Learning Classifier Systems for Understanding Patterns in Data

by

Yi Liu

A thesis
submitted to the Victoria University of Wellington
in fulfilment of the
requirements for the degree of
Doctor of Philosophy
in Computer Science.

Victoria University of Wellington
2021

Abstract

In the field of data-mining, symbolic techniques have produced optimal solutions, which are expected to contain informative patterns. Visualizing these patterns can improve the understanding of the ground truth of the explored domain. However, up to now, the symbolic algorithms struggle to produce optimal solutions for domains that have an overlapping distribution of feature patterns. Furthermore, the majority of problems have an overlapping distribution. Thus, novel techniques are needed to improve symbolic techniques' capacity to address overlapping domains, so that it is practicable to achieve the visualization of the underlying patterns.

Michigan-style Learning Classifier Systems (LCSs) are rule-based symbolic learning systems that utilize evolutionary computation to construct a population of rules to capture patterns and knowledge of the explored domains. LCSs have been applied to many data-mining tasks and have achieved good performance. Recently, employing visualization methods to improve the understanding level of models has become more and more popular in the data-mining community. In the LCSs field, visualization techniques orientated to explore feature patterns have not been developed. Investigating LCSs' models is commonly based on reading rules or viewing their distribution. However, LCSs' models may contain hundreds or even thousands of unduplicated rules, which makes the identification of patterns challenging.

Previously, Butz defined LCSs' optimal solutions as [O] sets, which are expected to utilize a minimal number of non-overlapping rules to present an explored domain completely and correctly. In the last two decades, rule compaction algorithms have been designed to search for [O]s by compacting LCSs' models, where rules that violate [O]'s definition are considered

as redundant rules. However, in many problems, an ideal [O] does not exist. Even if such a ruleset exists, redundant rules are often discovered in the compacted models. Furthermore, compaction often results in a decreased prediction performance. The LCSs community used to believe the reduced performance is an unavoidable and acceptable price for producing a compacted model. It is observed that across multiple LCS produced populations for the same problem, the irrelevant/redundant rules are varied, but useful/accurate rules are consistent. According to this observation, this thesis collects the common accurate rules and finds that for an arbitrary clean dataset, the common rules can form a determinate and unique solution, i.e. the proposed natural solution. A natural solution is composed of all consistent and unsubsumable rules under the global search space. A natural solution can correctly and completely represent the explored clean datasets. Furthermore, searching for natural solutions can produce concise correct solutions without reducing performance.

To visualize the knowledge in the solutions, three visualization methods are developed, i.e. Feature Important Map (FIM), Action-based Feature Importance Map (AFIM), and Action-based Average value Map (AFVM). FIM can trace how LCSs form patterns during the training process. Besides, AFIM and AFVM precisely present patterns in LCSs' optimal solution respectively regarding attribute importance and specified attribute's value's importance.

For the sake of efficiently producing natural solutions, a new type of compaction algorithm is introduced, termed Razor Cluster Razor (RCR). RCR is the first algorithm that considers Pittsburgh-style LCSs' conceptions to Michigan-style LCSs' compaction algorithms, i.e. compacting is based on multiple models. RCR was first designed for Boolean domains, then RCR has been extended to adapt to real-value LCSs' models. The conducted experiments demonstrated that natural solutions are producible for both Boolean domains and real domains.

The experiments regarding producing natural solutions lead this the-

sis to discover that LCSs have an over-general issue when addressing domains that have an overlapping distribution, i.e. optimal rules' encodings naturally share overlapped niches. The over-general issue causes LCSs to fail in maintaining the prediction performance, i.e. due to the optimal rules and over-general rules being repeatedly introduced and removed, training performance can never achieve 100% accuracy. This newly discovered issue inspires the development of the Absumption method as a complement to the existing Subsumption method, which continuously seeks to produce optimal rules by correcting over-general rules. Absumption not only improves LCSs' prediction performance in overlapping domains but also enables LCSs to employ hundreds of cooperative rules to precisely represent an explored domain.

The success of Absumption demonstrates LCSs' capacity in addressing overlapping domains. However, introducing Absumption to LCSs does increase the cost of computer resources for training, which results in a need for more efficient exploration. Furthermore, LCSs employed search techniques tend to evolve maximally generalized rules, rather than produce rules that do not overlap with the existing rules in the population. Thus, [O]s do not fit LCSs' fundamental search techniques. As a result, LCSs' accurate models often do not contain an optimal solution, which results in the LCSs produced models being poorly interpretable. This hampers LCSs from being a good data-mining technique. In this thesis, the Absumption and Subsumption based Learning Classifier System (ASCS) is developed. ASCSs consider natural solution as the search objective and promote the Absumption mechanism and Subsumption mechanism as the primary search strategies. Thus, it is possible to remove the traditional evolutionary search algorithms, i.e. crossover, mutation, roulette wheel deletion, and competition selection. Experiments demonstrated that ASCSs can use thousands of cooperative rules to represent an explored domain and enable easy pattern visualization that was previously not possible.

Acknowledgments

I would first like to express my deepest gratitude to my supervisors. Prof. Will N. Browne and Prof. Bing Xue, for their continuous support of my Ph.D. study. I much appreciate the amount of time and resources they have put into helping me with academic training. They professionally and convincingly guided me to conduct my research, including designing experiments, coding programs, writing reports, and publishing academic papers. I really appreciate their continuous encouragement and patiently supporting regarding improving my English ability. They especially spend enormous time improving my academic writing, providing insightful comments on my drafts of academic papers. They are the best and most patient supervisors that I have ever met during my time in study. Thanks sincerely for their help in supporting me in writing this thesis.

I am grateful to Victoria University of Wellington (VUW) for the school offers me an opportunity to study as a Ph.D. student at this school. It is impossible to conduct my experiments and complete this thesis without the school's material conditions. Besides, I wish to thank Prof. Meng Jie Zhang and the Evolutionary Computation Research Group (ECRG). Through the weekly meeting and seminars, the EGRG provides me a lot of useful academic training, i.e. exploring new techniques designed by ECRG members, presenting the published academic work in an appropriate method, and how to highlight the take-home message when writing academic papers. All these educational practices offered by the ECRG are highly valued, and I received lots of useful help from the group.

I want to pay my special regards to the staff of the School of Engineering and Computer Science (ECS). Thanks to Diana Siwiak for supporting my projects. Thanks to Mark Davies and Tim Exley for their technical supports. Thanks to Peter Andrea ("Pondy"), Patricia Stein and Sarah Dillion for their help and kindness. Besides, I want to thank my friends and colleagues in my office: Raghavender Deshagoni, Abhi Chatterjee, Trung Nguyen, Roshni Babu, and ZheMing Zhang.

Contents

Chapter 1

Introduction

In the field of data-mining, symbolic techniques, e.g. the Learning classifier systems (LCSs) have produced optimal solutions, which are expected to contain informative patterns. Visualizing these patterns can improve the understanding of the ground truth of the explored domain. However, up to now, the LCSs struggle to produce optimal solutions for some domains for unknown reasons. Therefore, the LCS community requires a pipeline that can allow LCSs to consistently and continuously produce optimal solutions for an arbitrary explored dataset, so that the visible patterns can be provided consistently.

This thesis considers the clean datasets (e.g. clean artificial boolean problems and UCI datasets) and provides visualization techniques for improving the understanding of the LCSs' learning scheme and the produced models. The visualized result demonstrates that LCS can generate solutions that reflect the ground truth for an arbitrary clean dataset. Furthermore, such an interpretable solution is determinate and unique for the same clean dataset. A hypothesis that explains this phenomenon is proposed, i.e. the natural solution hypothesis [1]. Based on the hypothesis, rule

[1] The natural solution hypothesis describes that LCSs can utilize all the unsubsumable and correct rules to build a fully representative model, such model naturally contain the ground truth of the explored domain, this model terms natural solution in the thesis.

compaction algorithms are developed. These algorithms can consistently produce optimal solutions by compacting multiple LCS trained population for the same domain. Furthermore, the thesis also conduct experiments regarding executing the compaction mechanism during the learning process rather than after learning. The experiments demonstrate that such a modification enables LCS to produce optimal solutions for more complex problems.

1.1 Problem Statement

Learning classifier systems (LCS [99]) are a paradigm of rule-based machine learning methods that contain a discovery component and a learning component. LCSs excel in evolving representative models that can reflect the underlying patterns of single-step problems, e.g. artificial Boolean problems. In the field of LCS, there are two most popular LCSs architectures, i.e. Michigan-style [91] and Pittsburgh-style [6]. Both Michigan-style LCSs and Pittsburgh-style LCSs employ a set of rules to represent an explored domain. However, these two types of LCSs have two main differences, i.e. the manner in applying crossover and forming results. In Michigan-style LCSs, a rule is an individual, which is the basic unit for applying evolutionary computation operators. The crossover method is executed to create new rules by randomly exchanging a part of the selected rules' encoding. After exploration, the output result includes all the preserved rules. As a comparison, in Pittsburgh-style LCSs, an individual is a ruleset. The crossover method is invoked for exchanging rules in different rulesets. When the exploration is completed, the best individual (ruleset) is selected as the output. Pittsburgh-style LCSs allow a high diversity of solutions that increases the possibility of detecting a set of classifiers that can accurately and completely cover all available instances of an explored domain. Thus, the community used to believe that Pittsburgh-style LCSs' production's prediction performance is more likely to be better

than Michigan-style LCSs' results [73]. This conception altered when the community realized that an LCS's optimal solution is independent of the learning schema, i.e. Butz et.al proposed the optimal solution [O] sets [2] and produced the [O] by compacting Michigan LCSs' results [17].

An [O] is expected to completely and correctly represent a domain with a minimal number of maximal generalized and non-overlapped rules. Precisely, an [O] is required to reach four characteristics simultaneously, i.e. completeness, correctness, minimality, and non-overlapping [3]. Furthermore, Butz et al. proposed that an [O] can reflect the nature of the explored domains with discernable patterns, i.e. [O]s have good interpretability. In general, there is no mathematical definition to assess a model's interpretability. Interpretability is the degree to which humanity can understand how a model makes a decision. Thus, visualization techniques are good methods to verify the interpretability of [O]s [51].

Visualizing patterns from LCSs' optimal solution is a difficult task. Although many LCS orientated visualization methods have been developed [90] [92], these methods present vague patterns for the majority of the tested problems. These visualized patterns are not simple to interpret, hard to be employed to reveal the ground truth of the explored domain. Thus, visualization methods that can precisely describe the underlying patterns are needed to improve LCS's practicality and suitability in data-mining.

Producing an optimal solution from LCSs directly or from compacting an LCS produced model are difficult tasks. Due to the stochastic nature of the employed evolutionary computation [8], an LCS produced model unavoidably contains problematic rules such as over-general rules and

[2] [O] is one of LCSs' ideal format solution that can completely and correctly represent an explored dataset with all the member rules are maximally generalized and non-overlapped with each other [17].

[3] Overlapping describes two rules, which have an overlapped represented niche in their condition part, e.g. 00#:0 and 0#0:0 are two overlapped rules, as they are overlapped at the instance 000:0. As a comparison, 00#:0 and 1#0:0 are non-overlapped rules.

over-specific rules [4]. Thus, compaction is essential processing for obtaining an LCS's optimal solution. However, the existing rule compaction algorithms cannot precisely distinguish good performing over-general rules with maximally generalized rules. Thus, these compaction algorithms frequently failed in compacting an LCS's model to its optimal format. Thus, rule compaction algorithms need to be improved in terms of identifying over-general rules [5] to ensure optimal solutions can be produced efficiently and consistently.

Ternary representation based LCSs [39] consider the majority of problems have an overlapping distribution, i.e. the maximally generalized rules naturally shares overlapped niches. However, LCSs suffers from addressing the majority of the overlapping domains due mainly to inappropriately replacing the maximally generalized rules by good performing over-general rules. In this scenario, an LCS produced model may lack important maximally generalized rules, which are an indispensable part of the optimal solution. Therefore, an efficient search technique is needed to introduce to LCSs' exploration phase to address over-general rules so that LCSs can produce optimal solutions for overlapping domains.

The main objective of this thesis is that precisely visualizing the patterns from LCSs' optimal solutions. Both Michigan-style LCSs and Pittsburgh-style LCSs can produce optimal solutions, but Pittsburgh-style LCSs are much more computationally costly. Thus, this thesis is mainly based-on Michigan-style LCSs so that optimal solutions of complex problems, e.g. the 70-bits Multiplexer problem, can be produced in a reasonable time.

1.2 Motivations

The main objective of this thesis is that precisely visualizing the patterns from LCSs' optimal solutions.

[4] over-specific describes rules that can be covered by other correct and higher general rules.
[5] over-general describes rules that violate the truth of the training set.

1.2.1 Challenge of Visualization Patterns

Symbolic learning systems (e.g. LCSs) are expected to produce models that contain informative patterns that can reveal the ground truth of the explored domains. Translating these underlying patterns into discernable graphs improves the interpretability of the models, which ensures the discovered knowledge to be easy comprehended by humans. However, visualizing patterns of LCSs produced models is a difficult task, especially in domains that have an overlapping distribution. These tasks are challenging due mainly to two reasons, which are informative patterns only exist in an LCS's optimal solution [42], and obtaining an LCS's optimal solution is difficult.

In the LCSs area, correct patterns can precisely reflect an explored domains' ground truth, e.g. class distribution, feature interaction, and redundant features. These patterns are formed by the distribution of the ratio of the specified attributes (i.e. attribute importance). Thus, the quality of member rules in a model determines the quality of the underlying patterns [39]. Redundant rules and irrelevant rules can change the underlying patterns, which may obscure important knowledge. Thus, only optimal solutions are expected to contain correct patterns. However, due to the stochastic characteristic of LCSs employed search operators, LCSs rarely directly produce an optimal solution. Thus, compaction is a necessary processing for obtaining an LCS's optimal solution.

Obtaining an optimal solution is difficult due mainly to good performing over-general rules and accurate maximally generalized rules are difficult to be precisely distinguished. A good performing over-general rules can achieve 98%-99% accuracy, which is only 1%-2% lower than a maximally generalized rule's accuracy, i.e. 100%. When prediction performance is similar, compaction algorithms are prone to preserve good performing over-general rules, due to these rules have an advantage in generalization level compares with maximally generalized rules. Replacing maximally generalized rules with over-general rules causes the original

training accuracy reduced after compacting, which can be observed in all the existing compaction algorithms. Thus, existing compaction algorithms cannot produce optimal solutions consistently.

1.2.2 Why Multiple-Populations

A Michigan style LCS utilizes a population of rules to explore and represent an observable domain. Introducing multiple-population based techniques to Michigan style LCSs is expected to ensure consistently producing optimal solutions and efficiently training because of the following reasons:

- A single LCS produced model unavoidably contains problematic rules, i.e. over-general rules and over-specific rules. In a set of models for the same problem, although the problematic rules are varied, the maximal generalized accurate rules are common. Thus, compaction algorithms, which apply to multiple models for the same problem, are expected to identify maximal generalized accurate rules successfully.

- LCSs allow a combination of complementary over-general rules and over-specific rules to achieve good prediction performance. An LCS's trained model is possible to reach the maximum accuracy without including all the member rules of the optimal solution. Hence, compacting can fail in detecting an optimal solution when running on a single LCS's model that even possesses high performance in prediction.

- LCSs place all rules in one population, such that when applying operators (e.g. matching, subsumption and, rule discovery) all the rules need to be considered. Placing rules in multiple populations that are categorised by the rules' advocated class label and possessed generalization levels, can ensure a quick rule selection when invoking

search operators. Thus, the computational cost for exploration is expected to be reduced.

1.2.3 Limitations of Existing Work

Previously, an LCS's optimal solution needed to satisfy the definition of [O], which requires to reach completeness, correctness, minimality, and non-overlapping simultaneously. Due to [O]'s non-overlapping characteristic, LCSs can not produce [O]s that can represent overlapping domains. However, LCS can produce completely accurate models for overlapping domains. Thus, a new format of LCSs' optimal solutions needs to be defined to adapt to overlapping domains.

Existing LCS-orientated visualization techniques consider using Hamming distance to cluster rules as a pre-processing for visualizing the patterns. However, there is no evidence to support that Hamming distance can reflect the correlation between rules. These techniques fail to produce visualization results that can precisely reveal the ground truth of the explored domain from LCSs produced models. Thus, new visualization techniques that can precisely describe underlying patterns in an LCS's optimal solution are needed.

Existing compaction algorithms are designed to maintain the correctness or completeness of a single model. Although the original model is compacted, the output results usually do not satisfy any definition of LCS's optimal solution formats. These models do not contain the patterns that can reveal the ground truth of the explored domain. Thus, compaction algorithms that specifically aim to produce LCS's optimal solutions need to be developed.

Overlapping domains naturally contain good performing over-general rules. Hence, LCSs cannot distinguish good performing over-general rules and accurate maximal generalized rules. When exploring overlapping domains, LCSs' models can be dominated by good performing over-general

rules. As a result, LCSs fail in producing models that can correctly and completely represent the target domains. This issue arises the demand of developing methods to assist LCSs to adapt to overlapping domains. Furthermore, LCSs are expected to produce optimal solutions directly with techniques that can precisely identify and remove good-performing overgeneral rules during the exploration phase.

1.3 Goals

The thesis is that considering multiple, instead of a single population, enables LCSs to produce optimal solutions, adapt overlapping domains and clearly visualize the underlying patterns in complex domains. To achieve this overall goal, a set of research objectives have been established to guide this research, which can be seen as follows.

1 Propose a new definition of LCS optimal solution format. This format is expected to ensure the LCS's optimal solution can completely and correctly represent an overlapping domain together with underlying patterns that can be interpretable.

   [O] sets have good interpretability, but this optimal solution format does not suit overlapping domains. In the LCSs field, the majority of the addressed domains have an overlapping distribution. LCSs are prone to utilize a combination of maximally generalized overlapped rules to represent the overlapping domains rather than a ruleset that only contains non-overlapped rules. As a result, [O] sets cannot be produced by compacting LCSs' models of overlapping domains.

2 Develop visualization techniques that are based on clustering rules by rules' generalization level[6], and then translating the patterns in

[6] generalization level describes a rule covered niche size, and this niche size can be quantified, i.e. the number of # in the condition part. A higher generalized rule is expected to have a larger number of # compare with a lower generalized rule.

the clustered rulesets to human-discernable graphs. These newly proposed visualization techniques are expected to improve the understanding level for both LCSs produced models and LCSs themselves. For instances, these visualization algorithms are supposed to achieve a good understanding of tracing how LCSs form patterns during the exploration phase, answering how features have interacted with each other, and explaining why model make a decision.

Existing LCS-orientated visualization techniques utilize Hamming distance based clustering. These techniques provide a vague visualization result, which cannot precisely describe the underlying patterns. Rules that have the same generalization level are expected to have a common strategy in selecting specified rules. Thus, visualizing the specification ratio of features based on the rulesets after generalization level clustering, are expected to reveal the ground truth of the explored domain precisely, e.g. class distribution, feature interaction, and redundant features.

3 Develop rule compaction algorithms that apply to multiple LCSs' models for the same problem. These compaction algorithms are expected to ensure optimal solutions of ternary representation based LCSs to be produced consistently. Then extend these compaction algorithms to upper and lower boundary representation based LCSs.

LCS's optimal solutions contain patterns that can reveal the ground truth of the explored domain. However, existing compaction algorithms are designed to produce a maximally compacted model rather than consider an optimal solution as an output. Furthermore, existing compaction algorithms apply to a single model, but a single accurate model can lack a part of member rules of an optimal solution. As a result, although the compacted results are accurate, the contained patterns are poorly interpretable due to lack of member rules of an optimal solution. Thus, applying compaction algorithms

to multiple models for the same problem is expected to precisely identify all the important member rules of an optimal solution.

4 Develop a new Pittsburgh-style LCS that involves an ensemble learning structure and considers a Michigan-style LCS as an individual rather than a ruleset. This newly proposed LCS is expected to adapt to overlapping domains by producing models that can correctly and completely represent arbitrarily observed datasets.

LCSs allow a combination of complementary over-general rules and specific rules to achieve good training performance. However, in a single population, subsumption hampers newly evolved correct specific rules from being introduced if the population has contained over-general rules, which can cover the specific rules. Especially, to overlapping domains, where good performing over-general rules commonly exist. Thus, LCSs' produced models are frequently be dominated by over-general rules when exploring overlapping domains. As a result, LCSs fail in producing models that can completely and correctly represent an overlapping dataset. Ensemble learning, (i.e. boosting) can ensure all the evolved over-general rules to be completely removed periodically by invoking rule compaction algorithms at the end of each layer. Thus, introducing ensemble learning to LCSs is expected to reduce over-general rules' negative influence in preventing important specific rules from being introduced.

5 Develop a specify operator that can precisely identify and remove over-general rules during the exploration phase for Michigan-style LCSs. This specify operator is expected to create important specific rules by correcting over-general rules to ensure optimal solutions to be produced consistently and efficiently.

Existing specify operators are designed to reduce the identified over-general rules' numerosity rather than directly remove over-general

rules. Thus, over-general rules still have the potential to dominate a population when good performing over-general rules exist, e.g. in overlapping domains. Existing over-general rules will prevent important specific rules being introduced because of the subsumption mechanism. Thus, immediately removing the identified over-general rules is expected to ensure the newly evolved important specific rules can be inserted successfully so that Michigan-style LCS can address overlapping domains.

6 Develop a new LCS that considers optimal solutions as the final output. This LCS promotes the specify operator, subsumption, and informed mutation as the primary search techniques, removes other obsolete evolutionary search algorithms, i.e., crossover, mutation, roulette wheel deletion, and tournament selection. This LCS is expected to directly produce optimal solutions for the explored domains consistently and efficiently.

Due to the stochastic nature of LCSs employed search strategies, over-general rules are introduced repeatedly. It is hypothesized that transforming the search process from stochastic to deterministic can benefit LCSs in evolving optimal solutions and removing the need to tune search parameters to the problem. Furthermore, existing LCSs invoke compaction algorithms after the exploration phase. It is expected that applying compaction algorithms during the exploration phase can enable LCSs incrementally to form an optimal solution during the exploration phase.

1.4 Organization of the Thesis

The rest of this thesis is organised as follows. Chapter 2 presents the literature review of related work, the experimented benchmarks, and the parameters settings. The main contributions of the thesis are presented in

Chapters 3-7, shown in Figure 1.1. Chapter 8 concludes the thesis.

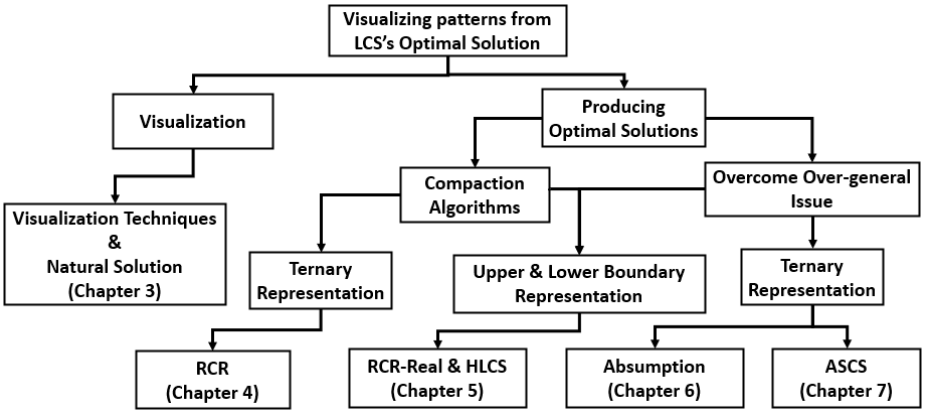


Figure 1.1: The overall structure of the contributions.

Chapter 2 presents the essential background of Learning Classifier Systems (LCSs) in detail with the four most important aspects, i.e. the basic structure of rules, the fundamental learning schema, the searching strategies, and the optimal ruleset. It also reviews the most popular rule compaction algorithms during the last two decades. It then discusses the tested benchmarks and the parameters setting.

Chapter 3 proposes a new definition of LCS's optimal solution, which is termed *natural solution* that complements the [O] hypothesis. It then proposes three LCS-orientated visualization techniques, i.e. Feature Important Map (FIM), Action-based Feature Importance Map (AFIM), and Action-based average Value Map (AFVM) respectively aim to trace how patterns are constructed, reveal how features have interacted, and explain the cause of a model's decisions. A set of experiments are conducted on commonly used artificial Boolean domains of varying difficulty. The results are then presented and analysed.

Chapter 4 proposes three rule compaction algorithms, i.e. Razor Cluster Razor (RCR), Razor Cluster Razor 2 (RCR2), and Razor Cluster Razor 3 (RCR3) for ternary representation based LCSs. These algorithms inno-

vatively handle the compaction based on multiple LCS produced models, rather than applying the compaction to a single model like other standard compaction algorithms. RCR, RCR2, and RCR3 respectively aim to produce [O] set, *natural solution* from well-trained models, and *natural solution* from insufficiently trained models. The proposed algorithm is then examined and compared with the existing most popular compaction algorithms, i.e., Compact Ruleset Algorithm (CRA), Fu's approaches (Fu1, Fu2, and Fu3), Alternative Reduction Algorithm (CRA2), Kharbat's approach (K1), Quick Rule Compaction (QRC), and Parameter Driven Rule Compaction (PDRC) in terms of the capacity of produce optimal solutions, the prediction performance, and the computational time.

Chapter 5 introduces an altered version of RCR, termed RCR-Real, that can adapt to real-valued domains to produce *natural solutions*. RCR-Real is orientated to upper and lower boundary representation based LCSs. Then, this chapter describes how to adapt FIMs to visualize the underlying patterns in RCR-Real compacted models. This chapter also proposes a novel LCS termed Hierarchical Learning Classifier System (HLCS) to overcome domains have an over-general issue. The HLCS's novelty is that it considers rule compaction as a part of the exploration stage, rather than invoking compaction after exploration has finished. The performance of RCR-real and HLCS is examined on a set of UCI datasets that have an over-general issue.

Chapter 6 proposes a new operator termed as *informed mutation* that aims to search for maximally generalized rules by correcting over-general rules. Then, based on informed mutation, a new mechanism named *Absumption* is proposed to enable the LCSs to produce a single model that contains all the member rules of a natural solution. This complements the subsumption operator that was designed for improving over-specific populations. Besides, a new version of RCR terms Razor Cluster Razor in Single Ternary alphabet (RCR-ST) is also described that aims to extract the natural solution from an Abusmption based LCS produced results. Their

performances are then examined on artificial Boolean domains of varying difficulty.

Chapter 7 proposes a novel LCS that considers *natural solution* as the search objective, i.e. Absumption and Subsumption based learning Classifier System (ASCS). ASCS promotes Absumption, Subsumption, and informed mutation as the primary search strategies, rather than the standard evolution operators such as crossover, mutation, roulette wheel selection, and tournament selection. ASCS's performance is then examined and compared with XCS and UCS on artificial Boolean domains of varying difficulty in terms of the prediction performance, the computational time and the interpretability.

Chapter 8 summaries the work and draws overall conclusions of the thesis. Contributions of the thesis are ascertained. It also suggests some possible future research directions.

1.5 Publication List

1 Yi Liu, Will, N. Browne and Bing Xue. "Visualisation and optimisation of learning classifier systems for multiple domain learning". Asia-Pacific Conference on Simulated Evolution and Learning. Springer, 2017. pp. 448-461.

2 Yi Liu, Will, N. Browne and Bing Xue. "Adapting bagging and boosting to learning classifier systems". International Conference on the Applications of Evolutionary Computation. Springer, 2018. pp. 405-420.

3 Yi Liu, Will, N. Browne and Bing Xue. "Hierarchical Learning Classifier Systems for Polymorphism in Heterogeneous Niches". Australasian Joint Conference on Artificial Intelligence. Springer, 2018. pp. 397-409.

4 Yi Liu, Will, N. Browne and Bing Xue. "Absumption to complement subsumption in learning classifier systems". Proceedings of the Genetic and Evolutionary Computation Conference. 2019. pp. 410-418.

5 Yi Liu, Will, N. Browne and Bing Xue. "Absumption and subsumption based learning classifier systems". Proceedings of the Genetic and Evolutionary Computation Conference. 2020. pp. 368-376.

6 Yi Liu, Will, N. Browne and Bing Xue. "Visualizations for Rule-Based Machine Learning". Natural Computing 2020.

Chapter 2

Background

This chapter describes Learning Classifier Systems (LCSs) [98] in detail with the five most important aspects related to this work. The first aspect is about the fundamentals of LCSs including the structure of rules, the learning schema [61]. The second aspect illuminates what are LCSs' over-specific and over-general issues, together with surveys the LCSs' searching strategies [67]. Afterward, optimization in rulesets of LCSs is introduced by reviewing the definition of LCSs' ideal optimization solutions and the most popular rule compaction algorithms in the last two decades [58]. The fourth aspect describes how visualization techniques apply to LCSs in order to present LCSs' explored results in a human-discernable manner [93]. In the fifth aspect, a brief description of benchmark problems and datasets is provided. The training parameters setting for this project are also presented.

2.1 Learning Classifier System (LCSs)

Learning Classifier Systems (LCSs) [35] are a class of adaptive rule-based learning systems that explore problems through identifying a population of attribute-dependent rules that collectively store knowledge in a niche-based manner. LCSs' rules are designed to be human-readable, be com-

pact, and contain important information regarding the represented domains. LCSs have forty years of history and have been applied to identify patterns in many different fields, including multilocus interaction [87] [94] , non-linear attribute interactions [100], heterogeneity in artificial Boolean domains [40] [42], epidemiological data sets [89] [90], and other diverse domains [57] [66] [71] [84] [88]. These works demonstrate that LCSs are promising techniques for highlighting and characterizing the underlying patterns of "attribute to action" association in data mining tasks.

In general, there exist two different types of Learning Classifier Systems (LCSs), which are the Michigan-style LCSs and Pittsburgh-style LCSs [22]. Holland [36] designed the Michigan-style approaches, while De Jong and his students [77] [78] introduced the Pittsburgh-style LCS. The two most fundamental distinctions between these two styles are the structure of an individual and the output result. In Michigan-style approaches, an individual is a classifier, the system attempt to evolve a population of classifiers to cooperatively represent an explored domain, and the final output is the population (shown in Figure 2.1). As a comparison, in Pittsburgh-style approaches, an individual is a set of classifiers, a population consists of a set of individuals, rules can crossover between individuals, and the best individual will be selected as the final output (shown in Figure 2.2).

One of the most popular accuracy-based Michigan-style LCSs is XCS [98], which is a rule-based symbolist learning system that employs Evolutionary Computation (EC) techniques [64] to construct a population of rules to represent an explored domain. XCSs have been frequently applied to data mining tasks. One of the main reasons is that evidence shows that XCSs can produce models containing human-discernible patterns [9].

Previously conducted works have demonstrated that LCSs are suitable techniques for highlighting and characterizing the underlying patterns of attribute association for addressing data mining tasks, i.e. how different attributes work together to determine an output action [21]. However, in practice, an LCSs' model can have thousands of rules, which makes the

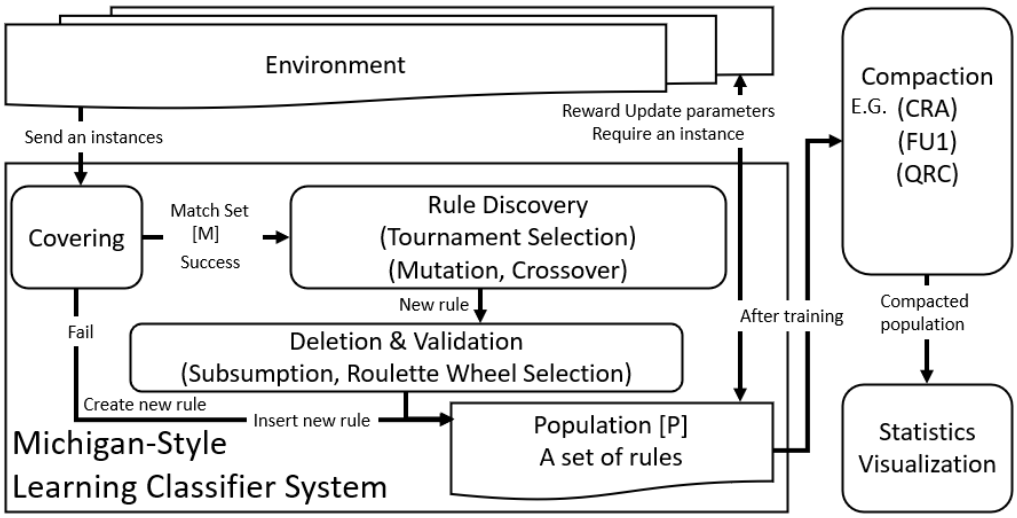


Figure 2.1: The standard learning schema of the Michigan-style LCSs where a population [P] preserves all the evolved rules, and evolutionary computation is used as the primary search technique. After training, the trained population will go through a compaction process to remove redundant rules. Then statistical analysis methods will be applied to the compacted ruleset in order to translate the underlying knowledge into human discernable patterns.

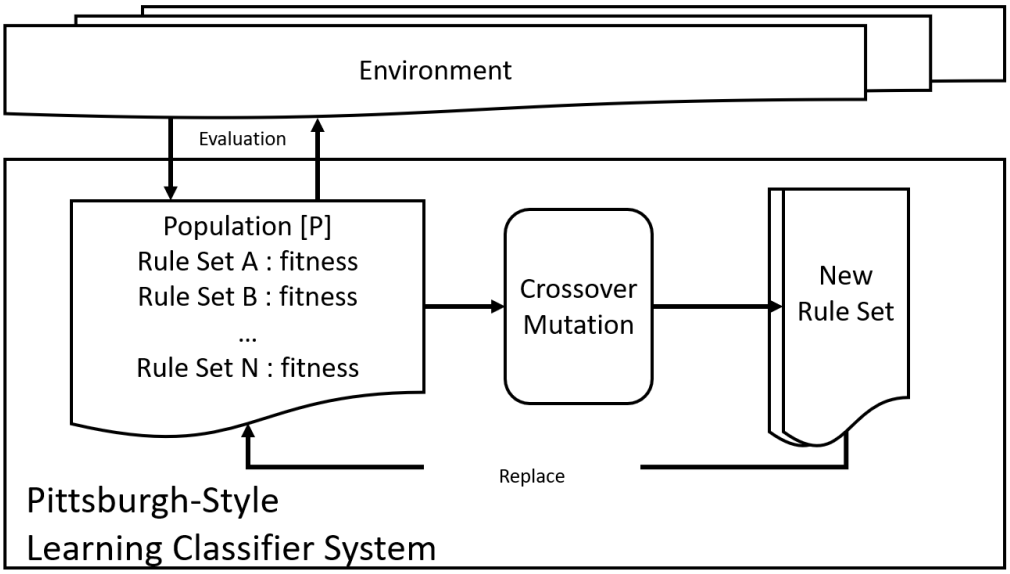


Figure 2.2: A Pittsburgh-style LCS considers a ruleset as an individual, a population has a set of rulesets, each ruleset evolves non-cooperatively, and the best ruleset is selected as the final output.

patterns underlying the model are difficult for humans to comprehend [17].

Furthermore, due to Michigan-style LCSs continuously introducing new rules during the whole exploration phase, a trained model unavoidably contains a considerable number of problematic rules (over-general rules or over-specific rules) [52]. These problematic rules frequently obscure the critical patterns [63]. Thus, LCSs frequently require post-processing to extract useful knowledge. For example, the LCSs' trained population needs to be compacted. Then statistical methods, such as calculating the specification ratio of attributes can be applied to transform the underlying knowledge to discernable patterns [45].

2.1.1 Michigan-style LCS's Rules (Classifiers)

Michigan-Style LCSs [99] reserves the discovered knowledge in a set of rules, see Figure 2.3. Each rule is composed of a condition part, an action part, and a set of training parameters that records the training history and training performance. The condition part represents the niches (environmental instances) that this rule covers. LCSs support many different types of representation format for describing the covered niches.

The ternary representation is one of the most common representation formats that is employed by LCSs. It is commonly used to address Boolean domains and classification tasks based on non-continue value problems. In each condition, the ternary representation encodes all the environmental attributes and labels these attributes with one of the two states, either specified or generalized. A specified attribute will be assigned with one of the plausible values of this attribute in the training set, e.g. in Boolean domains 0, 1. On the other hand, a generalized attribute will be marked with a "do not care" symbol #. # automatically matches the labeled attribute's all plausible values, e.g. 0,1 in Boolean classification tasks. A rule matches an instance when this instance and the rule have the same value among

all the specified attributes.

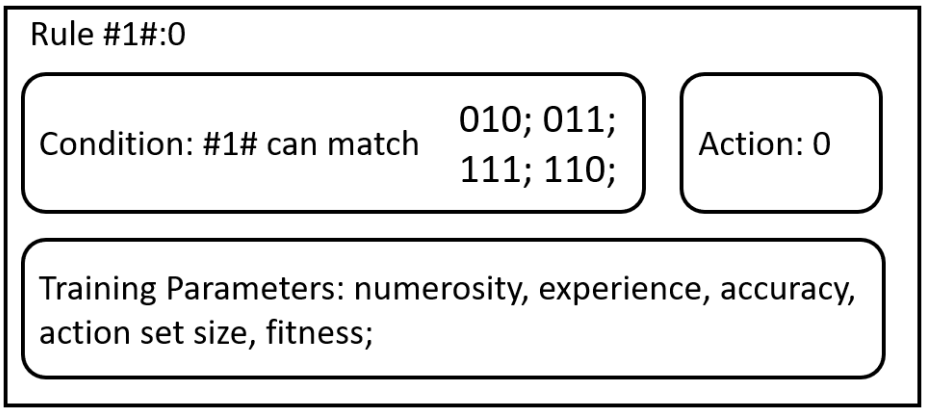


Figure 2.3: A LCSs' rule, where the condition part employes the ternary representation format, action is specified as 0.

The action part specifies the action that this rule advocates. For example, in binary classification tasks, action can be either 0 or 1 [41]. In multi-label classification, a rule's action also can be assigned a set of labels, e.g. [$label_0$, $label_1$...] [3].

The most common training parameters in LCSs' rules are *numerosity*, *experience*, *accuracy*, *action set size*, and *fitness* [56]. *numerosity* indicates the number of duplicates to a rule. In LCSs if rules have the same encoding and advocate the same action, then these rules are considered as duplicated. *experience* relates to a rule's training time, and experience increases by one each time after a rule covers an instance. *action set size* records a rule's approximate niche size. Note the value of this parameter is related to but not equal to the number of covered environmental instances in the training set of a rule. *accuracy* is a function of error, and *fitness* shows a rule's potential performance based on the above parameters, which also need to be taken into account to determine the worth of a rule to a solution.

2.1.2 Michigan-style LCS's Learning Schema

Michigan-style LCSs' learning schema is shown in Figure 2.1. An LCS has a population [P] that possesses all the evolved rules that capture the knowledge of the observed environment [75], e.g. patterns of attribute association or the importance of each attribute in determining the output action. LCSs' learning mechanism can be divided into three main parts, including covering, rule discovery, and rule validation & deletion.

The learning process of LCSs is step-wise, and the basic unit is an iteration. Each iteration of LCS begins with receiving an environmental instance, and then the covering method is invoked when forming a matching set [M] from [P] with rules, which can cover the observed instance if no sufficient rules can be found. The covering will randomly generalize several attributes of the instance to introduce new rules to [P], so that this instance is matchable by [P] next time. Note, the number of new rules is dependent on the searching strategies of an LCS. If LCSs consider a complete map as the objective, this number is equal to the number of actions of the explored problem. Instead, if the best action map is the objective, in this step, covering creates one rule that advocates the instance's associated action.

After covering, if [M] possesses at least one rule, LCSs apply the rule discovery process to try to evolve better new rules. The assumption is that parents have good performance are more likely to produce offsprings that can be better adapted to the explored domains, generation by generation. Thus, the tournament selection method is frequently considered as the strategy to select the parent rules to evolve offspring rules by EC methods, i.e. crossover and mutation.

EC methods have a stochastic characteristic, which leads to the newly created offsprings being potentially redundant. Thus, before inserting into [P], offsprings need to be verified. In [P], if any rule is the same or able to subsume an offspring (the subsumption mechanism will be described in Section 2.2.1), then the numerosity of this rule will increase by one to

substitute the process of inserting the offspring into [P].

After the rule discovery process, LCSs count the total *numerosity* of all rules in [P], i.e. total number of rules. When the value of this total number violates the capacity limitation of [P], LCSs execute the rule deletion method. Traditionally, LCSs consider roulette wheel selection (RWS) as the technique to implement the rule deletion in terms of selecting candidate rules. When a rule is selected by RWS, LCSs reduce the numerosity of this rule by one. A rule is removed from [P] when its numerosity decreases to zero. LCSs repeatedly execute rule deletion until the total number of numerosity satisfies the [P]'s capacity limitation.

LCSs complete the training process when a pre-defined number of iterations to explore the target problem has been performed or some other criteria have been satisfied, e.g. a set training performance level is reached.

At this stage, the produced [P] unavoidably contains many rules that are not trained sufficiently. This leads to the [P] containing many redundant rules (over-specific rules or over-general rules), which may obscure the critical underlying knowledge. Thus, rule compaction algorithms, e.g. compact ruleset algorithm (CRA) [101], Fu's first approach (FU1) [34], or quick rule compaction (QRC) [85], have been applied to remove these problematic rules from [P]. In most cases, compaction can significantly reduce the number of rules in [P]. As a result, the underlying knowledge becomes apparent. Furthermore, statistical analysis methods and visualization techniques can be applied to the compacted [P] so that the underlying knowledge can be translated into human-discernable patterns [1].

The community used to believe that the Pittsburgh-style LCSs are better than Michigan-style LCSs regarding the produced models' prediction performance, i.e. Pittsburgh LCSs are more likely to produce solutions

[1] There is no mathematic definition of models' interpretability. Interpretability is a degree to which humanity can understand the cause a model makes a decision. Human-discernable patterns mean the patterns in LCSs' model can be represented in a simple understandable graph.

that can represent an explored domain correctly and completely [73]. This conception altered when the community realize that an LCS's optimal solution is independent of the learning schema, i.e. Butz et. al proposed the optimal solution [O] sets and produced the [O] by compacting Michigan LCSs' results [17].

2.2 LCSs' Learning Strategies

LCSs are designed to construct a population of rules to represent an explored domain. Among all the available rules under the global search space, there exist two types of problematic rules, i.e. over-specific rules and over-general rules. These problematic rules can lead to many issues. For example, over-specific rules increase the [P]'s unnecessary diversity, obscuring the underlying patterns. Besides, over-general rules add biases to the [P], reducing the model's prediction performance. Therefore, to produce compact and accurate models, LCSs have developed many learning strategies to avoid, identify, and remove the problematic rules, e.g., subsumption, *specify* operator, and roulette wheel deletion.

2.2.1 over-specific Rules & Subsumption

In LCSs, "over-specific" describes rules, whose encodings are redundantly specific. For example, in the 6-bits Multiplexer problem, the rule $0000\#\# : 0$ is an over-specific rule since it can be subsumed by the optimal rule $000\#\#\# : 0$ (shown in Algorithm 1)). Therefore, the over-specific rules are expected to be replaced by other higher generalized rules without reducing prediction performance. LCSs employ the subsumption mechanism to remove the over-specific rules to keep a population's compactness.

In general, LCSs have two subsumption mechanisms, i.e. action set subsumption (shown in Algorithm 2) and genetic algorithm (GA) subsumption (shown in Algorithm 3). The action set subsumption effects

Algorithm 1: Judge whether rule1 can subsume rule2

Input: rule1's encoding en1;
rule2's encoding en2;

```
for attribute_i ∈ en1 & attribute_i ∈ en2 do
    if en1.attribute_i!=en2.attribute_i & en1.attribute_i!=# then
        return False rule1 cannot subsume rule2
    end
end
return True rule1 can subsume rule2
```

on rules in the $[A]$ that aims to address the over-specific rules in $[A]$. At the end of each iteration, when over-specific rules are identified, their numerosity will reduce by one, and the rule will be removed once its numerosity is reduced to zero. The GA subsumption prevents newly created over-specific rules to be introduced to the population. When an offspring has been identified as an over-specific rule, rather than being inserted into the population, the existing more general rule's numerosity will increase by one.

Normally the GA subsumption is always activated. Otherwise, the produced population cannot be dominated by the most general rules. The action set subsumption frequently replaces the optimal rules by good performance over-general rules. Thus, in domains that have overlapping distribution, activating action set subsumption frequently reduces the produced models' prediction performance. Therefore, for the majority of tasks, the action set subsumption is not activated.

2.2.2 over-general Rules & Specify Operator

over-general describes rules that matched instances with different actions. For example, in the 6-bits Multiplexer problem, the rule 00####:0 is an over-general rule, which covers the instance 001000:1. The over-general

Algorithm 2: Action set subsumption

```
Input: an action set [A];
foreach rule ∈ [A] do
    foreach rule' ∈ [A] except rule do
        if rule can subsume rule' then
            rule.numerosity++;
            rule'.numerosity- -;
            if rule'.numerosity==0 then
                remove rule'
            end
        end
    end
end
```

rules always negatively influence a model's prediction performance. However, LCSs are designed to produce a model as general as possible. As a result, over-general rules are unavoidable to be produced.

Previously, Lanzi [59] designed the *specify* operator to counteract the negative influence of the over-general rules. When one over-general rule is identified, the *specify* reduces the problematic rule's numerosity by one, and then creates a new rule by specifying one of the over-general rule's generalized attribute with one plausible value. An over-general rule will be removed, once its numerosity has been reduced to zero.

2.2.3 Roulette Wheel Deletion

LCSs utilize Evolutionary Computation (EC) as the main technique for knowledge discovery, but various LCSs may employ different EC search strategies, which results in the difference in LCSs' search performance. The search strategy determines the number, ability, and diversity of rules in a [P] and hence the interpretability.

Algorithm 3: rule discovery subsumption

Input: a population $[P]$;
offsprings created by rule discovery $[offsprings]$

```
1 foreach rule ∈ [offsprings] do
2     foreach rule' ∈ [P] do
3         if rule == rule' or rule' can subsume rule then
4             rule'.numerosity++;
5         end
6     end
7 end
```

XCSs [99] are the most popular reinforcement learning LCSs, as being accuracy based makes them suited to discovering patterns in datasets. XCSs employ crossover and mutation as the main searching strategies for evolving rules. The subsumption method is the most important secondary search strategy for XCSs as it serves as a preventive method against over-specific rules. The state that activates subsumption to discard newly generated rules is when the population has had a sufficiently trained rule that is correct [2] and more general than the new rule.

XCSs employ roulette wheel selection (RWS) for selecting rules to delete when the population grows too much. In XCS, this rule removal strategy considers the rule's numerosity, generalization level and accuracy. Thus, XCSs will primarily delete less-general rules, when there are no conspicuous [3] inaccurate rules in the population. Such a rule removal strategy enables XCSs' models to be compacted but this also leads to issues, e.g. XCSs frequently omit important specific rules and incorrectly replace important specific rules with high-performance over-general rules.

When assessing whether a rule is over-general, XCSs utilize the error

[2] correct means a rule and all its matches instance support the same action

[3] XCS's RWS utilize a rule's action set size, numerosity, and fitness, a conspicuous inaccurate rule is expected to have a relatively low value in fitness.

threshold ϵ_0 [4], which is a constant number regardless of the assessed rule's generalization level. When ϵ_0 is set weakly,i.e. the assigned value of ϵ_0 is lower than the number of instances (niches) that represented by good performing over-general rules, over-general rules can be incorrectly considered as verified accurate rules by the subsumption mechanism, which will hamper the important specific rules from being inserted into the [P]. This is another reason why XCSs may fail in identifying over-general rules.

As a comparison, UCSs [5] [10] are also accuracy-based but learn under a supervised scheme, when labeled data is available. This enables UCSs to precisely assess each rule's accuracy. Thus, all the UCSs' main search strategies are pure accuracy-based, including the RWS and the rule discovery mechanism that is composed of crossover and mutation. UCSs also consider subsumption as a secondary search strategy for preventing [P]s from introducing over-specific rules. UCSs give priority to remove over-general rules and protect all the correct rules. Thus, UCSs can efficiently reach high performance in prediction, but the produced models unavoidably contain many redundant rules, due to the lack of methods for efficiently removing over-specific rules that evolved at early generations.

2.3 LCS's Optimal Solution

Butz and Kovacs [18] considered that accuracy-based LCSs are designed to search for correct, maximally general rules, which enables the produced models to be accurate and compact. Based on this hypothesis, Butz and Kovacs [18] proposed an optimal ruleset [O] to represent the optimal model of LCSs, see Figure 2.4. A proper [O] should simultaneously satisfy four characteristics: completeness, correctness, minimality, and non-overlapping.

Completeness and correctness require that [O]s can precisely represent the whole problem domain. Furthermore, minimality and non-overlapping

[4] The error threshold under which the accuracy of a classifier is set to one.
[5] UCS stands for sUpervised Classifier System

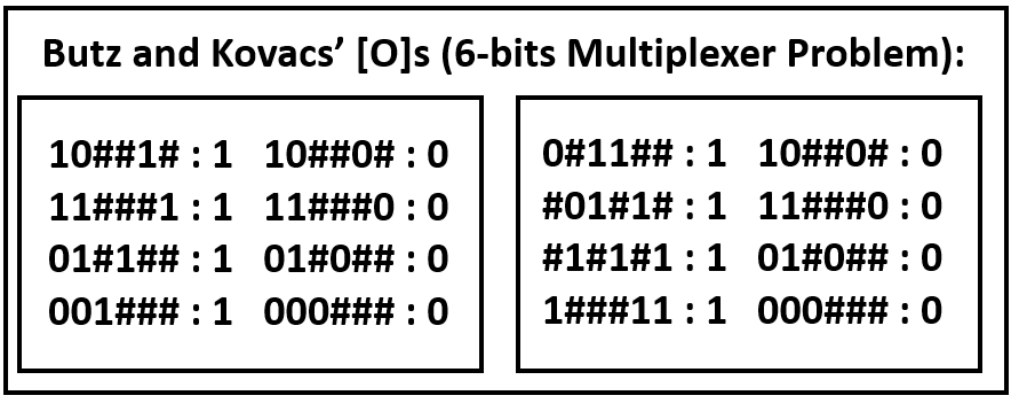


Figure 2.4: Two samples regarding Butz and Kovacs' optimal ruleset [O] of the 6-bits Multiplexer problem. An [O] has the characteristics of completness, correctness, minimality, and non-overlapping. Note, there can be alternative [O]s for the same problem [17].

indicate that redundant rules are intolerable to [O]s. Ideal [O]s have been found in the Multiplexer problem from the compacted models of both XCSs and UCSs. However, in practice, LCSs' trained [P]s rarely contain [O]s. This is because LCSs do not consider the non-overlapping condition as a searching restriction when introducing new rules to [P]. Thus, unduplicated rules can have an overlapped condition part. Removing overlapped rules may result in a [P] that violates the completeness characteristic, leading to failure regarding any attempt to compact [P] to an [O].

2.3.1 LCSs' Compaction Algorithms

When an LCS ceases its training process, as new rules were continuously introduced, the produced model unavoidably contains a large number of rules that are either over-general or over-specific. These problematic rules obscure the valuable underlying knowledge in LCSs' models, which hampers humans from better understanding the patterns in these problems by

identifying these models. Thus, LCSs frequently employ rule compaction algorithms, which serve as post processing in order to improve the interpretability of the produced models by excluding unnecessary rules.

In general, rule compaction algorithms can be categorized into two collections according to their primary compaction objective. The first collection considers preserving the original training accuracy as the primary compaction objective. This collection includes algorithms such as Compact Ruleset Algorithm (CRA) [101] and Fu's approaches, e.g. Fu1, and Fu3 [34]. When compacting, CRA, and Fu's approaches remove rules sequentially, and accuracy re-evaluation is executed after each removal. This strategy can maximally preserve the original accuracy of the model, as incorrect removal can be promptly informed by the resulting decreased accuracy from the re-evaluation. In many cases, these rule compaction algorithms successfully improve LCSs' models' readability by removing the majority of the problematic rules. However, the re-evaluation procedure is accompanied with the issue of expensive computation cost, which impedes the adoption to domains with a large training set or [P] containing a large number of rules, e.g. the 37-bits Multiplexer problem, where the training set includes around 34 billion different instances, and the 14-bits Majority-On problem, where [P] may contain more than 10 thousand rules.

The second collection emphasize is on maintaining the original models' completeness regarding covering [6] the training set. This collection includes the Alternative Reduction Algorithm (CRA2) [25] and its analogs, e.g. Kharbat's approach (K1) [45] and Urbanowicz's methods [85], i.e. Quick Rule Compaction (QRC), and Parameter Driven Rule Compaction (PDRC). Distinct from the first category, these algorithms remove the re-evaluation procedure of prediction performance, as sustaining the original accuracy is no longer the primary objective. CRA2 and its analogs utilize the environmental instances to determine whether a rule should be

[6] Covering ensures that all environment instances are matched by at least one rule

Algorithm 4: Compact Ruleset Algorithm (CRA)

Input: all rules in a trained population P;
all instances in the training set T;
Output: final compacted ruleset *F_Set*

```
Rank rule ∈P by numerosity or experience (descending order);
Stage One:
Original_Accuracy= Accuracy(P, T);
Subset= empty ruleset;
i=0;
Rest_Accuracy=0;
while Rest_Accuracy<Original_Accuracy do
    Add ith rule to Subset;
    Rest_Accuracy=Accuracy(Subset, T);
    i++
end
Stage Two:
foreach rule ∈ Subset do
    Remove rule from Subset;
    Rest_Accuracy=Accuracy(Subset, T);
    if Rest_Accuracy < Original_Accuracy then
        Add rule to Subset;
    end
end
Stage Three:
Rank rule ∈ Subset by the number of each rule matched instances
  (descending order);
while Size(Subset) > 0 and Size(T) >0 do
    Add first rule in Subset to F_Set;
    Remove first rule in Subset;
    for instance ∈ T do
        if Match(F_Set, instance) then
            Remove instance in T
        end
    end
end
```

retained. When compacting, all the instances in the training set will be sequentially reviewed to select and preserve the most suitable rule for each instance. Due to omitting the re-evaluation processing, the execution efficiency of compacting is greatly improved so that CRA2 and its analogs can better adapt to domains that have a large number of instances compared with CRA and Fu's approaches. However, due to the employed loose rule removal criterion, CRA2 and its analogs are under the risk of replacing important specific rules with over-general rules, especially the over-general rules that possess high accuracy performance. This drawback may result in the reduced interpretability of the compacted [P]s as the mistakenly kept over-general rules are likely to obscure the underlying patterns of the compacted [P]s. Incidentally, after Urbanowicz's work [85], the enthusiasm for researching LCSs' rule compaction algorithms waned as the community considered it a fully developed research area.

2.3.2 Compact Ruleset Algorithm (CRA)

CRA [101] starts by ranking rules according to experience or numerosity in descending order (shown in Algorithm 4). The ranking is an efficient method to identify the most promising candidates to keep in a [P]. This is because only good rules are expected to survive constantly under evolutionary competition. Then, compared with inferior rules, good rules have a higher value for trained parameters, e.g. experience or numerosity, which has a positive correlation with the number of survived generations.

After ranking, CRA is composed of three stages that are applied in sequence. The priority of the first two stages is to preserve the original accuracy when compacting. This demand necessitates step-wise accuracy evaluation to ensure the impartiality of removing each rule. Stage one builds a subset by sequentially receiving a rule from the original [P] in a top-down manner (ranked by numerosity). The process of receiving rules ceases when the subset reaches the accuracy of the [P]. Stage two

Algorithm 5: Stages 1&2 in Fu1

Input: all rules in a trained population P;
all instances in the training set T;
Output: a set of rules *S_Set*

```
Stage One:
Rank rule ∈ P by numerosity(descending order);
Original_Accuracy= Accuracy(P, T);
Rest_Accuracy=Original_Accuracy;
while Rest_Accuracy>=Original_Accuracy do
    Remove last rule in P;
    Rest_Accuracy=Accuracy(P, T);
end
Add the newest removed rule to P;
Stage Two:
foreach rule ∈ P do
    Remove rule from P;
    Rest_Accuracy=Accuracy(P, T);
    if Rest_Accuracy < Original_Accuracy then
        Add rule to S_Set;
        Original_Accuracy=Rest_Accuracy
    end
end
```

removes each rule in the subset in turn. After each removal, an evaluation is invoked to assess the current subset for accuracy, where a previously removed rule will be added back if the accuracy decreases. Maximally removing rules from a subset without weakening the capacity of covering training instances is the objective of the third stage. Stage three begins with iterating the training instances to count the number of covered instances for each rule. Then, rules are ranked by the number of their covered instances in an descending order. After ranking, rules in the subset are selected in sequence from top to bottom in the rank to construct the final output. The process of construction terminates when the final output can completely cover the entire training instances.

2.3.3 Fu1

Fu1 [34] is similar to CRA, and also contains three stages. The first two stages also involve a step-wise accuracy evaluation. Stage one starts by ranking rules in the descending order of numerosity. Then, in the ranked [P], rules are continuously removed from the bottom to top order based on numerosity, which differs from CRA. After each removal, the accuracy of the current [P] is evaluated. The process of removing is terminated when the accuracy decreases, at which point, the rule that causes the decrease is reinserted.

Stage two aims to find a subset of the [P], where each member rule is important to maintain the prediction performance. In this stage, the rules in the [P] are removed by turns to identify appropriate rules to place into the output subset. After each removal, the accuracy of the remaining rules is tested. The most recently removed rule will be preserved in the subset if the removal of this rule causes accuracy to decrease. After all the rules have undergone this process, the subset of the preserved rules is passed to stage three, which is the same as the last stage of CRA.

Fu1 follows the CRA's priorities and principles of compaction. In Fu1's

Algorithm 6: Incremental deletion procedure in Fu2

Input: a trained population P;
all instances in the training set T;
Output: a set of rules *S_Set*

```
foreach rule ∈ P do
    Remove rule from P;
    Rest_Accuracy=Accuracy(P, T);
    if Rest_Accuracy < Original_Accuracy then
        Original_Accuracy=Rest_Accuracy
    end
    else
        rule.numerosity minus one ;
    end
    if rule.numerosity> 0 then
        Add rule to S_Set;
    end
end
```

first stage, rules are removed from the bottom rather than the top (based on numerosity in the descending order). This is due to when a population lacks training, the rules for triggering termination of rule removal frequently are located near the bottom position in the ranked population. The altered direction of removing improves Fu1's efficiency when addressing the poorly trained populations. The alteration in Fu1 Stage two, as shown in Algorithm 5, reduces the computational complexity compared with CRA.

Algorithm 7: Stage3 in Fu3

Input: remaining rules after first two stages R_Set;
all instances in the training set T;
Output: R_Set

```
1 Rank rule ∈ R_Set by fitness or experience (descending order);
2 Original_Accuracy=Accuracy(R_Set, T);
3 foreach rule ∈ R_Set do
4 |   Remove rule from R_Set;
5 |   Rest_Accuracy=Accuracy(R_Set, T);
6 |   if Rest_Accuracy < Original_Accuracy then
7 |   |   Add rule to R_Set;
8 |   end
9 end
```

2.3.4 Fu2

Fu2 [34] is based on Fu1, which replaces the Fu1's first two stages with an incremental deletion procedure (shown in Algorithm 6). The procedure is similar to Fu1's second stage, but instead of directly removing rules that do not match the criterion (inferior rules), the procedure punishes an inferior rule by subtracting one from its numerosity, and a rule will be removed only if its numerosity reaches zero. After the procedure, rules are

ranked by numerosity in the descending order. Then, rules will undergo the process that is the same as CRA's last stage. Although Fu2 achieved an unsatisfactory performance on the examined domains, Fu2 (July 2002) [34] is the first attempt to compact rules with the priority of preserving the capacity of covering training samples, which is slightly earlier than Dixon's work (September 2002) [25].

Algorithm 8: CRA2

Input: all rules in a trained population P;
all instances in training set T;
Output: a set of rules *S_Set*

```
foreach instance ∈ T do
    M_Set=Match(instance,P);
    C_Action=MaxWeightAction(Match_Set);
    C_Set=FindCorrectSet(M_Set,C_Action);
    Find a rule with hightest numerosity*accuracy in C_Set;
    if rule ∉ S_Set then
        Add Mark_rule to S_Set
    end
end
```

2.3.5 Fu3

Fu3 follows the first two stages of Fu1 together with a modified last stage (shown in Algorithm 7). Fu3's last stage starts by ranking rules in descending order by fitness or experience. Then, the remaining process is the same as the second stage of CRA. This approach benefits from the stepwise accuracy evaluation that runs throughout the whole compaction process. According to previous experiments in [33], Fu3 preserves the original training accuracy after compaction. However, the advantage of preservation is at the price of increasing the computation complexity.

2.3.6 Alternative Reduction Algorithm (CRA2)

CRA2 gives priority to preserving the capability of covering instances but reduces the importance of maintaining the original accuracy. CRA2 starts with creating an empty ruleset to store the most useful rules. In CRA2, the usefulness of a rule is quantified by computing the product of its accuracy and numerosity. CRA2 iterates the trained instances to build match-sets $[M]$s.

A $[M]$ is an aggregation of all the rules in the $[P]$ that cover a specific instance. In a $[M]$, the weight of each possible action is calculated by summing the products of numerosity and fitness of all its supporters (rules). Then, the action that possesses the highest weight is considered as the action to effect. Afterward, an action-set $[A]$ is constructed by collecting all rules in $[M]$, which advocate the chosen action. At the end of each iteration, a rule that owns the highest value of usefulness[7] in $[A]$ is preserved in the ruleset. Algorithm 8 shows the pseudo-code of CRA2.

2.3.7 The K1 Approach

K1 [45] works on an offline environment and follows the architecture of CRA2, except for that K1 involves a pre-processing stage to re-evaluate each rule's performance by calculating its entropy. The calculation of a rule's entropy begins with finding the difference between the number of its correct and incorrect matched instances. Then the entropy is obtained by dividing this difference with the number of training instances. In the subsequent process, K1 uses the entropy measure to identify the most useful rules. In each iteration, K1 preserves the rule that possesses the highest entropy (shown in Algorithm 9).

[7] a rule's usefulness in CRA2 is equal to the product of accuracy and numerosity

Algorithm 9: K1's pre-processing stage

Input: all rules in a trained population P;
all instances in the training set T;
Output: all rules with entropy value in the trained population P

```
1  foreach instance ∈ T do
2      M_Set=Match(instance,P);
3      foreach rule ∈ M_Set do
4          if rule.action == instance.action then
5              rule.correct++;
6          else
7              rule.incorret++;
8          end
9      end
10     foreach rule ∈ M_Set do
11         rule.entropy = (rule.correct − rule.incorrect) / Size(T)
12     end
13 end
```

2.3.8 Quick Rule Compaction (QRC)

QRC is similar to CRA with a modification in the last stage. QRC begins by ranking rules by fitness. After ranking, QRC creates an empty set [T] for storing all the training instances. Then, from top to bottom, rules are interrogated in order by the number of correctly covered instances in [T]. QRC preserves rules that cover at least one instance. Meanwhile, the covered instances are removed permanently from [T]. The compaction process terminates when [T] is empty, or QRC has iterated through all the rules.

Algorithm 10: Quick Rule Compaction (QRC)

Input: all rules in a trained population P;
all instances in training set T;
Output: a sub-set rules R_Set

```
Rank rule ∈ Subset by fitness (descending order);
while Size(P) > 0 and Size(T) >0 do
    select first rule in P;
    match_size=0;
    for instance ∈ T do
        if Match(rule, instance) then
            Remove instance in T;
            match_size++
        end
    end
    if match_size >0 then
        Add rule to R_Set
    end
    Remove rule from P
end
```

2.3.9 Parameter Driven Rule Compaction (PDRC)

PDRC follows the structure of CRA2. The only modification of PDRC is that the method for quantifying a rule's usefulness is altered. In PDRC a rule's usefulness is estimated by a product of three parameters, which are the numerosity, accuracy, and generalization in encoding (equal to the number of "#" symbol in an encoding).

2.4 Visualization techniques

Visualization techniques have been widely applied into the LCSs field, especially for the purposes of investigating the LCS's training performance, or representing an LCS's produced model in a human-discernable form. For example, a two-dimensional map of training accuracy and learning iterations is the most common technique (Figure 2.5) for illustrating the LCSs' models' development of the prediction performance. Besides, a three-dimensional map (Figure 2.6) of fitness, numerosity, and the product of these two parameters, can be used to identify the discrepancy between superior rules and inferior rules.

LCSs evolved rules are human-readable as they follow the format of if "condition" then "action" [15]. Once accurate, the rules have been considered to contain useful knowledge that can reflect the nature of the explored domains, but few works have been conducted to visualize the patterns under LCSs' models. This is because the traditional optimization methods of LCSs cannot precisely identify the LCSs captured patterns, which hampers the effort of successfully translating LCSs' underlying patterns into human-discernable graphs.

Visualization techniques have been employed to investigate the difference of learning theory between LCS and other systems. This was achieved by comparing the graphs that describe the system's learning performance and the graphs that reflect the trained population's distribution. For exam-

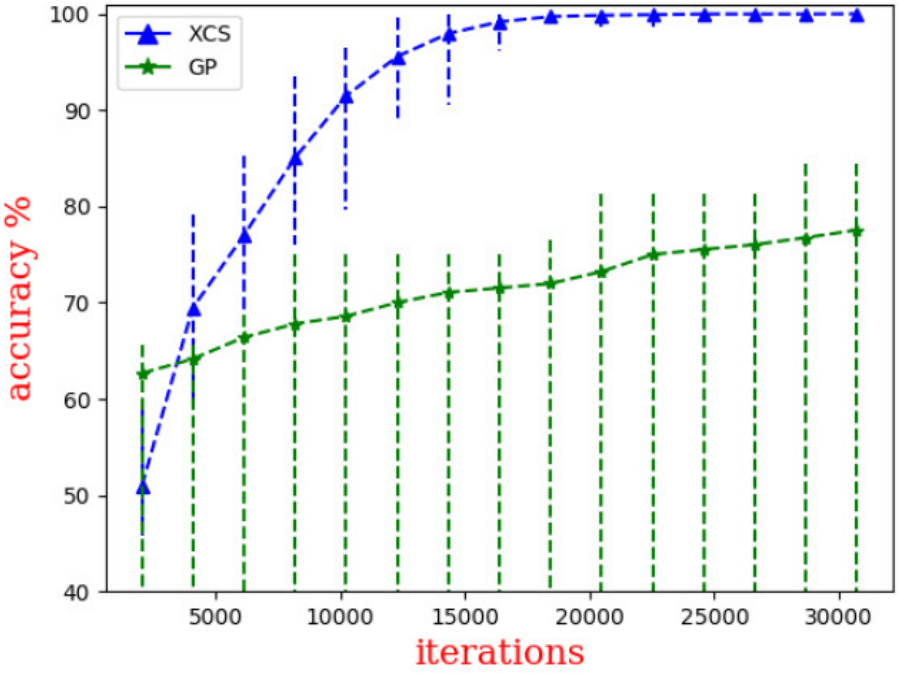


Figure 2.5: A sample of visualization for comparing XCSs' training performance with other techniques, e.g. Genetic Programming (GP) regarding the 11-bits Multiplexer problem. Green graph describes the XCS and blue graph depicts GP. Y-axis: the training accuracy, which is the average of 30 exploits tasks, plus error bars (shown with dotted line), X-axis: the number of environmental instances that have been received. This visualization technique reflects the development of the training performance, which can be applied to investigate the learning progress of the visualized system.

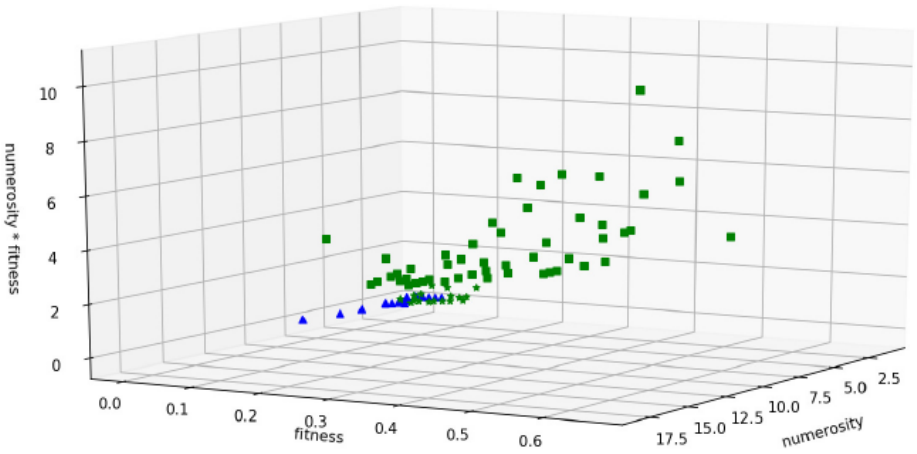
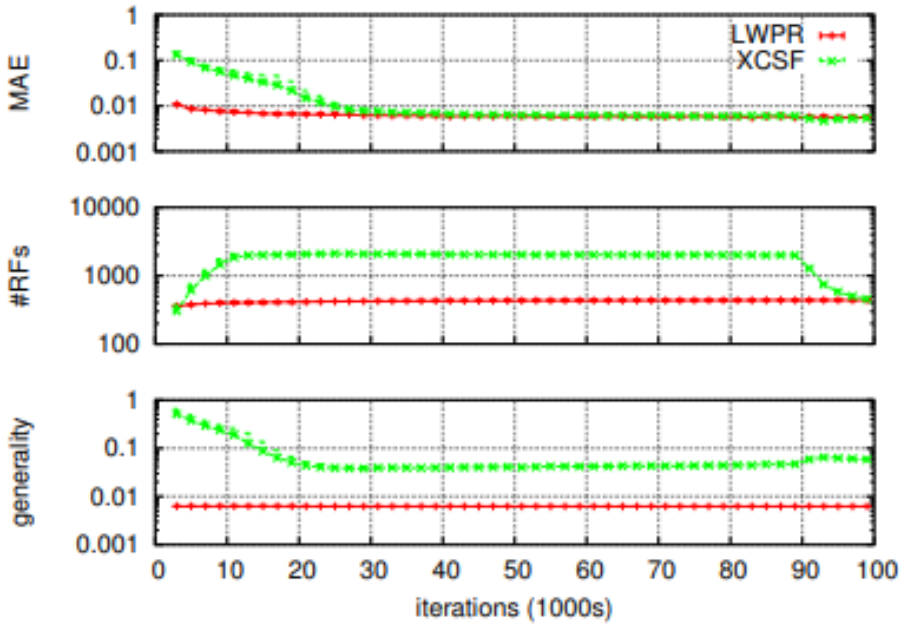
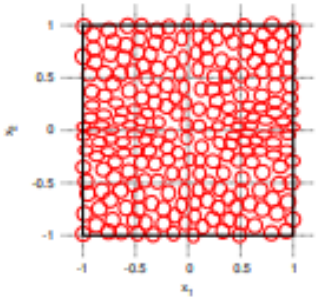


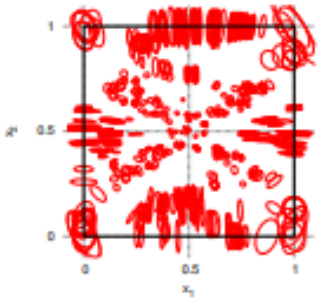
Figure 2.6: Visualization for a trained model that aims to distinguish the superior rules and the inferior rules regarding the 6-bits Multiplexer problem. Green and blue respectively color the superior rules and the inferior rules. This technique illustrates the superior rules' eigenvalue in a single LCSs produced model ([P]).



(a) Performance on Crossed Ridge



(b) LWPR's Population



(c) XCSF's Population

Figure 2.7: XCSF and LWPR's performance on the Crossed Ridge function. In (a), the horizontal axis represents the iterations and the vertical axis respectively record the MAE (mean absolute error), RFs (number of classifiers), and generality (mean plus standard deviation). (b) and (c) respectively represent the distribution of LWRP's and XCSF's population (ellipsoidal lines represent the inflection point of the Gaussian activity),x1 and x2 are two parameters of the Crossed Ridge function. The figure shows both systems are well applicable to the problem. LWPR reaches a low error faster than XCSF. However, XCSF is more likely to find a concise solution [79].

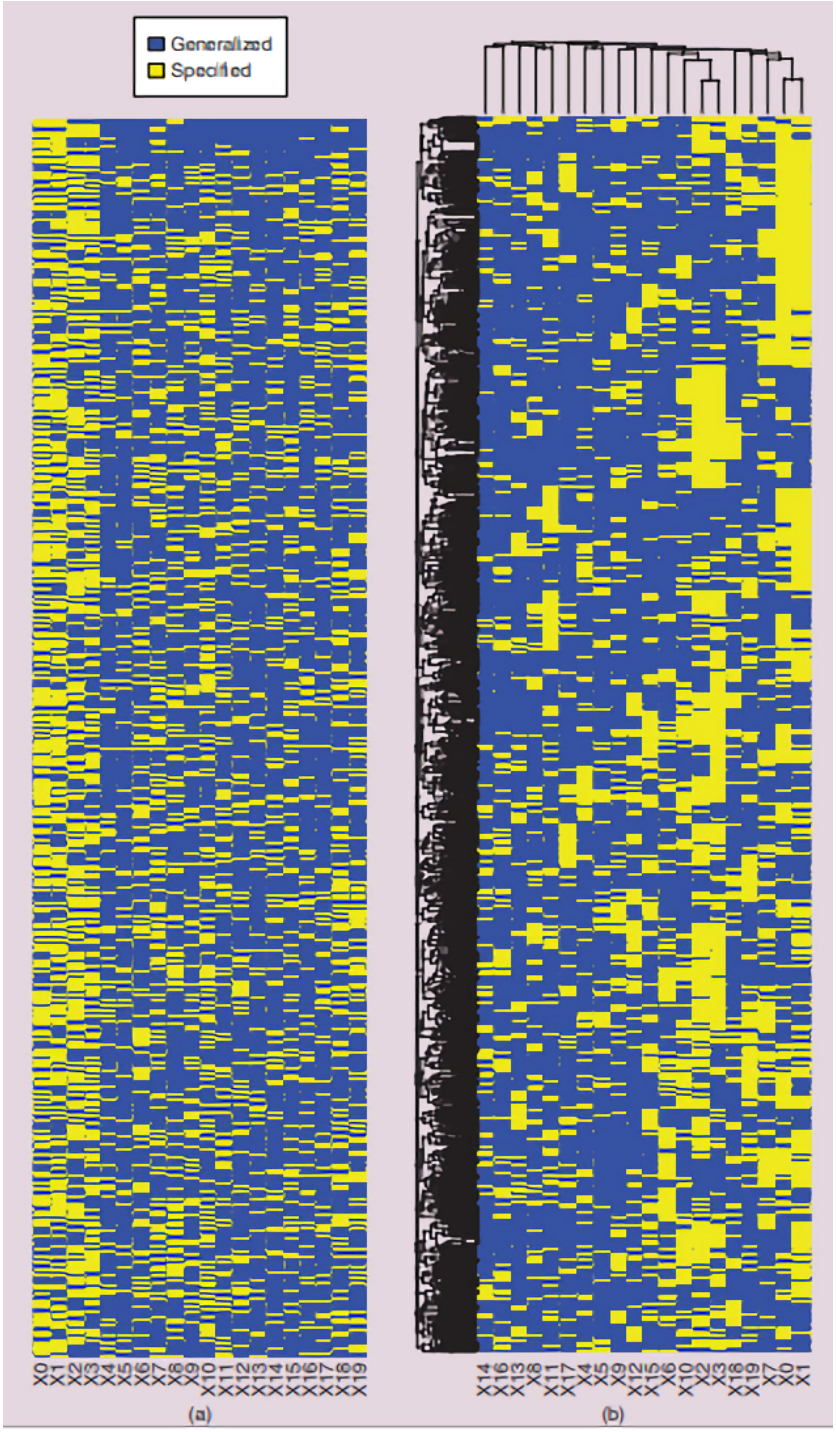


Figure 2.8: Heat maps visualize the LCS produced population. (a) without clustering, each row is a rule, each column is a feature, this figure modelled x0, x1, x2, and x3 as predictive features. (b) the same population after clustering, feature shows two pairs of interacting features, i.e. (x0,x1) and (x2, x3) are the most important features [92].

ple, Butz had conducted a visualization-based experiment [79] regarding investigating the difference between XCSF[8] [102] and Locally Weighted Projection Regression (LWPR [9] [96])'s difference in addressing function approximation problems.

For both systems, this work recorded three types of information, i.e. the mean absolute error, the number of classifiers of both systems (in LWPR, a classifier is called a receptive field (RFs), thus, RFs represent the number of classifiers), and generality measure (mean plus standard deviation). Afterwards, three visualizations (shown in Fig.2.7)that reflect the system's training performance are built with the horizontal axis for iterations and the vertical axis for the recorded information. By comparing these visualizations with the trained populations' distribution, LCS and LWPR can be compared objectively regarding computational effort, structuring capabilities, and parameter sensitivity in addressing function approximation problems [79].

It is possible to utilize visualization technique to reveal the ground truth of LCS explored problems by clustering the rules of an LCSs' trained population, and then visualizing the patterns in the rule clusters, e.g. Urbanowicz's work [92]. This work utilizes the Agglomerative Hierarchical Clustering (AHC [10] [72]) to cluster the trained rules. Afterwards, a heat map is employed to visualize the clusters' rules' generalization and specification characteristics (shown in Fig.2.8). Through the visualization results, the patterns of feature importance distribution are highlighted.

AHC works based on the assumption that Hamming distance can precisely reflect the correlation between rules. This results in Urbanowicz's work cannot adapt to boolean domains. For example, in the natural solution of the 6-bits Multiplexer problem, AHC considers $\#\#0000$ and $000\#\#\#$

[8] XCSF is a genetics-based machine learning algorithm, which is a modified version of XCS.

[9] LWPR is a statistics-based machine learning technique that excels in addressing function approximation problems.

[10] A Hamming distance-based clustering technique

in the same cluster as these two rules' Hamming distance is 0. The specification ratio cannot reflect the difference between the first two address bits and the last four data bits for this cluster.

2.5 Benchmarks

This thesis aims to identify and visualize the patterns in LCSs' models. Therefore, three artificial boolean domains with clear patterns are considered as the benchmarks, i.e. the Multiplexer problem, the Carry problem, and the Majority-On problem. Besides, for the sake of investigating the newly proposed techniques' potential in adapting to real-world applications, datasets from the UC Irvine Machine Learning Repository (UCI) [5] are also tested, including Iris, Sonar, Wine, Australian, German, Wisconsin Breast Cancer Diagnostic (WBCD), Lung Cancer, Hill & Valley, Ionosphere, and Zoo.

2.5.1 The Multiplexer Problem

The **Multiplexer** problems (MUX) are based on the Multiplexer electronic circuit and have an inherent epistasis data distribution as the values in the address bits affecting the importance of the data bits.

A Multiplexer accepts m input signals then outputs a signal. The first k bits are the address, and the remaining 2^k bits are data bits. For example in 6-bits multiplexer, if the input is *010100*, then the output will be 1 as the first two bits *01* represent the index 1 (in base ten), which is the second bit following the address. Previous works proved that LCSs are capable of addressing large-scale MUXs, e.g. with a population of around 20 thousand rules, LCSs can correctly represent the $4.3 * 10^{40}$ instances in 70-bits MUX.

None of the existing compaction algorithms had yet successfully compacted large-scale MUXs to their optimal format. Thus, it is interesting to

determine if any of the tested compaction algorithms can find the optimal ruleset for a large-scale MUX domain. Furthermore, the attempt of visualizing patterns contained by compacted rulesets for large-scale MUXs can investigate whether our proposed visualization methods can be applied to large-scale problems.

2.5.2 The Carry Problem

In **Carry** problems (CARs), two binary numbers of the same length undergo the process of addition. If the addition triggers a carry bit, then the output is 1 otherwise 0. For example, the action in *1010* is 1, while the action for *0110* is 0.

CARs have two characteristics that make them become difficult problems for compaction. Firstly, member rules of an optimal solution for a CAR have a naturally over-lapping distribution. This means that for rules that support the same action, their encodings may have an overlapping area, e.g. in 6-bits CAR, $111\#\#1:1$ and $\#111\#1:1$. When a problem has an over-lapping distribution, LCSs frequently evolved and kept over-general rules in their final population. Compaction algorithms will fail in CARs if algorithms cannot identify over-general rules. Secondly, CARs' optimal rules have different generalization levels. Important but specific rules may have relatively low training parameters, e.g. numerosity and fitness. Therefore, these important specific rules are under the risk of being replaced by over-general rules when compacting.

2.5.3 The Majority-On Problem

In the **Majority-On** domain (MAJs), the correct classification is the majority value of the N input bits. For example, the output is 1, when the majority of the input bits are 1s, and the output is 0, when the majority of the input bits are 0s or the number of 0s and the number of 1s are equal. For example, *0011*'s action is 0, and *0111*'s action is 1.

MAJs are difficult to compact for two reasons. Firstly, the number of rules in an optimal ruleset for a MAJ is much larger than for other domains, e.g. an optimal set for 11-bits MAJ has 924 rules whereas for the 70-bits MUX only has 128 rules. When compacting, even incorrectly removing one of the optimal rules, the performance of the compacted population will decrease, and the readability of the compacted populations contained patterns will become obscure. Secondly, MAJs have a severe over-lapping distribution. In this scenario, LCSs continually evolve good performance over-general rules, and LCSs incorrectly assign training parameter values for these problematic rules when training. Therefore, it is difficult for compaction algorithms to distinguish optimal rules from over-general rules.

2.5.4 Datasets from UCI

The UCI datasets have been widely employed in the data mining area as benchmarks [2] [30] [31] [74]. In this work, ten UCI datasets for continuous real domains are studied, six relatively simple datasets are selected to interrogate the correctness of the proposed work, including Iris [53], Sonar [48], Wine [11], Australian [11], German [103], and Wisconsin Breast Cancer Diagnostic (WBCD) [43]. Furthermore, four other complex domains are selected to investigate the novel method's limitations due to high dimensionality e.g. Lung Cancer [43], artificial problems e.g. Hill and Valley [20], and multiple actions with a low number of instances e.g. Zoo [70], and natural domains, e.g. Ionosphere [24]. As the aim is not to test LCSs capability in dealing with missing data, all the instances that contain missing data have been removed to avoid this confounding variable rather than a lack of capability in LCSs.

The **Iris Dataset** [32] is a very simple domain that refers to a type of three different iris plants, i.e. Setosa, Versicolor, and Virginia. This dataset has three classes of fifty instances each and four attributes, i.e.

sepal length, sepal width, petal length, and petal width.

The **Sonar, Mines vs. Rocks Dataset (Sonar)** [105] is a real-world task that records sonar signals bounced off a metal cylinder and those bounced off a cylindrical rock. The Sonar contains two classes, i.e. rock and mine. The rock class contains 97 instances and the mine class has 111 instances. Each instance has sixty attributes, which are valued in the range of 0.0 to 1.0. Each attribute represents the energy within a particular frequency band, integrated over a certain period.

The **Wine dataset** [104] is a simple domain for classification tasks, which represent the results of a chemical analysis of wines grown in the same region in Italy but derived from different cultivars. This dataset has 178 instances, which belong to three classes. Besides, each instance has thirteen attributes, i.e. Alcohol, Malic acid, Ash, Alcalinity of ash, Magnesium, Total phenols, Flavanoids, Nonflavanoid phenols, Proanthocyanins, Color intensity, Hue, OD280/OD315 of diluted wines, and Proline.

The **Australian Credit Approval dataset** [28] records credit card applications from Australian. This dataset has two classes and 690 instances. Each instance has 14 attributes. This dataset is interesting because there is a good mix of attributes, including continuous, nominal with small numbers of values, nominal with larger numbers of values, and missing values.

The **South German Credit Dataset (German)** [29] shows customers' credit information. This dataset has 21 attributes and 1000 instances with 700 instances as good credit and 300 instances as bad credit.

The **Breast Cancer Wisconsin (Diagnostic) Dataset** [82] reports the attributes that are computed from a digitized image of a needle aspirate of a breast mass. The attributes describe characteristics of the cell nuclei present in the image. This dataset has 2 classes and 569 instances. Each instance has 32 attributes.

The **Lung Cancer Dataset** [37] describes 3 types of pathological lung cancer, i.e. three classes. This dataset has 32 instances and each instance has 56 attributes.

The **Hill & Valley Dataset** [20] is an artificial problem that each instance describes a graph of either hill or valley. This dataset has 606 instances, each instance represents 100 points (attributes) on a two-dimensional graph.

The **Zoo dataset** [50] is a multiple class problem containing 101 instances, that describes 7 types of creatures. In this dataset, an instance has 16 boolean-valued attributes, i.e. hair, feather, eggs, milk, airborne, aquatic, predator, toothed, backbone, breaths, venomous, fins, legs, tail, domestic, and catsize.

The **Ionosphere Dataset** [105] was collected by a radar system in Goose Bay, Labrador. This dataset has 34 attributes, 351 instances, and two classes that are "Good" and "Bad". "Good" records radar returns that are showing some evidence of some type of structure in the ionosphere. "Bad" documents the radar returns that pass through the ionosphere.

2.6 Parameters Settings

This thesis's models are produced by three types of LCSs, i.e. XCS, UCS, and Absumption and Subsumption based Learning Classifier System (ASCS) (ASCS is proposed in this project and is described in detail in Section 7). The description and default values of the training parameters for XCSs are represented in Table 2.1, and Table 2.2. The UCSs' parameters are in Table 2.3. Besides, Table 2.4 shows the parameter settings for ASCS.

In this thesis, 18 Boolean problems use the default values of XCSs and UCSs, i.e. the 6-bits, 11-bits, and 20-bits Multiplexer problem, the 6-bits, 8-bits, 10-bits, 12-bits, 14-bits, and 16-bits Carry problem, the 6-bits, 7-bits, 8-bits, 9-bits, 10-bits, 11-bits, 12-bits, 13-bits, and 14-bits Majority-On problem. Besides, all the UCI datasets are tested by the training parameter's default values.

Regarding the 37-bits and 70-bits Multiplexer problem, the $P_{\#}$ is respectively increased to 0.66 and 0.99 in both XCSs and UCSs. Other pa-

Table 2.1: Table of training parameters for XCSs

Parameter	Description	Default
α	The fall off rate in the fitness evaluation.	0.1
β	The learning rate for updating fitness, prediction, prediction error, and action set size estimate in XCS's classifiers.	0.2
γ	The discount rate in multi-step problems.	0.95
δ	The fraction of the mean fitness of the population below which the fitness of a classifier may be considered in its vote for deletion.	0.1
ν	Specifies the exponent in the power function for the fitness evaluation.	5
θ_{GA}	The threshold for the GA application in an action set.	25
ϵ_0	The error threshold under which the accuracy of a classifier is set to one.	10
θ_{del}	Specified the threshold over which the fitness of a classifier may be considered in its deletion probability.	20
pX	The probability of mutating one allele and the action in an offspring classifier.	0.8

Table 2.2: Table of training parameters for XCSs

Parameter	Description	Value
selectTolerance	The tolerance of select candidate parents.	0.05
pM	The probability of mutating one allele and the action in an offerspring classifier.	0.04
tournamentSize	The fraction of classifiers participating in a tournament from an action set.	0.4
$P_{\#}$	The probability of using a don't care symbol in an allele when covering.	0.3
predictionErrorReduction	The reduction of the prediction error when generating an offspring classifier.	1.0
fitnessReduction	The reduction of the fitness when generating an offspring classifier.	0.1
θ_{sub}	The experience of a classifier required to be a subsumer.	20
predictionIni	The initial prediction value when geneerating a new classifier (e.g in covering).	10
predictionErrorIni	The initial prediction error value when generating a new classifier (e.g in cvering).	0
fitnessIni	The initial fitness value when generating a new classifier (e.g in covering).	0.01

Table 2.3: Table of training parameters for UCSs

Parameter Name	Description	Value
ν	Power parameter used to determine the importance of high accuracy when calculating fitness.	5
χ	The probability of applying crossover in the GA.	0.8
υ	The probability of mutating an allele within an offspring.	0.04
$P_{\#}$	The probability of using a don't care symbol in an allele when covering.	0.33
θ_{GA}	The GA threshold; The GA is applied in a set when the average time since the last GA in the set is greater than theta_GA.	25
θ_{del}	The deletion experience threshold.	20
θ_{sub}	The subsumption experience threshold.	20
acc_sub	Subsumption accuracy requirement.	1.0
β	Learning parameter; Used in calculating average correct set size.	0.2
δ	Used in determining deletion vote calculation.	0.1
init_fit	The initial fitness for a new classifier.	0.01
fitnessReduction	Initial fitness reduction in GA offspring rules.	0.1
θ_{sel}	The fraction of the correct set to be included in tournament selection.	0.5

Table 2.4: Table of training parameters for UCSs

Parameter Name	Description	Value
Epoch Number	Number of epochs that are required to be completed before exploration is ceased.	10
Training Number	Number of instances that need to be explored in each training run.	50000
Compacting Number	Number of instances that need to tested in each compacting process.	5000
Maximum Rule Number	The maximum number of rules in $[P]$.	1000
Experience discount	The discount ratio of experience at the end of each compacting process.	0.33

rameters use the default values [95].

In ASCS, the 6-bits, 11-bits, and 20-bits Multiplexer problems, the 6-bits, 8-bits, and 10-bits Carry problems, and the 6-bits, 7-bits, 8-bits, 9-bits, and 10-bits Majority-On problems are using the default values. In the 37-bits Multiplexer problem, 12-bits and 14-bits Carry, and 11-bits, 12-bits, 13-bits, and 14-bits Majority-On problems, the epoch number is 15, the training number is 250,000, the compacting number is 25,000, and the maximum number of rules is 10000.

Chapter 3

Natural Solution & Visualization Techniques

This chapter proposes a new hypothesis, which is termed *natural solution* regarding the Learning Classifier System's optimal solution. Hence, three novel visualization techniques that aim to translate LCSs' underlying knowledge into human-discernable patterns are introduced. The three visualization techniques are termed Feature Importance Map (FIM), Action-based Feature Importance Map (AFIM), and Action-based Feature's average Value Map (AFVM), respectively.

FIM reflects the importance of each involved attribute. This visualization method is easy to implement and can be utilized to trace an LCS's performance in patterns, i.e. how patterns are constructed. AFIM shows attribute importance based on each action, AFIMs can be utilized to investigate the explored domain's data distribution. AFVM presents the specified attribute's average values based on actions, AFVMs are useful to detect the relationship between specific values and the output actions, i.e. the cause of a model's decisions.

3.1 Introduction

Previously, Butz et al. proposed that LCSs are designed to evolve rulesets that are characterized by completeness, correctness, minimality, and none overlapping to represent the explored domain [18]. Such rulesets are considered as the optimal results of LCSs and are termed optimal ruleset [*O*], which are assumed to contain interpretable patterns that can reflect the underlying nature of the addressed problem (see Section 3.4.3). However, in practice, LCSs use Evolunatory Computation (EC) techniques as the primary search algorithms. The stochastic nature of EC makes LCSs prone to produce high diversity rules. This indicates that LCSs cannot guarantee the correctness and minimality of the produced ruleset. Furthermore, LCSs employ subsumption to prevent redundant specific rules to be introduced to the population [*P*] [55]. The subsumption method ensures that the evolved rules are unsubsumable rather than that the rules are not overlapping. Hence, in the majority of LCSs produced models, an [*O*] does not exist.

In this scenario, a new type of optimal solution to be produced by LCSs is proposed by this work. This optimal ruleset is expected to represent explored domains completely and correctly. Furthermore, each member rule should be consistent and unsubsumable[1] under the global search space. *Consistentness* requires that a rule can either consistently support or consistently oppose the associated action. The newly defined LCSs' optimal solution is termed *natural solution*, which complements Butz and Kovacs' [*O*]. The *natural solution* has been discovered in all the XCSs [99], and UCSs [68] addressed domains. However, these optimal solutions for most domains still contain a considerable number of rules that cooperatively represent the knowledge in a manner that is too complex to be comprehended by humans [10]. Thus, assistive techniques are required to support hu-

[1] subsumable rules are the over-specific rules, unsubsumable rules are non-over-specific rules.

mans in understanding the critical knowledge in LCSs' models. This leads to the visualization techniques of Action-based Feature Importance Map (AFIM) and Action-based Feature's Average value Map (AFVM) being developed, which can precisely describe patterns of optimal solutions from different viewpoints.

3.2 Natural Solution

Natural solution is proposed in this work, which is a definition regarding a type of ideal form of LCSs' optimal solution. *Natural solution* describes a ruleset that includes all the consistent and unsubsumable rules under the global search space. *Natural solution* is unique under the global search space. This uniqueness unlike [*O*], which may also contain multiple ways of accurately describing the domain's patterns. Furthermore, a *natural solution* can correctly and completely represent arbitrary dataset, e.g. represent any off-line datasets such as overlapping boolean domains and real-valued domains in UCI datasets. This differs from [*O*], which can only adapt to non-overlapping datasets, e.g. Multiplexer problems. Thus, the discovery of *natural solution* reveals that LCSs can produce interpretable models for both overlapping domains and non-overlapping domains.

A *Natural solution* has a necessary and sufficient condition, that is a noiseless training set [T]. Assuming an *N*-features problem Pr_N can be globally described by *M* instances, where ($[I_0, I_1, ..., I_M] \subset Pr_N$). Meanwhile, in Pr_N, *m* instances form [T], where $[I_0, I_1, ..., I_m] \subset [I_0, I_1, ..., I_M]$.

In representations that reflect the importance of attributes, the global search space is naturally splittable. For example, in the ternary alphabet representation, the global search space (GS) can be split into N+1 subspaces (SP) that are distinguished by the number of generalized features in the encodings. For example, ($[SP_0, SP_1, ..., SP_N] \subset GS$) that are ranked from the most specific (all features are specified) SP_0 to the most general (all features are general) SP_N. Since SP_0 specifies all features, then, [T]

must be a subset of SP_0. $SP_0 \subset$ GS. Therefore, for any [T], in GS, there exists at least one [P] that can completely represent the [T] correctly.

In GS, among all the possible [P]s, there exists one solution that possesses all rules that satisfy two criteria: consistent and maximal generalization, under the global search space. Consistent requires a rule consistently supports or consistently opposes the covered instances' action. Maximal generalization expects a rule that can not be subsumed by any other correct rules in GS. The quality of being unsubsumable indicates that the encodings of maximal generalization for a specific [T] are deterministic. Therefore, for any [T], its *natural solution* is deterministic. This characteristic varies from [O], which can be alternative rulesets. Furthermore, in specific cases, some [O] can be a subset of a *natural solution* (show in Figure 3.1). When this occurs, similar to *natural solution*, [O]s will possess human-discernable patterns (show in Figure 3.3), e.g. some [O]s of the Multiplexer problems.

Each rule in a *natural solution* corresponds to a cluster of instances from [T]. The instances in the same cluster advocate the same action and share the same feature importance. A rule's encoding specifies the important common features to identify the rule covered instances from instances that advocate different actions. Therefore, the specified features in the encodings can reflect the knowledge for distinguishing instances possessing different actions, which are the underlying patterns of [T]. If [T] and Pr_N have a similar distribution, then the detected underlying patterns of [T] also are adapted to Pr_N. Hence, in practice, LCSs still have the potential to evolve an interpretable ruleset even if only a small portion of possible instances are observable, e.g. when exploring a 70-bits Multiplexer problem, LCSs typically observe around four million of all $1.18 * 10^{21}$ available instances.

Natural solutions have been discovered in both XCSs and UCSs' optimized models. Furthermore, for the same dataset, the *natural solutions* found by XCSs and UCSs are the same. This evidence supports the hy-

pothesis that *natural solution*s widely exist in various LCSs produced models and only dependent on the employed representation format.

Natural Solution:	Butz and Kovacs' [O]:
##1111 : 1 10##1# : 1 11###1 : 1 01#1## : 1 001### : 1 0#11## : 1 #01#1# : 1 #1#1#1 : 1 1###11 : 1 ##0000 : 0 10##0# : 0 11###0 : 0 01#0## : 0 000### : 0 0#00## : 0 1###00 : 0 #00#0# : 0 #1#0#0 : 0	10##1# : 1 10##0# : 0 11###1 : 1 11###0 : 0 01#1## : 1 01#0## : 0 001### : 1 000### : 0

Figure 3.1: Samples respectively describe a *natural solution* and a Butz and Kovacs' optimal solution [O] for the 6-bits Multiplexer problem. For a certain dataset, a *natural solution* is unique, but [O] can be multiple. Furthermore, [O] that be composed of only unsubsumable rules, are always subsets of the *natural solution*.

3.3 The Proposed Visualization Techniques

Three types of visualization techniques that aim to explore the discovered data patterns are proposed, i.e. Feature Importance Map (FIM), Action-based Feature Importance Map (AFIM), and Action-based Feature's Average value Map (AFVM). These methods have been used independently with a clear comparison of the strengths, weaknesses, and suitability for various visualization tasks.

3.3.1 Feature Importance Map (FIM)

In the LCS domains, calculating an attribute's ratio of specification among all the rules in an LCSs' model is the most common manner for assess-

ing each attribute's importance in addressing the explored problem. The majority of previous visualization techniques are based on visualizing the attributes' specification ratio. However, these techniques cannot precisely describe the patterns captured by LCSs. Thus, it is hypothesized that clustering the rules according to the rules' generalization level, and then calculating the specification ratio for attributes in each cluster can improve the clarity of the visualized patterns. Feature Importance Map (FIM) is developed based on this hypothesis.

FIMs consider three dimensions, i.e. the labeled attributes, the global search space that is composed of a set of different generalization levels, and the attribute importance. The X-axis represents all the involved environmental attributes. Here, all the attributes are labeled, e.g. when addressing an n-bits Boolean problem, attributes are labeled from 0 to $n-1$. Y-axis is the global search space that is divided into a set of generalization levels, e.g. for any n attributes problems, n+1 generalization levels exist that rank from 0 to n that completely satisfied with general respectively. Z-axis is the attribute importance that record an attribute's specification ratio (importance) at a specific generalization level, e.g. the generalization level i contains k_i rules, among the k_i rules, regarding the $attribute_j$, m_j rules specific the $attribute_j$, then the $attribute_j$'s specification ratio R_{ij} at generalization level i can be calculated by the Equation (3.1).

$$R_{ij} = \frac{m_j}{k_i} \tag{3.1}$$

FIMs are able to discover the difference in importance regarding the considered attributes and highlight the critical generalization levels for addressing the problem at global level. For example, in Figure 3.2, FIMs successfully visualize the difference between the address bits ($attribute_0$ and $attrbiute_1$) and the data bits (from $attribute_2$ to $attribute_5$) for the 6-bits Multiplexer problem. Importantly, this figure identifies that the generalization level 3 is the most critical generalization level for the explored domain.

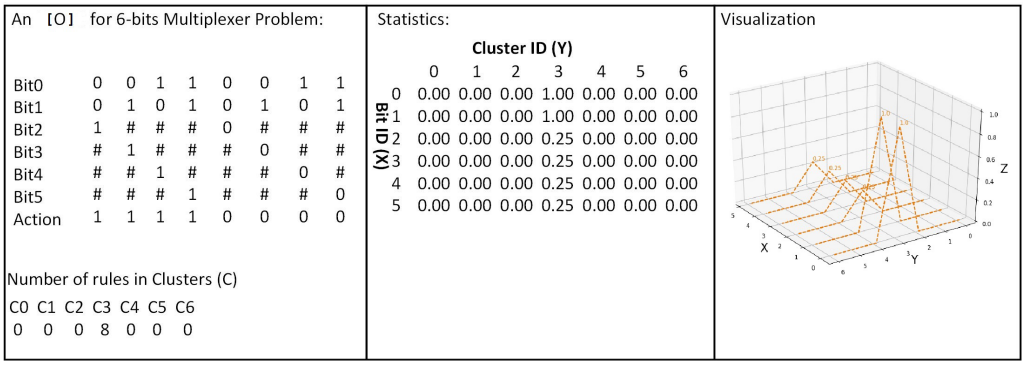


Figure 3.2: A figure shows the visualization process of Feature Importance Map (FIM) in the case of the 6-bits Multiplexer problem. First rectangle: An [*O*] for 6-bits Multiplexer problem; Second rectangle: calculated statistics (see Section 3.1 for the details); Third rectangle: visualize the statistics results. In the visualization part, X is the attribute ID; Y records the generalization levels that rank by the number of generalized bits; Z represents the importance of attributes for a specified attribute at a specific general level. The FIM shows the importance of each attributes in the specification for the explored domain. Furthermore, the importance among all the involved generalization levels also can be visualized. For example, this figure shows the generalization level 3 is crucial for the 6-bits Multiplexer problem.

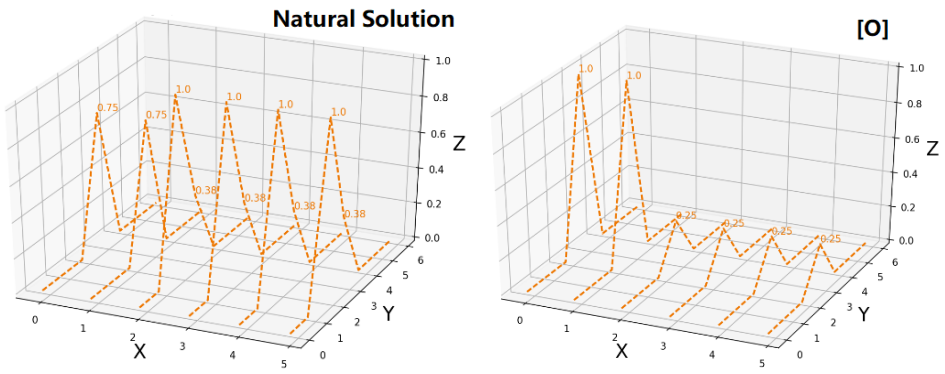


Figure 3.3: FIMs respectively describe a *natural solution* and an [*O*] for the 6-bits Multiplexer problem. Both *natural solution* and [*O*] that are built by unsubsumable rules, contain patterns that display a distinct difference between address-bits and data-bits. However, the member rules of [*O*] may not be unsubsumable under the global search space.

3.3.2 Action-based Feature Importance Map (AFIM)

FIMs visualize the underlying patterns at the macro level, which makes FIMs easy for human-discernment in comprehending the basic knowledge of the explored domains. However, LCSs' captured patterns have the potential to display more useful information on the addressed problems. Thus, it is hypothesized that visualizing the underlying patterns at the micro level can help a human to discover other hidden knowledge that can not be observed from FIMs. Action-based Feature Importance Map (AFIM) is developed as a technique for visualizing the attribute importance at the micro level, which are the important attribute values.

AFIMs consider three dimensions, which are attributes (X-axis), search space (Y-axis), and attribute importance (Z-axis). Different from FIMs, AFIMs begin with grouping the rules according to the rules' supported actions, then independently calculate the attribute importance in each grouped ruleset. Eventually, the statistics of each action will be respectively trans-

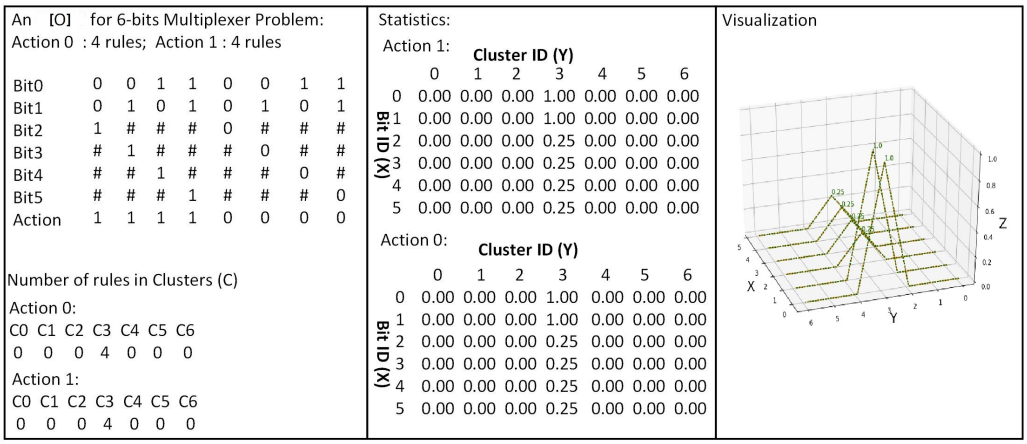


Figure 3.4: A sample describes the implementation of the Action-based Feature Importance Map (AFIM) regarding the 6-bits Multiplexer problem. First rectangle: An [*O*] for the 6-bits Multiplexer problem; Second rectangle: statistics calculation process (see Section 3.2 for the details); Third rectangle: the visualized statistics results, where X is the attribute ID, Y is the generalization levels that ranks by the number of generalized attributes, Z represents the attribute importance of specified attributes at a specific generalization level for the supported action. Graphs for action 1 and action 0 are colored with green and orange, respectively. This figure shows the explored domains' distribution and how the importance of attributes in determining the output action, e.g. the Multiplexer problems have a balanced distribution as patterns for both actions are completely overlapped, and the address-bits are very important in determining actions as the $attribute_0$ and $attribute1$ are specified by all the rules.

lated into a graph for visualization. When graphs for the different actions are represented in the same figure, these graphs will be distinguished by different colors. Thus, it is possible to discover knowledge regarding the different patterns between the available actions through the AFIMs.

Similar to FIMs, AFIMs also can reflect the important difference among the environmental attributes and identify the critical search space (generalization levels). However, AFIMs show this knowledge at the micro level, where patterns are related to an available action instead of to the whole problem domain. Furthermore, AFIMs are competent in discovering the explored domain's characteristic of class distribution. A problem has a balanced class distribution when the patterns for different actions are completely overlapped (shown in Figure 3.4). Otherwise, the problem has an imbalanced class distribution. Curiously, class imbalance has previously been counted from the input data, rather than automatically displayed in the output visualization.

3.3.3 Action-based Feature's Average value Map (AFVM)

A correlation between a rule's specified value and this rule's supported action can be observed in many LCSs' optimal solutions, e.g. in the Carry domain, all the optimal rules' specified values are the same as these rules' associated action's values. Thus, it is hypothesized that visualizing the patterns of the average value of the specified attributes can answer how attributes determine the output actions. AFVM is designed because of this hypothesis.

AFVMs are three-dimensional maps that consider attributes (X-axis), search space (Y-axis), and the average value of the specified attributes (Z-axis). The implementation of AFVMs is similar to AFIMs that start with grouping rules according to their advocated actions. Then the relevant statistics are calculated independently in each grouped ruleset, e.g. assuming a group contains k rules at generalization level i, among the k

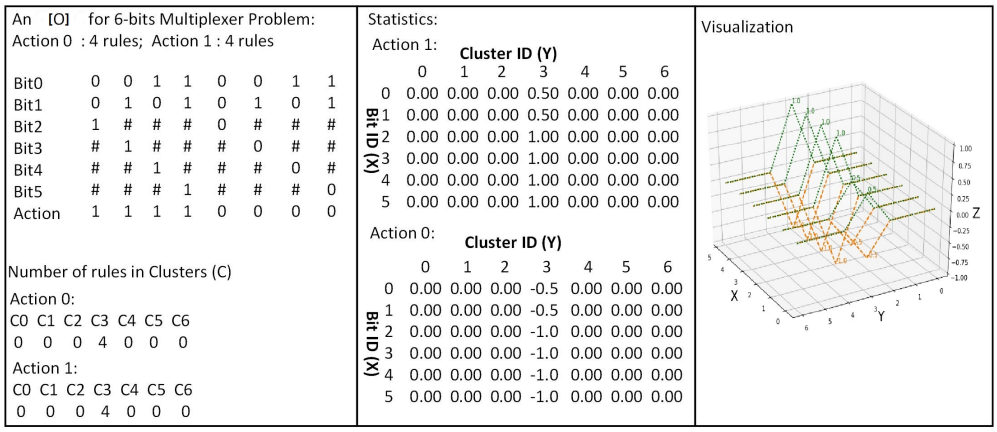


Figure 3.5: A figure illustrates how Action-based Feature's Average value Map (AFVM) is implemented on the 6-bits Multiplexer problem. First rectangle: An [*O*] for the 6-bits Multiplexer problem; Second rectangle: results of the calculated statistics (see Section 3.3 for the details); Third rectangle: graph of the visualized statistics results, where X is the attribute ID, Y is the generalization levels that rank by the number of generalized bits, Z represents the average value of specified attributes at a specific generalization level for the supported action. Graphs for action 1 and action 0 are colored with green and orange, respectively. This shows the relationship between the specific values of attributes and the output action, e.g. this figure shows that in the Multiplexer domains, the output action is determined by the specified value in the data-bits.

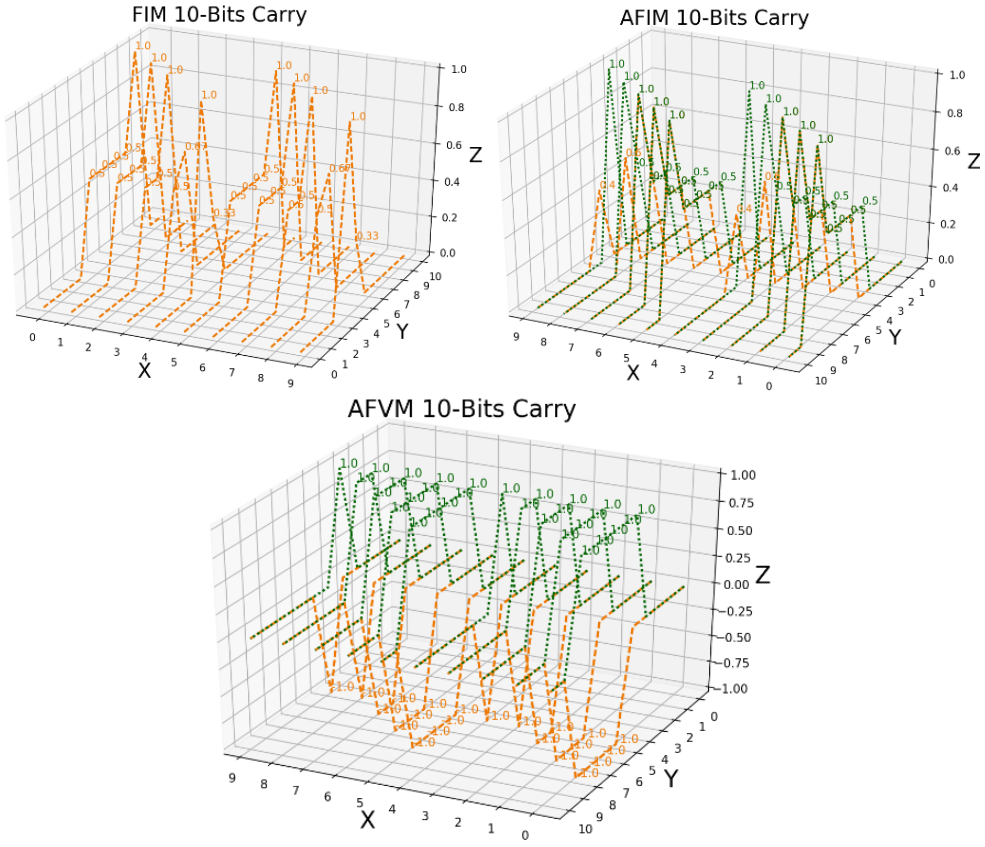


Figure 3.6: FIM, AFIM, and AFVM describe the 10-bits Carry problem. These graphs can reflect the natural information of the explored domain, e.g. in the Carry domain, all the considered attributes can be split into two parts. FIMs are easy to read, AFIMs can reflect the class distribution, and AFVMs can reflect the relationship between rules' advocated actions and the value of the specified attribute.

rules, m_j rules specified the $attribute_j$, then the average value of $attribute_j$ at generalization level i can be computed by Equation (3.2) where $\sum m_j$ means the summation of m_j rules specified value at $attribute_j$. Furthermore, in binary action Boolean encoded domains, in the graphs of action zero, the value of the specified attributes in rules will reduce by one before calculating the statistics, e.g. 1 be 0, and 0 be -1. This additional process intends to clarify the difference in graphs between two actions. The difference among three visualization techniques is shown in Figure 3.6.

$$Ave_{i,j} = \frac{\sum m_j}{m_j} \tag{3.2}$$

AFVM can reflect the importance of each attribute in identifying the action. For example, Figure 3.5 shows that for the 6-bits Multiplexer problem, the data bits determine the output actions.

3.4 Visualization Results & Discussion

The proposed visualization techniques have an effect on six fields for improving the understanding of the LCSs. Firstly, FIMs can be employed for tracing LCSs' training progress so that the learning performance among different types of LCSs can be interrogated. Secondly, FIMs can investigate how a specific training parameter impacts the learning process. Thirdly, FIMs can be applied to visualize the trained models from different types of LCSs. This allows the difference among various LCSs trained models to be interrogatable by visualizations. Fourthly, AFIMs can serve to assess compaction algorithms' performance on optimizing LCSs' produced models. And arguably most importantly, fifth, AFIMs and AFVMs can efficiently translate the underlying patterns to human-discernable knowledge of the explored domains, where knowledge is difficult to discern from directly reading the member rules in the optimal solutions. Lastly, FIMs for optimal solution visualizations can highlight redundant attributes.

The models produced by XCS, UCSs, Absumption and Subsumption based learning classifier system (ASCS [2]) are visualized in this section.

3.4.1 Tracing the Learning Progress

FIMs are simple to read, such that these FIMs' graphs can be employed to trace the LCSs' training progress, compared with the traditional training graphs that mainly record the training accuracy improvements and the changes in Macro vs Micro population. FIMs can reflect how the underlying patterns formed along with the learning progress, and how the rules' generalization changes as the evolving process moves forward.

Figure 3.7, Figure 3.8, and Figure 3.9 respectively show the learning progress of ASCS, UCS, and XCS in addressing the 6-bits Carry problem. ASCSs are designed to search for the *natural solution*, which is the optimal solution for the majority of the LCSs addressed domains. Hence, ASCSs are the only systems that can gradually form human-discernable patterns during the learning progress. Furthermore, due to ASCS employing deterministic searching algorithms, i.e. subsumption, absumption [3], and informed mutation, ASCSs can evolve accurate models within a much smaller number of iterations when compared with XCSs and UCSs, which are based on stochastic search algorithms, i.e. crossover, mutation, and roulette wheel deletion.

Due to XCSs having a preference to evolve general rules, when over-general rules are incorrectly preserved, XCSs' training performance may reduce. For example, Figure 3.9 shows how the over-general rules impact on reducing training accuracy, from iteration 6000 to iteration 10000, where the number of over-general rules at generalization level 5 is in-

[2] ASCS is a new version of LCSs introduced in this work, which varies from other LCSs, since ASCS considers the *natural solution* as the primary search objective, Chapter 7 will describe this system in detail

[3] Absumption is an search method that produce specific rules by correct overgeneral rules, this algorithm will be described in Chapter 6

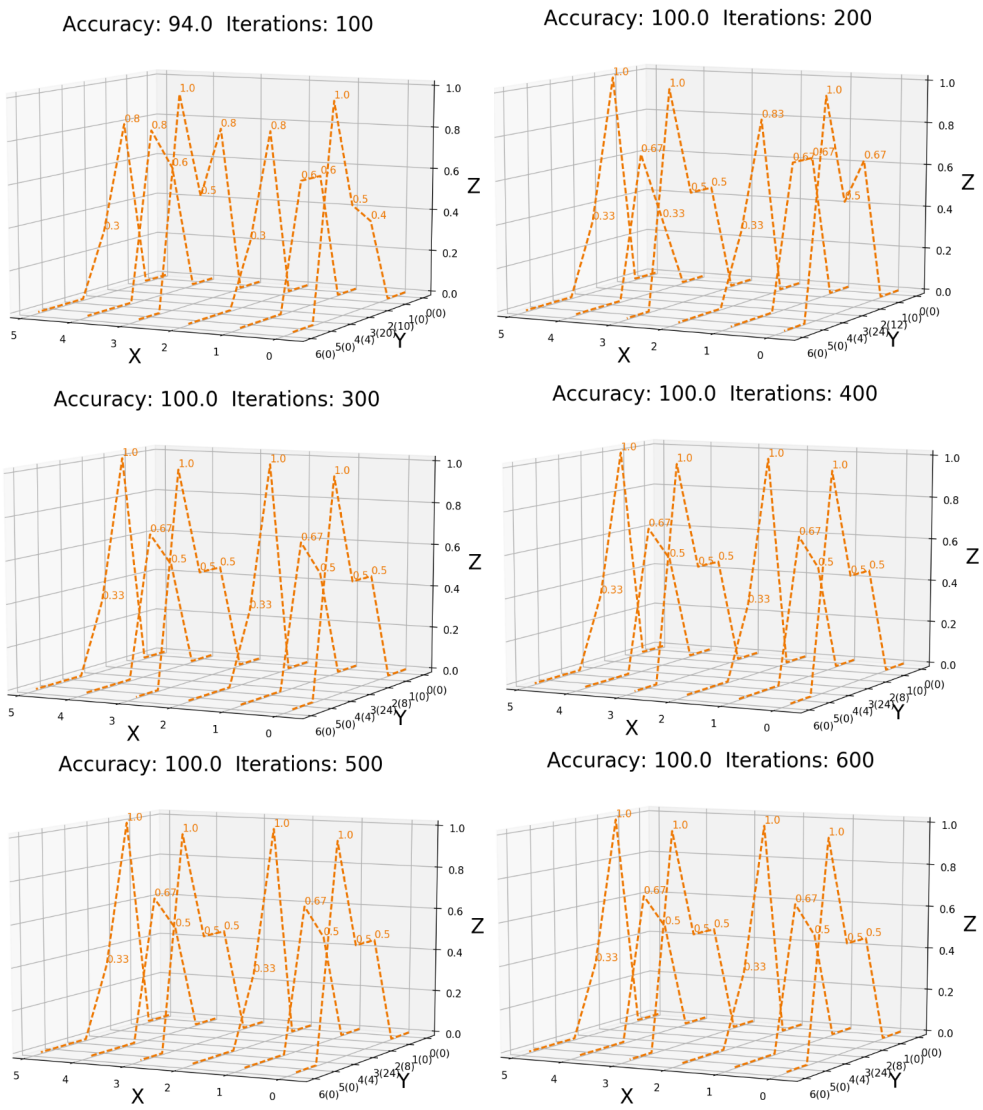


Figure 3.7: The detailed description of X, Y, Z for each graph is in section 3.3. The FIMs records the learning progress of an ASCS for the 6-bits Carry problem, the integers inside the bracket on the Y-axis represent the number of rules at the specific generalization level. FIMs show that ASCSs spend 200 iterations to reach the maximal accuracy. Afterward, 100 more iterations are consumed for evolving the accurate model to the optimal format so that the underlying patterns of the optimal solution can be visualized after 300 iterations.

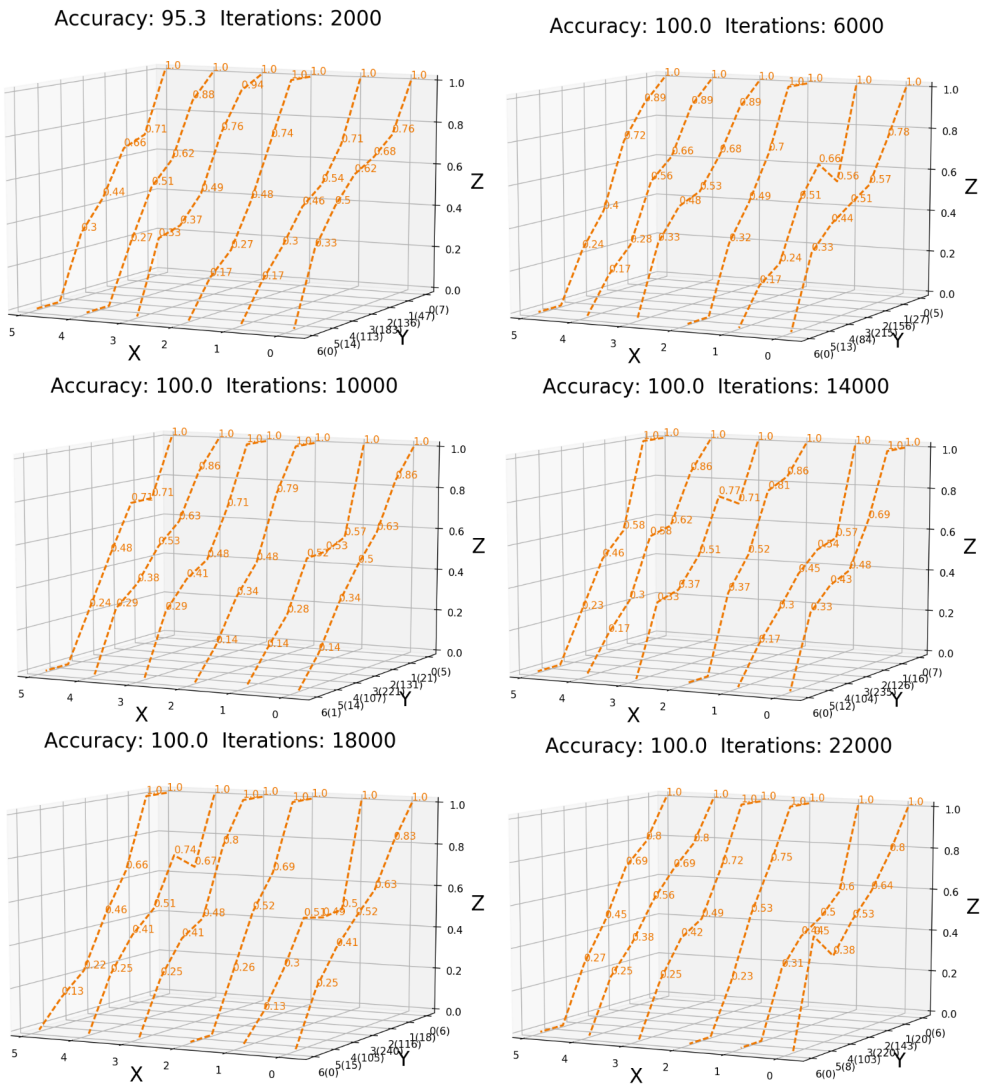


Figure 3.8: These FIMs represent the learning progress of the UCSs for the 6-bits Carry problem. Y-axis, the integers inside the brackets are the summation of the rules' numerosity at a specific generalization level. UCSs need around 6000 iterations to produce an accurate model. Due to UCSs being designed to evolve accurate rules instead of searching for the optimal rules, UCSs' models do not possess human-discernable patterns (see Figure 3.7).

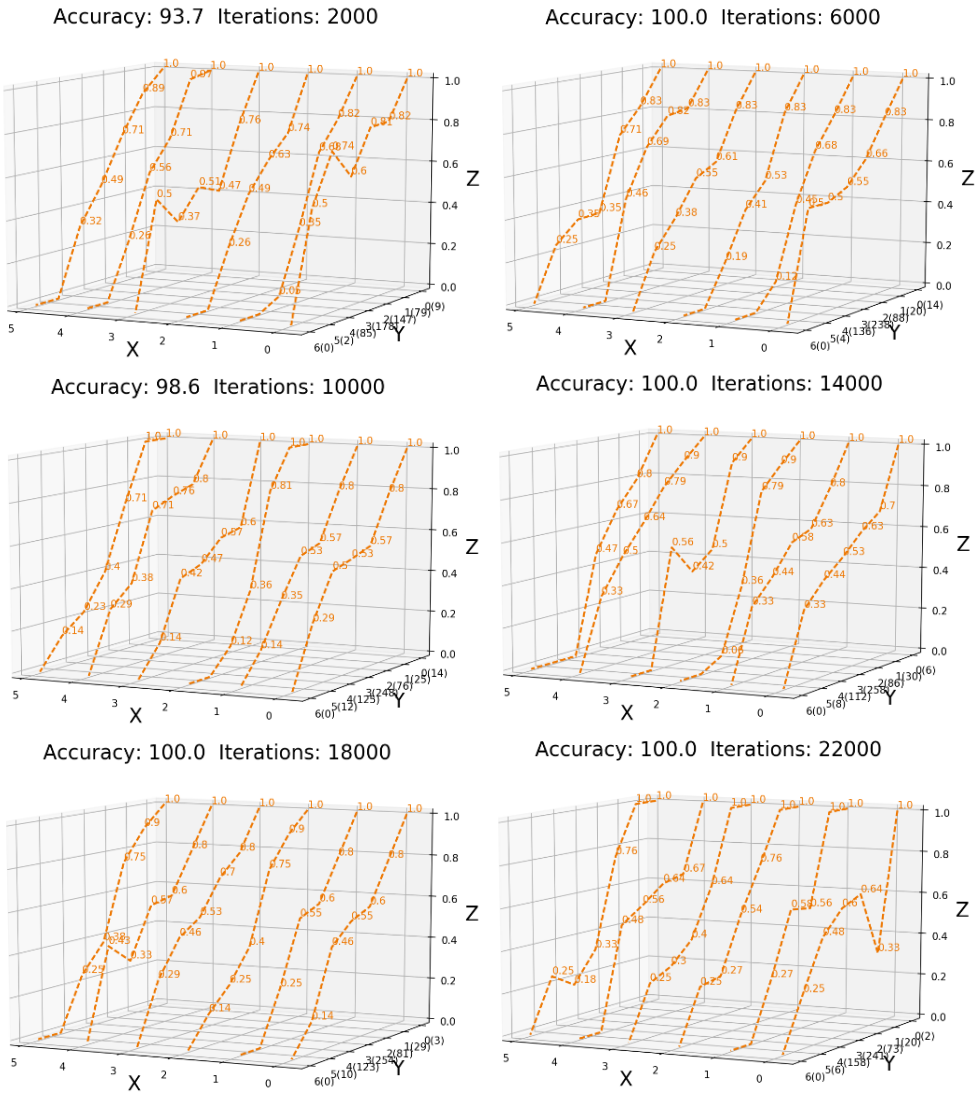


Figure 3.9: These FIMs describe the XCSs' training progress in addressing the 6-bits Carry problem. Y-axis, the integers inside the brackets are the summation of the rules' numerosity at a specific generalization level. XCSs spend 14,000 iterations for evolving consistently accurate models. Due to XCSs being prone to evolve rules that are maximally generalized and accurate, the patterns in XCSs' model only vaguely reflect that the attributes should be divided into two parts.

creased from 4 to 12, which results in XCSs' training accuracy being reduced to 98.6% from 100%.

3.4.2 Tracing the Learning Parameters

FIMs can reflect the LCSs' learning process at a very detailed level. Thus, it is possible to use FIMs to interrogate how the change of a specific training parameter impacts LCSs' training process, i.e. control the generalization level of the evolved rules.

Figure 3.10 and Figure 3.11 are FIMs recorded training processes (respectively from XCSs and UCSs), where $P_{\#}$ is set as a relatively low value (0.01) regarding addressing the 6-bits Carry problem. Comparing with these two figures, XCS is much better than UCS regarding the removal of early generated over-specific rules. This is consistent with the evolutionary characteristics of the two LCSs, where XCSs are prone to preserve correct general rules and that specific rules tend to be removed, but UCSs protects all the correct rules, where subsumption is the only mechanism that prevents UCSs from including over-specific rules.

Furthermore, when compared with the Figure 3.8 and Figure 3.9, where $P_{\#}$ is set as 0.33, the results show that a lower $P_{\#}$ value can benefit LCSs to improve the prediction performance with a much smaller number of learning iterations. However, the models are flooded with over-specific rules.

3.4.3 Investigating the Trained Models

FIMs not only could be employed to investigate the learning progress of LCSs but also can be applied to interrogate the trained models from LCSs. Hence, the search preference of various LCSs can be visualized. Figure 3.12 shows the visualized patterns from XCSs, UCSs, and ASCSs on three problems, i.e. 11-bits Multiplexer, 12-bits Carry, and 13-bits Majority-On,

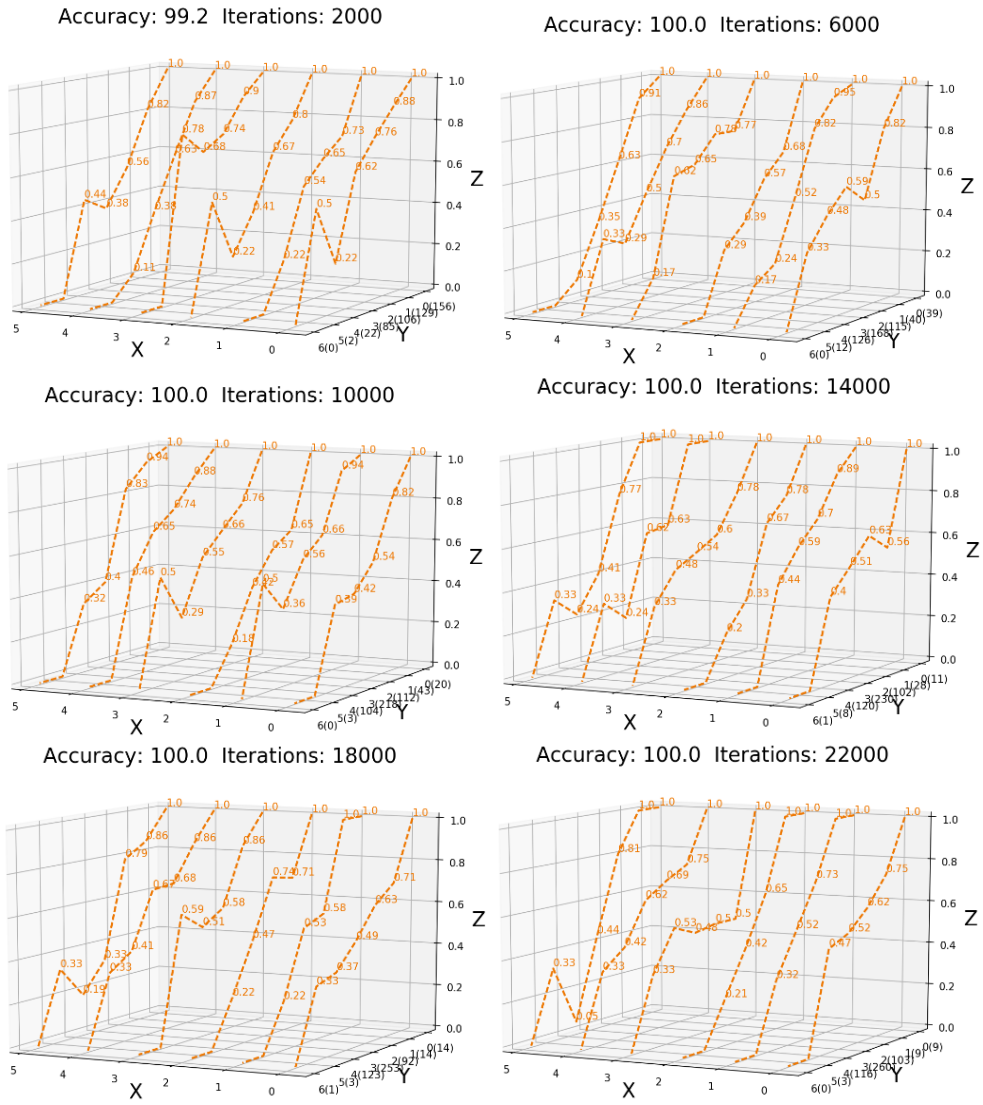


Figure 3.10: The FIMs for the XCSs' training progress regarding addressing the 6-bits Carry problem, where $P_{\#}$ is set with a relatively low value, that is 0.01. All the rules locate at the generalization level 0 and 1 are over-specific for this problem. Y-axis, the integers inside the brackets are the summation of the rules' numerosity at a specific generalization level. The FIMs show that at the beginning of the learning, the evolved model employed a lot of very specific rules to represent the explored domain, where 156 rules and 129 rules are located respectively at generalization level 0 and generalization 1, but after 20000 iterations, only 18 rules are in the generalization level 0 and level 1. This evidence shows that XCSs can effectively remove the over-specific rules as the learning progresses.

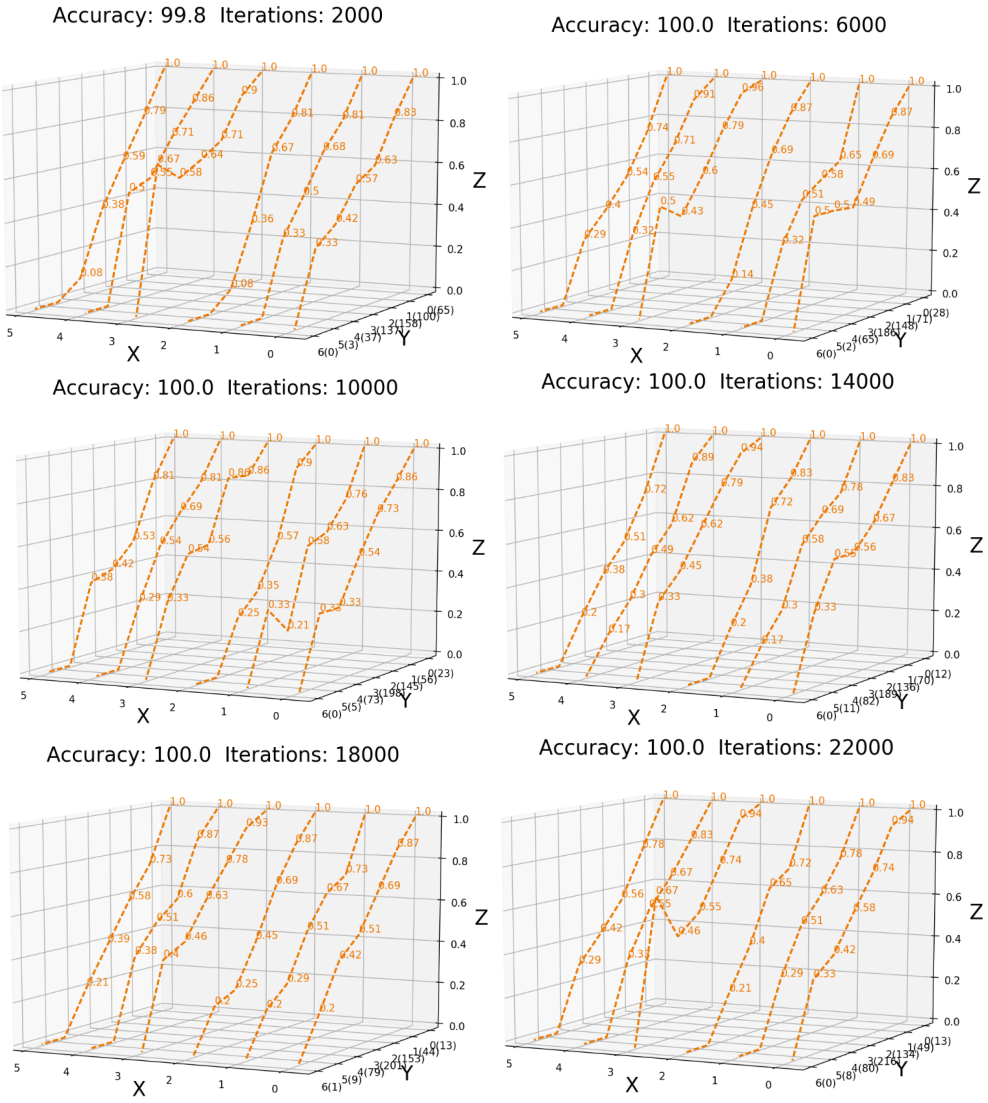


Figure 3.11: These FIMs regarding the learning progress of the UCSs for the 6-bits Carry problem with $P_{\#}$ is set as 0.01. All the rules are located at the generalization level 0 and 1 are over-specific for this problem. Y-axis, the integers inside the brackets are the summation of the rules' numerosity at a specific generalization level. The results show that at the beginning of the learning, a relatively low $P_{\#}$ can benefit UCSs in improving prediction performance. The results also evidence that UCSs does not excel in removing early generated over-specific rules, as at 10000 iterations there are 56 over-specific rules at generalization level 1, after another 12000 iterations, only 7 of the over-specific rules have been removed by the UCSs.

all the visualized models have been trained sufficiently to ensure the rules have converged to the most suitable generalization level.

All the LCSs can fully address the 11-bits Multiplexer problem. The results show that even in XCSs and UCSs' fully representative accurate models, many over-general and over-specific rules exist. This evidences that correct over-specific rules and optimal rules can neutralize the over-general rules' negative impact on prediction, which can be an advantage of using multiple rules to construct models. In general, XCSs possess a larger number of over-general rules but have a smaller number of over-specific rules compared with UCSs. This is consistent with XCSs having a preference for evolving general rules and UCSs having a lack of methods for removing over-specific rules.

The Carry problem and the Majority-On problem have an overlapping issue. This issue describes that in specific domains, the optimal rules share overlapping niches (represented instances) that results in LCSs generating good performance over-general rules such that LCSs' optimal models are under the risk of being dominated by the over-general rules. Due to this issue, XCSs and UCSs cannot generate completely accurate models for the Carry and Majority-On problems. The presented FIMs can visualize the over-general phenomenon caused by the overlapping issue for XCSs and UCSs. The results also evidence that ASCSs can adapt to the overlapping issue as ASCSs can produce the optimal solutions for domains that have an overlapping issue.

3.4.4 Assessing the Optimization Performance

LCSs produced models frequently contain a considerable number of redundant rules and irrelevant rules. For the sake of discovering the correct patterns under LCSs' models, rule compaction algorithms are developed that aim to remove the problematic rules from the LCSs' produced models. However, due to the lack of techniques for understanding the patterns in

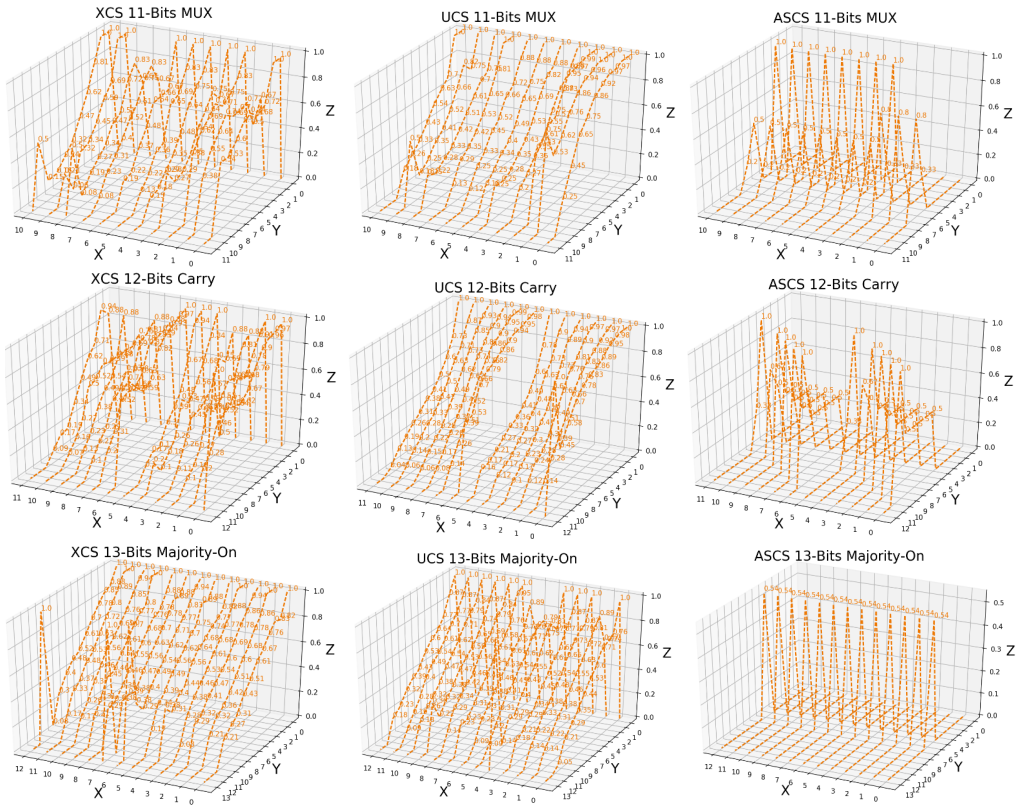


Figure 3.12: Nine FIMs for the trained LCSs models from XCS, UCS, and ASCS that addressed the 11-bits Multiplexer, 12-bits Carry, and 13-bits Majority-On problems. The results show how important it is to obtain the LCSs' optimal solutions, as only these solutions possess patterns that are easily human-discernable (shown in ASCS's results). The results show that XCSs' and UCSs' trained models contain many redundant rules (over-specific rules) and irrelevant rules (over-general rules). Hence, visualizing the results from XCSs and UCSs cannot precisely reflect the ground truth of the explored domain, i.e. the important difference between the address-bits and the data-bits in the Multiplexer problem and all the optimal rules for addressing the odd number attributes Majority-On problem only locating at a single generalization level. The results of the Carry problem from XCS and UCS still show that the attributes should be divided into two parts, but not as precisely as ASCS accomplishes.

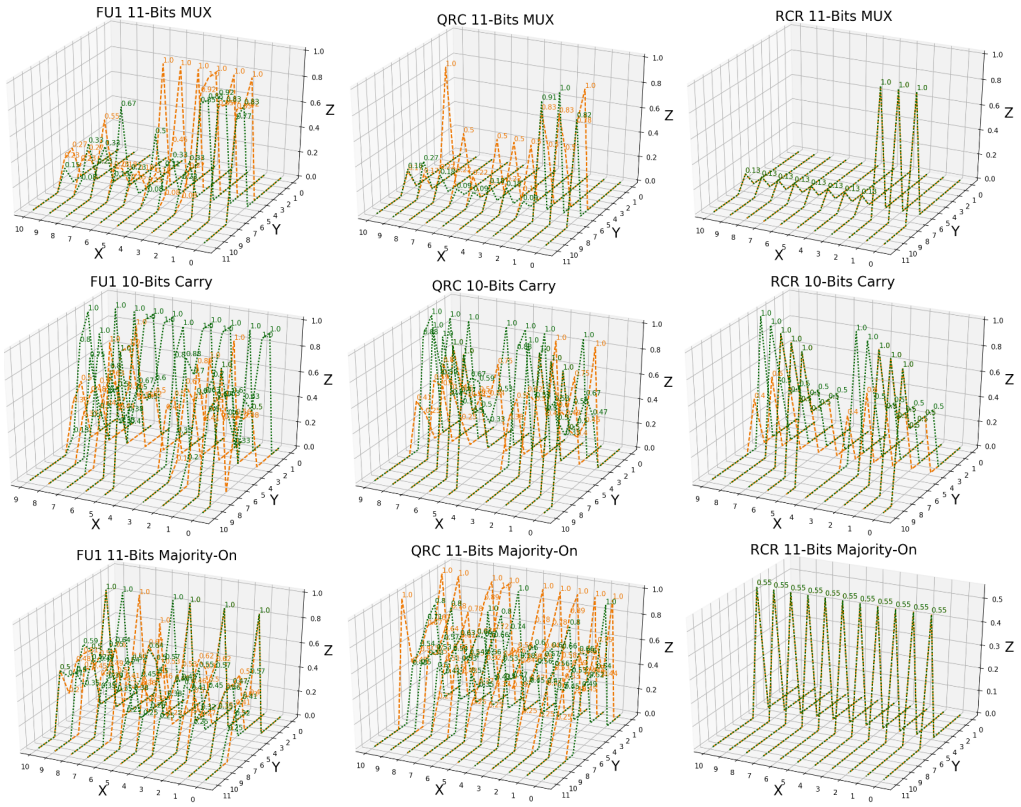


Figure 3.13: AFIMs visualize the optimized XCSs' models that are compacted by the Fu's first approach (FU1), Quick Rule Compaction (QRC), and Razor Cluster Razor (RCR) regarding the 11-bits Multiplexer (MUX), 10-bits Carry and 11-bits Majority-On problems. Graphs for action 1 and action 0 are respectively colored with green and orange. The results show that FU1 and QRC compacted models still contain redundant and irrelevant rules. In contrast, the RCR successfully compacts the models to the optimal format, so that the correct graphs can be produced.

LCSs' models, previously, the testing accuracy is the only measure for assessing a compaction algorithm's performance. As a result, although previous rule compaction algorithms frequently reduce the original models' training accuracy, no explanation exists for this phenomenon. The AFIMs enable the patterns in the compacted models to be interrogatable, which inspires a hypothesis to explain the reduced training accuracy. That is: the previous rule compaction algorithms cannot distinguish the good performance over-general rules and the optimal rules in a single model, which results in the reduction of the prediction performance.

Figure 3.13 shows the patterns in the compacted XCSs' models from Fu's first approach (FU1), Quick Rule Compaction (QRC), and Razor Cluster Razor (RCR [4]). FU1 is designed to remove rules that are redundant for maintaining the training accuracy, and QRC aims to detect the best rule for each instance in the training set, then construct a new model with these best rules. Both FU1 and QRC act on a single model. Whereas, RCR operates on multiple models, which intends to search for the correct unsubsumable rules in the global search space. The results show that compacting multiple LCSs' models is a promising method for discovering LCSs' optimal solutions. RCR successfully produces the optimal solutions for all the tested problems, even in domains in which over-general rules dominate models, i.e. the 10-bits Carry and 11-bits Majority-On problems, where the visualized graphs demonstrated there are over-general rules existing in FU1 and QRC compacted solutions.

All three compaction algorithms successfully remove the problematic rules from the models for the 11-bits Multiplexer problem, but only RCR produces a correct visualization result. This is because of the optimal solution of the Multiplexer problems can be either [O]s or a *natural solution*. However, FU1 and QRC frequently omit some member rules of *natural so-*

[4] RCR is a rule compaction algorithm that applies the compacting on multiple models to produce a subset that all member rules are consistent and unsubsumable. This algorithm will be described in Chapter 4

lution or incorrectly introduce the *natural solution*'s member rules to [*O*]s. As a result, although the FU1 and QRC compacted models only possess optimal individual rules, such rulesets do not satisfy the requirement of *natural solution* nor [*O*]s. Thus, these rulesets cannot enable a correct graph to be visualized.

3.4.5 Visualizing Patterns

LCSs' models are composed of human-readable rules. Thus, previously LCSs have claimed to be promising techniques to discover the ground truth of the explored domains. However, in specific problems, LCSs employ a large number of cooperating rules to construct an accurate model, which hamper humans from understanding the underlying patterns of LCSs' optimal solutions by reading the rules. As a result, LCSs' capability of applying to data mining tasks are underestimated. In this scenario, AFIMs and AFVMs are developed to improve the interpretability of the patterns in LCSs' optimal solutions.

LCSs excel in addressing the Multiplexer problem as they can cope with epistasis and heterogeneity, e.g. many versions of LCSs have completely solved the 70-bits case. According to such achievements, the community believes LCSs can adapt to large-scale domains by merely tuning the training parameters, e.g. control the initial generalization level of the covering mechanism, i.e. tuning $P_{\#}$. However, the transfer of the parameters control strategies to other fields cannot improve LCSs' adaptability in addressing these problems, e.g. XCSs even fails in addressing 12-bits Carry or 10-bits Majority-On regardless of how the training parameters are set. Previously, this phenomenon was hard to be explained because there exists an expected model for representing the Multiplexer problems. Such prejudice could result in only the expected rules in LCSs' solutions being appreciated. Other rules that are neither redundant nor irrelevant could be ignored since it is easy for humans to understand the patterns in

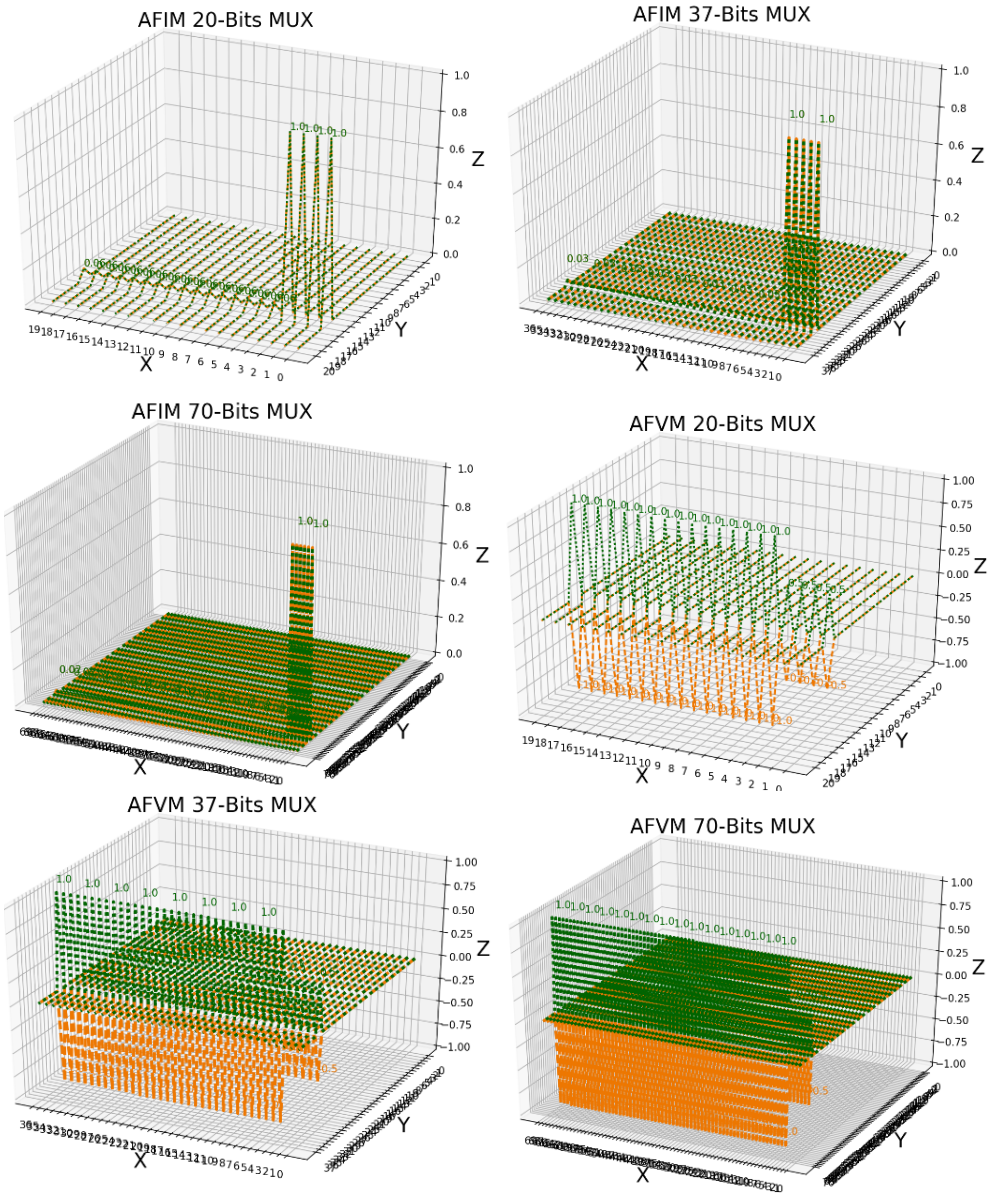


Figure 3.14: AFIMs and AFVMs visualize the 20-bits, 37-bits, and 70-bits Multiplexer problems. Patterns of action 1 and action 0 are respectively colored with green and orange. The visualized patterns consistently reflect the difference between address-bits and data-bits regardless of the number of the considered attributes. AFIMs identify that the Multiplexer problems have a balanced class distribution, as the patterns for both actions are completely overlapped. AFVMs show that the specified value in data-bits determine the output action.

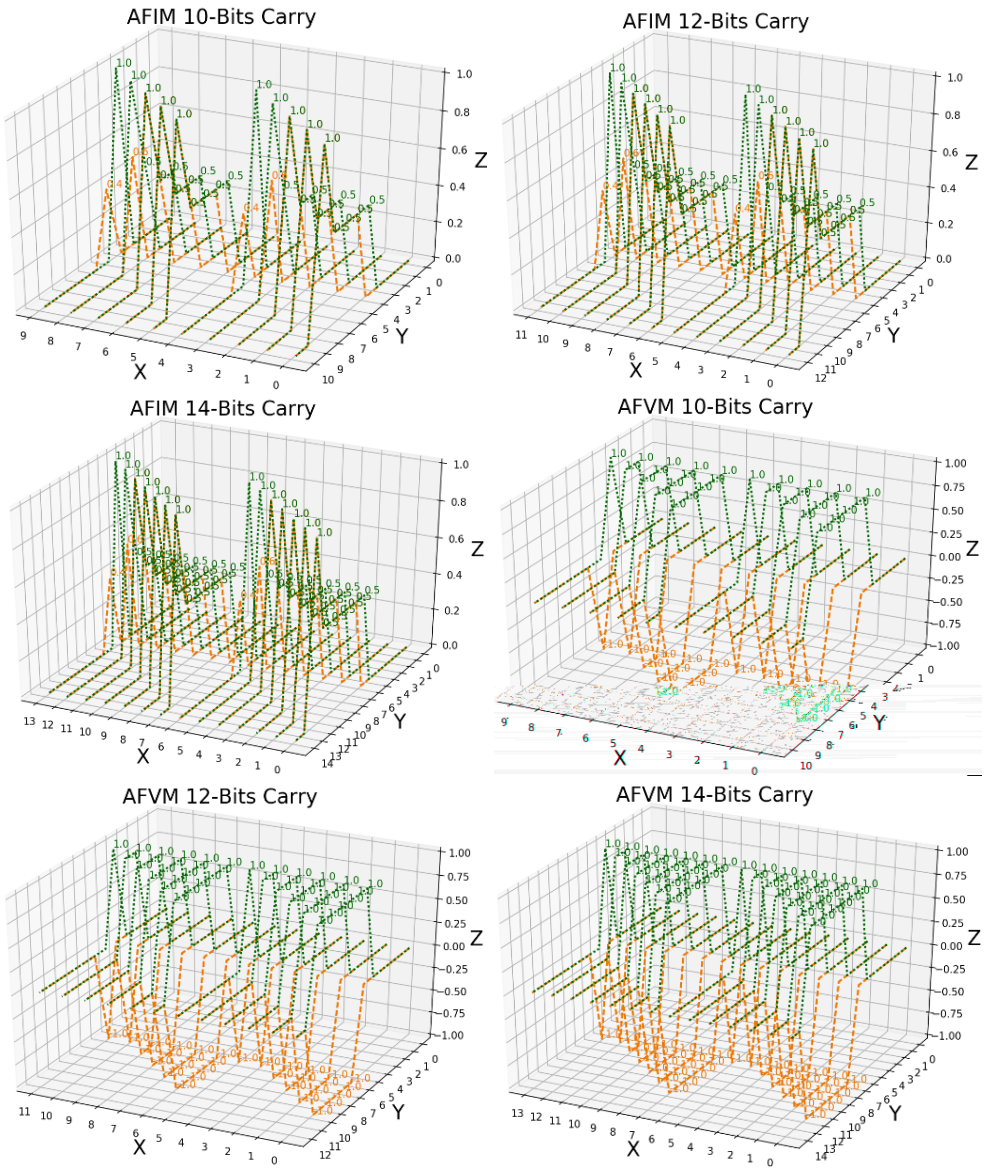


Figure 3.15: AFIMs and AFVMs describe the 10-bits, 12-bits, and 14-bits Carry problems. Patterns for action 1 and action 0 are respectively colored with green and orange. The patterns show that attributes should be split into two parts. AFIMs show that the Carry domains have an imbalanced class distribution with class 0 as the major class. This is because rules that advocated action 0 are located at more general generalization levels, which indicates these rules covered a larger number of instances compared with rules that support action 1. AFVMs show that the optimal rules' specific value is equal to the value of this rule's advocated action.

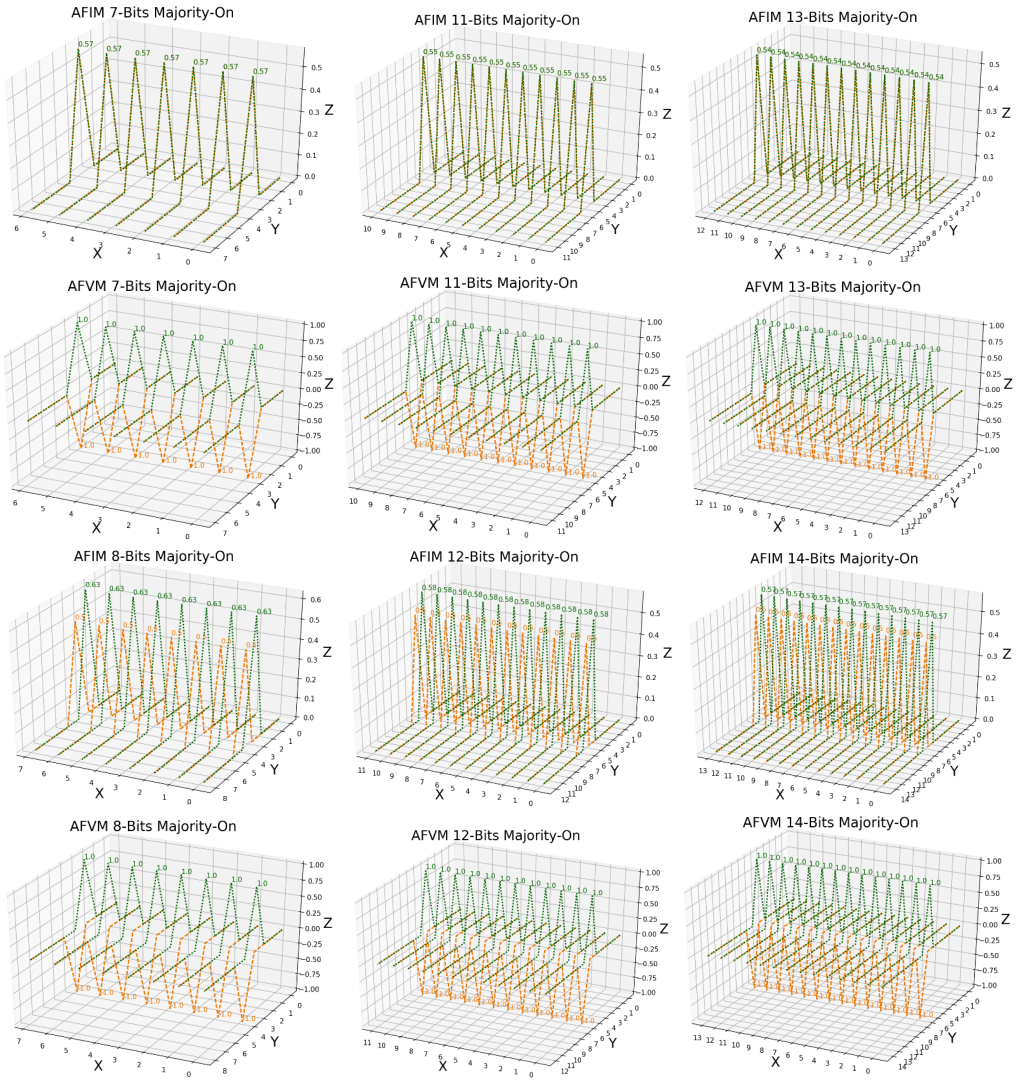


Figure 3.16: AFIMs and AFVMs visualize the 7-bits, 11-bits, 13-bits, 8-bits, 12-bits, and 14-bits Majority-On problems. Patterns of action 1 and action 0 are respectively colored with green and orange. AFIMs show that in the odd case, this problem is a balanced domain, whereas, in even case, this is an imbalanced problem. The problem description of the Majority-On domain does not include this discovery. AFVMs show that the output actions are determined by the majority of the specified value.

an expected model rather than LCSs' natural models, which is composed of a large number of cooperating rules.

In the last two decades, although LCSs have been applied to many fields, the community still has little insight regarding the optimal solutions of LCSs. Benefits of the proposed visualization techniques include that the underlying patterns in complex LCSs' optimal solutions can be interrogated, which leads to the discovery of the *natural solution*s. *Natural solution*s are composed of the consistent unsubsumable rules in the global search space. *Natural solution*s and [O] that are built by unsubsumable rules, contain human-discernable patterns that can reflect the ground truth of the explored domain, e.g. shown in Figure 3.12 is a *natural solution* of the 11-bits Multiplexer problem; in contrast, Figure 3.13 shows the patterns of 11-bits Multiplexer problem's [O]. All domains contain *natural solution*s, but fewer problems possess [O] that are only composed of unsubsumable rules.

Figure 3.14, Figure 3.15, and Figure 3.16 respectively show the AFIMs and AFVMs for the Multiplexer, Carry, and Majority-On domains. The visualization results show that in the same domain the patterns are consistent regardless of the number of considered attributes, i.e. in the Multiplexer domains, the difference between address-bits and data-bits, in the Carry domains, attributes should split into two parts, and in the Majority-On domains, optimal rules advocating the same actions are at the same generalization level. AFVMs show that the Multiplexer problems' output actions are determined by the data-bits, which is consistent with the ground truth of this domain. Furthermore, AFIMs indicate that in odd situations (an odd number of problem bits), the Majority-On problems have a balanced class distribution, whereas, in the case of even situations, the Majority-On domains possess an imbalanced class distribution with class 0 as the major class. Such discovery can not be found in the problem description, which shows that visualization techniques can help humans to understand previously unknown characteristics of the explored domains.

3.4.6 Highlighting Redundant Attributes

When LCSs' models reach the optimal format, the redundant attributes can be identified. Due to these attributes having no contribution in determining the prediction results, the attribute importance of these redundant attributes will be zero, consistently. This results in FIMs being practical methods in feature selection regarding highlighting the redundant attributes. Figure 3.17 shows FIMs successfully identifying all the redundant attributes in the explored hidden Boolean domains.

3.5 Conclusions of Visualization Techniques

A definition of a new style of LCSs' optimal solution (*natural solution*) is proposed. A *natural solution* is unique and deterministic for a certain training set, and this solution is a ruleset that includes all the consistent and unsubsumable rules in the global search space. The hypothesis of *natural solution* completes the Butz and Kovacs' [*O*] hypothesis and contains human-discernable patterns that reveal the ground truth of the addressed problems. Furthermore, three types of three-dimensional visualization techniques for LCSs are proposed, i.e. FIMs, AFIMs, and AFVMs. FIMs visualize the LCSs' models at the macro level. FIMs ease the understanding of patterns and can reflect the rules' generalization distribution and attribute importance distribution, especially in an LCSs' optimal model. AFIMs and AFVMs visualize the LCSs' model at the micro level. AFIMs excel in discovering the explored domains' class distribution. AFVMs can be adapted to investigate how attributes determine the output actions.

The presented work demonstrates that the proposed visualization methods are promising techniques in improving the comprehension level of LCSs. Where the first impressive outcome is that FIMs can precisely visualize the development of the rules' generalization in LCSs' learning process. This outcome not only enables the searching preference among var-

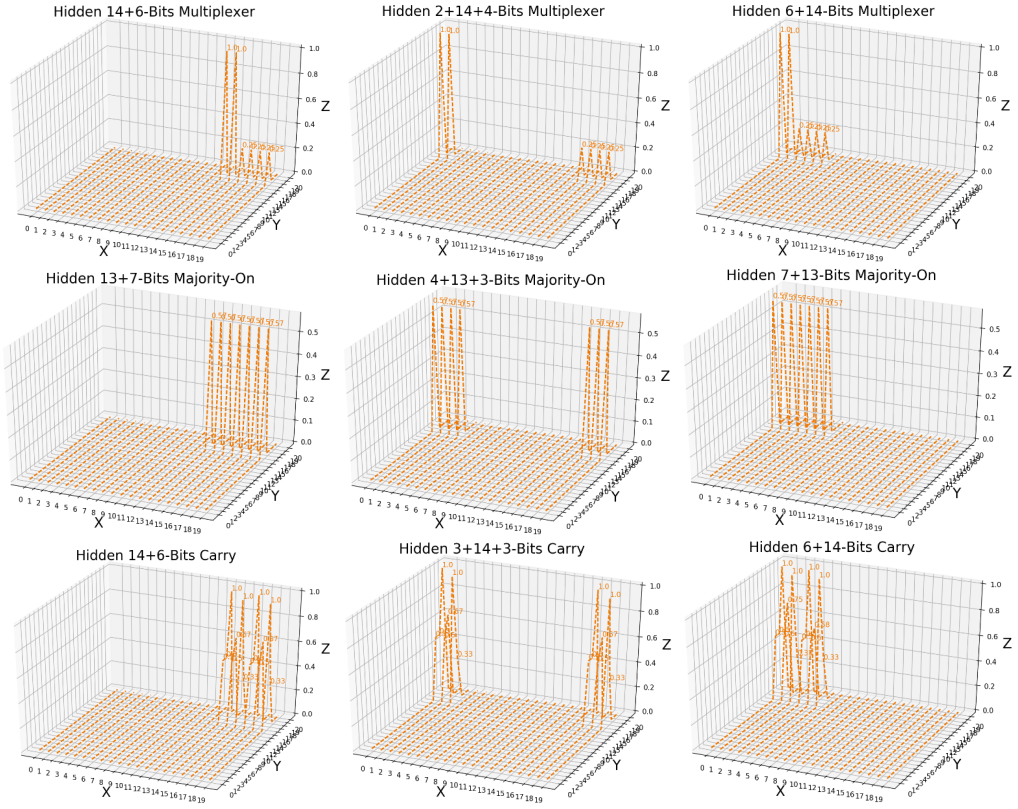


Figure 3.17: FIMs describe the 20-bits hidden problems, including the Multiplexer, Carry, and Majority-On domains, where redundant attributes are placed at different locations. Results show the visualizations can identify the redundant attributes despite a shift in the redundant attributes, as these attributes' importance is consistently zero.

ious types of LCSs to be discoverable but also facilitates the understanding of how a specific training parameter effects the learning process of an LCS. The second outcome is that FIMs, AFIMs, and AFVMs can visualize the underlying patterns in an LCSs' produced model. This supports the hypothesis of *natural solution* and shows these visualization methods are useful in assessing the performance of LCSs' compaction algorithms (optimization algorithms), i.e. evaluate the capability of compaction algorithms in ascertaining the optimal rules by visualizing patterns in a compacted model. Besides, FIMs for hidden problems demonstrate FIMs' potential in assisting LCSs in adapting to feature selection regarding identifying redundant patterns.

Previously, humans have a preference for detecting the expected rulesets from the trained populations of LCSs but ignore other possible rulesets. This is because the underlying knowledge is too difficult to understand in an LCSs' *natural solution* that may contain thousands of rules. The proposed visualization techniques can reduce this prejudice since AFIMs and AFVMs can precisely translate the underlying patterns in the LCSs' optimal solutions to human-discernable graphs. This enables the knowledge in a complex ruleset to be investigatable, which significantly advances human's understanding of the ground truth of the LCSs explored domains and LCSs themselves.

Four previous misconceptions of LCSs can now be clarified, which can be seen from the visualized results. (1) The optimal solutions of LCSs are formed of all the consistent unsubsumable rules in the global search space, rather than the minimum number of correct maximally generalized rules. (2) An LCSs' trained model can reach the maximum accuracy without including all the member rules of the optimal solution. Hence, compacting can fail in detecting an optimal solution when running on a single model that even possesses a high performance in prediction. (3) Although LCSs are practicable in addressing large-scale Multiplexer problems, due to the over-lapping issue, LCSs can not adapt to middle-scale over-lapping

domains without deterministic search algorithms, e.g. the absumption method. (4) The capability of LCSs in data mining has been underestimated, as LCSs can use a large number of cooperating rules to describe a complex problem, and the patterns in the solutions are naturally interpretable since these patterns can be translated into a human-discernable format, including highly complex solutions.

Chapter 4

Rule Compaction Algorithm

This chapter proposes a hypothesis for explaining why previous compaction algorithms may decrease the training accuracy after compaction. Subsequently, a novel way to compact LCS's trained models is also introduced. The new algorithm is termed Razor Cluster Razor (RCR), which innovatively handles the compaction based on multiple LCS produced models, rather than applying the compaction to a single model like other standard compaction algorithms. Furthermore, two variants of RCR, termed RCR2 and RCR3, are also described to further explore the benefits of this multiple population-based compaction algorithm. RCR2 considers the natural solution as the primary search objective, and RCR3 is designed to adapt to models that have insufficient training performance.

4.1 Introduction

The first rule compaction algorithm for LCSs was the Compact Ruleset Algorithm (CRA) [101], which was developed by Wilson in 2001. CRA was applied to an integer encoded LCS (XCSI) to search for a minimal rule subset from a trained population that was sufficient to solve a problem. A successful application was data mining on the Wisconsin Breast Cancer dataset (WBC), where CRA compacted a population with 100% training

accuracy into a much smaller subset while preserving the maximal training accuracy.

CRA was designed to function on well-trained populations that had 100% accuracy. Although CRA achieves excellent performance on the WBC problem, there are problems where LCSs rarely reach the required training accuracy. As a result, such a rigorous precondition of 100% training accuracy hampers CRA's applications to adapt to some LCSs' addressed problems.

In 2002, inspired by CRA, Fu and Davis developed three compaction algorithms named Fu1, Fu2, and Fu3 [33], which are specialized in managing populations with non-perfect training accuracy. CRA, Fu1, and Fu3 achieved the task of rule removal by analyzing rules from the macro viewpoint (population) and employed step-wise accuracy evaluation to quantify the importance of a rule to a population. Although the employed strategies achieved good performance, the extensive repetition of accuracy testing results in a severe computation complexity issue. This limits CRA and Fu's approaches to adapt to domains that involve a large number of training instances.

In this scenario, Dixon proposed the Alternative Reduction Algorithm [25], namely CRA2. CRA2 analyzes rules from the viewpoint of micro-population (i.e. match-set) and utilizes a product of several training parameters to guide compaction so that CRA2 is free from excessive accuracy evaluations. The performance of CRA2 depends on the quality of the rules. Due to the lack of methods for appraising the accuracy of rules when rules' parameters were assigned inappropriately, CRA2 has the risk of incorrectly replacing an important rule with a problematic rule.

To combat CRA2's primary disadvantage, Kharbat proposed a new approach (K1) [47]. K1 follows the architecture of CRA2 but utilizes the training set to re-evaluate the entropy for each rule, and K1 assumes the value of entropy can reflect a rule's performance. Therefore, the compaction of K1 is based on entropy. K1 surpassed other compaction algorithms in de-

tecting [O][1]. However, if a model does not possess an [O], K1's compacted results will have a much lower accuracy than the original accuracy.

Up to this point, rule compaction algorithms had the issue of reducing population size in conjunction with reducing the overall population performance. Motivated by this, Urbanowicz examined the above algorithms and proposed Quick Rule Compaction (QRC) together with Parameter Driven Rule Compaction (PDRC) [85]. Based on an empirical study, PDRC changes the selected parameter combination for searching for a ruleset. QRC slightly alters the architecture of CRA2. Both algorithms surpass previous algorithms in preserving the training accuracy. However, after compaction, these two algorithms still produce alternative rulesets, which contain inconsistent patterns that may produce incorrect visualized results, e.g. unimportant attributes may be highlighted incorrectly.

Previously, compaction algorithms are either inadequate for large domains (as they take too long to check for accuracy preservation) or lack of the interpretability (as they include superfluous rules) for the compacted results. In this scenario of such a predicament, the Razor Cluster Razor (RCR) was introduced as a pragmatic approach for compacting large-size populations in domains with a large number of attributes. RCR considers the [O] as the main compaction objective. If [O] does not exist, RCR attempts to produce a natural solution instead. To investigate whether a model can simultaneously have [O] and natural solution inside, RCR2 is designed, which considers natural solution as the only search objective. Furthermore, RCR's utmost goal is to preserve the underlying patterns of the original populations to enhance the interpretability of the compacted results. However, according to the conducted experiments by this work, RCR could not identify problematic rules when LCSs produced over-general rules that dominated the populations. Thus RCR3 is de-

[1] [O] sets is a style of LCSs' optimal solution that can correctly and completely represent a domain with a minimal number of non-overlapped rules

signed to adapt RCR to the models with insufficient training performance, i.e. not 100% testing accuracy.

A result finding from reviewing the above algorithms is that across multiple LCSs' populations for the same task, although the problematic rules can be different, the accurate rules are common. Furthermore, these common rules can construct a fully representative model, which contains patterns that can consistently refer to the nature of the explored domain, e.g. the data distribution or the importance of features for determining actions. This finding inspires us to investigate how compaction algorithms determine the quality of patterns in visualization.

4.2 Overlapping Distribution and Overlapping issue

Rule compaction algorithms are employed to eliminate rules from a population that do not influence the overall performance. In the articles related to compaction algorithms [4] [19] [46] [92], the most common issue is that the training accuracy is reduced together with the removed rules, e.g. a model's original training accuracy is 100%, after compaction this model's accuracy performance on the training set becomes 97%. Here the issue is termed as *reduced accuracy*. However, up to now, no article has proposed a convincing explanation for this issue. Here it is hypothesized that the incorrectly preserved over-general rules are responsible for the reduced accuracy. It is assumed that when a problem has an overlapping distribution, i.e. the niches of the optimal rules will have overlapping areas and similar generalization levels. Such optimal rules frequently produce over-general rules that have high training accuracy [34]. Such problematic rules are easy to be evolved but difficult to be removed. Thus, over many iterations, a [P] can be flooded by these problematic rules, and this will be a challenge for rule compaction algorithms to distinguish and remove these

problematic rules. Hence, the *reduced accuracy* issue occurs.

When training, LCSs unavoidably introduce over-general rules to their final populations as the GA is continuously searching for plausible better rules [27]. When an over-general rule possesses an innate high accuracy, its training parameters, e.g. numerosity, accuracy, and fitness, are under the risk of being assigned high values improperly. This issue is because LCSs assign rules' training parameters according to the rule's generalization and accuracy. For instance, in an 8-bit Majority-On problem, the accuracy of the over-general rule $000\#\#\#\#\# : 0$ is 96.875%, which is close to 100%. However, this over-general rule represents double the number of instances compared with $0000\#\#\#\# : 0$, which is the maximally general and correct rule. Note, the accuracy of an over-general rule can even reach higher than 99% in the Majority-On problems, which consider more than fourteen attributes. In these scenarios, compaction algorithms, which utilize training parameters to determine whether to remove a rule, can preserve over-general rules.

In practice, if a ruleset consists of over-general rules and over-specific rules that are complementary, then this ruleset still has the potential to reach the maximum accuracy of prediction [97]. This phenomenon is because LCSs utilize the votes of all matched rules to determine the output action for the observed states. Furthermore, each rule estimates its votes from its training parameters. For example, in a 4-bit Majority-On problem (ground truth of this problem shown in Fig.4.1), where, *num* stands for numerosity and *pre* is an abbreviation of prediction. Meanwhile, the value of a rule's vote is equivalent to the product of its *num* and *pre*. Say, a ruleset comprises of five over-general rules $0\#\#\# : 0$, *num*: 12, *pre*: 990; $\#\#0\# : 0$, *num*: 11, *pre*:990 ; $\#\#\#0 : 0$, *num*:12, *pre* 990; $11\#\# : 1$ *num*: 14, *pre* 990; and, $\#\#11 : 1$ *num* 14, *pre* 990; and two over-specific rules $\#100 : 0$, *num* 4 *pre*:1000; and, $00\#1 : 0$, *num* 4, *pre* 1000. Although all members in this ruleset possess a problematic encoding part, in case of prediction, this ruleset still reaches 100% accuracy. In practice, previous

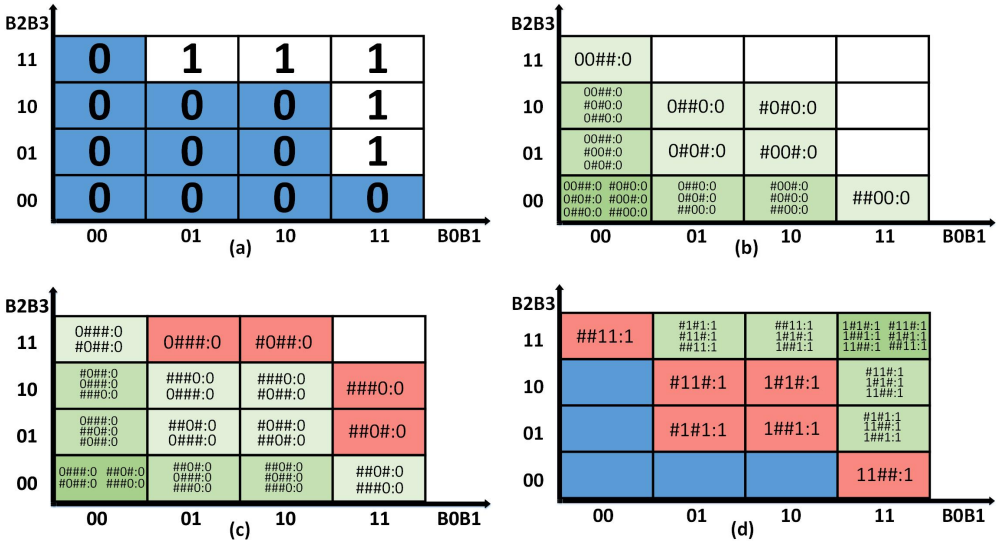


Figure 4.1: Four figures illustrate the over-general issue of LCSs. All figures are based on 4-bit Majority-On problem. An encoding is visualized through the form of $Bit_0Bit_1Bit_2Bit_3$. The horizontal axis depicts Bit_0, and Bit_1 together with the vertical axis that represents Bit_2, and Bit_3. (a) describes the ground truth of action distribution. (b) shows the maximum generalized correct encodings of action zero. (c), and (d) expose the over-general encodings of action zero and one, respectively. In the case of unmatched niches, blue and white apiece specify niches for action zero and action one. Meanwhile, green indicates correctly matched niches, whereas red marks incorrectly matched niches.

compaction algorithms frequently produce such rulesets. Due to the quality of interperability being determined by their encodings, the compacted results with such problematic encodings can result in incorrect patterns for the addressed problem domain.

In general, the risk for LCSs of improperly assigning training parameters to rules increases when exploring domains with an increasing number of features [60]. In order to counteract the negative influence of the incorrectly covered niches for maintaining accuracy more over-specific rules need to be engaged. However, the majority of LCSs follow the principle of searching for rules with a high generalization level. This principle results in issues when LCSs attempt to explore domains that have good performing over-general rules. In these domains, over-general rules can dominate a population, and then not enough capacity remains for storing rules that are specific but correct [26]. The domination does not mean that LCSs preserve the over-general rules. In these cases, LCSs continuously remove the over-general rules, but soon after the removal, LCSs evolve new over-general rules. When the velocity of removal is less than the velocity of evolving, a population can be observed as being dominated by over-general rules. When this issue occurs, all the rules in a population are inexperienced, i.e. low *experience*. In practice, such a problematic population still can reach high training accuracy with a set of over-specific rules that can correct the mispredictions caused by the over-general rules. However, such performance is vulnerable. When the over-specific rules are excluded from these populations, accuracy will decrease. The priority purpose of compaction algorithms is to remove redundant rules, e.g. over-specific rules. Furthermore, the majority of the previous compaction algorithms perform in a single population. Thus, it is anticipated that the compacted results will fail in preserving accuracy for populations that have a severe over-general issue.

4.3 Rule Compaction Algorithms

Three types of new rule compaction algorithms are introduced, which are Razor Cluster Razor (RCR), Razor Cluster Razor2 (RCR2), and Razor Cluster Razor3 (RCR3). RCR is the first algorithm that applies the compaction to multiple LCSs produced models (i.e. populations of rules) in order to produce an optimal solution, which contains patterns that can reflect the nature of the addressed problem. Thus, RCR considers [O] (Butz defined optimal solution, detail described in Section 2.3) as the primary objective. Since LCSs's models may not have an [O], then when [O] does not exist, RCR produces a natural solution as an alternative. As a comparison, the primary objective of RCR2 is to produce a natural solution, regardless of whether [O] exists. Both RCR and RCR2 cannot adapt to models that have insufficient training performance as these methods depend on good performing models. Thus, RCR3 is designed to address models that have an unsatisfactory performance with a rigorous rule verification process. Hence, RCR3 can distinguish optimal rules from high performance over-general rules. As a result, an optimal ruleset can be produced even from a set of problematic LCSs models, e.g. have insufficient training or have been flooded by over-general rules.

4.3.1 Razor Cluster Razor

The main innovation of the RCR is proposing a different definition for the $redundant$ rules. Formerly, being $redundant$ is equivalent to being "replaceable". Here "replaceable" characterizes rules that can be removed without reducing the original performance of the priority criterion, e.g. accuracy of training set in CRA or covering capacity in CRA2. In RCR, $redundant$ describes rules that are either over-general or subsumable rules. over-general depicts rules that have issues in representing a niche precisely. For example, a portion of a rule covered instance opposes this rule's action. Besides, "subsumable" specifies insufficiently general rules. For

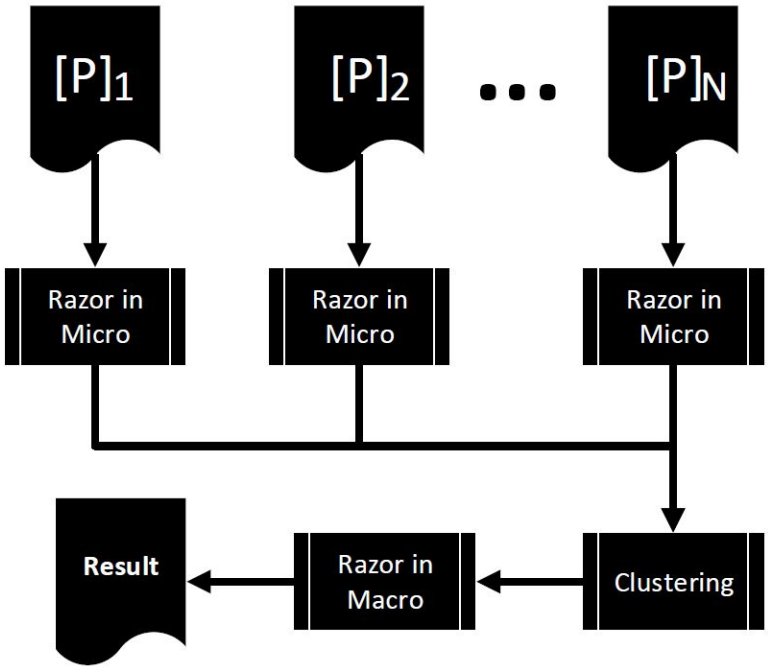


Figure 4.2: The implementation process of Razor Cluster Razor (RCR).

example, a rule is "subsumable", when this rule's covered instances can be completely covered by any other correct rule, which can also cover additional instances.

The altered $redundant$ definition results in RCR identifying redundant rules through investigating the encodings of rules, rather than reviewing the training sets. This transformation of identification ensures the computation complexity of RCR is independent of the number of training instances, which makes RCR preferable to previous compaction algorithms in large domains [2]. Because of the altered identification, RCR's rule selection criterion for rule removal is different from other compaction algorithms. During selection, rather than depending on a rule's contribution to the prediction performance of a population, RCR relies on both a rule's

[2] Large domains describe benchmarks that current computers can not iterate through all the available instances, e.g. 37-bits and 70-bits Multiplexer problems.

correctness and encoding's irreplaceability under the global search space, i.e. unsubsumable. This new selection strategy brings an additional benefit, which is liberating compaction methods from compacting in a single population to being able to compact multiple populations trained for the same problem. Thus, RCR gathers candidate rules from different populations trained on the same task together as the source for compaction. The enriched sources of candidate rules improve the probability of detecting optimal rules for the tested task. As a result, the underlying patterns can be highlighted.

Furthermore, in domains, when only a portion of all possible instances is reviewable due to a time constraint, previous algorithms cannot distinguish over-general rules from rules that are more specific but correct due to lack of the samples for inspection. However, RCR does not suffer this problem, as RCR verifies rules based on multiple populations. Since, although problematic rules can be various, optimal rules are common. Thus, selecting the most common rules from multiple populations can efficiently detect the optimal rules without testing the rules with the whole dataset.

An RCR is composed of three stages: Razor in Micro (RMI), Clustering, and Razor in Macro (RMA). The architecture of RCR is shown in Fig 4.2. RMI is a pre-processing step that removes inaccurate rules and inferior rules from individual populations. In RMI, the criterion for "inaccurate" is a rule's training accuracy. RMI removes rules, where training accuracy does not reach the maximum level of 100% accuracy (note the assumption that the environment contains patterns that enable 100% accurate rule to form). A rule's level of inferiority is estimated by the product of a rule's numerosity and fitness. RMI ranks all rules according to the product in descending order. Then, RMI removes rules in a lower rank (see Algorithm 11), say the last 40% of rules, where past work indicates the lower-ranked rules do not contribute to the solution.

In the attribute importance based representation (e.g. ternary alphabet representation), the global search space for any N-attributes domain

Algorithm 11: Razor in Micro (RMI)

Input: all rules in a trained population P;
the percentage of kept rules $Threshold$
Output: remained rules P

```
foreach rule ∈ P do
    rule.indicator = rule.numerosity*rule.fitness if
     rule.accuracy<1.0 then
        Remove rule from P
    end
end
Rank rule ∈ P by indicator (ascending order);
while P.Size > Threshold* P.Size do
    Remove the first rule in P
end
```

has N+1 sub-search spaces, which are distinguished by the number of the specified attributes. Clustering is responsible for clustering rules in their interrelated sub-search spaces for all the populations. After clustering, all the clusters are ranked by the number of specified attributes from the most general (zero) to the most specific (N). Furthermore, during the clustering process, if a rule possesses the minimum value of $prediction$ (completely incorrect rules), its action will be flipped. Meanwhile, all the duplicated rules are merged by accumulating numerosity and experience (shown in Algorithm 12). Note, the flipping process is only activated for binary class domains. In the case of multiple class domains, all the completely incorrect rules are discarded.

RMA employs subsumption (shown in Algorithm 13) to remove redundant rules from the clustered ruleset. Since subsumption depends on the encodings of rules, the improperly introduced incorrect encodings will adversely affect the correctness of the final results. In this scenario, RMA

Algorithm 12: Clustering

Input: a set of populations after RMI P_Set;
number of involved attributes N;
Output: a set of clusters C

```
1  Initial an empty rule clusters C (an N attribute domain possesses
     N+1 Clusters);
2  foreach P ∈ P_Set do
3      foreach r ∈ P do
4          ID= the number of specified attributes in r.encoding;
5          if r.prediction==0 then
6              flip r.action
7          end
8          if ∃ r' in C[ID], r'.encoding==r.encoding then
9              r'.numerosity+=r.numerosity;
10             r'.experience+=r.experience
11         end
12         else
13             Add r to C[ID]
14         end
15     end
16 end
```

Algorithm 13: Subsumption of encodings

Input: a relatively general rule $rule$;
a relatively specific rule $rule'$;
number of attributes in encoding N;
Output: Whether the $rule'$ can be subsumed

```
1 for i=0; i < N; i++ do
2     if rule.encoding[i] != don't care and rule.encoding[i] !=
        rule'.encoding[i] then
3         return False
4     end
5 end
6 Return True
```

invokes an Error Detector (ED) for diagnosing and removing incorrect rules before executing the subsumption method see Algorithm 14. LCSs intrinsically keep correct rules, and LCSs assign a relatively higher value for these rules in numerosity compare with incorrect rules. Based on this, ED quantifies a rule's risk of being incorrect by accumulating the numerosity of its $conflict$ rules (this accumulated value is termed $conflict$ value). Note, $conflict$ describes rules that have overlapping encodings but support different actions, e.g. rule $1\#\# : 1$ conflicts with rule $\#1\# : 0$.

ED iterates rules in each cluster, three times. In the first time, a rule will be compared with all other rules in more specific clusters. Hence this rule's more specific conflicting rules can be identified so that the conflicting value of this rule can be determined (shown in Algorithm 14). In this iteration, the rules, where their conflicting value surpasses their numerosity are categorized as incorrect rules, so ED removes these incorrect rules. Next, the direction of the comparison is reversed. A rule is compared with all the rules in more general clusters. As above, all the identified incorrect rules are removed. In the last turn, rules are interrogated by the alterna-

Algorithm 14: Error detector (first iteration)

Input: a clustered ruleset $Cluster$;
Output: a clustered ruleset after operating $Clusters$

```
1  foreach Cluster ∈ Clusters do
2      foreach rule ∈ Cluster do
3          Error_value=0;
4          foreach Cluster′ ∈ Cluster and Cluster′ is less general than
            Cluster do
5              foreach rule′ ∈ Cluster′ do
6                  if rule and rule′ have overlapping then
7                      Error_value+= rule′.numerosity
8                  end
9              end
10         end
11         if Error_value > rule.numerosity then
12             Remove rule from Cluster
13         end
14     end
15 end
```

Algorithm 15: Find the [O]

Input: the dominant cluster $cluster$;
Output: a subset of $cluster$ ($Bestsub$)

```
1  Rank the rules ∈ cluster by numerosity in descending order;
2  Maxim_numerosity=0;
3  Bestsub = an empty ruleset;
4  foreach rule ∈ clusters do
5      cluster′=clusters;
6      sub = an empty ruleset;
7      Add rule to sub;
8      foreach rule′ ∈ cluster′ do
9          if rule′ does not overlap with any rule in sub then
10             Add rule′ to sub
11         end
12     end
13     total_numerosity = sum the numerosity of rules in sub;
14     if total_numerosity>Maxim_numerosity then
15         Maxim_numerosity = total_numerosity;
16         Bestsub = sub
17     end
18 end
```

tive rules in the same cluster to identify and remove the last potentially incorrect rules.

After RMA executes the subsumption, RMA will check whether a dominant cluster (DC) exists among the clusters. A DC is a label for the phenomenon, where the majority of rules assemble in a single cluster e.g. in MUX domains. A rule's numerosity represents the number of duplications that a rule has. Then, the number of a cluster's possessed rules is equivalent to the sum of all its rules' numerosity. A DC is apparent, where the DC occupies more than say 80% of the total numerosity amongst all the rules (this threshold is straightforward to tune empirically).

RMA attempts to compact the clustered ruleset further when a domain is determined to have a dominant cluster. RMA achieves further compaction by searching the [O] from the dominant cluster, as displayed in the Algorithm 15. [O] is a set of rules that can completely represent the training samples correctly, and do not represent overlapping niches with each other. However, in some domains, even if a dominant cluster exists, an [O] still can be absent, e.g. the 7-bit Majority-on problem. Hence, rollback will be activated if the attempt of detecting [O] fails. On the other hand, if RMA finds an eligible [O], rules other than members of [O] will be removed.

4.3.2 Razor Cluster Razor 2 (RCR2)

RCR has a priority to search for [O]s. When an [O] exists, RCR does not produce a natural solution. To investigate whether [O] and the natural solution can exist simultaneously, RCR2 is proposed, concentrating on searching the member rules for the natural solution from multiple populations. Furthermore, RCR is designed completely for binary class domains. In RCR, only the completely correct rules are kept. However, in multiclass domains, completely correct rules and completely incorrect rules can carry different patterns. To keep the completeness of the underlying pat-

terns to enable a consummate visualization. After RCR, the RCR2 and RCR3 methods preserve all rules that are consistent (both completely correct rules and completely incorrect rules). Thus, RCR2 and RCR3 extend RCR's adaption to multi-class domains.

Algorithm 16: Clustering of RCR2

Output: $OpR \leftarrow$ Absumption created new rules;
Input: a set of populations after RMI P_Set;
number of involved attributes N;
Output: a set of clusters for completely correct rules C_1;
a set of clusters for completely incorrect rules C_2;
Initial two empty rule clusters C_1 and C_2 (an N attribute domain possesses N+1 Clusters);
foreach $P \in P\_Set$ **do**
  foreach $r \in P$ **do**
    ID= the number of specified attributes in $r.encoding$;
    if $r.prediction$==*0 (Minimum)* **then**
      if $\exists\, r' \in C_1[ID]$ and $r'.encoding$==$r.encoding$ **then** $r'.numerosity$+=$r.numerosity$;
      else Add r to $C_1[ID]$;
    end
    if $r.prediction$==*1000 (Maximum)* **then**
      if $\exists\, r' \in C_2[ID]$ and $r'.encoding$==$r.encoding$ **then** $r'.numerosity$+=$r.numerosity$;
      else Add r to $C_2[ID]$
    end
  end
end

Similar to RCR, RCR2 comprises three stages, RMI, Clustering, and RMA. The RMI of RCR2 abolishes the process of inferior rules removal,

as this processing is under the risk of incorrectly removing important but specific rules. Different from [O], some member rules of natural solutions can locate at low generalization levels, these rules may have a relatively low value in both numerosity and fitness. Therefore, this abolition of removing inferior rules is expected to benefit RCR2 in preserving member rules of natural solutions. As mentioned, RCR2 preserves consistent rules. Thus all the rules, which accuracy reached the maximum 100% or the minimum 0%, will be kept after RMI.

Clustering of RCR2 starts with grouping rules into two collections. The grouping is according to whether a rule's prediction is either maximum (completely correct rules) or minimum (completely incorrect rules). During grouping, rules are removed if they do not possess the extreme values of prediction. Furthermore, duplicated rules from different populations are merged by accumulating their numerosity. In each group, rules are clustered based on the number of specified features (details shown in Algorithm 16).

RMA of RCR2 follows RCR's ED and subsumption mechanism, but RCR2 removes the process of searching for [O]. This is because RCR2's primary objective is to search for a natural solution to represent the addressed problem. In general, an [O] has a smaller number of member rules than a natural solution, but this does not indicate [O]s are better under all criteria than natural solutions. First, [O] excludes a portion of accurate maximally generalized rules in the global search space, which may result in the visualized patterns being incomplete. Second, although [O] reveal patterns for the Multiplexer problems, in other domains, none interpretable [O] have been found from LCSs trained models, e.g. the Carry problem and the Majority-On problem.

Algorithm 17: Reviewing the training instances in RCR3

Input: a population of rules P;
all the training instances T;
Output: P with the reviewed training instances for each rule

```
foreach I ∈ T do
    foreach r ∈ P do
        if r cover I then
            If r supports I.action r.correct++;
            If r opposes I.action r.incorrect++;
        end
    end
end
```

4.3.3 Razor Cluster Razor 3 (RCR3)

over-general rules can dominate populations of LCSs when the explored domains have a severe over-general issue, e.g. the Majority-On problems. In these scenarios, LCSs improperly update the training parameters for over-general rules. Furthermore, a common problematic rule may dwell in all LCSs' produced populations. As a result, the strategy of analyzing rules from different populations does not help RCR and RCR2 to identify over-general rules. A practical way to address this issue is that re-evaluating each rule with the whole training set so that optimal rules can be distinguished from the massive problematic rules. Based on this strategy of re-evaluation, RCR3 is proposed.

RCR3 follows the architecture of RCR2, i.e. including RMI, Clustering, and RMA. The RMI of RCR3 begins by attaching two new training parameters to each rule, which are *correct* and *incorrect*. Afterward, the continuity of all involved rules is re-evaluated. During re-evaluating, each rule records the number of its correctly matched samples in the *correct* together with the number of incorrectly matched samples stores in the *in-*

correct (shown in Algorithm 17). A rule fails in evaluation if this rule's product of *correct* and *incorrect* is higher than zero. After evaluation, the RMI removes all rules that fail the evaluation. Note, due to RCR3 reviewing all members in the training set, RCR3 does not adapt to large-scale problems, e.g. the 70-bit Multiplexer problem.

After RMI, rules from different populations are merged, grouped, and clustered under Clustering, which is the same as the Clustering of RCR2. As a comparison of RCR and RCR2, RMA of RCR3 abolishes ED, due to the re-evaluation in RMI, having excluded all the potential over-general rules. RCR3's RMA starts from subsumption to remove over-specific rules. Then RMA ranks rules according to the generalization of their encodings. Lastly, rules undergo the process of Urbanowicz's Quick Rule Compaction (QRC) to remove rules that are redundant in representing the training set (detail of QRC is described in Section 2.3.8). As QRC is involved in the final process, RCR3 differs from RCR and RCR2 in the priority of searching, RCR3 has the priority of searching for a ruleset that has a minimal number of rules to represent the addressed problem but allows over-lapping, this is different from [*O*] or natural solution.

4.4 Results of RCR Evaluation

To examine the capacity of compacted populations from the novel methods to find patterns, three artificial Boolean domains at different scales are considered as benchmark problems. Specifically, **Multiplexer** problems (MUX), **Carry** problems (CARs), and **Majority-On** domains (MAJs). Action based Feature Importance Map (AFIM) and Action based Feature's Average Value Map (AFVM) are employed as the technique to visualize the underlying patterns. The detail of AFIM and AFVM is described in Chapter 3.

Table 4.1: The running time, the number of rules and the accuracy in a compacted population for MUX. Values in parentheses are the number of optimal rules. s, m, and, h are the symbols for the second, minute, and hour, respectively. "−" indicates that an implementation spends more than one week on computing to compact the ruleset, so the result for this implementation is absent.

Problem	MUX6(8)	MUX11(16)	MUX20(32)	MUX(64)	MUX70(128)
CRA	<1s	11s	5.1h	–	–
	8±0	24±6	69±14	–	–
	100%	99.97%	99.96%	–	–
FU1	<1s	29.6m	–	–	–
	9±1	35±10	–	–	–
	100%	99.98%	–	–	–
FU3	<1s	60.3m	–	–	–
	8±0	20±4	–	–	–
	100%	99.98%	–	–	–
CRA2	<1s	2.1s	49.8m	–	–
	9±1	35±10	74±13	–	–
	100%	100%	100%	–	–
K1	<1s	4.7s	1.76h	–	–
	8±0	29±6	64±6	–	–
	100%	100%	100%	–	–

Table 4.2: The running time, the number of rules and the accuracy in a compacted population for MUX. Values in parentheses are the number of optimal rules. s, m, and, h are the symbols for the second, minute, and hour, respectively. "−" indicates that an implementation spends more than one week on computing to compact the ruleset, so the result for this implementation is absent.

Problem	MUX6(8)	MUX11(16)	MUX20(32)	MUX(64)	MUX70(128)
QRC	<1s	<1s	7.1m	–	–
	9±1	35±10	62±10	–	–
	100%	100%	100%	–	–
PDRC	<1s	2.3s	51.85m	–	–
	9±1	30±7	65±7	–	–
	100%	100%	100%	–	–
RCR	<1s	25s	5.4m	5.8m	19.98h
	8±0	16±0	32±0	64±0	128±0
	100%	100%	100%	100%	100%
RCR2	<1s	25s	5.9m	7m	71.9h
	18±0	54±0	160±0	249±15	958±36
	100%	100%	100%	100%	100%
RCR3	<1s	25s	5.9h	–	–
	8±0	20±4	56±4	–	–
	100%	100%	100%	–	–

4.4.1 Result of Multiplexer Problems (MUX)

The over-general issue in MUX is not prevalent, as the domain is non-overlapping. Therefore, LCSs are capable of addressing large scale MUXs, e.g. 70-bits. However, as the scale of MUXs becomes larger, although over-general rules do not dominate a single population, the number of over-general rules in [P] unavoidably increases as the length of condition increases. Note, both the training accuracy and the testing accuracy of the compacted rulesets of evolved populations reach 100%.

According to the results shown in Table 4.1 and Table 4.2 . CRA, Fu1, and Fu3 fail in maintaining the original accuracy after compaction. This failure is because these algorithms' first two steps use a single rule's contribution to maintain global performance to determine the rule's removal. However, as mentioned, LCSs frequently represent domains with a set of rules that are problematic but complementary. Therefore, after the first two steps, a combination of over-general rules and over-specific rules can replace important rules.

Furthermore, the last step of CRA, Fu1, and Fu3 have the same objective, which aims to represent a training set with a minimal number of rules, regardless of the accuracy. Thus, the last step can remove rules that are over-specific but vital to maintaining performance. Straightforwardly, after compaction, the observed performance decreased. Note, due to additional time consumption that is caused by computation complexity, only RCR and RCR2 can adapt to 37-bits and 70-bits MUXs in these tested domains in a timely manner (under a week using a PC of 2.4GHZ CPU and 10GB RAM).

Continuity of Patterns in MUX

All maps shown in Fig. 4.3 represent patterns of feature importance from RCR compacted results, where results for any problem where the compaction algorithm needs more than one week are omitted. According to

the visualization, in all cases, the difference between the address bits and data bits have been highlighted. Furthermore, these patterns exist in all MUX problems regardless of the scale. This indicates that when artificial Boolean domains share the same background logic, no matter how the scale changes, the underlying patterns are similar. These results show the possibility of utilizing patterns at low scale to infer patterns at a larger scale. Patterns can be easily transformable between analogous domains. Moreover, visualizations show that in MUXs, action zero and action one share the same feature importance distribution.

Patterns Comparison Between Different Algorithms

AFVMs reflect the average value of each specified feature. According to the visualizations shown in Fig. 4.4. RCR, RCR2, and RCR3 have visualized that actions are determined by the data bits. For example, in the case of action one, all the features in data bits are specified as one. Meanwhile, when action is zero, in data bits, all the specified features are valued as zero.

Furthermore, RCR and RCR2 present the knowledge that in address bits, the possibility of each feature to be specified as one or zero is equal. Patterns of RCR3 do not show such a phenomenon, due to the lack of completeness when compared with natural solutions or have redundant rules when compared with N-ORCs.

Among visualizations for the three CRA2 analogous algorithms (shown in Fig. 4.5), K1 has the best patterns, which are close to that of all the three RCRs. The difference between the address bits and the data bits is highlighted. CRA2 and QRC also produced acceptable patterns, but their produced patterns are not as transparent as RCRs'.

In general, visualizations of the CRA analogous algorithms have the worst interpretability regarding their patterns (shown in Fig. 4.6). FU3 is the only algorithm that can visualize the underlying patterns of action zero correctly. However, there is noise in the patterns of action one, which may

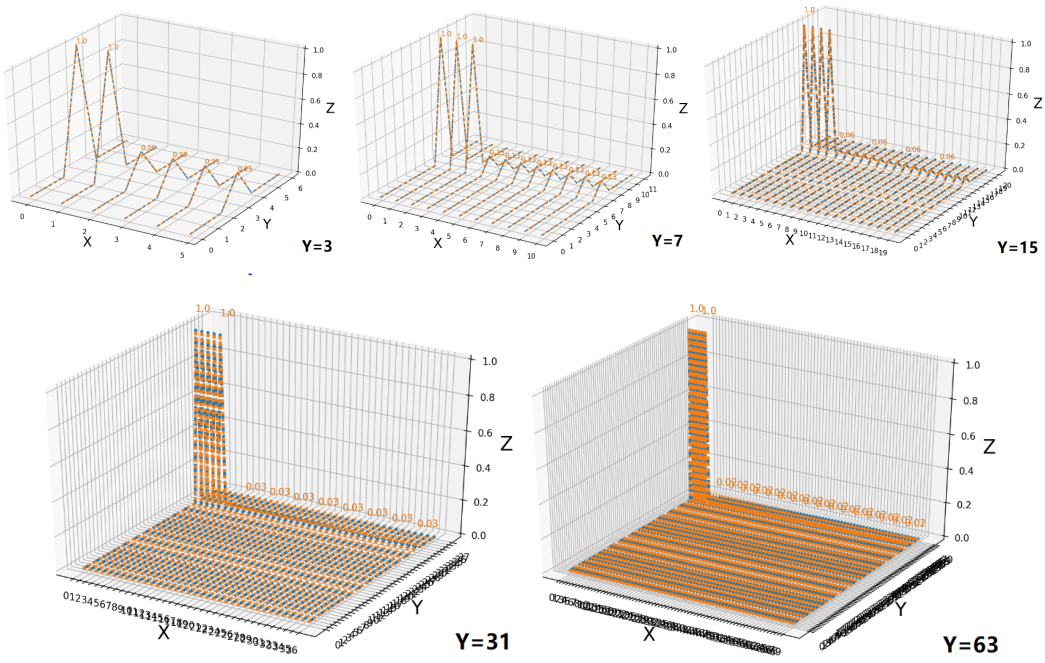


Figure 4.3: AFIMs for five MUX problems based on RCR compacted results, maps following orders of left to right and then from top to bottom are patterns of 6-bits, 11-bits, 20-bits, 37-bits, and 70-bits MUX. Blue represents the patterns of action zero, while orange marks action one. AFIMs aim to show the distribution of the instances for the explored domains. The graphs for action one and action zero are completely overlapped, which shows that the domain is evenly sampled. This demonstrates that MUXs are balanced domains. X-axis: Attribute Id, Y-axis: sub-search space Id, Z-axis: Feature Importance, which describes a feature's specified level at a certain sub-search space.

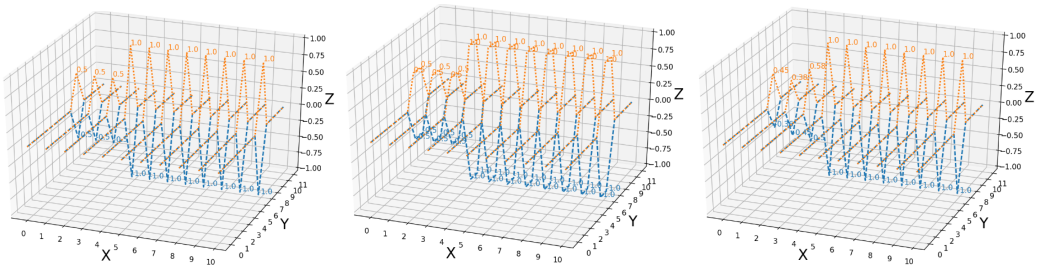


Figure 4.4: AFVMs for 11-bit MUXs, following the order of left to right, are from compacted results of RCR, RCR2, and RCR3. AFVMs aim to show how features determine the action. All visualizations show the difference between the address bits (first three) and data bits (last eight). Furthermore, visualizations show evidence that the value of address bits may determine the action. As a comparison, in RCR and RCR2, patterns for address bits for both actions are symmetric with the value of 0.5, which indicates for both actions, the same strategy is employed to identify address bits. However, patterns from RCR3 cannot show such a symmetric distribution. Therefore, the interpretability of RCR3's result is worse than RCR's N-ORC and RCR2's natural solution. Note, AFIMs (Fig. 4.3) show the importance of specifying a bit, whereas AFVMs (here) show the importance of the value of the bit to the class once specified.

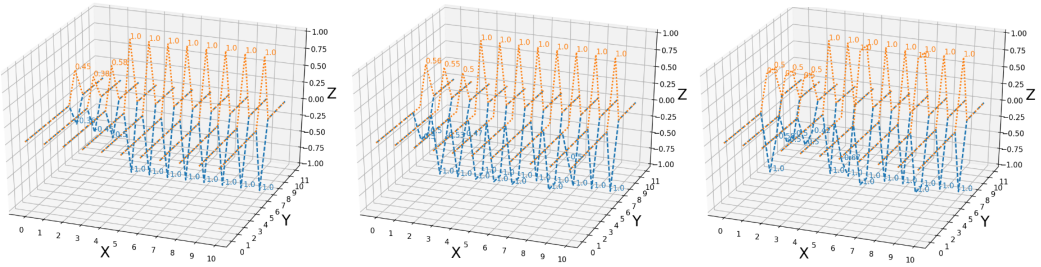


Figure 4.5: AFVMs for 11-bit MUXs, following the order of left to right, are from the compacted results of K1, QRC, and CRA2. Results from CRA2 analogous algorithms show vague patterns for the explored domains. However, compared with RCR's results, the visualizations are less transparent in terms of patterns for identifying address bits.

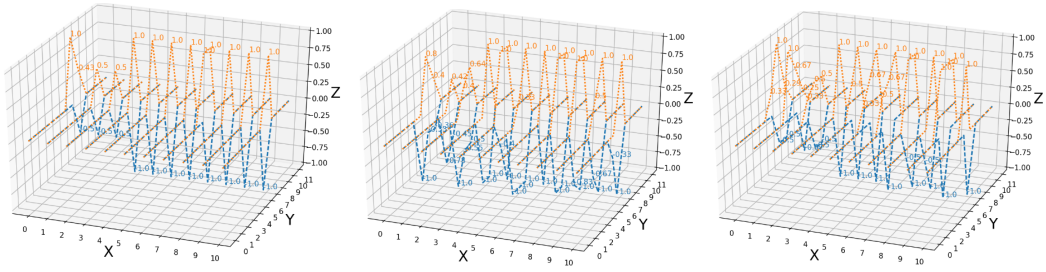


Figure 4.6: AFVMs for 11-bit MUXs, following the order of left to right, are from the compacted results of FU3, FU1, and CRA. The performance of visualization of FU3 is close to the CRA2 analogous algorithms. However, FU1 and CRA show evidence of over-general rules.

reduce the patterns' readability. FU1 and CRA cannot produce compacted results with good readability. These results indicate CRA analogous cannot produce informative patterns for domains, even those without any severe over-general issue.

[*O*] Versus Natural Solution for Different Patterns

RCR2 produces a natural solution, where the selected rules have maximal accuracy and maximal generalization. RCR generates a subset of a natural solution, where the training set is fully represented, and all members' encodings of this subset contain non-overlapping rules (this subset terms [*O*]). RCR3 attempts to compact RCR2's results, but do not consider the property of overlapping when compacting. Thus RCR3 produces alternative rulesets, which are similar to [*O*] that consider completeness, correctness, and minimality, but RCR3' results allow overlapped rules.

Visualizations of the 11-bits MUX problem are shown in Fig 4.7. Patterns related to a natural solution show the number of address-bits is equal to the number of covered sub-search spaces. The [*O*]s do not show such a pattern. However, patterns of [*O*] successfully highlight the difference between address bits and data bits. In the case of RCR3's results, patterns of feature importance are less transparent. Therefore, only natural solu-

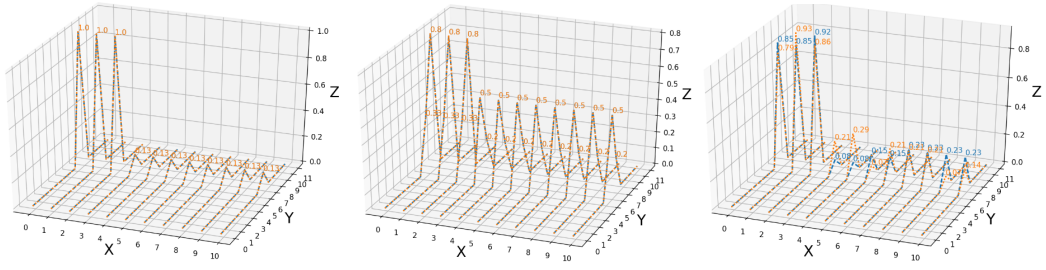


Figure 4.7: AFVMs for 11-bit MUXs, following the order of left to right, are from compacted results of RCR, RCR2, and RCR3. Visualizations show that both [O]s and natural solutions can reflect the MUXs are balanced as patterns for both actions are symmetric. However, patterns of RCR3 do not show such a symmetric relationship between both actions. As a result, RCR3's patterns fail in identifying such a distribution.

tions and [O]s are capable of producing patterns of feature importance that are very transparent. Incorrectly introducing or incorrectly removing only one rule in the compacted results will cause the visualized patterns to loose transparency at a certain level.

Furthermore, the attempt to search for a natural solution fails in MUXs, where the scale is larger than 11-bits. Two factors may be responsible for this failure. Firstly, compared with [O], the number of members in the natural solution is much larger. The difference in size will become more significant when the scale of MUXs enlarges. Therefore, it is difficult for LCSs to evolve and keep all members of the natural solution when the scale is large. Secondly, MUXs can be completely represented by an [O] at all scales, which means that in these cases, a portion of the natural solutions is replaceable. Hence, LCSs may be prone to remove these replaceable rules, which results in RCR2 failing in searching for natural solutions for MUXs in the cases of large scales.

4.4.2 Result of Carry Problems (CAR)

Among the tested artificial Boolean domains, the Carry problems (CARs) have a moderate over-general issue. In these scenarios, K1, as an algorithm that seeks the most general rules, always produces a solution that only comprises rules with a severe over-general problem. Therefore, reading K1's outcomes cannot achieve any informative patterns for the explored CARs. As a comparison, QRC shows promising performance in counteracting the adverse influence of over-general issue with a solution that is close to a natural solution. However, compared with natural solutions produced by RCRs, redundant over-specific rules are still involved in a QRC's result. Other algorithms, such as CRA, FU1, FU3, CRA2, and PDRC, show evidence of how algorithms can be disturbed by over-general rules when compacting.

Another interesting finding is that RCR, RCR2, and RCR3 all produce a natural solution as the final output (shown in Table 4.3 and Table 4.4). This phenomenon is because when using the ternary representation to represent CARs, a natural over-lapping distribution exists. This distribution results in a fully representative ruleset that must involve overlapping rules. Therefore, there is no suitable $[O]$ for CARS, so that RCR and RCR2 produce a natural solution as the output. The phenomenon of RCR3 producing natural solutions indicates that all members of a natural solution are vital to represent the full training set. This discovery evidences that a natural solution reaches the minimal number of rules in a ruleset to represent CARs. This is because alternative fully representative rulesets have to replace a maximally general rule with one or more less general rules, which may enlarge the number of involved rules compared with natural solutions.

Table 4.3: The running time, the number of rules and the accuracy in a compacted population for CAR. Values in parentheses are the number of optimal rules. s, m, and, h are the symbols for the second, minute, and hour, respectively.

Problem	CAR6(18)	CAR8(38)	CAR10(78)	CAR12(158)
CRA	<1s	9.78s	2.3m	2.72h
	17±4	55±26	98±49	345±145
	100%	100%	99.79%	99.69%
FU1	3,1s	3.72m	43.2m	9.64h
	17±1	38±2	76±7	115±12
	99.95%	99.38%	99.39%	98.78%
FU3	3.2s	3.92m	43.4m	9.72h
	15±3	32±5	56±10	54±9
	99.95%	99.38%	99.41%	98.93%
CRA2	<1s	<1s	2.12s	14s
	17±1	38±2	79±2	115±9
	100%	99.99%	99.89%	99.02%
K1	<1s	<1s	4.5s	31.44s
	5±0	5±0	5±0	5±0
	81.25%	77.71%	76.3%	75.35%

Table 4.4: The running time, the number of rules and the accuracy in a compacted population for CAR. Values in parentheses are the number of optimal rules. s, m, and, h are the symbols for the second, minute, and hour, respectively.

Problem	CAR6(18)	CAR8(38)	CAR10(78)	CAR12(158)
QRC	<1s	<1s	<1s	2.01s
	18±0	38±1	81±3	146±10
	100%	100%	100%	99.44%
PDRC	<1s	<1s	2.11s	14.44s
	17±1	34±3	59±9	73±11
	99.95%	99.09%	98.68%	98.26%
RCR	<1s	4.1s	37s	4.1m
	18±0	38±0	78±0	158±0
	100%	100%	100%	100%
RCR2	<1s	5.2s	40s	4.23m
	18±0	38±0	78±0	158±0
	100%	100%	100%	100%
RCR3	<1s	4.1s	37s	4.11m
	18±0	38±0	78±0	158±0
	100%	100%	100%	100%

Continuity of Patterns in CARs:

Although AFVMs contain more information and are simple to visualize patterns for rulesets with problematic rules. AFIMs are easy to read, and are transparent when understanding the continuity of patterns. Therefore, four AFIMs are employed to show the attribute importance distribution of CARs (shown in Fig. 4.8). Similar to MUXs, the underlying patterns of CARs also contain a property of continuity when the length of involved features scales. The patterns clearly show CARs should be split into two equal parts, which follows the policy of how CARs execute.

Furthermore, there is a significant difference between patterns for action zero and action one. This difference indicates that different from MUXs, CARs are imbalanced domains. Moreover, compared to action zero, features for action one are more frequently specified to ascertain an instance of action one. This phenomenon reflects that in CARs, the size of niches of action zero is larger than that of action one, which is consistent with the ground truth of CARs.

Patterns Comparison Between Different Algorithms:

The visualization of RCR's results (shown in Fig. 4.9) clearly shows that the generalization boundary of action one is between four to eight, and for action zero is between five to eight in the 12-bits CAR. This boundary indicates that for action zero, any involved rule that specified more than five features must be over-specific, and the most general rules have eight features that are generalized. Furthermore, this result also shows that in each rule, the value of each specified feature is consistent with the value of its supported action.

In general, the visualization of QRC is close to RCR's. However, there is evidence of QRC's compacted population containing redundant over-specific rules. For example, in action zero, rules specifying six features are kept. These incorrectly kept rules make the visualization of QRC less

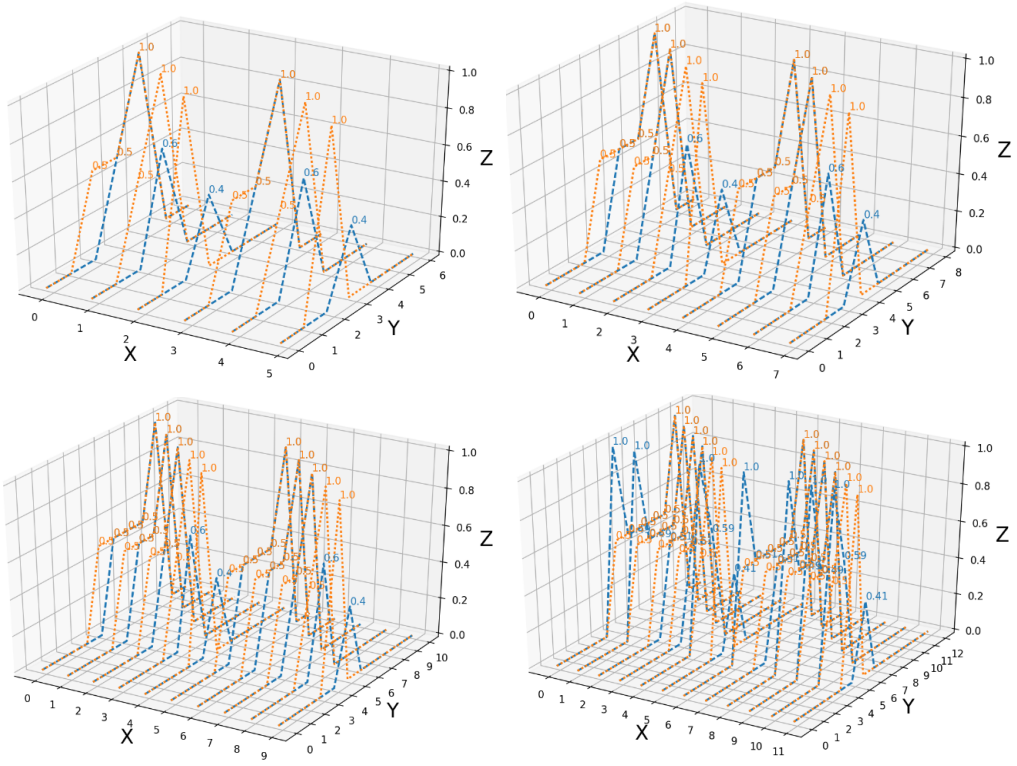


Figure 4.8: AFIMs for four CAR problems based on RCR compacted results, maps following orders of left to right and then from top to bottom are patterns of 6-bits, 8-bits, 10-bits, and 12-bits CARs. Blue represents the patterns of action zero, while orange marks the action one. Graphs demonstrate that CARs are imbalanced domains with action zero as the majority class.

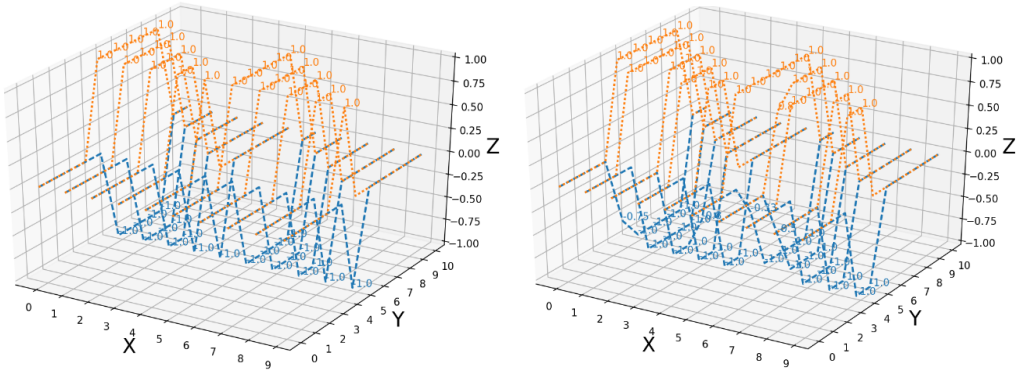


Figure 4.9: AFVMs for 10-bits CARs, following the order of left to right, are from compacted results of RCR, and QRC. Patterns of RCR show that features need to be split into two equal parts. Moreover, in each part, the features have front locations that are more critical in determining actions as these features are more frequently specified, e.g. bit0 and bit5. As a comparison, patterns of QRC cannot clearly show such insights.

transparent. As a comparison, PDRC and K1 show the issue of using over-general rules replacing more specific vital rules (show in Fig. 4.10). In the visualization of PDRC, for action one, all vital rules that generalized four features are not included. In the visualization of K1, due to only the severe over-compaction, no correct patterns can be visualized. In the cases of FU1 and FU3 (shown in Fig. 4.11), a large number of over-specific rules are kept together with over-general rules. This evidence confirms the proposed assertion of CRA and its analogous algorithms being prone to use a combination of rules, which are problematic but complementary, to maintain their performance.

4.4.3 Result of Majority-On Problems (MAJ)

MAJs suffer from a severe over-general issue, which is caused by these problems' overlapping distribution of samples (shown in Fig. 4.1). Therefore, over-general rules frequently dominate the populations of MAJs. In

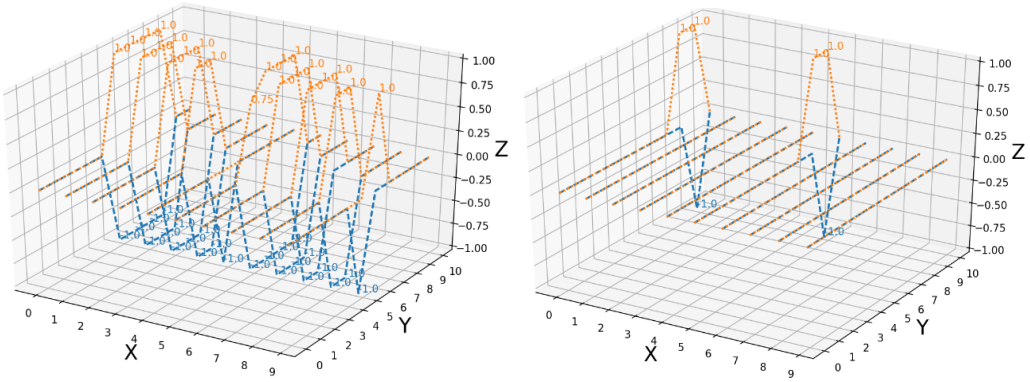


Figure 4.10: AFVMs for 10-bits CARs, following the order of left to right, are from compacted results of PDRC, and K1. The results of PDRC replace important rules with over-general rules. Thus a portion of patterns disappears. K1 is dominated by over-general rules, which show incorrect patterns that can misguide readers.

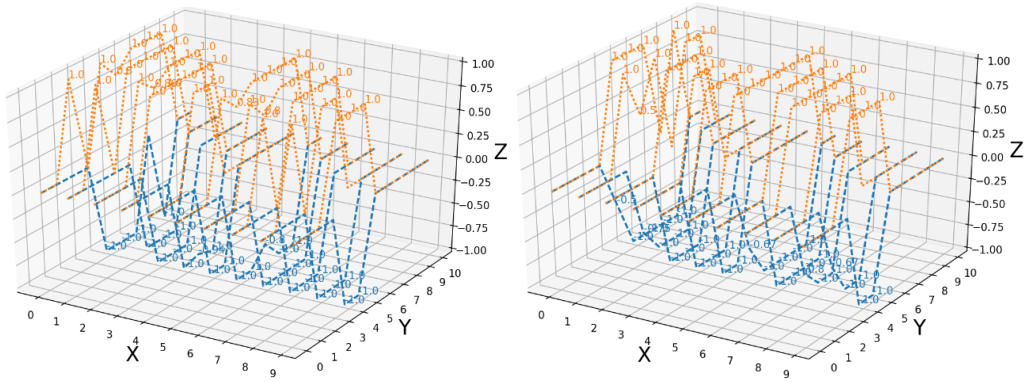


Figure 4.11: AFVMs for 10-bits CARs, following the order of left to right, are from compacted results of FU1, and FU3. Graphs here are based on a ruleset of over-general rules and over-specific rules. Thus, the actual patterns are indiscernible.

these cases, LCSs inappropriately assign a relatively high value of training parameters to some of the problematic rules. These rules with inappropriate valued parameters increase the difficulty of correctly executing the process of ascertaining whether a rule needs to be removed when compacting.

In general, MAJ's over-general issue becomes more serious when the number of involved features increases. The results in Table 4.5, Table 4.6, Table 4.7 and Table 4.8 support this hypothesis. Besides, results show the limitations of RCR and RCR2 on the 11-bits MAJ. In this case, common problematic rules with an inappropriate high value of training parameters can dwell in all the included populations. As a result, merging duplicated rules in different populations cannot highlight the most important rules through the value of the training parameters. This failure hampers RCR and RCR2 from searching for members of natural solutions. However, RCR3 completely addresses the 11-bit MAJ. This success indicates that LCSs' produced populations still contain members of natural solutions. Furthermore, re-evaluating rules' continuity can be a promising method to assist the compaction algorithms in adapting to populations that are dominated by problematic rules.

An interesting result in MAJ12 is that CRA reaches 98.84% after compaction, but RCR3 only achieves 81.3%. However, RCR3 produced a ruleset only with correct rules, which leads to uncovered instances causing all of the lost accuracy. As a comparison, CRA covers all the instances, but incorrectly predicts 1.16% of instances. Furthermore, the optimal number of rules for 12-bit MAJ is 1716. RCR3 detects around 1000 of these rules. However, CRA's produced ruleset only consists of around 300 rules, which indicates that the over-general rules have dominated the CRA compacted population. Furthermore, this is the only problem where FU's approaches consume less time than CRA. This is because LCSs cannot completely address this problem, and the highest training accuracy is 98%. Fu's approaches surpass CRA in running time, which conforms to the claim of

FU's approaches, where these approaches are designed to address insufficiently trained populations.

Continuity of Patterns in MAJs:

According to the visualizations in Fig. 4.12. The patterns of MAJs are different, depending upon whether the number of involved features is even or odd. In the case of being even, the patterns for two actions are separated. Meanwhile, for rules that support action one, features are more frequently specified compared with the rules of action zero. This phenomenon indicates that in the even situation, MAJs are imbalanced, and action zero possesses a larger size of niches compared with action one. In the case of being odd, patterns are completely duplicated, which implies these MAJs have a balanced distribution of niches. This observation is not immediate available in the problem definition.

Patterns Comparison Between Different Algorithms:

The visualization of RCR3's compacted result precisely identifies that in each rule for the 11-bits MAJs, shown in Fig. 4.13, six features need to be specified. Meanwhile, for all rules, the value of its specified feature is equal to the rule's supported action. These patterns indicate that the majority value of the involved features determines the output action, which follows the ground truth of MAJs. As a comparison, although the visualization of RCR is somehow readable, it does not offer much useful information. K1's visualization is based on a ruleset with a severe overgeneral issue, which may misguide readers. Patterns from QRC's, CRA's, and FU1's results are completely indiscernible.

Table 4.5: The running time, the number of rules and the accuracy in a compacted population for MAJ. Values in parentheses are the number of optimal rules. s, m, and, h are the symbols for the second, minute, and hour, respectively.

Problem	MAJ6(35)	MAJ8(126)	MAJ10(462)	MAJ12(1716)
CRA	<1s	12s	11.2m	39.9h
	26±7	82±15	136±18	312±55
	100%	100%	99.99%	98.48%
FU1	2.7s	4.1m	1.9h	16.3h
	33±2	125±5	348±22	491±19
	99.84%	99.73%	98.11%	95.64%
FU3	2.83s	4.16m	2.7h	18.39h
	29±3	104±13	175±15	278±25
	99.84%	99.73%	98.25%	97.19%
CRA2	<1s	<1s	3.3s	20s
	33±2	128±5	346±20	196±17
	99.99%	99.98%	97.67%	95.18%
K1	<1s	<1s	6.2s	37s
	13±1	20±3	28±5	34±3
	88.39%	83.95%	83.84%	83.98%

Table 4.6: The running time, the number of rules and the accuracy in a compacted population for MAJ. Values in parentheses are the number of optimal rules. s, m, and, h are the symbols for the second, minute, and hour, respectively.

Problem	MAJ6(35)	MAJ8(126)	MAJ10(462)	MAJ12(1716)
QRC	<1s	<1s	1.6s	6.3s
	35±0	129±3	485±5	285±35
	100%	100%	99.88%	93.05%
PDRC	<1s	<1s	3.3s	19.3s
	29±5	87±10	229±18	227±25
	98.39%	98.1%	97.19%	94.5%
RCR	<1s	6.1s	56s	3.2m
	35±0	126±0	462±0	649±93
	100%	100%	100%	90.16%
RCR2	<1s	7.2s	57s	3.8m
	35±0	126±0	462±0	700±50
	100%	100%	100%	90.26%
RCR3	<1s	7.3s	56s	17.51m
	35±0	126±0	462±0	1039±51
	100%	100%	100%	81.03%

Table 4.7: The running time, the number of rules and the accuracy in a compacted population for MAJ. Values in parentheses are the number of optimal rules. s, m, and, h are the symbols for the second, minute, and hour, respectively.

Problem	MAJ7(70)	MAJ9(252)	MAJ11(942)
CRA	2.1s	1.15m	3.78h
	53±17	109±11	307±62
	100%	100%	99.99%
FU1	23s	28.3m	6.54h
	70±2	245±3	376±12
	100%	99.56%	97.35%
FU3	22.3s	28.6m	6.52h
	66±4	162±11	208±14
	100%	99.56%	98.39%
CRA2	<1s	1.2s	8s
	71±2	252±3	347±13
	100%	99.76%	96.38%
K1	<1s	2.3s	17s
	17±3	22±2	24±6
	89.55%	87.33%	84.88%

Table 4.8: The running time, the number of rules and the accuracy in a compacted population for MAJ. Values in parentheses are the number of optimal rules. s, m, and, h are the symbols for the second, minute, and hour, respectively.

Problem	MAJ7(70)	MAJ9(252)	MAJ11(942)
QRC	<1s	<1s	4s
	72±1	254±2	620±29
	100%	100%	94.85%
PDRC	<1s	1.2s	8s
	59±11	124±6	270±17
	89.71%	96.7%	95.95%
RCR	2.1s	6.1s	1.4m
	70±0	252±0	874±23
	100%	100%	99.37%
RCR2	2.1s	10s	1.78m
	70±0	252±0	879±30
	100%	100%	99.17%
RCR3	2.2s	9.1s	1.3m
	70±0	252±0	924±0
	100%	100%	100%

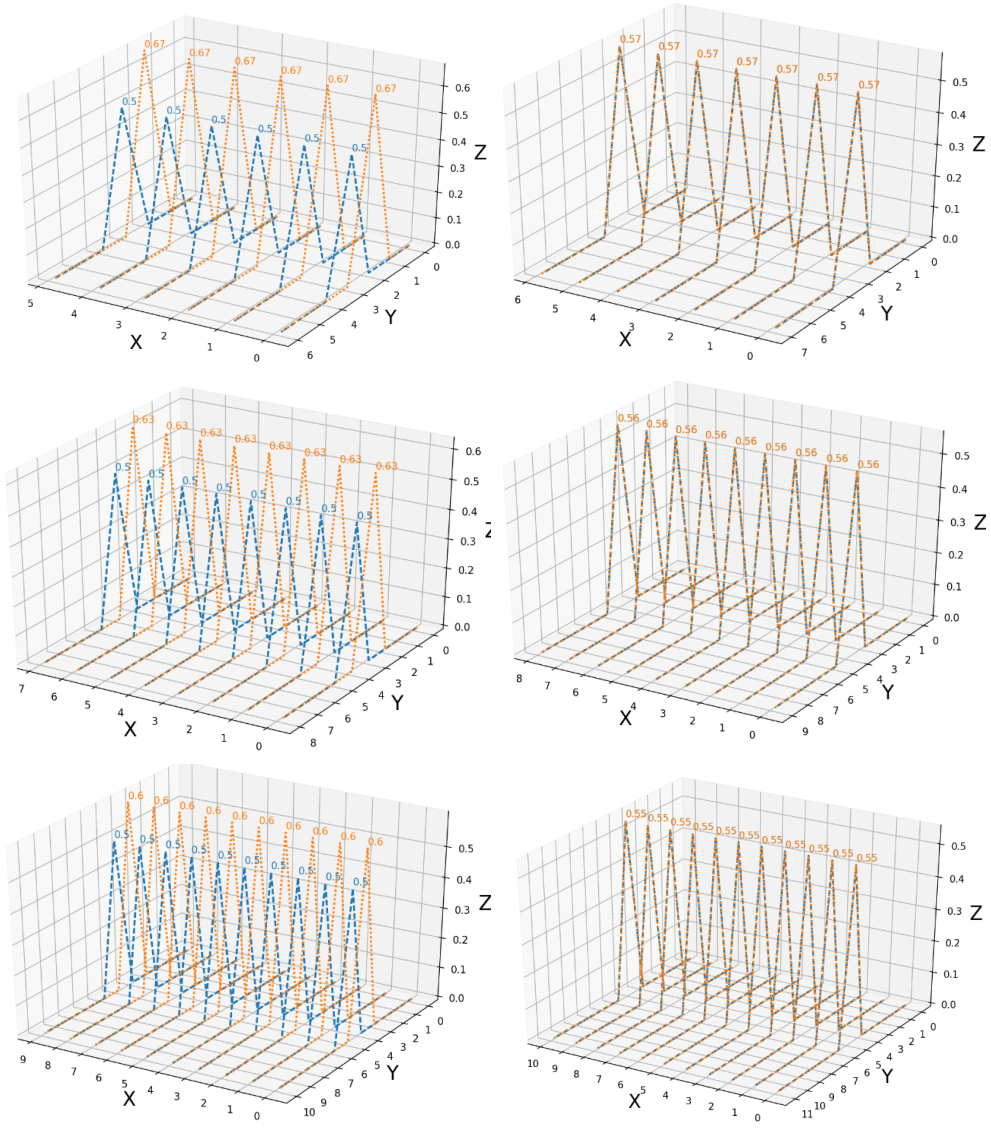


Figure 4.12: AFIMs for six MAJ problems based on RCR3 compacted results, maps following orders of left to right and then from top to bottom are patterns of 6-bits, 7-bits, 8-bits, 9-bits, 10-bits, and 11-bits MAJ. Blue represents the patterns of action zero, while orange marks the action one. Patterns show MAJs are balanced when the number of features is odd. In even situations, MAJs are imbalanced with the majority class zero. Note this discovered pattern can not be identified from the problem definition.

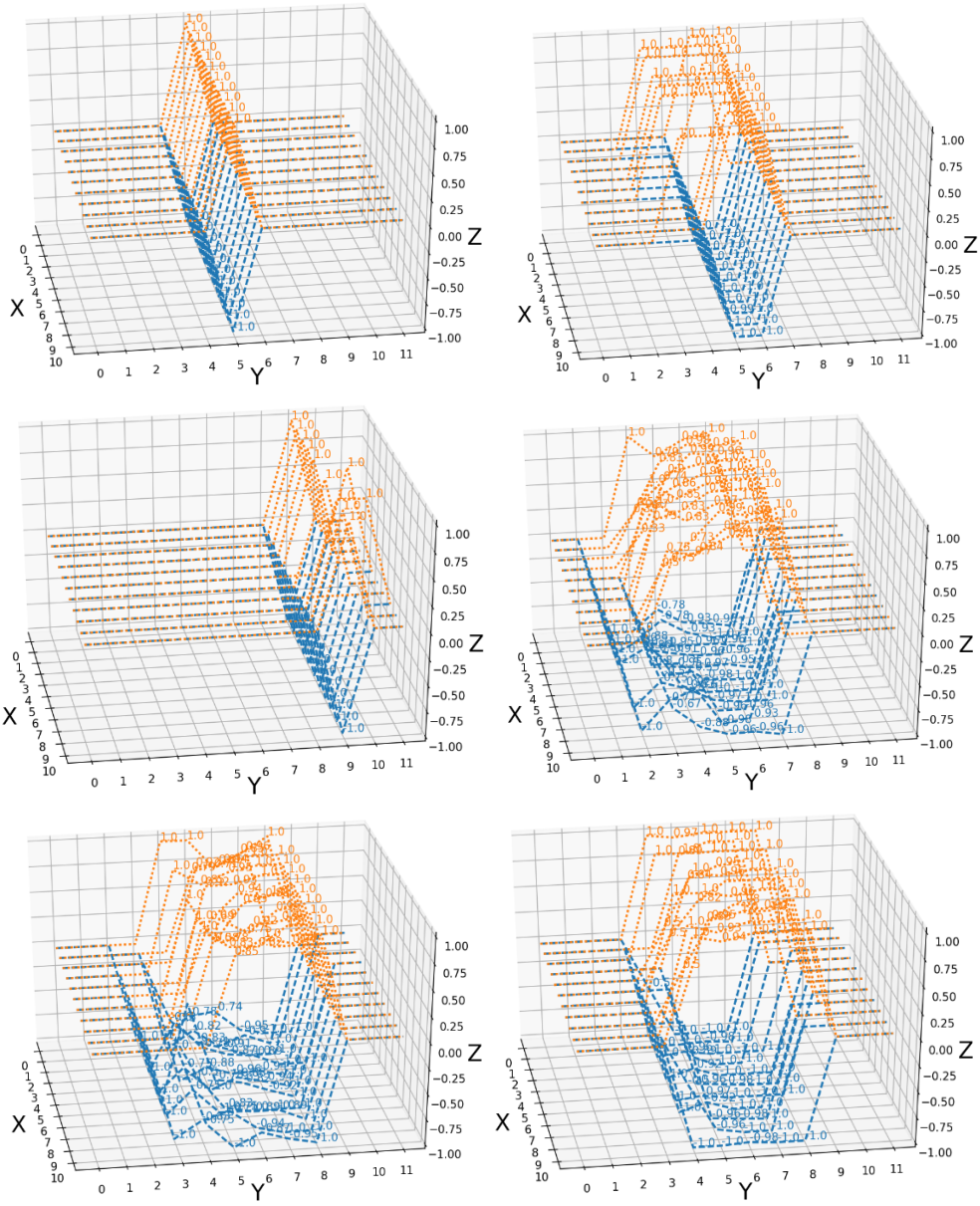


Figure 4.13: Six AFIMs for 11-bits MAJ problems, maps following orders of left to right and then from top to bottom are patterns of RCR3, RCR, K1, QRC, CRA, and FU1. Patterns of RCR3 demonstrate that in MAJ11, the majority of values of features determine the action, which is consistent with the ground truth. RCR and K1 show the patterns of over-general rules, which may misguide readers. The patterns of QRC, CRA, and FU1 consist of many problematic rules. Hence, these patterns are indiscernible in terms of identifying the ground truth.

4.5 Conclusions of Rule Compaction Algorithms

The proposed work has provided further evidence that LCSs is a suitable technique for producing natural, interpretable models in data mining tasks, which contribute to interpretable AI, an important future research direction. Noiseless artificial Boolean domains contain discernible patterns, which enable the effectiveness of interpretable techniques to be ascertained, are selected as benchmarks.

The most popular compaction algorithms in the last two decades have been reviewed to investigate how compaction algorithms can produce an effective population that the contained underlying patterns are visualizable in a transparent manner. According to the results of the conducted experiments, two issues in the LCSs field can be identified. Firstly, the patterns contained in the compacted population of high performance can be undiscernible when visualized. Secondly, due to time constraints, the strategy of repeatedly reviewing a training set will reduce the applicability of compaction algorithms in addressing large-scale problems. Experimental results suggest a solution for addressing these two issues. That is, when compacting, instead of preserving prediction performance, it is preferable to preserve the rules that have correct encodings, so that the underlying patterns of LCSs produced model can be kept after optimization (compaction). Successful compaction is demonstrated, especially on the 11-bits Majority-On problem that RCR3 correctly collects all 924 different interacting rules for constructing a fully representative model. Furthermore, for the first time, RCR and RCR2 successfully compact the large-scale 70-bits Multiplexer problem without reducing the training accuracy.

The results support the assumption that the natural solution and the [O] can simultaneously exist, as in any MUX that is less than 20-bits, RCR captures [O]s together with RCR2 detected natural solutions. In MUXs, both [O]s and natural solutions contain human-discernable patterns, which support the assertion that LCSs are able to produce natural interpretable

solutions for explored domains. Besides, results indicate that natural solutions are the same for the same problem ranther than [O]s can be multiple. Regarding the LCSs produced models, if the models have been trained properly, natural solutions can always be captured. However, in domains that have an overlapping distribution, LCSs models can not be used to produce [O] as the subsumption mechanism continuously removes the member rules in [O]s, which have specific encodings that could be considered as over-specific by the subsumption. Lastly, results identified that 12-bits Majority-On problem is the limitation of all the tested compaction algorithms, as none of these algorithms can capture an interpretable rule-set for this problem.

Chapter 5

Rule Compaction in Real-Value Domains

An interesting question is that whether the proposed natural solution hypothesis and visualization pipeline can adapt to the clean real-value attributes problems. To investigate this question, this chapter develops a new version of RCR, i.e. RCR-Real, that can compact a real-value attributes problem orientated model to its natural solution format. Furthermore, the visualization technique, i.e. the FIM has been extended to enable the RCR-Real compacted models' patterns to be visible. More importantly, to assist LCS to produce natural solutions for real-value domains, which have an overlapping distribution, a novel LCS termed Hierarchical Learning Classifier System (HLCS) is proposed. The HLCS's novelty is that it considers rule compaction as a part of the exploration stage, rather than invoking compaction after exploration has finished. The conducted experiments successfully find natural solutions for all the tested clean UCI datasets. Furthermore, the produced natural solutions contain visible patterns that reflect the ground truth of the explored UCI datasets.

5.1 Introduction

LCS's employed representation format has an inherent rich characteristic. Thus, LCSS can represent the underlying patterns in multiple (polymorphic) ways [12] [13] [14] [81] [83]. This may obscure the most informative patterns. A novel rule reduction algorithm that purposes to identify such patterns is proposed based on ensembles of multiple trained LCSs models to reduce the local diversity and global polymorphism [1]. Furthermore, a hierarchical learning architecture based on the newly proposed compaction algorithm is introduced to LCS, with the intention to assist LCSs in adapting to real-value domains with an overlapping distribution.

The objective is to develop a technique that can interrogate the hidden patterns in LCSs' trained populations and improve the LCSs' representation capacity in overlapping real-value domains. This will enable visualization of the importance of features in data groups (niches) that can contain heterogeneous patterns, i.e. even if different patterns result in the same class, the importance of features can be found.

Previously, the Razor Cluster Razor (RCR1, RCR2, and RCR3) described in Chapter 4 aims to search for the optimal global solutions in Boolean domains based on analyzing a set of LCSs' trained populations. These Boolean orientated RCRs compact rulesets for any such domain that LCSs can completely solve, i.e. reach 100% accuracy in the test set. This work aims to determine whether the convergent evolution phenomenon helps in real-value domains, which are inherently unlikely to be completely solvable by any classification technique due to their imprecise decision boundaries, e.g. caused by signal noise.

RCR-real is proposed here, which is tailored to the most common representation used in real-value domains, i.e. upper and lower bounds rather than the ternary alphabet of Boolean domains. An LCSs-based hierarchy

[1] Local diversity and global polymorphism frequently result in an uncertain visualization result, which might not precisely present important patterns.

learning architecture is also proposed to extract the common ruleset for LCSs' non-completely solvable domains.

The objectives are to firstly create a method to extract a common ruleset for an ensemble of LCSs that have separately explored the domain. Second, it also aims to estimate each involved attribute's distinguishability. Finally, to visualize a problem's detected underlying heterogeneous patterns (see Section 3). Ten different UCI datasets are used as benchmarks to visualize the learned knowledge. Note, this work does not seek to improve LCSs' prediction capability, instead the representation of underlying capability. Thus the system's performance is estimated by the training result only, where overfitting is to be detected through the visualization. The proposed work is based on XCS (a reinforcement-learning LCS) as it forms a complete map of inputs to outputs. The main alternative, UCS (a supervised-learning LCS), forms a best-action map, which lacks the consistent incorrect information of the observed domain needed to calculate attribute importance.

5.2 Compaction Algorithm for Real-Valued Domains

Since XCSs employ different representations for Boolean and real-value domains, the implementation between Boolean orientated RCRs and RCR-Real is different. The only commonality between them is the basic philosophy, which is that "entities are not to be multiplied without necessary", i.e. Occam's razor. This theory inspires an alternative way to handle the rule compaction problem, where it is hypothesized that rules can be compacted under a macro view rather than a micro view. The rule reduction methods prior to RCR focused on interrogating individual rule's performance in a population, despite how individuals interact with each other. Moreover, sampling all rules from a single population results in a substan-

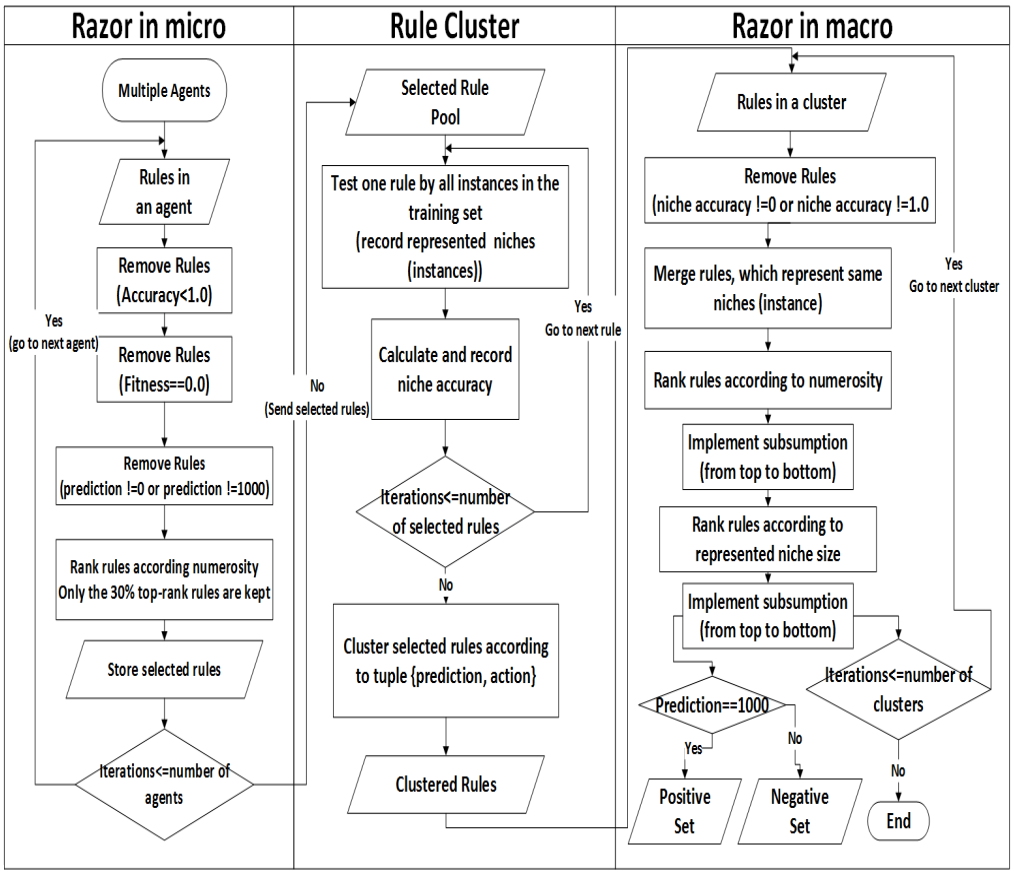


Figure 5.1: The following graph for the RCR-Real process. RCR-Real is composed of Razor in Micro that aims to reduce rules' diversity in individual models, Rule Clusters that intends to unite rules from different models into a single model, and Razor in Macro aims to reduce the rules' polymorphism in the merged model.

tial risk of being misguided by the variance in the ways the correct rules population can be formulated (termed polymorphism). This may explain why although LCSs are competent to generate optimal rules for the addressed problem, the previous rule reduction methods find it difficult to compact a population to its optimal state, i.e. just containing one set of only optimal rules.

The RCR-Real process is shown in Fig. 5.1. The three-phases of RCR-real are *razors in micro*, *rules cluster*, and *razors in macro*, which are processed in sequence. Razors seek to simplify as much as possible, but no simpler. Razors in micro aims to remove the poor performance rules in each population. Subsequently, the remaining rules in all populations will be gathered together to be clustered according to the number of attributes kept in each individual, where this process is termed as *rules cluster*. Finally, for the sake of reducing rules' polymorphism in the clustered set, the razor in macro is introduced.

5.2.1 Razor in Micro

In XCSs, the training population is larger than the optimal number of final rules to guarantee that XCS is capable of exploring multiple competing hypotheses simultaneously. Thus, a high diversity of rules is introduced to the actual final population. As a result, the quality of the evolved rules is varied. In certain domains, the majority of rules are redundant for representing the task's patterns. Moreover, a few rules may even make a negative contribution. Hence, this pre-processing phrase is needed to remove the inferior individuals in a population.

Generally, three factors can describe an inferior individual rule: inadequate training, inconsistent prediction behavior, and no contribution to the represent action of the task. Any such rule in a population should be excluded to reduce unnecessary diversity. The Razor in micro includes four dependent steps that are implemented in sequence (shown in Algorithm

Algorithm 18: Razor in Micro (RMI) in RCR-Real

Input: all rules in a trained population P;
the percentage of kept rules $Threshold_P$
Output: a compacted population P

```
foreach rule ∈ P do
    if rule.accuracy<1.0 then
        Remove rule from P
    end
    if rule.fitness==0.0 then
        Remove rule from P
    end
    if not rule.fitness==Maximum or Minimum then
        Remove rule from P
    end
end
Rank rule ∈ P by rule.numerosity (ascending order);
while P.Size > Threshold_P * P.Size do
    Remove the first rule in P
end
```

18). The first step intends to remove rules that may produce an incorrect prediction for a portion of the covered instances. As a sufficiently trained rule's *accuracy* can stay at the maximum value (100%), only if this rule produces consistent performance regarding prediction. Thus, this step only preserves the rules whose *accuracy* is equal to the maximum value.

The next two steps consider removing rules that have no contribution to retaining a population's prediction performance. The second step evaluates rules based on rules' *fitness*. As rules' *fitness* indicates a rule's potential performance in representing a problem. Thus, rules that has a minimal value (0) in *fitness* indicate these rules are irrelevant to a population's per-

Algorithm 19: Clustering

Input: a training set T;
a set of populations after RMI P_Set;
Output: a clustered population C_P

```
1  Initial an empty clustered population C_P according tuple {action,
     prediction} (an N actions domain possesses 2N Clusters);
2  foreach Population (P) ∈ P_Set do
3      foreach rule (r) ∈ P do
4          foreach instance ∈ T do
5              if r cover instance then
6                  if advocate the same action then
7                      Inscert instance.ID to r.record
8                  end
9                  r.number++
10             end
11         end
12         r.nicheaccuracy = r.number / size(r.record)
13     end
14     cluster r to C_P according {action, prediction}
15 end
```

formance. Hence, such rules will be removed in this step. The third step assesses rules according to the rule's *prediction*, and only preserve rules where their *prediction* reach either the maximum value or the minimum value discovered so far. Note this step can only be activated for the cases having no noise.

The last step aims to remove rules that are trained insufficiently, and to delete rules that are redundant potentially. This purpose is achieved by ranking all the remaining rules according to their *numerosity* and then removing the last 70% [2] bottom-ranked individual rules. This strategy is supported by empirical evidence from the previous experiments in Boolean domains, where XCSs tend to offer the most important individuals a higher numerosity value compared with the redundant ones. By implementing the Razor in micro, the majority of the redundant rules and irrelevant rules will be excluded from a population. Thus, a candidate population's rule diversity is reduced.

5.2.2 Rule Cluster

Commonly, XCSs employ the upper and lower bounds representation in real-value domains [[$attribute_0$ *high boundary*, $attribute_0$ *low boundary*], [$attribute_1$ *high boundary*, $attribute_1$ *low boundary*]...], which offer rules both precision and generality. However, due to the rich representation style, a rule's explicitly represented niches cannot be assessed directly in the condition part, which hampers efforts to detect the target's unique morphism.

In the implementation of the rule cluster (shown in Algorithm 19), two stages are included. In stage one, each selected rule reviews the training set to record the matched niches (instances) and the correctly represented niches. Hence, RCR can estimate each rule's niche accuracy by dividing the size of the correct niches by the size of the matched niches. The niche accuracy will be used in Razor in Macro to remove the remaining incon-

[2] This setting is based on experience, this setting value can be altered for different problems

Algorithm 20: Error Detector in RCR-Real

Input: a clustered population P;
Output: a compacted population P

```
1  foreach rule ∈ P.correct_set do
2      if not rule.niche_accuracy==1.0 then
3          Remove rule from P.correct_set
4      end
5  end
6  foreach rule ∈ P.incorrect_set do
7      if not rule.niche_accuracy==0.0 then
8          Remove rule from P.correct_set
9      end
10 end
```

sistent rules.

In the next stage, all the pre-processed rules will be clustered according to the prediction, action tuple, since after Razor in Micro, all the remaining rules' prediction can only be either the maximum value or the minimum value. Thus, the number of clusters is equal to double the number of actions. Note, if a rule's prediction is associated with the maximum value, then it is a correct rule. Otherwise, the rule is an incorrect individual. As XCSs naturally seek to form complete maps, which consist of both correct rules and incorrect rules, the inherent patterns of each can be different, especially in multi-action domains. Thus, without the clustering process, the effort of identifying the underlying patterns from a population cannot make progress, since important patterns are mixed together.

5.2.3 Razor in Macro

XCSs aim to form a complete map to represent the explored domains. However, due to the XCS' rich representation, extremely diverse rules are

Algorithm 21: Rule Merge in RCR-Real

Input: a clustered population P;
Output: a compacted population P_C

```
1  rank rules according to numerosity (descending order);
2  foreach rule1 ∈ P do
3      Add rule1 to P_C;
4      remove rule1 from P;
5      foreach rule2 ∈ P do
6          if rule1.record==rule2.record then
7              rule1.numerosity+=rule2.numerosity;
8              remove rule2 from P
9          end
10     end
11 end
```

generated, which obscures the discovered patterns. Razor in macro is designed to compact the XCS's trained populations to their single common state by reducing the populations' polymorphism. Three processes are involved, including error detection, rules merging, and two-level subsumption.

The error detection aims to eliminate all the remaining irrelevant individual rules by interrogating each rules' niche accuracy. For any correct rule, its niche accuracy reaches the maximum, and for any incorrect rule, its niche accuracy ought to decrease to the minimum. Otherwise, the rule must be irrelevant to the optimum population (shown in Algorithm 20).

The rule merging method focuses on merging rules within the same niche. During the merging process, attributes will be merged one by one, and only the common overlap interval will remain. Therefore, the final merged rule will approximate the target, which successfully removes any unsupported attribute interval. Merged rules' numerosity will be summed

Algorithm 22: First stage subsumption in RCR-Real

Input: a clustered population P;
Output: a compacted population P_C

```
1  rank rules according to numerosity (descending order);
2  foreach rule1 ∈ P do
3      Add rule1 to P_C;
4      remove rule1 from P;
5      foreach rule2 ∈ P do
6          if rule2.record∈rule1.record then
7              remove rule2 from P
8          end
9      end
10 end
```

(show in Algorithm 21).

In XCSs, subsumption focuses on addressing the redundant rules problem. In RCR-Real, a novel two-level subsumption is implemented. The first level (shown in Algorithm 22) ranks all the remaining rules according to their numerosity and invokes subsumption from top to bottom. Each rule will be compared with all their peer rules that have a lower rank. If a lower-rank rule can be subsumed, it will be deleted. In the second level (shown in Algorithm 23), all the rules will be ranked according to their represented niche size, and then subsumption is reactivated to ensure that the output set is as general as possible. Eventually, for the sake of visualization in the next step, all the selected individual rules will be placed into a positive set or a negative set, depending on whether their prediction is maximum or minimum.

Algorithm 23: Second stage subsumption in RCR-Real

Input: a clustered population P;
Output: a compacted population P

```
rank rules according to numerosity (ascending order);
foreach rule ∈ P do
    remove rule from P;
    if rule.record∉P then
        add rule to P
    end
end
```

5.3 Visualization in real-value domains

Attribute Importance Map visualizes the importance of each considered attribute to identify the importance of an attribute in determining the output action. In the Boolean domain, an attribute's importance is assessed by its generality level. In real-value domains, the employed representation format (upper and lower boundary representation) does not reflect the attributes' generality level. Here, the attribute importance is estimated by analyzing each attribute's independent distinguishability by comparing each attribute represented non-overlap space for each action between the RCR produced positive set and negative set.

$$AInf_{nm} = \frac{\sum_{i=m0}^{i=mi} \frac{(\sum_{j=m0}^{j=mj}(PSize_{nmi}+NSize_{nmj}-2*PNSize_{nmij}))}{N_{mlength}}}{\sum_{i=m0}^{i=mi} PSize_{nmi} + \sum_{j=m0}^{j=mj} NSize_{nmj}} \tag{5.1}$$

$$AImp_{km} = \frac{AInf_{km}}{\sum_{att=0}^{att=n} AInf_{attm}} \tag{5.2}$$

In real-value domains, an attribute's importance can be assessed by Equation 5.1 and Equation 5.2. Assume a problem, which has an action set A, A=[Act_0, Act_1...Act_m] and an attributes set N, N=[Att_0, Att_1,...Att_M],

and for an action *a* ($a \in A$), it associates a support ruleset I_m, I_m= [P_{m0}, P_{m1} ... P_{mi}] and an opposite ruleset [3] J_m ,J_m= [N_{m0}, N_{m1}...N_{mj}], in *i* ($i \in I$) positive ruleset or *j* ($j \in J$) negative ruleset, for any related attribute *n*,($n \in N$), its represented range is defined as $PSize_{nmi}$ and $NSize_{nmj}$. The overlap between *i* positive rule and *j* negative rule for attribute *n* is defined as $PNsize_{nmij}$, $N_{mlength}$ response for the number of negative rules for action *a* ($a \in A$), then the attribute *n*'s influence for action *a* $AInf_{na}$ can be estimated by Equation 5.1. Afterward, the attribute *k* ($k \in N$) importance to action *a* $AImp_{nm}$ can be calculated by normalizing the attribute influence as shown in Equation 5.2.

5.3.1 Hierarchical Learning Classifier System

The Hierarchical Learning Classifier System (HLCS) is proposed (see Fig. 5.2) to correctly and completely represent an identified dataset with a defined representation format. Furthermore, when a ruleset has represented a dataset, HLCS attempts to use this ruleset to produce the explored problem's visualizations that can present the underlying patterns through human discernable graphs. HLCS is inspired by homologues of ensemble learning, i.e. bagging and boosting. Besides, the ideas from population-based incremental learning (PBIL) also influence the HLCS, where learning is "adapted to new data without forgetting the existing knowledge".

HLCS's learning process is composed of a few layers of automatically splitting a dataset and incrementally exploring a domain, which resemble boosting. The number of layers depends on the explored dataset, where layers are automatically increased until the produced models represent the target dataset, completely and correctly. A layer includes a training set, several independent XCSs, a rule compaction mechanism based on RCR-real, and FIMs for translating the underlying patterns into discern-

[3] An opposite ruleset includes all rules, where their assigned accuracy is 0%, which indicates that these rules oppose their assigned class label.

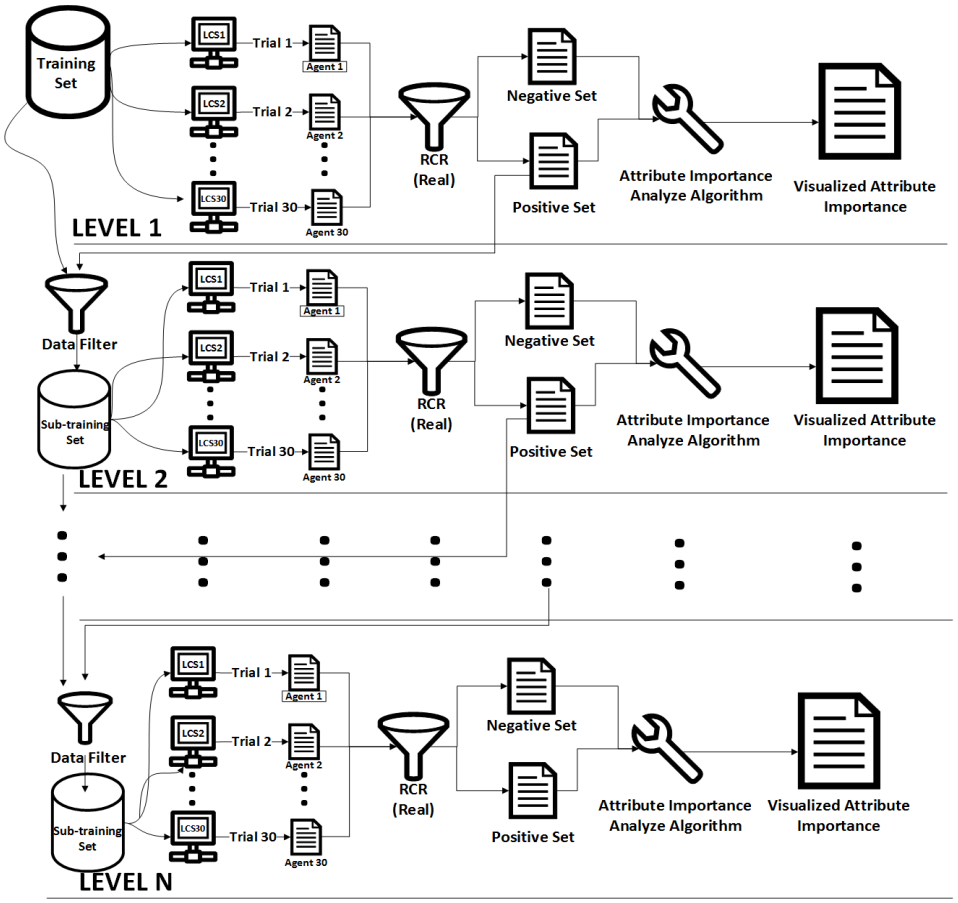


Figure 5.2: The architecture of the Hierarchical Learning Classifier System (HLCS). The number of layers will automatically be increased until the HLCS completely represents the target dataset. Each layer consists of five modules, a data filter based on the RCR compacted positive set, a bagging based training process, RCR, attribute importance analysis, and attribute importance visualization.

able graphs. Furthermore, after the first layer, any additional layer considers a subset of instances from the original dataset as the training set. This subset is composed of all the unrepresented instances that cannot be matched by the RCR-real compacted models' positive set in the former layers. Thus, former layers' compacted models perform the role of data filters to construct the training set for the next layers.

HLCSs utilize a bagging style structure for applying the exploring process, where multiple XCSs explore the same training set, synchronously. This structure avoids the errors caused by the XCS's stochastic characteristic in a single trained model. This is because important rules are always common, but erroneous rules can be varied in the populations. Thus, it is practicable for RCR-real to select the most crucial rules from multiple XCSs' models, as important rules are expected to appear repeatedly in different models, highlighting these rules' importance.

Furthermore, HLCS benefits by applying parallel computing, although multiple XCSs are employed, the computation in HLCS only introduces around 20% additional execution time than standard XCS. At the end of each layer, HLCS produces a visualization result based on an RCR-real compacted model. This visualization is achieved by implementing real-value domains orientated FIMs, which introducing in Section 3.3.

5.4 Experiments and Results of Real-Value Domains

Six basic datasets from continuous real-value domains in UCI are selected to interrogate the HLCS correctness, including Iris, Sonar, Wine, Australian, German, Wisconsin Breast Cancer Diagnostic (WBCD). Four complex domains for investigating the HLCS's limitations are also selected, including high dimensionality, (i.e. Lung Cancer), artificial problem, (i.e. Hill and Valley), multiple classes with a low number of instances, (i.e. Zoo), and

Table 5.1: RCR, XCS, HLCS results regarding the number of rules and training accuracy, where [low, high] values of average over 30 runs are presented. "+" is a symbol for distinguishing different layers, where instance numbers or rule numbers in different layers are split by this symbol. "acc" represents for accuracy, size represents the number of instances in the training set.

Domian	Iris	Wine	Australian	Sonar
size	150	178	680+10	208
XCS size	[1557,1675]	[2736,2813]	[2611,2714]	[2924,2958]
RCR size	9	13	115+9	208
XCS acc	[99.5%,100%]	[100%,100%]	[93.9%,97.2%]	[99%,100%]
RCR acc	100%	100%	98.56%	100%
HLCS acc	100%	100%	100%	100%

natural domains, (i.e. Ionosphere).

Among all the ten explored datasets, HLCS achieves 100% training accuracy, whereas standard XCS failed in four of them (see Table 5.1 and Table 5.2). For all the domains, more than 90% of the introduced rules in XCSs are removed after RCR-real has been implemented. Note, results in Table 5.1 and Table 5.2 are not testing accuracy for prediction.

RCR-Real produces a stable performance model consisting of rules that have different represented niches. Importantly, the rulesets' performance can be interrogated not only by the accuracy but also by their generality and overfitting. For example, the results for Lung cancer and Sonar problems indicate that these domains need to be represented by the most specific rulesets (Lung cancer is a high-dimension problem and Sonar is a difficult domain). Unexpectedly, a very general ruleset is produced for the Zoo dataset, which contains seven classes. Traditionally, LCSs are not systems that have good performance for domains with a high number of classes but low number of instances.

Table 5.2: RCR, XCS, HLCS results regarding the number of rules and training accuracy, where [low, high] values of average over 30 runs are presented. The "+" symbol is used for distinguishing different layers, where instance numbers or rule numbers in different layers are split by this symbol. "acc" represents for accuracy, the size represents the number of instances in the training set. Note as pattern extraction, rather than prediction, is considered all data is used for training. Overfitting can be identified through the visualization, i.e., many very specific rules.

Domian	**Zoo**	**WBCD**	**Ionosphere**
size	101	683	351
XCS size	[2756,2834]	[2576,2653]	[2764,2848]
RCR size	12	37	154
XCS acc	[100%,100%]	[99.6%,100%]	[97.5%,100%]
RCR acc	100%	100%	100%
HLCS acc	100%	100%	100%
Domain	**German**	**Lung Cancer**	**Hill Valley**
size	940+60	18+9	559+47
XCS size	[2642,2745]	[2944,2963]	[2981,2994]
RCR size	242+14	18+9	267+31
XCS acc	[78.5%,86.8%]	[63.1%,70.3%]	[79.7%,86.8%]
RCR acc	94%	66.66%	100%
HLCS acc	100%	100%	100%

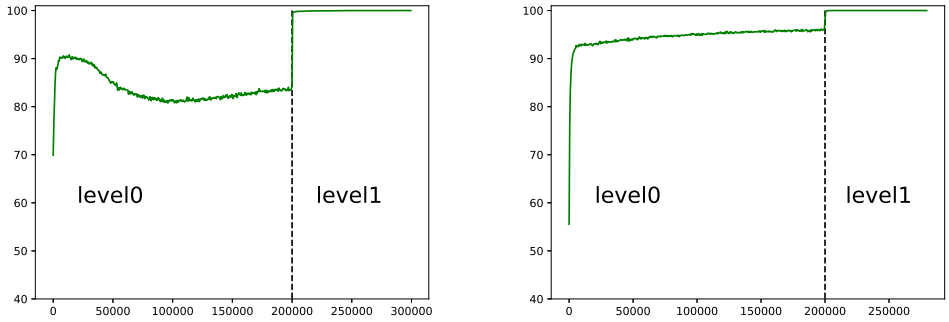


Figure 5.3: HLCS' Training performance for the German and Australia problems. The graph shows how HLCS produces a fully representative model with two layers.

The training graph (Figure 5.3) shows that there is a limitation on producing a fully representative model by XCSs, without the edition of the method to add hierarchies. Layer 0 is essentially a standard XCS's training performance, where a clear steady-state error range exists, regardless of how many additional training iterations are introduced. This can be caused by overlapping niches suited to polymorphic rulesets, where the XCS identifies multiple main patterns. Hierarchically removing the main patterns and associated data enables the XCS to discover the next most important patterns and so forth. Each of such patterns can contain epistatic relationships between features, unlike decision trees where each hierarchy layer focusses on an individual feature. Ultimately, the last hierarchy could consist of the hard to classify outlier data points where no generality is possible, giving rise to specific rules. These can be identified through equal importance being given to each attribute (feature).

After the HLCS splits the problem domain into successive comprehensive parts, the completely represented solutions are obtained. Thus, it is practicable for HLCSs to completely represent domains. The training map for the German domain exhibits how XCS's population is dominated by over-general rules in the first stage of training (layer 0) as when they form

performance drops. This does not occur for the Australia dataset, which suggests specific rules are formed.

Table 5.3: HLCS and traditional feature rankers' results for the Iris problem. Iris problem has three class labels, i.e. Setosa, Versicolour, and Virginica.

Feature Rank	Attribute Importance
HLCS Setosa	Petal Length, Petal Width: 50% Sepal length, Sepal width: 0%
HLCS Versicolour	Petal length: 32.8%, Petal Width:31% Sepal width: 18.3%, Sepal length:17.9%
HLCS Virginica	Petal Width:29.4%, Petal length: 28.9% Sepal length:21.3%, Sepal width: 20.4%
PCA	Petal length, petal width
Relief	Petal width, petal length, sepal width, sepal length
CFS	Petal length, petal width

5.4.1 Feature Ranking

When FIM is applied to RCR-real compacted models, the calculated attribute importance can be considered as an assessment for feature ranking. Thus, HLCS can be employed as a feature ranking technique, which reaches an action-based fine-grained level and can reflect how important an attribute is in determining output actions. The Iris dataset's ranked results (see in Table 5.3) display the order of attributes' importance identified by HLCS. Besides, in order to compare HLCS with traditional attribute ranking techniques, i.e. Principal Components Analysis (PCA), Relief, and Consecution based Feature Selection (CFS), Table 5.3 is presented.

HLCSs produced feature (attribute) ranking is action-based, as in multiple class domain, different classes may have a different attribute impor-

tance ranking. Thus, HLCSs produce a feature ranking for each considered action. In the Iris problem, three classes are involved, accordingly, HLCSs output three independent feature rankings (shown in Table 5.3 ,which are HLCS Setosa, HLCS Versicolour, HLCS Virginica).

In all classes, HLCS identifies that petal length and petal width are the most important attributes, which is supported by PCA and CFS. Moreover, HLCS also identifies that petal length and petal width having similar attribute importance. Meanwhile, sepal width and sepal length having similar attribute importance. This discovery can also be found by using Relief. The Iris domain indicates that although HLCS is based on investigating the LCS produced rules, the novel proposed system obtains common attributes' importance with traditional statistics-based algorithms but displays the achievement in a much clearer manner.

5.4.2 Learning from Feature Ranking

Once the output is known, the solution can be interrogated, such that overfitting can be examined, i.e. the discovered patterns can be observed. It appears that LCSs learn some strange patterns, e.g. in the Zoo dataset (see Table 5.4: *feather* attribute is not important to classify birds, but *milk* is. LCSs consider birds are animals that can be distinguished by [not] producing milk, laying eggs, and [not] having hair. *Mammals* need to seriously consider their fins, where this is an excellent example of a heterogeneous niche. To distinguish all the animal classes, *milk* is really important. Amphibians contain fewer species, which does not distract HLCS as it identifies that *backbone type* and the *number of tails* are important to categories these species. Also, [not] *feathers* has the same importance level as *fin* for identifying fishes.

If LCSs' detected underlying patterns are directly compared with common human knowledge, there is obviously a huge gap between human's and LCS's comprehension of the world. However, if the training dataset

Table 5.4: HLCS highlighting mammal and fish classes and traditional feature rank results for Zoo problem.

Feature Rank	Attribute Importance
HLCS Mammal	Milk:17.4%, fins:17.1%, hair:8.8%, tail:8.5% airborne:5.8%, Eggs, toothed, backbone:4.4% breathes, venomous, domestic:4.4%, Legs:4.3% catesize:4.3%, aquatic:4.2%, predictor: 3.4%, feathers:0%
HLCS Fish	Hair, feathers, eggs, milk, airborne: 7.3% aquatic, predator, toothed, backbone: 7.3% fins, legs, tail, domestic: 7.3%, Venomous, catesize: 2.4% breathes:0%
PCA	Feathers, eggs, milk, airborne, aquatic, predator, toothed, backbone, breathes,venomous
Relief	Venomous, breathes, tail, milk, backbone, domestic, predator, eggs, fins, airborne, feathers, legs, aquatic, toothed, catesize
CFS	Airborne, breathes, venomous, fins, domestic, catesize

of learned knowledge is further investigated, interesting patterns emerge. Firstly, Zoo is a typically unbalanced dataset, where mammals occupy around 41% of the dataset. This indicates that identifying differentiating common attributes for the mammals class is the most important task, and among all the species, only mammals produce milk. Moreover, due to the involvement of sea mammals, only mammals have a non-consistent *fin* number. That is the main reason why LCSs identify *fin* and *milk* as the most important attributes to distinguish mammals from others. Meanwhile, birds only occupy 20% of the dataset, and *feathers* only could be used for birds, such that this attribute's function can be replaced by a

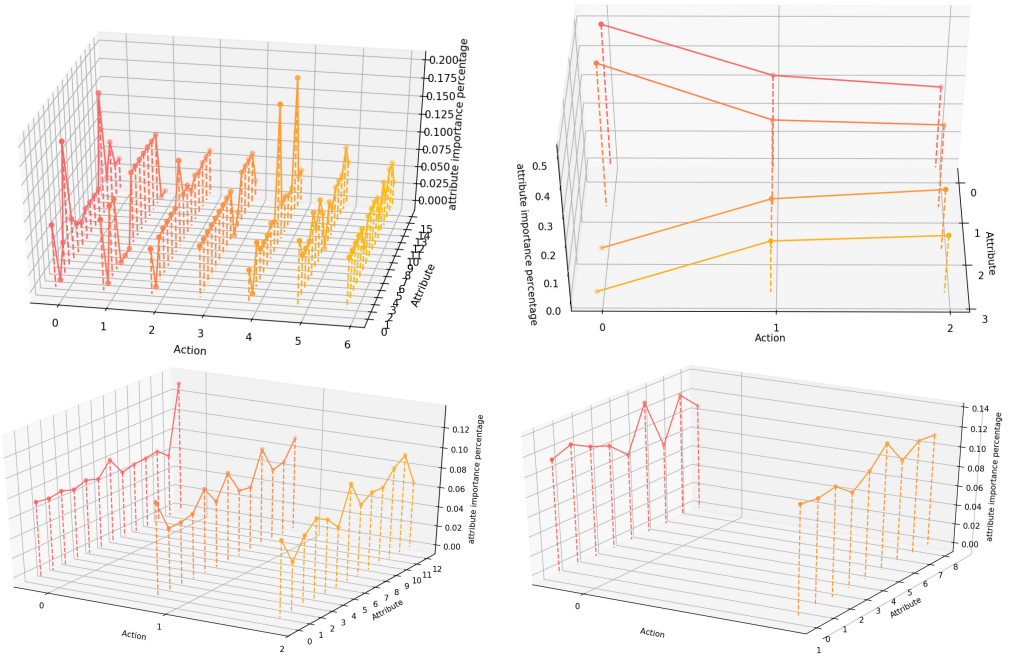


Figure 5.4: First line, the result for Zoo and Iris problems, respectively. Second line, the result for Wine and WBCD problems, respectively.

combination from the first ten attributes. Thus, the *feather* is a redundant attribute in this case, e.g. CFS removes *feather*, and in Relief, *feather* gets a low rank, which is 10th. Not only bird, but the mammal, amphibian, and reptile also note *feather* is redundant. Fish and bird do have a similar attribute importance distribution. Therefore, to distinguish these two species, certain discriminative attributes need to be considered. Thus, in fish support rules, the *feather* attribute is considered.

The ranked features could also be visualized with FIM (show in Figure 5.4). In the Wine domain, HLCS identifies *proline*, *color intensity*, and *OD280 / OD315* of diluted wines as the most important attributes for class0, class1, and class2, respectively. *Flavanoids* need to be considered for all the classes. The HLCS identifies that the most important attributes for WBCD are *Bare Nuclei* and *Normal Nucleoli* in both actions. Generally, as WBCD only contains two actions, the results for each action's attribute

importance distribution became very similar. However, the attributes' potential range is still different, which is why these two distributions are not exactly the same.

The result shows that compared with humans, LCSs have an advantage, which is to get rid of the primacy effect. LCSs are impartial to the underlying patterns for the domains. Therefore, by visualizing these hidden patterns, humans could understand the patterns in the dataset better as prejudice can be avoided. In contrast with standard attribute rank algorithms, the proposed HLCS offers a fine-grained level ranked attributes, which helps researchers not only know which attributes are important but also hints about why these attributes are important.

5.5 Conclusion of Real-Value Domain Compaction Algorithm

This work extends the RCR to adapt to real-value domains. The proposed RCR-real successfully produces optimal solutions by analyzing multiple XCSs' models based on upper and lower boundary representation format. In RCR-real's compacted models, the member rules are consistent and unsubsumable. Similar to natural solutions, the patterns in an RCR-real's solution is interpretable. Subsequently, a real-value domain orientated Feature Importance Map (FIM) is developed to translate the underlying patterns to discernable graphs. However, due to the rich nature of the employed representation format, rules represent the same niches (phenotypes) can have different encoding (genotype), thus different from natural solutions, RCR-real's can produce alternative solutions, which are neither deterministic nor unique.

A hierarchical architecture is introduced to XCSs. This architecture applies incremental learning and ensemble learning to LCSs, and the proposed LCS termed HLCS. HLCS divides the learning process into a set of

layers to incrementally collect the learned knowledge. The number of layers increases, automatically, until the evolved ruleset in these layers can complementarily represent the target dataset. For all the tested dataset, HLCSs produce a fully representative model with a defined representation format. Such models can reflect the attributes' importance in determining each action. Thus, HLCS's models can be applied to feature ranking tasks, where the underlying knowledge can reveal the ground truth of the explored dataset and avoid human prejudice.

Chapter 6

Absumption

Previous chapters have demonstrated that LCSs are prone to evolve models composed of all correct, unsubsumable rules under the global search space (natural solution). However, due to the issues caused by over-general rules, in overlapping domains, multiple LCSs' models are necessary for identifying all member rules in order to produce a natural solution. Producing multiple models and introducing additional optimization algorithms increase the complexity of the LCSs and the computation costs. This drawback of producing the natural solution was the inspiration of developing an operator that can assist LCSs to adapt to overlapping domains. Thus, a new operator termed as *informed mutation* is introduced to search for optimal rules based on correcting over-general rules. Furthermore, based on informed mutation, a new mechanism named *Absumption* is proposed to enable the LCSs to produce a single model that contains all the member rules of a natural solution. This complements the subsumption operator that was designed for improving over-specific populations. Besides, a new version of RCR terms Razor Cluster Razor in Single Ternary alphabet (RCR-ST) is also described that aims to extract the natural solution from an abusmption based LCS produced results.

6.1 Introduction

Visualizing the output of an algorithm is important for understanding the patterns in the data and how the algorithm produced solutions for the classification problem. However, producing a correct visualization result is not easy if the patterns in the underlying model are unnecessarily complicated, such as information distributed among edges and nodes of a network. Many sub-symbolic algorithms struggle in this regard [106] [86] [65]. Symbolic approaches offer a more human-readable alternative, but they struggle to produce maximal generalization, maximal accuracy, and minimal redundancy in their models [49] [62] [54].

Although the LCSs' produced population of rules [P] is often accurate, it is also inundated with a number of over-general rules and over-specific rules [44]. During exploration, LCSs do not have any explicit method to remove such rules. These issues are especially severe in over-lapping domains as different rules attempt to share the same data niche.

This chapter introduces the Absumption method as a complement to the Subsumption method that addresses over-specific rules. The Absumption method uses an over-general rule's inconsistency to identify this type of rule, then using an informed mutation operator attempts to correct the rule. Furthermore, a rule compaction algorithm that depends on Absumption is also proposed, termed Razor Cluster Razor in Single Ternary alphabet (RCR-ST), to remove all the irrelevant and redundant rules in the trained population [P]. Although the developed methods can work with real-world data, Boolean domains with different scales of condition length are selected as the example problems as the over-general issue-level can be controlled by tuning the data distribution. These permits investigate how Absumption acts on over-general issues. Moreover, the discovered underlying patterns in these domains can be more easily compared to the ground truth patterns. Within each domain, the problem size (number of bits) can scale, where accurate visualization of the solutions will enable

the equations for the optimal size of solutions for a problem to be verified, which increases user confidence in future solution verification.

6.2 Methods

Three methods are introduced in this section. The *informed mutation* which attempts to detect optimal rules based on correcting the over-general rules. *Absumption*, a new mechanism that aims to assist LCSs to adapt overlapping domains. *RCR-ST*, which is a rule compaction algorithm that is developed to discover natural solutions inside Absumption based LCSs' results.

6.2.1 Informed mutation

XCSs consider optimal rules are excepted to be located at a highly general search space. Thus, XCSs utilize tournament selection and Subsumption to guide the searching process to guarantee that the newly created rules have a higher generalization level than their parent rules. As a result, when exploring, XCSs tend to search the general search space but frequently misjudge the importance of specific regions of the search space. Recent experiments have demonstrated that highly specific rules can be necessary to address the target problem in overlapping domains. Hence, XCSs need an operator to explore new rules with highly specified conditions. In this scenario, the informed mutation is proposed to create specific rules based on almost nearly correct over-general rules.

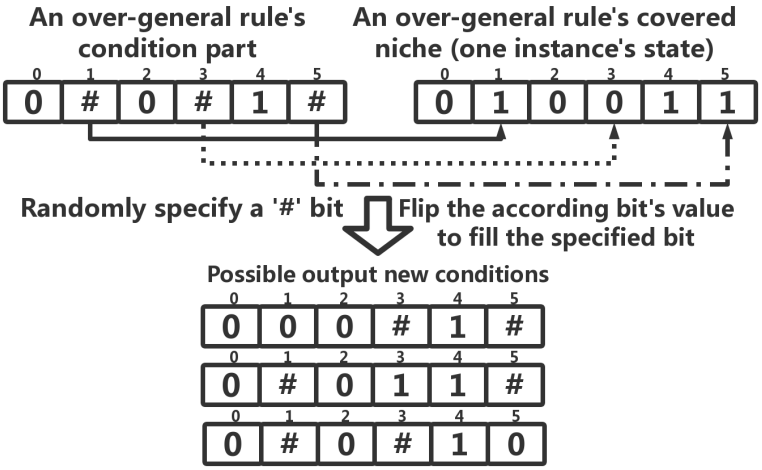


Figure 6.1: Informed mutation: Creating new conditions based on correcting an over-general condition. Randomly select a $'\#'$ bit to become the specific bit. Afterward, record the value of the corresponding bit on the instance. Create a new condition by replacing the selected $'\#'$ bit by the flipped recorded value. Hence, the new rules will not cover the observed instance any more.

Optimal rules are expected to make a consistent prediction for all their matched instances. Thus, an inconsistent rule must be an over-general rule. Informed mutation utilizes the information that an optimal rule is consistent. When a rule is identified to be inconsistent (over-general) by an instance, then the informed mutation receives this inconsistent rule and this instance. Afterward, the informed mutation creates new rules by excluding the instance from the inconsistent rule's encoding (shown in Figure 6.1).

For example, in the 6-bits Majority-On problem, *00####:0* is an over-general rule. Assuming that when exploring, this over-general rule is identified when it matches the instance *001111*. Then, the informed mutation will create a new rule based on rule *00####:0* and instance *001111*. The process is that informed mutation randomly selects a generalized attribute from the over-general rule, and sets this attribute value different

Algorithm 24: Informed Mutation

Input: an over-general rule r;
an instance that can identify this rule is over-general i
Output: a corrected rule nr

1 Randomly select a generalized attribute A_r in r;
2 find the corresponding attribute in i,A_i;
3 Create nr by Specifying the A_r with a different value of A_i;

from the instance, as the mutation is informed, this instance leads the rule being over-general. Here, assume the third attribute is selected, the newly produced rule is *000###:0*, which will no longer cover the instance *001111*. Hence, step by step, the informed mutation will produce suitable rules by correcting the over-general rules' niches (shown in Algorithm 24).

6.3 Absumption

To assist XCSs to overcome the over-general issue so that XCSs can adapt to overlapping domains, a novel mechanism termed Absumption is proposed. Absumption identifies over-general rules based on over-general rules' non-consistency. Absumption hypothesis that in XCSs, a rule's consistency can be assessed by this rule received environmental rewards' consistency. Thus, Absumption introduces two new training parameters, the number of positive rewards (NPR) and the number of negative rewards (NNR) to XCSs in order to assess a rule's consistency. NPR records the number of times a rule receives a positive reward from the environment, whereas NNR traces the number of negative rewards. Both NPR and NNR's initial value is zero. Absumption will be activated to rules, where the product of NPR and NNR exceeds zero, as this indicates a rule has become inconsistent.

Absumption uses a rule's experience and the size of its represented

Algorithm 25: Absumption

Data: rules in action set A; rules in population P;
the observed instance S; initialize an empty condition set for the Absumption created rules *OpC*;
initialize an empty rule set for final output rules *OpR*;
Result: Non-duplicated Absumption created new rules *OpR*

```
(REAthreshold, NPR, and NNR are defined in section 6.3 );
foreach rule ∈ 𝒜 do
    if rule.NPR * rule.NNR > 0 then
        if rule.REA < REAthreshold then
            Remove rule from A;
            Remove rule from P;
        end
        else
            for i = 0;i++;i < rule.numerosity do
                condition = Specify(rule.condition, S);
                if condition ∉ OpC then
                    Add condition to OpC;
                end
            end
            Remove rule from A; Remove rule from P;
        end
    end
    foreach condition ∈ 𝒪pC do
        rule=CreateRule(condition);
        Add rule to OpR;
    end
end
```

niches for assessing the probability of an over-general rule's encoding carrying correct genotypic information. In Absumption, this assessment probability termed, Represented niche and Experiment Assessment (REA), acts as a trigger of strategy selection for decomposing over-general rules. Absumption executes one of the two strategies, either *removing* or *replacing* for any identified over-general rule. When an XCS is exploring, in each iteration, all the identified over-general rules will undergo Absumption. For any over-general rules, their REA value is higher than the predefined REA threshold, then Absumption considers these rules' genes are highly likely to be close to being correct. Thus, it is reasonable for Absumption to invoke informed mutation to create new rules by correcting these over-general rules' niches. Afterward, the newly created rules replace the over-general rules in [P]. Regarding rules that associate with low REA, the condition part rarely carries correct genes. Hence, the *removing* strategy is evoked in order to eliminate these over-general rules from [P], so that the adverse impact of these rules on classification performance is prevented, immediately. The proposed work empirically set the threshold of REA as 1 (shown in Algorithm 25).

Absumption is similar to the specialization mechanism of Anticipatory Classifier System2 (ACS2) [80]. As a comparison, Absumption identifies over-general rules based on continuity rather than a determined probability as employed by ACS2. The altered distinction strategy affords Absumption the capability of addressing domains with severe over-general issues but results in the assisted LCSs suffering in the case of noisy environments. As Absumption requires a rule and its matched instances advocate the same class label, if a dataset contains noise instances, Absumption must result in producing rules overfit these noise instances. This is a tradeoff for overcoming the over-general issue.

Another similar mechanism is Lanzi's specific operator [59], but it differs from Absumption in three aspects. Firstly, these two techniques have different objectives. The specific operator is designed to enrich the diver-

sity of specific rules in [P], whereas Absumption aims to search for an optimal rule by iteratively correcting over-general rules. Secondly, they have varying strategies to identify over-general rules. The specific operator removes one of over-general rule's numerosity, and only removes the over-general rule, where the numerosity reaches zero. As a comparison, Absumption removes general rule, immediately, after they have been identified. This difference is because the specific operator considers that an over-general rule can have a positive impact on prediction performance. However, the Absumption hypothesis is that an over-general rule's importance in maintaining prediction performance can be completely replaced by the optimal rules. Thirdly, they create new rules via a different method. The specific operator randomly selects a generalized attribute and specifies this selected attribute with a random value. As a comparison, the Absumption specifies the selected attribute with an informed value, which excludes the incorrectly matched instance from the niche.

6.4 Razor Cluster Razor Single Ternary

XCSs continuously discover rules during the exploration process in case better patterns are discovered or the domain is dynamic. Thus, when XCSs cease the exploration phase, it is not guaranteed that all the final rules have been trained sufficiently. Besides, in order to achieve the capability of simultaneously exploring multiple competing hypotheses, XCSs employ a training population [P] that is likely to be much larger than the optimal number of rules. Hence, irrelevant and redundant rules unavoidably exist in the final [P], which obscures the informative patterns.

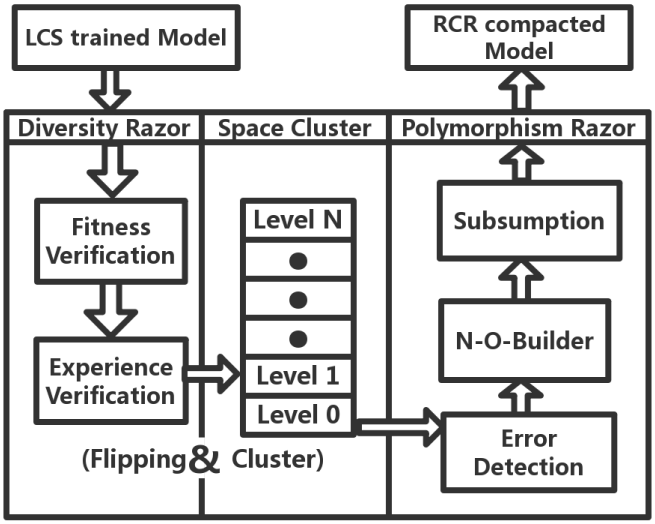


Figure 6.2: The flowchart of RCR-ST, that consists of diversity razor, space cluster, and polymorphism razor. RCR-ST is Absumption dependent. RCR-ST aims to compact a XCSs produced [P] to the most straightforward form.

For the sake of improving the Absumption-based on XCSs' produced models' interoperability to the state-of-art level, the Razor Cluster Razor Single Ternary (RCR-ST) method is proposed, as shown in Figure 6.2. This is an Absumption dependent rule compaction algorithm. This aims to optimize an XCS trained [P] to the most straightforward form. This method is inspired by the Occam's Razor theory, which is "Entities are not to be multiplied without necessity". The RCR-ST method composes of three phases, which are Diversity Razor, Space Cluster, and Polymorphism Razor.

Diversity Razor has a Fitness Verification Operator(FV) and an Experience Verification Operator (EV). These two operators aim to remove irrelevant rules by interrogating individuals' training performance according to the rules' training parameters. With Absumption, the irrelevant rules in the final [P] are expected to be either non-contributing to form the solution [P] or insufficiently trained. FV utilizes individuals' fitness to es-

timate a rule's contribution, removing the rules whose fitness reaches the minimum value, i.e. 0. EV removes any insufficiently trained rules when a rule's experience is less than its average niche. In this work, a rule's niche is set equal to its niche size.

Space Cluster is responsible for removing redundant rules by implementing the flipping and clustering process. In binary classification problems, flipping a completely incorrect rule's action could make it become a completely correct rule. Thus, the flipping executes on all the rules, whose action is associated with the minimum prediction (completely incorrect) to conduct the correction.

In ternary based XCSs, the global search space of exploring any N-bit binary problem consists of N+1 *sub-search spaces* that are distinguished by the number of specified condition attributes N_{sat}. For example, a 6-bit problem has seven *sub-spaces* ranging from the bottom (i.e. no specified bits, e.g. *######:1*) to the top (i.e. all specified bits, e.g. *011010:1*). Besides, each sub-space Sp_i is capable of independently fully representing the global problem domain. Clustering manages to place each rule into its related sub-space for the subsequent Polymorphism Razor.

Alternative rule combinations that represent the same niches cause ambiguous explanations for the underlying patterns for data, which is named the polymorphism issue [63]. Error Detection, Non-over-lapping set builder (N-O Builder), and Subsumption methods constitute the Polymorphism Razor. This is designed to solve the polymorphism issue by removing all redundant rules from [P], such that the simplest [P] with fully representative capacity can be constructed (a unique morphism).

The Absumption leads to potentially over-general rules' experience being slightly lower than their companions in the same sub-space. R stands for a single rule, where *exp* records the associated experience, Sp_i represents a sub-space, and $|Sp_i|$ is equal to the number of included rules, then $\forall\ R \in Sp_i$, if $R.exp < 0.2 * \frac{\sum_{R \in Sp_i}(R.exp)}{|Sp_i|}$, R are removed by Error Detection. Additionally, in any fully representative [P], if both optimal rules and in-

correct rules exist, then conflict must occur, i.e. two rules have overlap in a search space but propose different actions. In Error Detection, each rule is compared with the others, when conflict happens, record the sum of the numerosity as the conflicting value. Finally, remove any tested rule where its numerosity is less than its conflicting value.

In non-over-lapping domains, e.g. in the Multiplexer problem, there are optimal rule sets ([O] set), where no member rules share niches and assemble at the same sub-space. In these cases, the replaceable, redundant issue happens. All the replaceable rules are neither incorrect nor subsumable. Therefore, it is impossible to distinguish replaceable rules by analyzing their accuracy or generalization. An alternative way to address this problem is attempting to form a non-over-lapping and fully representative rule set on the sub-space, which has the highest sum of all the related rules' numerosity (Algorithm 26). If the N-O builder finds such a satisfied rule set, it removes all the other rules. Otherwise, it activates Subsumption in order to remove all the subsumable rules.

6.5 Result and Experiments of Absumption

Three types of artificial Boolean domains with different condition length are considered as benchmarks, i.e, the Majority-On, Carry, and Multiplexer problem domains. All the results experiments have been run independently thirty times.

6.5.1 Absumption Learning Performance

In all the domains, the covering method is responsible for covering any unmatched observations in the initial training stage. Because of the stochastic nature of the covering process, many over-general rules are placed into the population. This results in absumption's removing operator being highly activated at the start of the training. Once the covering process ceases, the

Algorithm 26: Find the fully representative non-overlap rule set from the sub-space that has the highest sum of numerosity

Data: The size of the problem's niche *GN*;
All rules in the selected sub-search space *S*;
initialize an empty rule set for temporary store S_{temp};
initialize an empty rule set for final output rules *Subset*;
Result: A fully representative rule set *Subset*

```
foreach rule ∈ 𝒮 do
    S_temp ← S;
    Add rule to Subset;
    Remove rule from S_temp;
    foreach rule′ ∈ 𝒮_temp do
        if ∀ r ∈ Subset; r ∩ rule′ =∅ then
            Add rule′ to Subset;
        end
    end
    SN ← 0 Subset represented niches;
    foreach rule′ ∈ 𝒮_temp do
        SN+=rule′.niche
    end
    if SN=GN then
        break;
    end
    else
        Subset ← an empty rule set;
    end
end
```

removing operator's activeness also reduces. However, instead of ceasing, the removing operator eventually maintains a steady activation level. This is due to XCSs employing the crossover search operator, which unavoidably generates incorrect rules, Figure 6.3 shows the rate of absumption application.

Although the removing operator continuously deleted rules from the population, the population's capability of representing niches did not reduce. That is because of the Absumption's specification operator's capability in protecting potentially correct genes in non-optimal rules.

For example, in the 6-bit Majority-On problem, *11####:1* is a typically good performing, but non-optimal rule, which has 87.5% accuracy, only incorrectly predicting niche *110000*. In this case, the informed mutation is invoked to produce new rules based on specifying an attribute to expel the problematic niche from the incorrect rule's condition. Therefore, after specification operator, four candidates may be introduced, which are *111###:1*, *11#1##:1*, *11##1#:1*, *11###1:1*. Hence, an incorrect rule's correct represented niche is generated. Since, in overlapping domains, the crossover search is prone to generate many good performance incorrect rules, such that, the specification operator is relatively active, whereas it is inactive in non-overlapping domains, e.g. Multiplexer.

Compared with the Carry problem, the Majority-On problem has a more complex over-lapping distribution, which results in producing over-general rules instead of over-specific rules, so that the Majority-On problem depends much more on the Absumption compared with Subsumption.

The GA Subsumption aims to prevent over-specific rules joining [P], and the Action set Subsumption is responsible for removing the over-specific rules in [P]. Previously, due to the difficulty of distinguishing over-general rules and optimal rules, the activation of the Action set Subsumption could introduce an adverse effect, as potentially optimal rules were incorrectly removed. However, the Absumption method continuously re-

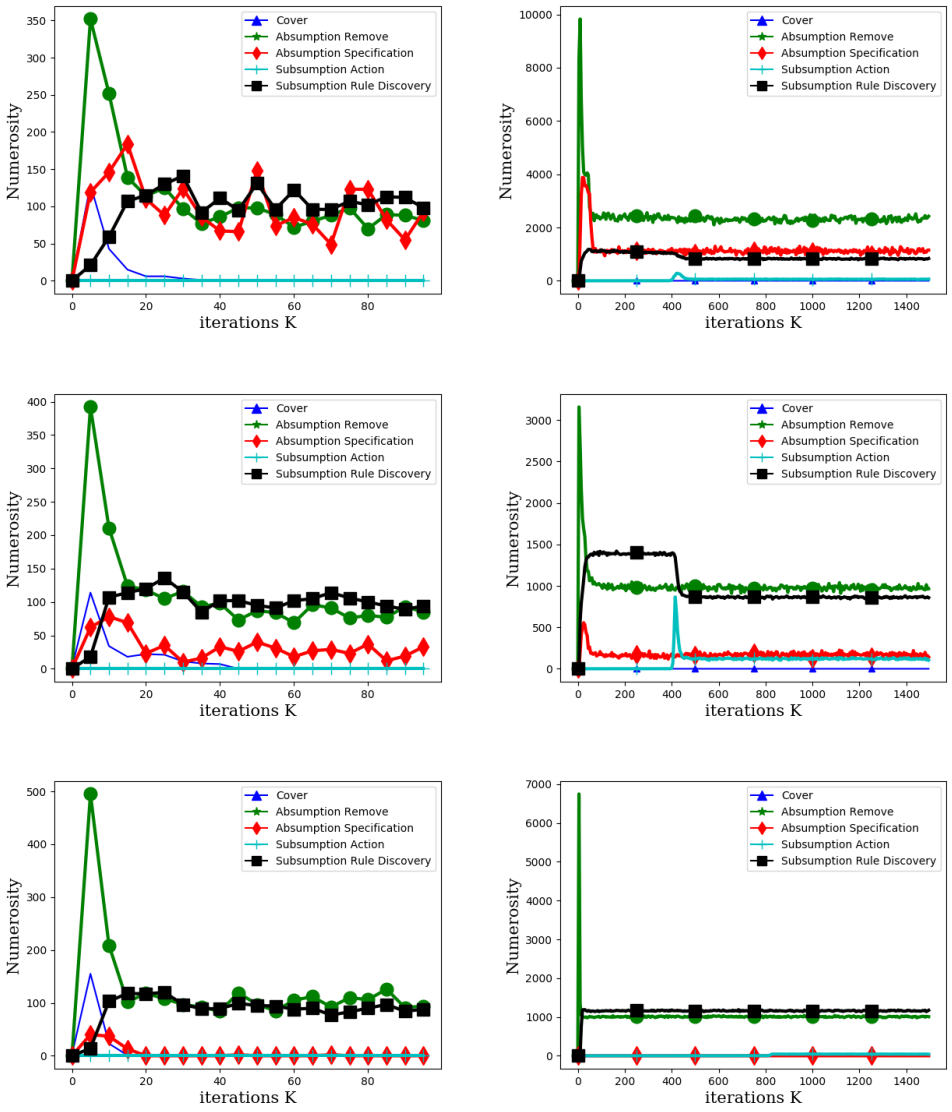


Figure 6.3: The impact of Absumption assisting XCSs in solving over-lapping domains. A clear positive correlation exists between Absumption's activation level and the problem's over-lapping level. First line: 6-bits and 10-bits Majority-On; Second line: 6-bits and 10-bits Carry; Last line 6-bits and 11-bits Multiplexer.

moves over-general rules, which guarantees the long survived rules' quality. For instance, *#11:1*, which represent two instances, needs to be verified with over 200 experience.

Hence, once Absumption is activated, the Action set Subsumption can also be activated. The Action set Subsumption does not contribute to 6-bit problems, as the training process is too short of marking the "long survival" rules. For the 10-bit Carry, and Majority-On problems, the Action set Subsumption demonstrates the ability to remove over-specific rules efficiently. Once the Action set Subsumption is highly activated, the possibility for activating the GA Subsumption [1] decreases, which indicates that the rule discovery process generates less over-specific rules.

6.5.2 Absumption to Addresses the Over-General Issue

The primary role of Absumption is assisting XCSs to overcome overlapping domains, to address the over-general issue. Previously, the community considers that XCSs can produce acceptable models for overlapping domains. However, all these models' training accuracy can be at most 99%, which violates the natural solution hypothesis. Such a phenomenon happens because XCSs represent the explored domain with a combination of over-general rules and over-specific rules rather than optimal rules, which is the over-general issue. The produced problematic models are likely to possess useful patterns. Absumption enables XCSs to address the over-general issue by removing the over-general rules, immediately and consistently, so that optimal rules can be preserved during exploration. Although, improving the accuracy by 1%-2% may appear as a small contribution, in terms of pattern understanding, the ability to ascertain optimal rules is a major improvement.

The Majority-On domains have a severe over-general issue. These do-

[1] Executing the Subsumption operator to the GA evolved rules before inserting this newly created rule into a population.

Table 6.1: Samples of XCSs produced rules from both Absumption is active or inactive for 8-bits Majority-On problem. In this domain, the correct optimal rules for action 1 or action 0 need to specify five 1s or four 0s, respectively, otherwise, they are over-general rules.

No-Absumption	Absumption
1##1#1#1:1	11#1#1#1:1
#111###1:1	1111###1:1
11#1##1#:1	11#1##11:1
0##0##0#:0	0##0##00:0
###0#00#:0	###0000#:0
#0#0###0:0	00#0###0:0

mains' optimal rules have an inherent horizontal distribution, i.e. many similar rules assemble at the same sub-space. This distribution results in XCSs having a tendency to produce good performance over-general rules. This tendency gradually obscures the optimal rules. Hence, the over-general rules are incorrectly highlighted so replace the optimal rules in $[P]$. For example, in 8-bit Majority-On, *1111####:1* is a highlighted incorrect rule, where error only occurs in instance *11110000* among the represented sixteen instances. As these incorrect rules have a relatively good performance, XCSs may incorrectly highlight them, especially in reinforcement-based XCSs. Moreover, due to the employed crossover search operator, XCSs may produce many such incorrect rules. Eventually, these over-general rules dominate the population, which causes further misclassification.

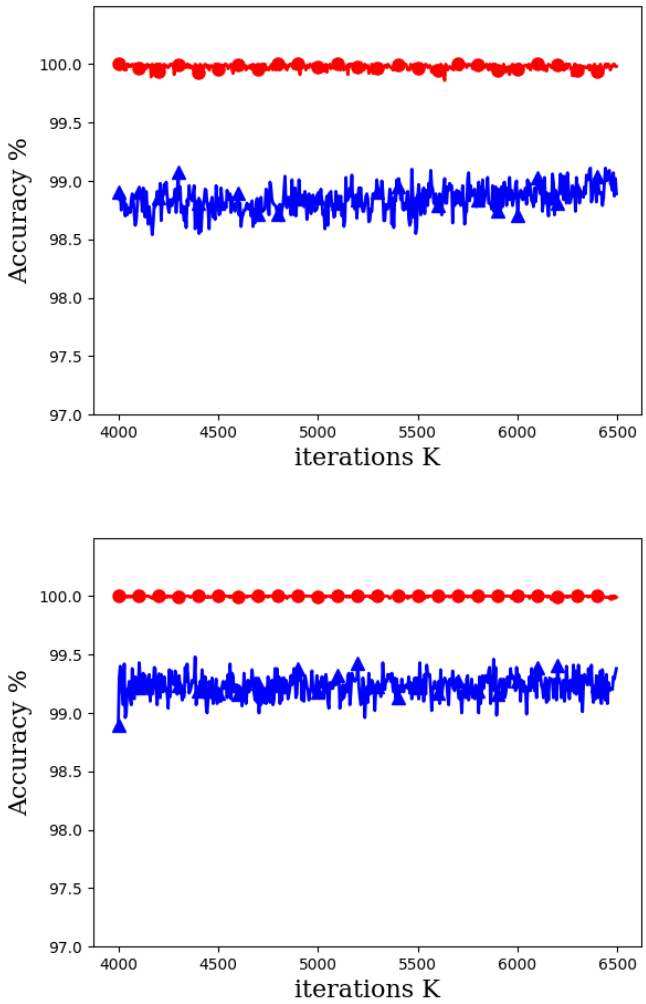


Figure 6.4: Red circles stand for XCSs with Absumption, blue trangles represent XCSs without Absumption. For clarify, only the training performance on the last 2.5 million iterations is shown. Left to right: 10-bit Majority-On, 12-bit Carry. The Mann-Whitney U test output results for 12-bits carry problem, and 10-bits majority-on problem are $3.03e^{-13}$, and $3.11e^{-13}$, respectively. Note, dependent axis scale.

The Carry domains have a vertical distribution optimal rule set, i.e. optimal members are non-uniformly distributed in several sequential sub-

spaces. In these cases, good performing over-general rules occur in the middle of the related sub search space, which omits some very specified optimal rules. In general, the Carry problem's over-lapping level is less than the Majority-On problem. Thus, XCSs can achieve better performance, but still cannot produce a model to fully represent the problem, when the domains' number of attributes is larger than eight.

Figure 6.4 demonstrates that Absumption does assist XCSs to address overlapping domains. Furthermore, the results of both domains are much smaller than the critical value of U, which is 0.05 in the Mann-Whitney U-Test. The results from the conducted U test on the final performance of two systems show statistically significant performance in their 1% improvement regarding accuracy.

6.5.3 Absumption Avoids Over-General Rules

XCSs utilize reinforcement learning techniques, e.g. Q-learning, to estimate each rule's performance. As a result, good performing over-general rules might be incorrectly highlighted. Subsequently, the employed Subsumption methods will continuously subsume vital rules that are relatively more specific. Gradually, a [P] will be dominated by these problematic rules. Since standard XCSs lack methods for counteracting over-general rules, they are unable to completely represent any overlapping domains that contain more than eight attributes (8-bits). Therefore, XCSs have been incorrectly labeled as a technique that cannot learn all Boolean functions [38].

A defective [P] not only suffers from unsatisfactory classification performance but also has trouble with the transparency of the mined underlying patterns, e.g. the top map in Figure 6.5, which is based on a standard XCS produced [P] that has had the incorrect rules and the subsumable rules removed completely. This map merely ambiguously reflects that the 6+6 Carry problem has a symmetrical attribute importance distribution.

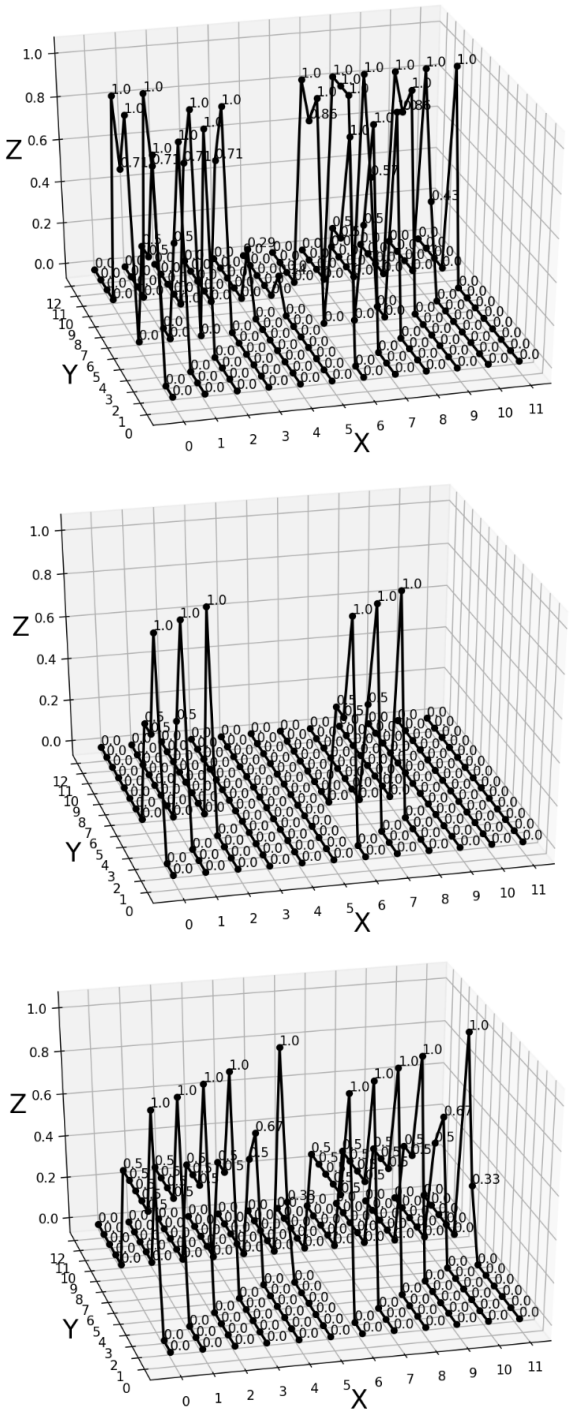


Figure 6.5: X: Attribute Id, Y: sub-search space Id, Z: Attribute Importance. Three attribute importance maps for three compacted [P]s of 6+6 Carry problem. Top to bottom: first, a standard XCS's [P]; second, optimal rules in a standard XCS's [P] showing that attribute 2, 3, 4, 5, 8, 9, 10, 11 are missing; third, XCS with activated Absumption produced [P].

As a comparison, the bottom map, which is based on an optimal rule set, visualizes the symmetrical patterns precisely.

The second map visualizes the patterns of optimal rules in a defective [P], which only partly exhibits the symmetrical characteristic of the 6+6-bits Carry problem. Since the optimal rules with a relatively more specific encoding are missing, the second map cannot offer any pattern about bit_3 to bit_5, and bit_8 to bit_{11}. The first two maps demonstrate that without the assistance from Absumption, in Carry domains, over-general rules result in either obscuring or fragmenting visible patterns. Similar maps of the Majority-On domains are not presented, due to all the optimal rules being replaced by a good performing over-general kindred. Therefore, after compaction, only one over-general rule is kept.

6.5.4 Razor Cluster Razor Single Ternary

RCR-ST is a rule compaction algorithm, which builds on Absumption. Absumption avoids over-general rules in the produced rule set. Thus the binding compaction algorithm does not suffer from the poor performance caused by incorrectly keeping the over-general rules but removing the optimal rules. RCR-ST aims to mine the optimal rule set from a single XCS's trained [P]. Any N attributes Boolean problem BP_N consists of M instances ($BP_N \supset$ [Ins_0, Ins_1, ..., Ins_M]) the ternary representation's global search space (GS) can naturally split into N+1 sub-spaces ($GS \supset$ [SP_0, SP_1, ..., SP_N]), based on the number of specified attributes N_{sat} of an encoding. Each SP consists of $C_N^{N_{sat}} * 2^{N_{sat}}$ available rules (r). In SP_N, all the attributes are specified. Hence, SP_N = BP_N. Therefore, for any BP_N, at least one fully-representative model in GS can be found. Straightforwardly, $\forall\ Ins$, $Ins \in BP_N$, $\exists\ r$, $Ins \in r$, r is completely accurate, $r \in GS$. Hence, $\exists\ P$, $P \supset$ [r_0,r_1,... ,r_i], $BP_N \in P$.

In a fully representative P, $\forall\ r$, if r is not completely accurate, then r is irrelevant to BP_N. $\forall\ r_i$, $r_i \in P$, $r_i \in SP_i$, if $\exists\ r_j$, $r_j \in P$, $r_j \in SP_j$, i>

Table 6.2: Non-Absumption XCS [*P*] size and accuracy versus Absumption XCSs, S-size and S-Accuracy respectively stand for number of rules and classification accuracy for standard XCSs without Absumption, whereas, A-size and A-Accuracy record the results for the Absumption based XCSs. With the assistance of Absumption, XCSs achieve maximal accuracy and optimal rule set (sizes are recorded in the A-size) for all the tested domains.

Problem	S-Size	S-Accuracy	A-Size	A-Accuracy
6-bit MUX	172	100%	8	100%
11-bit MUX	222	100%	16	100%
20-bit MUX	4216	100%	32	100%
37-bit MUX	6822	100%	64	100%
6-bit Carry	124	100%	18	100%
8-bit Carry	512	99.8%	38	100%
10-bit Carry	852	99.4%	78	100%
12-bit Carry	950	99.2%	158	100%
6-bit Maj-On	157	100%	35	100%
7-bit Maj-On	561	100%	70	100%
8-bit Maj-On	2672	99.6%	126	100%
9-bit Maj-On	6305	99.4%	252	100%
10-bit Maj-On	9244	98.6%	462	100%

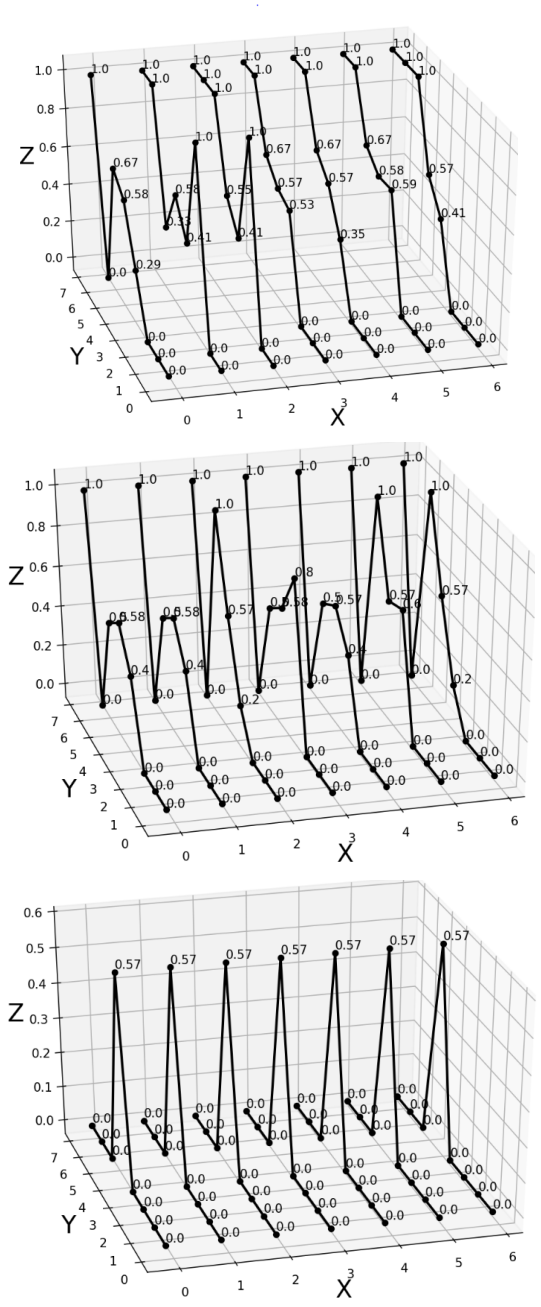


Figure 6.6: Three FIMs show that an Absumption based XCS's [P]'s underlying patterns are highlighted by RCR-ST removing incorrect rules and redundant rules in sequence for the 6-bits Majority-On problem. X: Attribute Id, Y: sub-search space Id, Z: Attribute Importance. The top graph presents the patterns inside an XCS's [P], the middle graph describes the patterns of [P] when incorrect rules (over-general rules) are removed by RCR-ST's diversity razor. The last graph shows the patterns after RCR-ST have been completed, which are optimal.

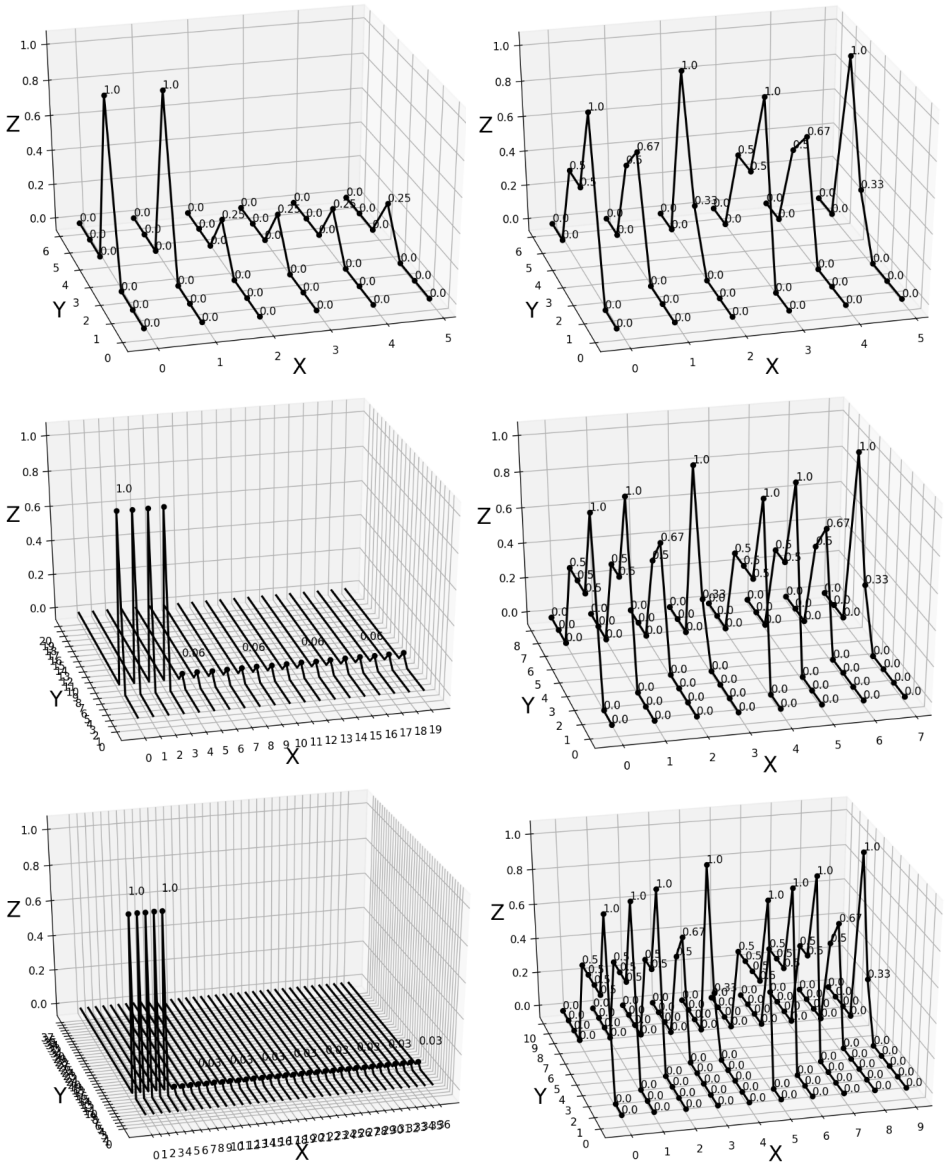


Figure 6.7: Six FIMs show how a Boolean problem's patterns change with the problem' size scale. X: Attribute Id, Y: sub-search space Id, Z: Attribute Importance. The figures in the first column are for 2+4 bits, 4+16 bits, and 5+32 bits Multiplexer, where all the difference between address bits and data bits are highlighted. The second column's figures respectively represent the 3+3 bits, 4+4 bits, and 5+5 bits Carry problems, which clearly distinguish the difference between the two parts of each problem.

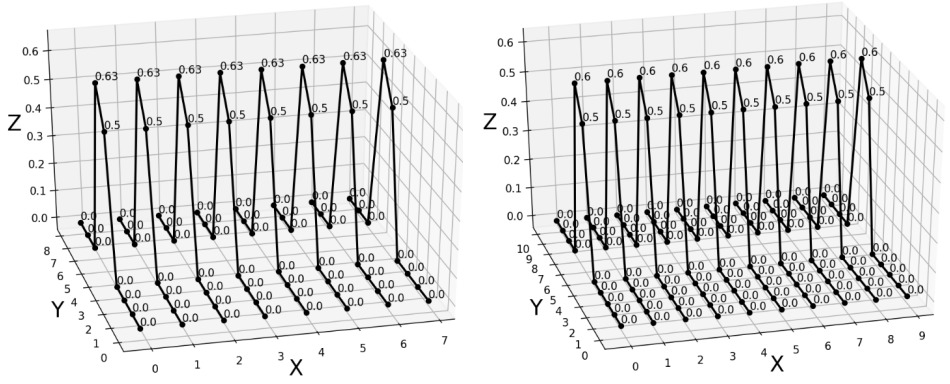


Figure 6.8: Two FIMs show how a Boolean problem's patterns change with the problem' size scale. X: Attribute Id, Y: sub-search space Id, Z: Attribute Importance. The two figures show the optimal rules of even cases (8-bits and 10-bits) in the Majority-On problems locate at the same generalization level, which explains why the Majority-On domains having a severe overlapping issue.

j, $r_i \in r_j$, then r_i is redundant. The number of representative instances of an r is equal to $2^{N-N_{sat}}$. Hence r's representation size is ≥ 1 once a P matches all the instances, P is fully representative. Thus for any BP_N the minimum member of rules of a fully representative P must satisfied that $P \leq M$. Straightforwardly, XCSs with ternary representation could be considered as an automatic data compression algorithm, once the dataset is successfully compacted, the informative patterns of the dataset become apparent.

The attribute importance following the sub-space clustering in RCR-ST collected classifiers is shown in Figure 6.6 (6-bits Majority-On problem). Essentially, for a given attribute level, this shows how likely a given attribute is to be specified. This clearly visualizes the different underlying patterns in each problem domain. Moreover, by implementing this method in the same domain with various scales (length of the condition), the hidden patterns that influence the attributes at a specific scale have

been detected.

Figure 6.7 and Figure 6.8 show how the attribute importance gradually becomes clarified by removing irrelevant rules based on diversity and removing redundant rules based on polymorphism. In all the Multiplexer problems, all the optimal rule members were limited to one subspace for all the results. This supports the Multiplexer problems are non-overlapping. The difference of importance between address bit and the data bit is highlighted. Moreover, although the data bits' attribute importance is decreased as the problem scales, all the address bits' attribute importance is kept at 1.0. From the 3+3-bits to 6+6-bits Carry problems, the RCR-ST visualizes a natural symmetry to the attribute importance distribution. Additionally, when the problem is increased by 2 bits, only one additional sub-space is introduced. In the Majority-On problems, the visualization reflects a distribution relationship between the odd condition length and even condition length problems. In even length problems, all optimal rules cross two sub-spaces, whereas, in odd length, optimal rules gather in one sub-space.

The RCR-ST successfully compacts all the thirteen tested problems' model to their maximal accuracy, maximal generalization, and minimal redundancy state, which is also the problems' optimal rule set ([O] or *natural solution*), shown in Table 6.2. The last 1% accuracy is the most difficult part for XCSs to represent in over-lapping domains. The 100% accuracy [P] has the potential to highlight all the optimal rules, whereas a 99% accuracy [P] in certain domains, may highlight none of the optimal rules, as shown in Table 6.1. An additional result of this work is the equation for calculating each problem's optimal rule size. In Multiplexer problems, all optimal rules contain the specified address bits and one data bit that reflect the action. Hence the size of the optimal rule set for the k-bits address Multiplexer problem equals 2^k.

All the Carry problems' optimal rules have a regular distribution, e.g. in 6-bits the number of preserved rules on sub-space 2-4 are [2,12,4]. When

the Carry problem scales by 2 bits the covered sub-space will increase by one, e.g. in 8-bits [2,4,24,8], on sub-space 2-5, 10-bit [2,4,8,48,16], and 12-bit [2,4,8,16,96,32]. Hence, the Carry problems' optimal rule set size = $2^{\frac{N}{2}+1} + \sum_{k=1}^{k=\frac{N}{2}-2} 2^k$, $(N > 4)$. In N-bits Majority-On problems, the size of the optimal set needs to consider the length of condition; in the odd situations, size=$2 * C_N^{\frac{N+1}{2}}$, whereas even situations, size=$C_N^{\frac{N}{2}+1} + C_N^{\frac{N}{2}}$. It would be very difficult (and has not been achieved previously) to mine these rules from a single standard XCS produced population.

6.6 Conslusion for Absumption

An informed mutation operator is designed to search for important specific rules by correcting over-general rules. Based on informed mutation, a mechanism named Absumption is introduced to assist the XCS symbolic rule production system to address Boolean problems that have an over-lapping issue. The training analysis reflects that the use of the Absumption method has a positive correlation between the method's activity level and the problem dataset's over-lapping level. Moreover, for all the tested over-lapping domains, Absumption enables the generation of fully representative models.

An Absumption dependent rule compaction algorithm, RCR-ST, is also introduced, devoted to identifying an optimal rule set from a fully representative model by removing irrelevant diversity and redundant polymorphism rules. Moreover, according to the optimal rules' distribution, the formulations for calculating the number of optimal rules for the Multiplexer, Majority-On, and Carry problems can be defined.

Chapter 7

Absumption and Subsumption based LCS (ASCS)

Learning Classifier Systems (LCSs) have been identified as useful data mining techniques that can obtain optimal results that contain human-discernable patterns [76] for a number of tasks. Previously, Butz described LCSs are designed to evolve a minimal number of non-overlapping rules to represent an explored domain, accurately and completely [16]. Such optimal rulesets term as [O] sets. However, due to [O] sets not representing the non-overlapping domains, and LCSs were not designed to construct a non-overlapping ruleset, LCSs rarely produce [O] sets as the final result. According to the experiments in Chapter 6, another style of the optimal ruleset consisting of consistent, unsubsumable rules is discovered. This new form of optimal rulesets are termed *natural solution*s, which also have good interpretable patterns.

Furthermore, *natural solution*s can be employed to represent domains that have an overlapping distribution and can be easily constructed by LCSs' employed search mechanism, i.e. Subsumption. Such a discovery inspired the development of a new LCS that considers *natural solution*s as the search objective. Furthermore, this novel LCS is designed to promote Absumption, Subsumption, and informed mutation as the primary search

strategies, rather than the standard evolution operators such as crossover, mutation, roulette wheel selection, and tournament selection. Thus, this novel LCS is termed Absumption and Subsumption based learning Classifier System (ASCS). This chapter introduces the ASCS. The conducted experiments are to demonstrate that ASCS enables training efficiency and easy patterns visualization.

7.1 Introduction

Previously, Butz et. al [18] defined the optimal ruleset [O] of LCSs and demonstrates that the optimal results contain human-discernable patterns. [O] sets are expected to completely, correctly represent an explored domain with a minimal number of non-overlapped rules. However, in the past two decades, no LCS considers [O] sets explicitly as the final results. The employed searching strategies should be responsible for this strategy. Thus, a change to the traditional evolutionary operators, i.e. crossover, mutation, Subsumption, roulette wheel deletion, and tournament selection is needed. When these operators work cooperatively, the only characteristic of the evolved rules that can be guaranteed is that rules will be highly generalized, but they cannot guarantee that the evolved rules are non-overlapped [7]. In another word, standard LCSs naturally cannot produce [O] sets for representing the majority of the commonly tested domains.

Experiments shown in Chapter 5 and 6 have evidenced the existence of another style of optimal rule sets that contain all the consistent, unsubsumable rules under the global search space, i.e. *natural solution* [63]. *Natural solution*s also possess useful underlying patterns that can reflect the nature of the explored domains. Meanwhile, unlike [O]s that can be multiple for a single noiseless dataset, the relevant *natural solution* is deterministic and unique. This indicates that, once a noiseless dataset's *natural solution* is identified, the underlying nature of the dataset can be visualized. Fur-

thermore, constructing a *natural solution* is technically supported by one of the LCSs' fundamental searching mechanisms, i.e. Subsumption, which is designed to assist LCSs to search for the maximally generalized rules.

LCSs have an over-general issue when attempting to explore domains that have an overlapping distribution [1] [69], e.g. the Carry problems and the Majority-On problems. This is because, LCSs employ crossover and mutation as the primary search strategies, which results in the evolved rules' possible generalization levels being determined by $P_{\#}$ [1]. When $P_{\#}$ is inappropriately set with a high value, LCSs omit the important specific rules and evolve over-general rules, resulting in a cover and delete cycle [23]. Conversely, if $P_{\#}$ is set too low to fit the target problem, LCSs lack the motivation to discover the important general rules, leading to an inefficient exploration performance. Thus, an inappropriate $P_{\#}$ setting can result in a crossover and mutation based LCS producing a poor performance model.

Absumption is a complementary mechanism to Subsumption (see Algorithm 25 in Chapter 6). Absumption is designed to generate specific rules by correcting over-general rules. When Absumption is activated as a secondary search strategy, LCSs can overcome a domains' over-general issues. However, the processing of Absumption has previously been in the same paradigm as mutation and crossover in evolving new specific rules, which wastes computational resources. Furthermore, LCSs still fail in overlapping domains when an LCS' frequency of its rule discovery mechanism introducing new over-general rules surpasses the efficiency of Absumption identifying over-general rules, so that they can be removed. All these circumstances lead to the necessity of designing a new LCS, which can consistently evolve an optimal result is, simple to control, and efficiently exploresthe domain.

The Absumption and Subsumption based Learning Classifier System (ASCS) is proposed, which considers *natural solution*s as the search objec-

[1] probability of generalizing attributes when covering

tive. Furthermore, Subsumption, Absumption, and informed mutation are promoted as the only search methods for discovering and removing rules. ASCS replaces all the LCSs' current obsolete training parameters (see Table 7.1 and Table 7.2) with five new parameters that do not affect or restrict the searchable generalization levels. Moreover, ASCS is designed to efficiently and consistently produce models that contain human-discernable patterns for noiseless domains.

Three artificial Boolean domains from 6-bits to 70-bits noiseless environments are used as benchmarks. The proposed system addresses the 37 and 70-bits Multiplexer problems, showing whether the new search strategies retain the ability to adapt to large scale Boolean domains. Successfully visualizing patterns is demonstrated as in all the experienced domains, the new system produced models that can be directly translated into graphs that contain human-discernable patterns, especially on the 13-bits and 14-bits Majority-On problems that respectively require 3432 and 6435 different interacting co-operating rules in the optimum solution to be uniquely identified to enable correct visualization.

7.2 Proposed Methods

Absumption and Subsumption Learning Classifier System (ASCS) is a rule-based machine learning system with a generalization-level based architecture for exploring noiseless domains (shown in Fig. 7.1). The primary objective for ASCS is to search for rules that contain human-discernable patterns from a given search space for an observable dataset. According to previous experiments, [O] sets can possess such interpretable patterns. However, [O] sets are difficult to be evolved by LCSs, and [O] sets fail in representing domains with an overlapped distribution. Recent experiments have demonstrated that the *natural solution* also contains human-interpretable patterns, which coincides with the proposed system's objective. Furthermore, the processing of producing *natural solution*s is sup-

ported by an LCSs' fundamental search mechanism, i.e. Subsumption that assistant LCSs to search for the maximally generalized (unsubsumable) rules. Hence, ASCS considers *natural solution* as the search objective. This makes ASCS differ from previous LCSs, as ASCSs have a certain searching objective rather than utilize alternative ruleset to represent an explored domain.

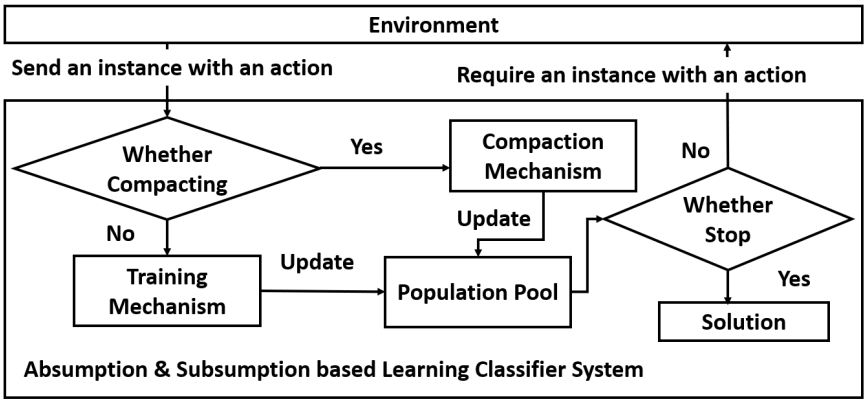


Figure 7.1: The figure shows one epoch of ASCS for learning. A learning process of ASCS contains Ep_{num} epochs. Each epoch runs the training mechanism with review T_{num} number of environmental instances and invokes the compaction mechanism once. ASCS terminates this process after all the predefined epochs.

The member rules of an *natural solution* are expected to be **consistent** and **unsubsumable** under the global search space (see Section 3.2). Thus, ASCSs consider these two characteristics as the most important condition to determine the evolved rules that need to be kept. For efficient exploration, ASCS manages rules in a class label and generalization based population pool, so that when introducing new rules, the computation resource that is spent on identifying whether a rule is redundant is reduced.

Furthermore, ASCSs altered the primary search techniques that leads to ASCSs omit the unnecessary system complexity for evolving target rules.

As optimal rules are expected to be located at different generalization levels for the most domains, ASCSs introduce incremental learning to enable the system to gradually identify the target solution for getting rid of the negative effect of the good performing over-general rules. Thus, ASCSs divide the learning process into a set of epochs, i.e. supervised batch learning. Each epoch executes certain iterations of the training mechanism for exploration and then invokes the compaction mechanism to remove rules that are either over-general or over-specific for maintaining the populations' compactness. The learning process ceases once ASCS has completed all the epochs. ASCSs introduce five new parameters in controlling the exploration phase and three new parameters for tracing each rules' training performance. However, ASCSs abandon all the obsolete training parameters of XCS and UCS (show in Table 7.1 and Table 7.2) in order to deliver a system that is easy to control.

7.2.1 Rules in ASCSs

An ASCS is supervised and multiple rules-based. A rule is the most basic unit for capturing the learned patterns from the environment. A rule has an condition part for identifying the covered niches, an action part for recoding the output action, and three parameters for tracing this rule's learning performance. ASCSs consider the ternary alphabet representation as the encoding format. For example, in Boolean domains, an encoding considers all the attributes and is a string formed by {0,1,"#"}, where "#" is the don't care symbol that matches both 0 and 1. The compaction of the encoding is feasible because the condition's representation ensures the formation of generalization, the rules whose condition has "#", that can match multiple instances. A rule's action is one of the plausible actions in the environment.

ASCSs are expected to identify consistent, unsubsumable rules. Therefore, each rule in ASCSs has two parameters: the **positive experience** (P_e)

and **negative experience** (N_e) that respectively record the number of times that this rule advocates or opposites a matched instance, with initial values as zero. P_e and N_e can assess a rule's consistency, effectiveness and distinctiveness. Since any consistent rule's product of P_e and N_e is expected to be zero.

The Subsumption mechanism can determine a rule's unsubsumable characteristic. However, due to new rules being continuously introduced during the evolutionary training process, and rules with different generalization levels requiring a different number of experience to guarantee this rule's correctness. Thus, a constant experience threshold may fail in identifying over-general rules that are highly generalized and possess high prediction performance. These problematic rules have the potential to incorrectly subsume specific optimal rules, which results in an incomplete ruleset. For the sake of ensuring each rule has been trained sufficiently to avoid the issues caused by good performing over-general rules, ASCSs assign each rule a **Subsume state** (P_{state}) parameter. P_{state} identifies whether a rule is authorized to subsume other rules. The initial authority of a rule's P_{state} is false, and this authority will be altered to true if the rule compaction process preserves this rule at the end of an epoch.

7.2.2 Parameters for controlling ASCSs

LCSs have a number of parameters for guiding the system to construct a representative model. These parameters are designed based on one of the two primary objectives: either aiming to identify a newly created rule's generalization level, or aiming to support LCSs to determine whether a rule should be preserved.

Finely tuning these parameters assists LCSs to adapt to various problem domains. However, setting these parameters with arbitrary values can lead to LCSs to produce models that perform poorly. For example, XCSs and UCSs employ the probability $P_{\#}$ to identify the initial generalization

Table 7.1: Training parameters setting table. Note, in LCSs, GA Threshold θ_{GA} is the threshold for applying the GA application in an action set.

System	Parameters for controlling system
XCS	Learning Rate, Exponent, GA Threshold, Crossover Probability, Mutation Probability, Covering Probability, Deletion Fitness, Reduction Fitness, Tournament Ratio, Fitness Fall Off Rate, Deletion Threshold, Prediction Error Threshold, Subsumption Threshold, Population Size, Iteration;
UCS	Learning Rate, Exponent, GA threshold, Crossover Probability, Mutation Probability, Covering Probability, Deletion Fitness, Reduction Fitness, Tournament Ratio, Subsumption Threshold, Population Size, Iteration;
ASCS	Epoch Number, Training Number, Compacting Number, Maximum Rule Number, Experience Discounting Ratio.

Table 7.2: A rule's parameters

System	Parameters for Recording rule's Performance
XCS	Prediction, Prediction Error, Accuracy, Fitness, Numerosity, Experience, Action Set Size, Time Stamp;
UCS	Numerosity, Fitness, Accuracy, Action Set Size, Deletion Vote, Time Stamp GA, Match Count, Correct Count;
ASCS	Positive Experience, Negative Experience, Subsumption State.

levels for rules produced by the covering process. Furthermore, when crossover and mutation are considered as the primary searching strategies, the consequence is $P_{\#}$ can determine the searchable generalization level range. Thus, an inappropriate setting in $P_{\#}$ can result in the majority of the evolved rules possessing a generalization level that does not fit the explored domain. The **Subsumption threshold** is an indispensable parameter for identifying whether a rule has the ability to subsume other rules. Mis-setting this parameter can lead to a model being overwhelmed by over-general rules.

To avoid the issues caused by inappropriately setting parameters, ASCSs deliver a simple control system that removes all LCSs' obsolete training parameters (show in Table 7.1 and Table 7.2). ASCSs automatically determine where a newly created rule' generalization level is, and whether a rule should be kept, rather than be intervened by thresholds that need to be predefined. To enable automatic exploration, ASCSs split the whole exploring process into a set of epochs, so that ASCSs can adaptively adjust

the evolving rules' generalization to the most suitable levels for representing the target problem, epoch by epoch. The **Epoch Number** (E_N) defines the number of epochs that need to be completed before the exploration ceases.

An epoch has a training phase and a compacting phase. Each epoch reviews an equal number of environmental instances and their relevant actions, where these instances are split into two parts, i.e. for training and compacting. **Training Number** (T_N) specifies the number of instances that need to be received in the training process. **Compacting Number** (C_N) defines how many instances the compacting process reviews. ASCSs use a combination of E_N, T_N, and C_N together to define the total number of iterations. This definition differs from other LCSs that use a single constant number for setting the number of training/testing instances. This change is necessary for ASCSs, as ASCSs exclude the identified redundant generalization levels at the end of each epoch. Thus, ASCSs are able to automatically narrow the search space. T_N implies the number of rules that will be evolved. Thus, when setting T_N, it is necessary to consider the number of member rules in the target optimal solution. C_N relates to the ASCSs' capacity to distinguish over-general rules. Increasing C_N is a practicable strategy to aid ASCSs to address domains that have an overlapping distribution. Allowing adjusting the E_N, T_N, and C_N permits the ASCS to efficiently adapt to domains with different sizes of search space and different data distributions.

An ASCS is a multiple populations based system and a population contains multiple rules. Thus, ASCSs are at risk of being very slow if there are too many candidate rules in the model when exploring. Thus, ASCSs introduce **Rule Maximal Number** (RMN) to identify a suggested maximum number of rules in the model. Different from other LCSs, ASCSs do not accept duplicated rules, so ASCSs' rules do not possess numerosity. Thus, ASCS's number of rules is equivalent to the number of the rules in the model, rather than the sum of all the possessed rules' numerosity. Be-

sides, RMN offers a suggested number, rather than a number that strictly restricts the maximum number of rules. This is because RMN only affects the rule discovery mechanism but does not impact the covering process. New rules still can be introduced through the covering process, even after the limit of RMN has been exceeded. Meanwhile, RMN does not function as a trigger for invoking rule deletion, e.g. when a model's actual rule number exceeds the defined RMN.

ASCSs provide an **experience discounting ratio** (D_{rate}) to specify a rate for discounting all the preserved rule's P_e and N_e at the end of each epoch. Reducing preserved rules' experience facilitates the processing of subsuming early epoch produced over-specific rules by optimal rules detected in a later epoch. Otherwise, these over-specific rules require a large number of epochs to be removed.

7.2.3 Population Pool in ASCS

LCSs verify the redundancy of a newly created rule by comparing this rules with all rules that exist in the population. However, this verifying process only needs to consider rules that are more general or have the same generalized as the new evolved rule, and any rule possesses a less generalization level is irrelevant to the verification. Thus, it is hypothesized that storing the evolved rules in a structure that is based on the rules' generalization level can avoid the unnecessary computation cost incurred the verification process. Hence, ASCSs introduce a new structure for keeping rules, which is termed the *population pool*. This clusters rules according to their advocated actions, possessed consistency tendency, and located generalization level.

A population pool organizes rules in three stages. In the first stage, ASCSs categorize rules into a set of groups [G_A] according to their advocated actions. This is because optimal rules that advocate the same action have a similar pattern in constructing their encodings, i.e. specify a simi-

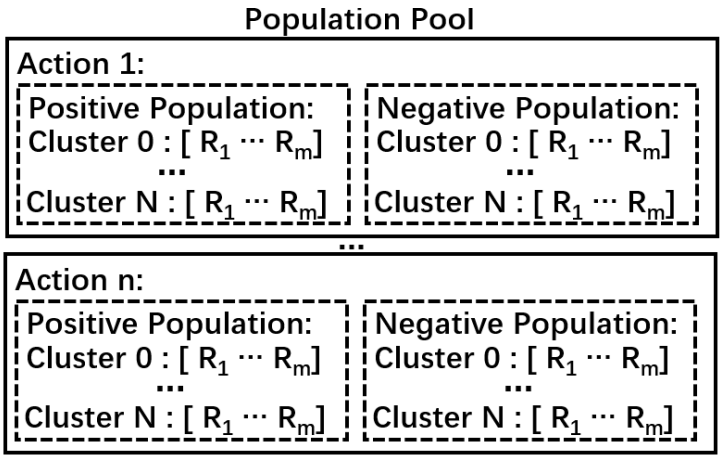


Figure 7.2: ASCS's population pool containing populations in the form of [Action [P_p, P_n]...], P_p and P_n are respectively the positive and negative population for rules that are completely correct and completely incorrect. In the populations, rules are clustered according to the number of generalized attributes.

lar number of attributes or specify attributes with the same value. In the second phase, in each [G_A], ASCSs group rules into different populations according to the rules' consistency tendency. ASCSs consider two types of consistency tendency, either positive or negative. Thus, each [G_A] subordinates two populations that are termed positive population [P_p] and the negative population [P_n]. Rules that advocate all the covered instances' actions are considered as completely correct rules [2]; these rules have a positive consistency tendency and will be stored in [P_p]. Oppositely, the completely incorrect rules are assigned to [P_n]. This stage enables ASCSs to support both the complete map and the best action map. When ASCS accepts both [P_p] and [P_n], the produced models are complete maps, which is the same as XCSs. Once an ASCS excludes the [P_n], the produced models are the best action maps , which are the same as the UCSs' results.

In [P_p] and [P_n], the last stage clusters the rules based on the number of generalized attributes in rules' encodings. For example, an N-attributes domain, a population contains N+1 clusters that are ranked from the most specific $cluster_0$ to the most general $cluster_N$ (shown in Fig. 7.2). This stage not only reduces the computation costing for inserting new rules but also facilitates the implementation of Absumption and Subsumption. Besides, an additional benefit from this clustering architecture is to enable the ASCSs to exclude redundant search spaces as the exploration progresses. As each cluster represents a portion of the search space, when an redundant search space is identified, ASCSs can deactivate these redundant clusters when evolving new rules. As a result, ASCSs can narrow the search space to the most reasonable range. To enable the capacity to control the search space, ASCSs assign each cluster a property A_{state} to identify whether this cluster is available to explore for searching. Rule discovery strategies, e.g. the covering and the informed mutation, are forbidden to produce rules that belong to an unavailable cluster, but this restriction

[2] In ASCSs, a completely correct rule's N_e are expected to be zero, whereas a completely incorrect rule's P_e must be zero.

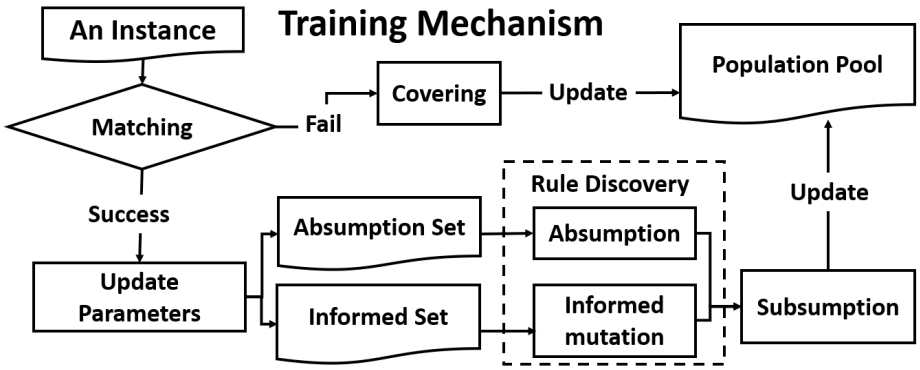


Figure 7.3: ASCS's training mechanism only considers the generalization-level based methods as the search algorithms: Subsumption for removing over-specific, Absumption for exploring specific and removing over-general, and an informed mutation for exploring general rules.

does not affect Absumption. The search ability of clusters is always available, except after compaction, when a cluster and its nearest more specific cluster do not contain any rules.

7.2.4 Training Mechanism in ASCS

ASCS innovatively considers Absumption, Subsumption, and informed mutation instead of crossover and mutation as the primary strategies for the evolutionary process, i.e. covering process and rule discovery. The main motivation for altering the search strategies is to produce *natural solution*s as the final result. This enables ASCSs' results to contain human-discernable patterns, rather than merely possess accurate prediction performance. Furthermore, for efficient removal of potentially irrelevant rules or redundant rules, ASCS replaces the traditional roulette wheel deletion as both Absumption and Subsumption to delete problematic rules (shown in Figure 7.3).

ASCS is a supervised learner. In each learning iteration (shown in Al-

Algorithm 27: Matching process in ASCSs

Data: Population Pool P_{pool};
An environmental instance I;
Result: Match Sets $[Ms]$ (Each population has one [M])

```
foreach G_A ∈ P_pool do
    [M] ← empty set
    foreach rule ∈ P_p do
        if rule match I then
            Add rule to [M]
        end
    end
    foreach rule ∈ P_n do
        if rule match I then
            Add rule to [M]
        end
    end
    Add [M] to [Ms]
end
```

Algorithm 28: Covering & updating process in ASCSs

```
Data: Population Pool P_pool;
An environmental instance I;
Match Sets [Ms] from Matching process;
foreach [M] ∈ [Ms] do
    if [M] is empty then
        find the corresponding G_A in P_pool;
        for Cluster ∈ P_p ∈ G_A from most general to most specific do
            if Cluster is available for search then
                in this Cluster, create a rule covers I;
                break;
            end
        end
        for Cluster ∈ P_n ∈ G_A from most general to most specific do
            if Cluster is available for search then
                in this Cluster, create a rule covers I;
                break;
            end
        end
    end
    else
        foreach rule ∈ [M] do
            if rule advocate I's action then
                rule.P_e++
            end
            if rule opposite I's action then
                rule.N_e++
            end
        end
    end
end
```

gorithm 27), the learning mechanism begins with stochastically receiving an instance I that follows the format of $state : action$. For each [G_A], from its [P_p] and [P_n], ASCS attempts to form a match set [M] that is composed of all rules that can match I's state. Thus, the evolution of each [G_A] is independent of others. For any [G_A], if its [M] does not contain any rule, the covering method is invoked to create two new rules to match the I's state with the conditions that respectively belong to the most general available cluster in [P_p] and [P_n]. To any [M], which is not empty, ASCS updates the matched rules' E_P and E_N. Once a rule and the matched I advocate the same action, E_P increases by one. Otherwise, add one to E_N (shown in Algorithm 28).

Algorithm 29: categorize matched rules into Absumption set or informed set

Data: Match Lists $[Ms]$ from Matching process;
Result: an Absumption Set $[AB]$;
an Informed Set $[IN]$;

```
foreach [M] ∈ [Ms] do
    foreach rule ∈ [M] do
        consistency = rule.P_e*rule.N_e;
        if consistency != 0 then
            Add rule to [AB]
        end
        else
            Add rule to [IN]
        end
    end
end
```

After updating the experience, the matched rules will be categorized into either the Absumption set [AB] or the Informed set [IN], respectively, for the consistent rules and inconsistent rules (show in Algorithm 29). The

consistency is assessed according to the product of a rule's P_e and N_e, if the product is not zero, then the rule is inconsistent, oppositely, the rule is consistent.

Algorithm 30: Informed Mutation

Data: an Informed Set $[IN]$;
Result: a set for new generated rules $[new]$;
Clustering rules $\in [IN]$ into a set of clusters (generalization level);
foreach *cluster* $\in$ *clusters* **do**
    if *rules in Cluster has specified attributes* **then**
        Randomly select a rule;
        Create a rule by randomly generalize a specified attribute;
        Add the newly created rule to $[new]$;
    end
end

Rule Discovery is composed of Absumption mechanism and Informed mutation. Absumption mechanism affects all the rules in [AB] that attempt to evolve important specific rules by correcting the over-general issues of these identified inconsistent rules. The implementation detail of Absumption mechanism can be reviewed in Chapter 6. The informed mutation attempt to improve the model's generalization level by evolving higher generalized rules based on [IN]. In ASCS, Informed mutation begins with clustering all [IN]'s rules according to the number of generalized attributes, and then informed mutation randomly selects one rule from each cluster. If the selected rule has specified attributes, the informed mutation randomly chooses one of the specified attributes, which is then substituted with the don't care symbol to represent the chosen attribute.

After the rule discovery, ASCS verifies all the newly created rules' redundancy, i.e. whether the new rule exists in the population pool, or other rules can subsume the new rule. In ASCSs, redundancy verification is

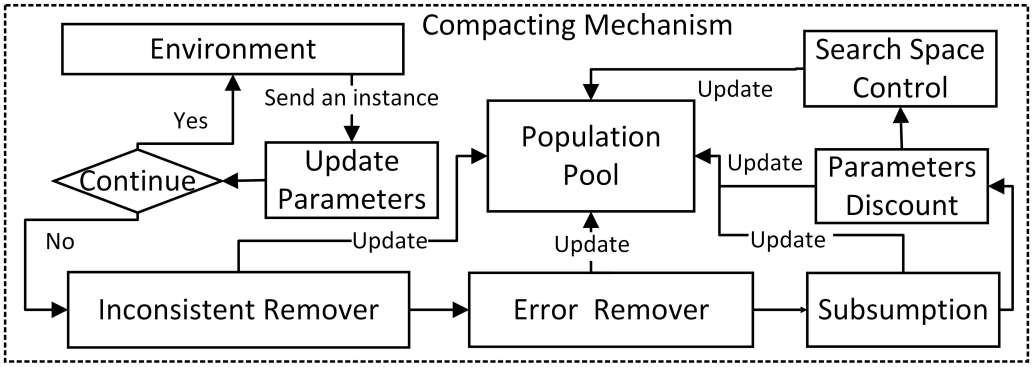


Figure 7.4: ASCS's compacting mechanism: functions are executed sequentially to maintain the compactness of the populations and adjust the range of the possible search space.

simple. ASCSs check the cluster, where the new rule will be inserted, to identify whether this rule exists. Then, ASCSs compares the new rules with old rules in the same population that are located at higher generalized level clusters to identify whether the rule can be subsumed. In ASCSs, old rules that can subsume new rules need to simultaneously satisfy three pre-conditions: an old rule can completely cover a new rule's represented niches, the two rules advocate the same action, and the old rule's P_{state} is true. ASCS removes all the identified redundant new rules.

At the end of each iteration, ASCSs insert all the remaining new rules into the population pool. Meanwhile, Absumption removes all the rules in $[AB]$ from the population pool. Note, Absumption is the only mechanism that has the authority to remove existing rules in ASCSs during evolution. Compacting during training also removes unnecessary rules.

7.2.5 Compacting Mechanism in ASCS

The compaction mechanism (shown in Fig. 7.4) is designed to maintain the compactness of the populations by eliminating problematic rules that are

either over-general or over-specific. At the beginning of the compaction, a set of environmental instances is reviewed to ascertain potential over-general rules. This process also ensures that all the current rules have been trained appropriately. Then, *inconsistent remover* and *error remover* act to remove over-general rules. Inconsistent remover utilizes the same strategy as Absumption to identify over-general rules. Whereas, for each $[G_A]$, Error remover ascertains the remaining over-general rules by comparing rules in $[P_p]$ with rules in $[P_n]$ (shown in algorithm 31).

Algorithm 31: Error Remover for a Positive Population

Input: all rules in a positive population P_{pos};
all rules in a negative population P_{neg};
Output: a sub-set rules for removal task S_{rem}

```
1  S_rem ← an empty list;
2  foreach R ∈ P_pos do
3      R_Error ← 0;
4      foreach R' ∈ P_neg do
5          if R and R' have overlapping in condition then
6              R_Error+=R'.E_N
7          end
8          if R_Error  R.E_P then
9              S_rem append R;
10             break;
11         end
12     end
13 end
```

Subsumption is responsible for removing over-specific rules by deleting any subsumable rules from the populations. After removing problematic rules, ASCS discounts all remaining rules' P_e and N_e by the D_{rate} rate so that over-specific rules evolved during early epochs can be properly

subsumed in future epochs. Finally, ASCS applies the search space control strategy for adjusting the available range of the search space to the most reasonable area for the next epoch.

7.2.6 Prediction in ASCS

The use of typical least mean squares (Widrow-Hoff) learning rule is also omitted with ASCS. Given an instance, ASCS makes a "best guess" prediction of the "weight" to be expected for each possible class label. ASCS creates a Weight List WL for storing all the weights. The weight for a class label is experience-based that is equivalent to the summation of E_P of all matched advocated rules in P_p minus the summation of E_N of all matched opponent rules in P_n. Two summations are considered since P_p records the knowledge of which is an instance's expected class label, whereas P_n shows which class label is unexpected.

7.3 Results

The difference in training performance among UCS, XCS, and ASCS is shown in Fig. 7.5. UCSs utilize rules' accuracy to guide the searching process. When the Subsumption mechanism for the action set is inactive, UCSs are prone to keep all the evolved completely correct rules. This searching preference enables UCSs to produce an accurate model, quickly. However, the UCSs' models include a large number of rules, where most of them are irrelevant for producing informative patterns (shown in Table 7.3 and Table 7.4, UCSs' models have proper performance in prediction but contain very few patterns).

When the crossover algorithm serves as the primary evolutionary method, the newly evolved rules' available generalization is limited by the initial generalization of rules, which are introduced by covering. In other words, $P_\#$ determines the generalization of the newly evolved rules. Hence, it

Table 7.3: A - training accuracy, T - time when the system first reaches 100%. If a system cannot produce a fully representative model, T will be marked as $--$. Symbols as s, m, h, and d denote the second, minute, hour, and day, respectively. ASCS completely addresses all the tested domains as it can adapt to domains that need specific rules or have an over-lapping issue.

Problem	XCS		UCS		ASCS	
	T	A	T	A	T	A
6 MUX	1m	**100%**	5s	**100%**	**2s**	**100%**
11 MUX	9m	**100%**	1.2m	**100%**	**27s**	**100%**
20 MUX	1.6h	**100%**	3.5m	**100%**	**56s**	**100%**
37 MUX	1.2d	**100%**	1.5h	**100%**	**47m**	**100%**
70 MUX	5d	**100%**	3d	**100%**	**13h**	**100%**
6 CAR	2.3m	**100%**	5s	**100%**	**1s**	**100%**
8 CAR	9.7m	**100%**	33s	**100%**	**25s**	**100%**
10 CAR	3h	**100%**	18m	**100%**	**47s**	**100%**
12 CAR	$--$	99.6%	3.9h	**100%**	**34m**	**100%**
14 CAR	$--$	99.18%	$--$	99.6%	**2h**	**100%**
8 MAJ	74m	**100%**	11s	**100%**	**4s**	**100%**
11 MAJ	$--$	98.7%	**7.4m**	**100%**	22m	**100%**
12 MAJ	$--$	97.36%	$--$	98%	**42m**	**100%**
13 MAJ	$--$	95.6%	**27m**	**100%**	3h	**100%**
14 MAJ	$--$	95.3%	$--$	97%	**12h**	**100%**

Table 7.4: Models' capacity for producing patterns: the number of rules in [O] for Multiplexer problems, and in *natural solutions* for Carry and Majority-On problems shown in brackets. The rule numbers in brackets are for best action maps (UCS). The numbers should be double in the case of complete maps (XCS, ASCS). N is the number of rules in a single model, and P is the percentage of a single model capturing important rules. A model can produce a correct visualization result only if its P is 100%. ASCS is the only system that can precisely evolve all the important rules for patterns in a single model.

Problem	XCS		UCS		ASCS	
	N	P	N	P	N	P
6 MUX (8)	117	**100%**	1269	**100%**	**32**	**100%**
11 MUX (16)	1243	**100%**	2744	**100%**	**108**	**100%**
20 MUX (32)	4085	**100%**	6786	99%	**320**	**100%**
37 MUX (64)	5194	**100%**	7174	71%	**768**	**100%**
70 MUX (128)	13448	70%	30857	30%	**1726**	**100%**
6 CAR (18)	192	**100%**	780	**100%**	**36**	**100%**
8 CAR (38)	607	**100%**	3796	**100%**	**76**	**100%**
10 CAR (78)	836	98.5%	4954	56.2%	**156**	**100%**
12 CAR (158)	1716	64.8%	6978	26.3%	**316**	**100%**
14 CAR (318)	2339	22%	8978	14.9%	**636**	**100%**
8 MAJ (126)	978	99.2%	2599	88%	**252**	**100%**
11 MAJ (924)	3097	42%	6490	55%	**1848**	**100%**
12 MAJ (1716)	6778	38%	8490	17%	**3432**	**100%**
13 MAJ (3432)	8503	7.7%	8739	7%	**6864**	**100%**
14 MAJ (6435)	9114	1.2%	10747	3.6%	**12870**	**100%**

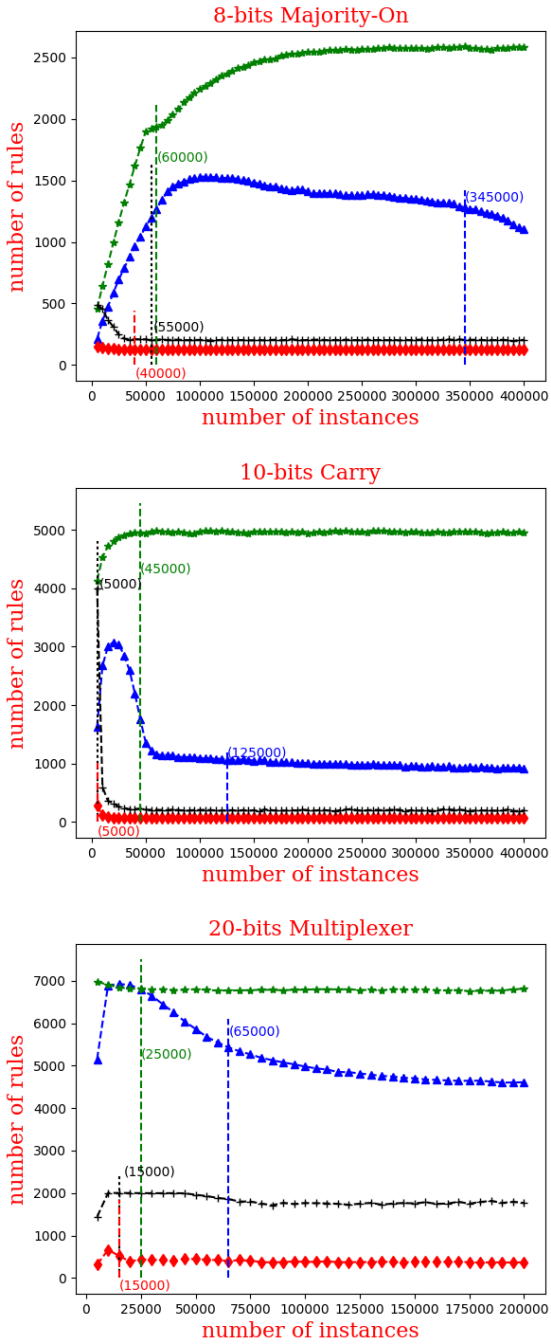


Figure 7.5: A training performance comparison graphs among XCS, UCS and ASCS. The green star and the blue triangle stand for UCS and XCS, respectively. The black cross and the red rhombus record the number of rules an ASCS possesses before compaction and after ASCS compaction. Vertical bars mark when the training performance first reaches 100% accuracy.

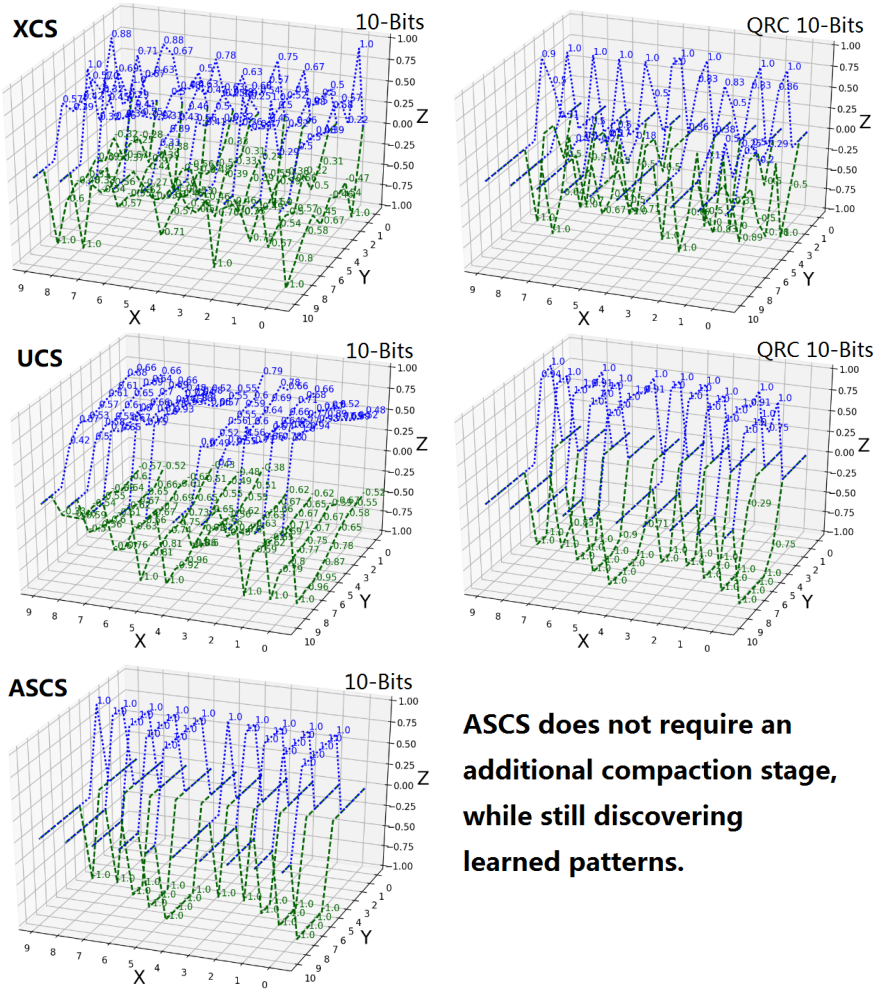


Figure 7.6: Patterns for action 1, and action 0 are colored with blue and green, respectively. X is the attribute ID, Y is the cluster ID that ranks by the number of generalized bits, Z represents the average value of specified attributes. From top to bottom rows are the results from XCS, UCS, and ASCS. The 10-bits Carry's models that respectively before and after compacted by quick rule compaction (QRC) are visualized. This shows ASCS's models can be visualized without being processed by additional compaction algorithms.

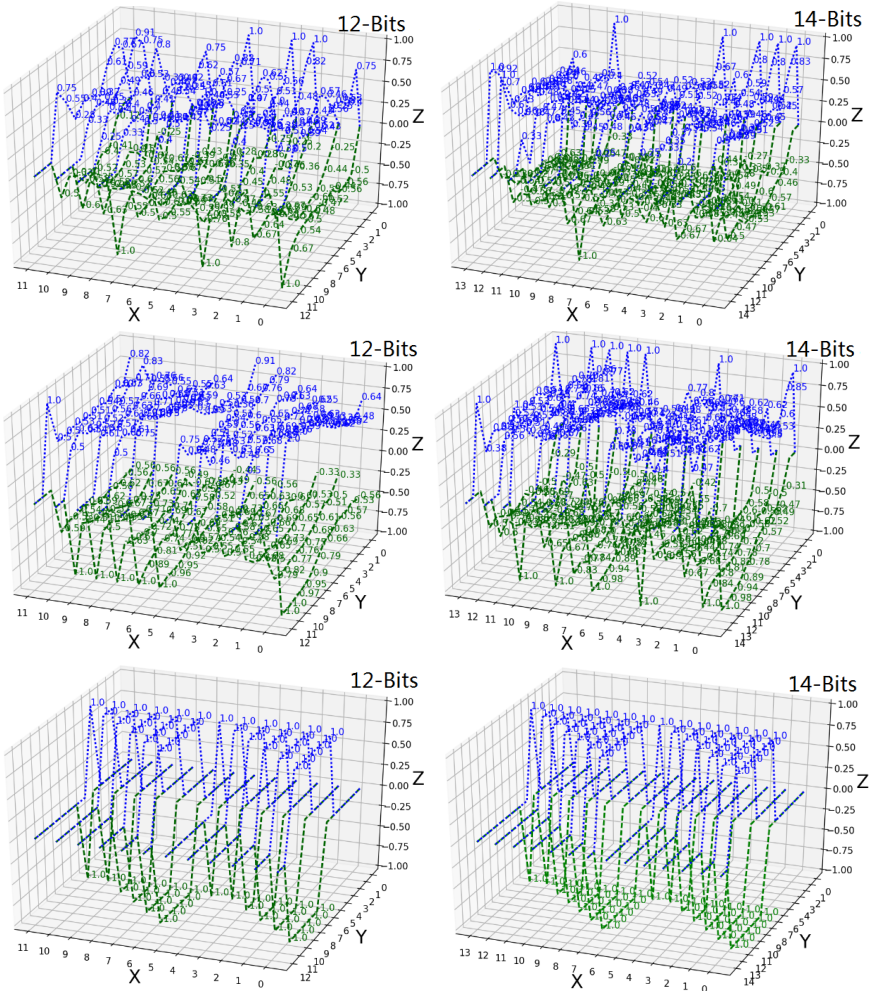


Figure 7.7: Patterns for action 1, and action 0 are colored with blue and green, respectively. X is the attribute ID, Y is the cluster ID that ranks by the number of generalized bits, Z represents the average value of specified attributes. From top to bottom rows are the results from XCS, UCS, and ASCS. The graphs reflect the visualizations for 12-bits and 14-bits Carry problems. This shows ASCS has an advantage in producing graphs that contain human-discernable patterns over XCSs and UCSs in complex domains. Besides, results show patterns for the same domains will be consistent regardless of the change of scales.

is practicable for crossover-based LCSs to address complex domains once the solution rules' generalization is known, e.g. UCSs and XCSs can address the 70-bits Multiplexer by carefully tuning the $P_{\#}$. However, due to the limitation of crossover, it is difficult for UCS to fully address domains that need a set of complementary rules across many different generalization levels. For example, UCS fails on the 14-bits Carry problem, which requires rules across seven different generalization levels to construct a fully representative model. Besides, due to the employed error threshold ϵ_0 [3], Subsumption does not consider the relationship between an individual rule's generalization and its experience, so good performing over-general rules risk being incorrectly considered as a Subsumption candidate. Hence, UCS cannot address situations in the Majority-On problems, that have a severe over-general issue, e.g. 12-bits and 14-bits.

XCSs tend to evolve and preserve the most general rules, which results in XCSs' models being more compact and containing more patterns than UCSs. However, the priority of considering rules' generalization leads XCSs to need more iterations to achieve a satisfactory prediction performance than the other two systems. Besides, XCSs frequently omit important specific rules. As a result, XCSs fail in completely address any Carry problem that is larger than 10-bits. Similar to UCSs, XCSs have issues in identifying high-accuracy over-general rules. Hence, XCSs do not work properly for the Majority-On domains.

ASCSs seek the completely correct unsubsumable rules. During the training process, only the inconsistent rules are removed by Absumption. An inconsistent rule must be an over-general rule, which is poor to describe the problem domain. Thus, ASCSs ensure that important rules will not be incorrectly removed before they have been trained sufficiently. As a result, ASCSs are the only system that produce models that not only reach the maximum accuracy but also contain full patterns for all the tested domains.

[3] ϵ_0 is the error threshold under which the accuracy of a classifier is set to one.

Previously, in crossover based LCSs, the rules' searchable generalizations are determined by $P_{\#}$. When the $P_{\#}$ is set lower than the optimal value, LCSs' models will be flooded with redundant specific rules, and lose the motivation to search for rules that are more general. Oppositely, when $P_{\#}$ is unnecessarily high, over-general rules are continuously introduced to models, then LCSs omit the important specific rules. As ASCS can search the whole search space, it can completely address problems that need rules that are located at different generalization levels, e.g. the Carry problems. However, a too large search space brings an issue that ASCS cannot converge in a small number of iterations, e.g. in the 37-bits Multiplexer problem, where although ASCS's model achieves satisfactory performance in both prediction and discovery of patterns, the model contains many redundant rules. In the 70-bits Multiplexer problem, due to the huge search space, similar to UCSs and XCSs, ASCSs need to limit the available generalization level to 60 and 68 [4] to grant ASCS the capability to address this problem in an efficient manner, (ASCS spends 13 hours for achieving the maximal accuracy, whereas XCS and UCS need 5 days and 3 days, respectively).

7.3.1 Pattern Visualization

Producing a correct visualization for an explored domain is difficult, as this requires a model to contain all the member rules of a *natural solution* or an [O] and also remove all the redundant/irrelevant rules. Previously, it was practicable to achieve correct visualizations by compacting multiple XCSs' or UCSs' models. However, such a process of compacting consumes a lot of computing resources, e.g. around 50 UCSs' models that have been trained for 4 million iterations are required to collect all the 1716 different interacting rules for producing the correct graph for the 11-bits Majority-

[4] The available generalization level is set as 60 to 68 that aims to imitate the UCS and XCS's assigned value of $P_{\#}$ for the 70-bits Multiplexer problem.

On problem. Now, ASCS can produce the same graph within 260 thousand iterations in a single run. Due to ASCS's high-efficiency in searching for *natural solutions*, it is plausible to visualize the patterns for more complex domains, which have not been done previously. For example, the 13-bits, the 14-bits Majority-On problems respectively require 3432, and 6435 different interacting co-operating rules (note, these rules are not linearly separable), and 14-bits Carry problem that expects 158 rules, which are located at seven different generalization levels (shown in Figure 7.6 and Figure 7.7).

7.4 Conclusions on ASCS

ASCS replaces the traditional crossover and mutation operators in LCSs with the Subsumption and Absumption mechanisms as the primary search methods for the evolutionary process. ASCSs preserve the correct unsubsumable & unabsumable rules, rather than UCSs which protect all the fully correct rules, and XCSs pursue the most generalized rules. ASCS retains previous LCSs' capacity to adapt to large complexity domains, e.g. addressing the 37-bits and 70-bits Multiplexer problems. This supports Butz's conclusions regarding the crossover operator having a minor influence on niche support, and once the most generalized rules are found that, the mutation operator is disruptive.

Furthermore, ASCS achieves fully representative models for the 13-bits, 14-bits Majority-On, and the 14-bits Carry problems, which overcomes XCSs' and UCSs' drawbacks in exploring domains that require specific rules and have an overlapping distribution. This can be ascribed to ASCSs' quick response to remove identified over-general rules, and ASCSs only being allowed to remove identified over-general rules in training mechanism. Such strategies reduce the over-general rules' negative influence in omitting important specific rules in GA Subsumption. Hence, the essential specific rules can be evolved and kept. Furthermore, since ASCSs

consider the *natural solution*s as the search objective, the produced models of ASCSs can be directly transformed into human-discernable graphs without any additional compaction procedures.

Chapter 8

Conclusions

The theme of this work is that considering multiple, instead of a single, populations enables Learning Classifier Systems to produce natural solutions, overcome overlapping niche issues and clearly visualize the underlying patterns in complex domains.

LCSs produced optimal solutions are expected to contain informative patterns that can reflect the ground truth of the explored problems. However, previously the community fails in finding LCSs defined optimal solutions (i.e. [O]) in many LCS addressed domains. This leads to a need to find a new style of LCS's optimal solution that can be adapted to all LCS addressed problems.

This thesis proposed *natural solution* that complement the hypothesis of the optimal solutions of LCSs, i.e. [O]. *Natural solution* expects member rules that are consistent and subsumable under the global search space. This differs from the [O], which requires member rules are none-overlapped. Experiments in Section 3.4.5 first time demonstrated that a *natural solution* can consistently represent overlapping problems with 100% accuracy, which [O]s used to fail. Thus, *natural solution* contributes to support that LCSs can technically produce models that can fully represent overlapping domains.

An LCS optimal solution can contain thousands of different interact-

ing rules, which makes it impracticable to understand patterns by directly reading rules. This raises the requirement for developing visualization techniques that can precisely describe this underlying patterns in an LCSs' optimal solution.

Three visualization techniques, i.e. Feature Importance Map (FIM), Action-based Feature Importance Map (AFIM), and Action-based average Feature Value Map (AFVM) are proposed. FIM is easy to understand patterns, it can be employed to trace how LCSs develop patterns during the exploration phase. AFIM presents how each attribute determines the output actions. AFVM answers why a model makes a certain decision.

These visualization techniques utilize rules' generalization level to cluster rules for category rules that may carry a common pattern. This differs from previous visualization algorithms, which utilize Hamming distance to cluster rules. For the first time, the patterns captured by an LCSs' optimal solutions for Boolean domains are precisely described and the process of how LCSs develop patterns are visualized (see experiments in Section 3.4). Thus, the proposed visualization techniques contribute to improving the understanding of both LCSs produced models and LCSs themselves.

Due to the stochastic nature of the employed EC operators, LCSs unavoidably produce models that contain over-general rules and over-specific rules. These problematic rules may obscure important patterns. Thus, the LCSs field needs rule compaction algorithms that can consistently produce optimal solutions.

Four rule compaction algorithms are proposed, i.e. Razor Cluster Razor (RCR), Razor CLuster Razor 2 (RCR2), Razor Cluster Razor 3 (RCR3), and Razor Cluster Razor Real (RCR-real). RCR, RCR2, and RCR3 are Boolean domain orientated specifically for models that employ ternary representation format, whereas RCR-real is designed for real-value problems that have been addressed by upper and lower boundary representation based LCSs. RCR has the priority to produce [O] sets, and RCR2, RCR3, and RCR-real aim to produce *natural solution*s. Different from RCR2

and RCR-real, RCR3 can adapt to domains that have a serve over-general issue.

The proposed RCRs consider multiple LCSs produced models for the same problem as the source for compacting. This differs from classic compaction algorithms, which act on a single model. Furthermore, RCRs primarily considers selecting rules based on rules' consistency and unsubsumable characteristic under the global search space. As a comparison, previous compaction algorithms select rules based on a rule's contribution to either maintaining original models' capacity in prediction performance or representing the training set.

The innovation regarding the rule selection strategies results in RCRs to be the first algorithm that can consistently identify optimal solutions from LCSs addressed domains (see the experiments in Sections 4.4 and 5.4.2). Thus, RCRs contribute to improving the quality of LCSs produced models.

LCSs did not consistently produce optimal solutions for domains that contain overlapping niches. However, the majority of real-valued problems have overlapping niches. Thus, LCSs needed to be improved to adapt to overlapping domains. Hence, an LCS with layered learning architecture is proposed, i.e. Hierarchical Learning Classifier System (HLCS). HLCS is a Pittsburgh-style LCS, which considers a Michigan-style LCS as an individual. This differs from classical Pittsburgh-style LCSs, which consider a population as an individual. HLCS shows the ability to produce models that can 100% correctly represent arbitrary datasets (see experiments in Section 5.4). This contributes to extending LCSs' adaption to real-valued domains.

An LCS's search operator is proposed, i.e. Absumption. Absumption counters the over-general issue by creating new rules by correcting over-general rules and removing identified over-general rules immediately. This differs from *specify* operator, which utilizes over-general rules to produce specific rules randomly and reduce over-general rules' numeros-

ity. Experiments demonstrated that with Absumption, for the first time, a single LCS can produce the *natural solution* for overlapping domains (see Section 6.5.3). Thus, Absumption contributes to overcoming the over-general issue.

Absumption and Subsumption based Learning Classifier System (ASCS) is created. ASCS promotes the Absumption, Subsumption, and informed mutation as the primary search strategies, but removes the traditional evolutionary computation algorithms, i.e. crossover, mutation, roulette wheel deletion, and tournament selection, which differs from standard LCSs. For the first time, ASCS successfully produced [O] for 70-bits Multiplexer and detected the *natural solution* for the 14-bits Majority-On problem, which is composed of 12870 cooperative rules (see Section 7.3). ASCS contributes to enabling LCSs to address overlapping domains efficiently and allowing easy visualization.

8.1 Major Contributions

This thesis makes the following major contributions.

1 In Chapter 3, this thesis proposes *natural solution*, which describes a format of LCSs' optimal solution. A *natural solution* consists of all the consistent and maximally generalized rules under the global search space. *Natural solution* complement the [O] hypothesis by allowing overlapped rules to be member rules of an optimal solution. *Natural solution* is the first LCS's optimal solution format that can be produced by LCSs to represent overlapping domains correctly and completely. In addition a *natural solution* contains interpretable patterns.

Part of this contribution has been submitted in:

Yi Liu, Will, N. Browne and Bing Xue. "Rule Compaction Algorithms for Learning Classifier Systems". Transactions on Evolution-

ary Learning and Optimization 2020.

2 In Chapter 3, this thesis proposes three visualization techniques, i.e. Feature Important Map (FIM), Action-based Feature Importance Map (AFIM), and Action-based average Value Map (AFVM). FIM, AFIM and AFVM respectively aim to trace how patterns are constructed, reveal how features have interacted, and explain the cause of a model's decisions. For the first time, underlying patterns in an LCS's optimal solution can be precisely visualized. The proposed visualization techniques improve the understanding level of LCS's produced models and LCS themselves.

Part of this contribution has been published in:

Yi Liu, Will, N. Browne and Bing Xue. "Visualizations for Rule-Based Machine Learning". Natural Computing 2020.

3 In Chapter 4, this thesis proposes three rule compaction algorithms that are orientated to ternary representation based LCSs, Razor Cluster Razor (RCR), Razor Cluster Razor 2 (RCR2), and Razor Cluster Razor 3 (RCR3). These algorithms innovatively handle the compaction based on multiple LCS produced models, rather than applying the compaction to a single model like other standard compaction algorithms. RCR, RCR2, and RCR3 respectively aim to produce [O], *natural solution* from well-trained models, and *natural solution* from insufficiently trained models. For the first time, the proposed compaction algorithms ensure LCS's optimal solutions can be produced consistently and efficiently.

Part of this contribution has been submitted in:

Yi Liu, Will, N. Browne and Bing Xue. "Rule Compaction Algorithms for Learning Classifier Systems". Transactions on Evolutionary Learning and Optimization 2020.

Part of this contribution has been published in:

Yi Liu, Will, N. Browne and Bing Xue. "Visualisation and optimisation of learning classifier systems for multiple domain learning". Asia-Pacific Conference on Simulated Evolution and Learning. Springer, 2017. pp. 448-461.

Yi Liu, Will, N. Browne and Bing Xue. "Adapting bagging and boosting to learning classifier systems". International Conference on the Applications of Evolutionary Computation. Springer, 2018. pp. 405-420.

4 In Chapter 5, this thesis proposes Razor Cluster Razor Real (RCR-real), which is a rule compaction algorithm that applies to the upper and lower boundary representation based LCSs to produce *natural solution*. RCR-real can adapt to real-valued problems that have continuous-valued features. For the first time, optimal solutions of LCSs that addressed real-valued problems can be produced.

Part of this contribution has been published in:

Yi Liu, Will, N. Browne and Bing Xue. "Hierarchical Learning Classifier Systems for Polymorphism in Heterogeneous Niches". Australasian Joint Conference on Artificial Intelligence. Springer, 2018. pp. 397-409.

5 In Chapter 5, this thesis proposes the Hierarchical Learning Classifier System (HLCS), which introduces incremental learning and ensemble learning to a Pittsburgh-style LCS and considers a Michigan-style LCSs as an individual rather than a ruleset. HLCS is the first LCS that can produce models that can completely and correctly represent arbitrary overlapping datasets with continuous-valued features. In addition, HLCS can precisely visualize the underlying patterns of the produced models.

Part of this contribution has been published in:

Yi Liu, Will, N. Browne and Bing Xue. "Hierarchical Learning Classifier Systems for Polymorphism in Heterogeneous Niches". Australasian Joint Conference on Artificial Intelligence. Springer, 2018. pp. 397-409.

6 In Chapter 6, this thesis proposes a new search operator termed as *informed mutation* that aims to search for maximally generalized rules by correcting over-general rules. Then, based on *informed mutation*, a new mechanism named *Absumption* is proposed to enable the LCSs to produce a single model that contains all the member rules of a natural solution. Absumption successfully improves the training performance of XCSs by counteracting over-general rules. Moreover, Absumption enables the produced rule-set to be compacted, such that underlying patterns can be visualized precisely. Furthermore, because Absumption assists XCSs to address the complex overlapping domains, the optimal rules' distribution is identified so that the formulations for calculating the number of optimal rules for the Multiplexer, Majority-On, and Carry problems can be identified.

Part of this contribution has been published in:

Yi Liu, Will, N. Browne and Bing Xue. "Absumption to complement subsumption in learning classifier systems". Proceedings of the Genetic and Evolutionary Computation Conference. 2019. pp. 410-418.

7 In Chapter 7, this thesis proposes a new type of LCS that considers *natural solutions* as the search objective, i.e. absumption and subsumption based learning classifier system (ASCS). ASCS promotes the absumption, subsumption, and informed mutation as the primary search strategies and removes other obsolete evolutionary search algorithms, i.e. crossover, mutation, roulette wheel deletion, and tournament selection. ASCS enables training efficiency and easy patterns visualization for complex Boolean domains that have an overlapping distribution. For the first time, the 14-bits Majority-On prob-

lem is addressed, with 6435 different interacting co-operating rules in the optimum solution to be uniquely identified to enable correct visualization.

Part of this contribution has been published in:

Yi Liu, Will, N. Browne and Bing Xue. "Absumption and subsumption based learning classifier systems". Proceedings of the Genetic and Evolutionary Computation Conference. 2020. pp. 368-376.

8.2 Main Conclusions

This thesis finds that LCS can produce optimal solutions for clean datasets, which have overlapping distribution characteristic. Furthermore, the generated optimal solutions contain visible patterns that can reflect the ground truth of the explored datasets.

This section discusses the main conclusions for the five research objectives drawn from the five contribution chapters, i.e. Chapter 3 to Chapter 7.

8.2.1 Patterns Visualization

Chapter 3 proposed the natural solution hypothesis and three visualization techniques that allowing translating LCSs detected patterns to visible graphs.

It is found that technically LCSs can produce a natural solution format solution for an arbitrary clean dataset.

A natural solution naturally captures the patterns that reflect the ground truth of the problem domains, e.g. data distribution and attributes importance.

A natural solution consists of all the unsubduable correct rules from the global search space. One limitation of the natural solution is that collecting all the member rules is difficult when the target dataset has a huge global

search space. For example, a clean dataset contains a large number of unduplicated instances.

Results demonstrates that utilizing generalization-based clustering can efficiently cluster rules that have the same strategies for classifying the covered instances. Visualizing the specification ratio of the clustered rules can highlight the difference of each attribute's importance. Then, the underlying ground truth such as data distribution, attribute association, and irrelevant attributes become apparent in the present graph. Such graphs can improve the understanding of LCSs' produced solution and LCSs themselves. This is because the visualization technique can present the most important information in a simple graph for models, which contain thousands of different rules, and can track how LCSs form the patterns through the learning process.

There does not exist a standard visualization regulation, which normalizes the format of representing patterns with graphs. Thus, readers may fail to understand the presented patterns. Hence, the quality of the produced visualizations depends on subjective rather than objective. This results in a limitation of the proposed visualization technique, i.e., the generated visualization results require the readers to have pre-knowledge of the explored problem to ensure the graphs captured patterns to be understood appropriately.

8.2.2 Rule Compaction for Boolean Problems

Chapter 4 proposed RCR and RCR2 that respectively search [O] and natural solution from multiple LCS produced solutions for the same problem. Furthermore, a modified version of RCR2 is also proposed, i.e. RCR3, that specialized for adapting to models that have not been trained sufficiently, e.g. models that have low training accuracy.

It is found that although LCS can fully represent the same dataset with alternative rulesets, these different rulesets contain common rules, which

are the member rules of the natural solution for the target dataset. Thus, it is practicable to form the target dataset's natural solution by collecting rules from multiple LCS produced models for the same dataset.

The conducted experiments demonstrate that LCSs can only adapt to high dimensional problems, which can be represented by an [O]. It is evidenced that not all domains can be represented by an [O], but all the tested problems have a corresponding natural solution. Meanwhile, for the same dataset, there may exist alternative [O]s, but the natural solution is determinate and unique. Furthermore, if a domain has a [O], then the [O] is always a subset of the domain's natural solution.

Results show that the proposed RCRs fails to form an [O] or natural solutions when the problem domain has an overlapping distribution. This is because such domains contain good-performing over general rules, which hampers LCS from preserving correct maximally generalized rules, which are the member rules of natural solutions or [O]s.

8.2.3 Rule Compaction Algorithms for Real-Value Attribute Domains

Chapter 5 extend the RCR2 for adapting to the real-value attribute problems, i.e. the RCR-Real. Meanwhile, HLCS is proposed to produce natural solutions by exploring clean real-value attribute datasets. HLCS is based on Pittsburgh-style architecture, consider an XCS as an individual, and invokes RCR-Real during the training process rather than after training.

The thesis finds that the natural solution hypothesis also suits the clean real-value attribute problems as RCR-Real successfully producing the natural solution for all the tested UCI problems. HLCS successfully addressed tested overlapping domains, which supports the hypothesis that LCS can overcome overlapping domains with evolved models that reach 100% training accuracy. Results also show that invoking the compaction process in the training process can relieve the issue of good performing rules replace

the optimal rules.

A limitation of HLCS is that this system is inefficient. This is because HLCS increase the number of the involved XCSs to counteract the negative influence of good performing over general rules. Such a strategy can be inefficient when the target domain has a serious overlapping issue.

8.2.4 Absumption for Overlapping Domain

This thesis proposed a new search operator, i.e. informed mutation that aims to identify and remove over-general rules efficiently. Based on the informed mutation, Chapter 6 proposed the Absumption to assist LCS to address overlapping domains.

It is found that immediately remove the identified over-general rules can efficiently prevent LCS from replacing the optimal rules with good performing over-general rules. Furthermore, creating rules by specifying over-general rules can improve training efficiency. The results show that LCS can produce a natural solution for overlapping domains in a single model when the invoked Absumption can overcome the problem's overlapping issue.

One limitation of the proposed informed mutation and Absumption is that they cannot adapt to the noise domains. Furthermore, absumption fails when the target problem has a severe good performing issue, e.g., problems that naturally contain over-general rules, in which training accuracy is higher than 99%.

8.2.5 ASCS for Efficient Learning

This thesis proposed a new version of LCSs for efficiently producing natural solutions for clean datasets, i.e. the ASCS. Chapter 6 proposes ASCS, which promotes Absumption and Subsumption as the main search strategies. Meanwhile, ASCS removes the crossover, mutation and route wheel selection.

Results show that replaces the crossover and mutation with informed mutation and absumption can improve LCSs' efficiency in producing natural solutions. Furthermore, it is found that when changing LCSs' search strategies from stochastic to deterministic, LCSs can address the overlapping domains.

A limitation of ASCS is that this system cannot adapt to noise dataset. This is because a natural solution does not exist in noise datasets. Another limitation is that for complex problems, i.e. 70-bits Multiplexer problem, ASCS requires manually tune the training parameters.

8.3 Future Work

This section highlights key areas of future work.

8.3.1 Pattern Visualization on Code Fragment based LCSs

This thesis focuses mainly on visualizing patterns from LCS's models that employ representation formats, that are analogous to the Genetic Algorithms' linear binary representations, e.g. the ternary representation and the upper and lower boundary representation. This thesis has shown it is possible to visualize patterns from LCS's optimal solutions. The visualized patterns can reveal the ground truth of LCSs explored domains, e.g. feature interacting, redundant features, and class distribution. In practice, LCSs support Genetic Programming's tree-like representations, e.g. code fragment. In future, it will be interesting to investigate how to visualize patterns from code fragment based models.

8.3.2 Identify the Optimal Solution for Code Fragment based LCSs

This thesis proposed the *natural solution*, which is the first optimal solution format that can be produced by ternary representation based LCSs with the capacity of representing an overlapping domain completely and correctly. The *natural solution* has shown that contain patterns that can be translated into human-discernable graphs. Furthermore, *natural solution* has been extended to adapt to the upper and lower boundary representation based LCSs. In future, it is interesting to investigate whether code fragment based LCSs can produce a *natural solution* to represent an explored domain.

8.3.3 Adapting Deductive Learning to LCSs

Two very distinct and opposing learning approaches are deductive and inductive learning. Both approaches can offer certain advantages, but the biggest difference is the process of training. In deductive learning, a model is formed in a top-down way (generate knowledge based on an assumed truth). This approach is very solution-centred. Conversely, inductive learning is a much more data-centred approach, where learning is implemented in a bottom-up manner (search the truth based on the knowledge).

LCSs are inductive learning system, i.e. generate knowledge by exploring datasets. This thesis proposed methods, that can extract knowledge from LCSs' optimal solutions. The conducted experiments have shown that different GA-based representation formats can reveal different ground truths of an explored domain. For example, the ternary representation excels in showing feature interaction, redundant features, and class distribution. As a comparison, the upper and lower boundary provides a grained level feature ranking result. Thus, it is assumed that intelligent models can be produced by using tree-like representations to evolve functional

program based on exploring the discovered truth, which have been produced by GA based LCSs, e.g. introducing deductive learning to LCSs. In future, it is interesting to develop LCSs that allow deductive learning to produce smart models, e.g. a model can represent a Boolean domain regardless of the number of attributes.

Bibliography

[1] AENUGU, S., AND SPECTOR, L. Lexicase selection in learning classifier systems. In *Proceedings of the Genetic and Evolutionary Computation Conference* (2019), pp. 356–364.

[2] AGHILA, G., ET AL. A survey of naive bayes machine learning approach in text document classification. *arXiv preprint arXiv:1003.1795* (2010).

[3] AHMADI ABHARI, K., HAMZEH, A., AND HASHEMI, S. Voting based learning classifier system for multi-label classification. In *Proceedings of the 13th annual conference companion on Genetic and evolutionary computation* (2011), pp. 355–360.

[4] ALVAREZ, I. M., BROWNE, W. N., AND ZHANG, M. Compaction for code fragment based learning classifier systems. In *Australasian Conference on Artificial Life and Computational Intelligence* (2016), Springer, pp. 41–53.

[5] ASUNCION, A., AND NEWMAN, D. Uci machine learning repository, 2007.

[6] BACARDIT, J., AND KRASNOGOR, N. Empirical evaluation of ensemble techniques for a pittsburgh learning classifier system. In *Learning Classifier Systems*. Springer, 2006, pp. 255–268.

[7] BACARDIT, J., AND KRASNOGOR, N. Smart crossover operator with multiple parents for a pittsburgh learning classifier system. In *Proceedings of the 8th annual conference on Genetic and evolutionary computation* (2006), pp. 1441–1448.

[8] BÄCK, T., FOGEL, D. B., AND MICHALEWICZ, Z. Handbook of evolutionary computation. *Release 97*, 1 (1997), B1.

[9] BERNADÓ, E., LLORA, X., AND GARRELL, J. M. Xcs and gale: A comparative study of two learning classifier systems on data mining. In *International Workshop on Learning Classifier Systems* (2001), Springer, pp. 115–132.

[10] BERNADÓ-MANSILLA, E., AND GARRELL-GUIU, J. M. Accuracy-based learning classifier systems: models, analysis and applications to classification tasks. *Evolutionary computation 11*, 3 (2003), 209–238.

[11] BRUNZELL, H., AND ERIKSSON, J. Feature reduction for classification of multidimensional data. *Pattern Recognition 33*, 10 (2000), 1741–1748.

[12] BULL, L. *Applications of learning classifier systems*, vol. 150. Springer Science & Business Media, 2004.

[13] BULL, L. Learning classifier systems: A brief introduction. In *Applications of learning classifier systems*. Springer, 2004, pp. 1–12.

[14] BULL, L., AND ADAMATZKY, A. A learning classifier system approach to the identification of cellular automata. *Journal of Cellular Automata 2*, 1 (2007).

[15] BULL, L., BERNADÓ-MANSILLA, E., AND HOLMES, J. *Learning classifier systems in data mining*, vol. 125. Springer, 2008.

[16] BUTZ, M. V. Learning classifier systems. In *Springer Handbook of Computational Intelligence*. Springer, 2015, pp. 961–981.

[17] BUTZ, M. V., KOVACS, T., LANZI, P. L., AND WILSON, S. W. How xcs evolves accurate classifiers. In *Proceedings of the Third Genetic and Evolutionary Computation Conference (GECCO-2001)* (2001), Citeseer, pp. 927–934.

[18] BUTZ, M. V., KOVACS, T., LANZI, P. L., AND WILSON, S. W. Toward a theory of generalization and learning in xcs. *IEEE transactions on evolutionary computation 8*, 1 (2004), 28–46.

[19] BUTZ, M. V., LANZI, P. L., AND WILSON, S. W. Function approximation with xcs: Hyperellipsoidal conditions, recursive least squares, and compaction. *IEEE Transactions on Evolutionary Computation 12*, 3 (2008), 355–376.

[20] CERVANTE, L., XUE, B., ZHANG, M., AND SHANG, L. Binary particle swarm optimisation for feature selection: A filter based approach. In *2012 IEEE Congress on Evolutionary Computation* (2012), IEEE, pp. 1–8.

[21] CHANDA, P., YANG, J., ZHANG, A., AND RAMANATHAN, M. On mining statistically significant attribute association information. In *Proceedings of the 2010 SIAM International Conference on Data Mining* (2010), SIAM, pp. 141–152.

[22] DE JONG, K. Learning with genetic algorithms: An overview. *Machine learning 3*, 2-3 (1988), 121–138.

[23] DEBIE, E., AND SHAFI, K. Implications of the curse of dimensionality for supervised learning classifier systems: theoretical and empirical analyses. *Pattern Analysis and Applications 22*, 2 (2019), 519–536.

[24] DHANDE, J. D., AND DANDEKAR, D. Pso based svm as an optimal classifier for classification of radar returns from ionosphere. *International Journal on Emerging Technologies 2*, 2 (2011), 1–3.

[25] DIXON, P. W., CORNE, D. W., AND OATES, M. J. A ruleset reduction algorithm for the xcs learning classifier system. In *International Workshop on Learning Classifier Systems* (2002), Springer, pp. 20–29.

[26] DRUGOWITSCH, J., AND BARRY, A. Towards convergence of learning classifier systems value iteration.

[27] DRUGOWITSCH, J., AND BARRY, A. M. A formal framework and extensions for function approximation in learning classifier systems. *Machine Learning 70*, 1 (2008), 45–88.

[28] EGGERMONT, J., KOK, J. N., AND KOSTERS, W. A. Genetic programming for data classification: Partitioning the search space. In *Proceedings of the 2004 ACM symposium on Applied computing* (2004), pp. 1001–1005.

[29] FAHRMEIR, L., AND HAMERLE, A. Kategoriale regression in der betrieblichen planung. *Zeitschrift für Operations Research 25*, 4 (1981), B63–B78.

[30] FATIMA, M., PASHA, M., ET AL. Survey of machine learning algorithms for disease diagnostic. *Journal of Intelligent Learning Systems and Applications 9*, 01 (2017), 1.

[31] FERNÁNDEZ-DELGADO, M., SIRSAT, M., CERNADAS, E., ALAWADI, S., BARRO, S., AND FEBRERO-BANDE, M. An extensive experimental survey of regression methods. *Neural Networks 111* (2019), 11–34.

[32] FISCHER, I., AND POLAND, J. Amplifying the block matrix structure for spectral clustering. In *Proceedings of the 14th annual machine learning conference of Belgium and the Netherlands* (2005), Citeseer, pp. 21–28.

[33] FU, C., AND DAVIS, L. A modified classifier system compaction algorithm. In *Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation* (2002), Citeseer, pp. 920–925.

[34] FU, C., WILSON, S. W., AND DAVIS, L. Studies of the xcsi classifier system on a data mining problem. In *Soft Computing and Industry*. Springer, 2002, pp. 655–664.

[35] HOLLAND, J. H., BOOKER, L. B., COLOMBETTI, M., DORIGO, M., GOLDBERG, D. E., FORREST, S., RIOLO, R. L., SMITH, R. E., LANZI, P. L., STOLZMANN, W., ET AL. What is a learning classifier system? In *International Workshop on Learning Classifier Systems* (1999), Springer, pp. 3–32.

[36] HOLLAND, J. H., AND REITMAN, J. S. Cognitive systems based on adaptive algorithms. In *Pattern-directed inference systems*. Elsevier, 1978, pp. 313–329.

[37] HONG, Z.-Q., AND YANG, J.-Y. Optimal discriminant plane for a small number of samples and design method of classifier on the plane. *pattern recognition 24*, 4 (1991), 317–324.

[38] IOANNIDES, C., BARRETT, G., AND EDER, K. Xcs cannot learn all boolean functions. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation* (2011), pp. 1283–1290.

[39] IQBAL, M., BROWNE, W. N., AND ZHANG, M. Extracting and using building blocks of knowledge in learning classifier systems. In *Proceedings of the 14th annual conference on Genetic and evolutionary computation* (2012), pp. 863–870.

[40] IQBAL, M., BROWNE, W. N., AND ZHANG, M. Extending learning classifier system with cyclic graphs for scalability on complex, large-scale boolean problems. In *Proceedings of the 15th annual conference on Genetic and evolutionary computation* (2013), pp. 1045–1052.

[41] IQBAL, M., BROWNE, W. N., AND ZHANG, M. Learning overlapping natured and niche imbalance boolean problems using xcs classifier systems. In *2013 IEEE Congress on Evolutionary Computation* (2013), IEEE, pp. 1818–1825.

[42] IQBAL, M., BROWNE, W. N., AND ZHANG, M. Extending xcs with cyclic graphs for scalability on complex boolean problems. *Evolutionary computation 25*, 2 (2017), 173–204.

[43] JOSHI, J., DOSHI, R., AND PATEL, J. Diagnosis and prognosis breast cancer using classification rules. *International Journal of Engineering Research and General Science 2*, 6 (2014), 315–323.

[44] KHARBAT, F., BULL, L., AND ODEH, M. Revisiting genetic selection in the xcs learning classifier system. In *2005 IEEE Congress on Evolutionary Computation* (2005), vol. 3, IEEE, pp. 2061–2068.

[45] KHARBAT, F., BULL, L., AND ODEH, M. Mining breast cancer data with xcs. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation* (2007), pp. 2066–2073.

[46] KHARBAT, F., ODEH, M., AND BULL, L. New approach for extracting knowledge from the xcs learning classifier system. *International Journal of Hybrid Intelligent Systems 4*, 2 (2007), 49–62.

[47] KHARBAT, F., ODEH, M., AND BULL, L. A new hybrid architecture for the discovery and compaction of knowledge from breast cancer datasets. *The International Arab Journal of Information Technology 11*, 3 (2008), 116–123.

[48] KHISHE, M., MOSAVI, M., AND KAVEH, M. Improved migration models of biogeography-based optimization for sonar dataset classification by using neural network. *Applied Acoustics 118* (2017), 15–29.

[49] KIDRON, I., AND TALL, D. The roles of visualization and symbolism in the potential and actual infinity of the limit process. *Educational Studies in Mathematics 88*, 2 (2015), 183–199.

[50] KOIVISTO, M., AND SOOD, K. Exact bayesian structure discovery in bayesian networks. *Journal of Machine Learning Research 5*, May (2004), 549–573.

[51] KOVACS, T. Xcs classifier system reliably evolves accurate, complete, and minimal representations for boolean functions. In *Soft computing in engineering design and manufacturing*. Springer, 1998, pp. 59–68.

[52] KOVACS, T., AND KERBER, M. What makes a problem hard for xcs? In *International Workshop on Learning Classifier Systems* (2000), Springer, pp. 80–99.

[53] KUMAR, V., AND RATHEE, N. Knowledge discovery from database using an integration of clustering and classification. *International Journal of Advanced Computer Science and Applications 2*, 3 (2011), 29–33.

[54] KUZNETSOVA, N., WESTENBERG, M., BUCHIN, K., DINKLA, K., AND VAN DEN ELZEN, S. Random forest visualization.

[55] LANZI, P. L. An analysis of generalization in the xcs classifier system. *Evolutionary computation 7*, 2 (1999), 125–149.

[56] LANZI, P. L. *Learning classifier systems: from foundations to applications*. No. 1813. Springer Science & Business Media, 2000.

[57] LANZI, P. L. Mining interesting knowledge from data with the xcs classifier system. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)* (2001), Morgan Kaufmann San Francisco, CA 94104, USA, pp. 958–965.

[58] LANZI, P. L. Learning classifier systems: then and now. *Evolutionary Intelligence 1*, 1 (2008), 63–82.

[59] LANZI, P. L., ET AL. A study of the generalization capabilities of xcs. In *ICGA* (1997), Citeseer, pp. 418–425.

[60] LANZI, P. L., AND LOIACONO, D. Standard and averaging reinforcement learning in xcs. In *Proceedings of the 8th annual conference on Genetic and evolutionary computation* (2006), pp. 1489–1496.

[61] LANZI, P. L., AND RIOLO, R. L. A roadmap to the last decade of learning classifier system research (from 1989 to 1999). In *International Workshop on Learning Classifier Systems* (1999), Springer, pp. 33–61.

[62] LE-RADEMACHER, J., AND BILLARD, L. Symbolic covariance principal component analysis and visualization for interval-valued data. *Journal of Computational and Graphical Statistics 21*, 2 (2012), 413–432.

[63] LIU, Y., BROWNE, W. N., AND XUE, B. Adapting bagging and boosting to learning classifier systems. In *International Conference on the Applications of Evolutionary Computation* (2018), Springer, pp. 405–420.

[64] MICHALEWICZ, Z. A survey of constraint handling techniques in evolutionary computation methods. *Evolutionary programming 4* (1995), 135–155.

[65] MOSAVI, A., HOFFMANN, M., AND MILANI, A. Adapting the reactive search optimization and visualization algorithms for multiobjective optimization problems; application to geometry. In *Conference of PhD Students in Computer Science* (2012), Citeseer.

[66] NAKATA, M., KOVACS, T., AND TAKADAMA, K. A modified xcs classifier system for sequence labeling. In *Proceedings of the 2014*

Annual Conference on Genetic and Evolutionary Computation (2014), pp. 565–572.

[67] ORRIOLS, A., AND BERNADÓ-MANSILLA, E. The class imbalance problem in learning classifier systems: a preliminary study. In *Proceedings of the 7th annual workshop on Genetic and evolutionary computation* (2005), pp. 74–78.

[68] ORRIOLS-PUIG, A., AND BERNADÓ-MANSILLA, E. A further look at ucs classifier system. *GECCO* (2006), 8–12.

[69] PRATI, R. C., BATISTA, G. E., AND MONARD, M. C. Class imbalances versus class overlapping: an analysis of a learning system behavior. In *Mexican international conference on artificial intelligence* (2004), Springer, pp. 312–321.

[70] RAY, R., AND DASH, S. R. Comparative study of the ensemble learning methods for classification of animals in the zoo. In *Smart Intelligent Computing and Applications*. Springer, 2020, pp. 251–260.

[71] RUDD, J., MOORE, J. H., AND URBANOWICZ, R. J. A multi-core parallelization strategy for statistical significance testing in learning classifier systems. *Evolutionary intelligence 6*, 2 (2013), 127–134.

[72] SASIREKHA, K., AND BABY, P. Agglomerative hierarchical clustering algorithm-a. *International Journal of Scientific and Research Publications 83* (2013), 83.

[73] SATO, K., AND TAKADAMA, K. Waterbus route optimization by pittsburgh-style learning classifier system. In *SICE Annual Conference 2007* (2007), IEEE, pp. 1150–1154.

[74] SHRIVASTAVA, P., SHUKLA, A., VEPAKOMMA, P., BHANSALI, N., AND VERMA, K. A survey of nature-inspired algorithms for feature selection to identify parkinson's disease. *Computer methods and programs in biomedicine 139* (2017), 171–179.

[75] SIGAUD, O., AND WILSON, S. W. Learning classifier systems: a survey. *Soft Computing 11*, 11 (2007), 1065–1078.

[76] SMITH, R. E., AND CRIBBS III, H. B. Is a learning classifier system a type of neural network? *Evolutionary Computation 2*, 1 (1994), 19–36.

[77] SMITH, S. A learning system based on genetic algorithms. *Ph. D. Dissertation (Computer Science), Univ. of Pittsburgh* (1980).

[78] SMITH, S. F. Flexible learning of problem solving heuristics through adaptive search. In *IJCAI* (1983), vol. 83, pp. 422–425.

[79] STALPH, P. O., RUBINSZTAJN, J., SIGAUD, O., AND BUTZ, M. V. A comparative study: Function approximation with lwpr and xcsf. In *Proceedings of the 12th annual conference companion on Genetic and evolutionary computation* (2010), pp. 1863–1870.

[80] STOLZMANN, W. An introduction to anticipatory classifier systems. In *International Workshop on Learning Classifier Systems* (1999), Springer, pp. 175–194.

[81] STONE, C., AND BULL, L. For real! xcs with continuous-valued inputs. *Evolutionary Computation 11*, 3 (2003), 299–336.

[82] STREET, W. N., WOLBERG, W. H., AND MANGASARIAN, O. L. Nuclear feature extraction for breast tumor diagnosis. In *Biomedical image processing and biomedical visualization* (1993), vol. 1905, International Society for Optics and Photonics, pp. 861–870.

[83] TAKADAMA, K., NAKASUKA, S., AND SHIMOHARA, K. Robustness in organizational-learning oriented classifier system. *Soft Computing 6*, 3-4 (2002), 229–239.

[84] TAKADAMA, K., YAMAZAKI, D., NAKATA, M., AND SATO, H. Complex-valued-based learning classifier system for pomdp envi-

ronments. In *2019 IEEE Congress on Evolutionary Computation (CEC)* (2019), IEEE, pp. 1852–1859.

[85] TAN, J., MOORE, J., AND URBANOWICZ, R. Rapid rule compaction strategies for global knowledge discovery in a supervised learning classifier system. In *Artificial Life Conference Proceedings 13* (2013), MIT Press, pp. 110–117.

[86] ULTSCH, A. Data mining and knowledge discovery with emergent self-organizing feature maps for multivariate time series. In *Kohonen maps*. Elsevier, 1999, pp. 33–45.

[87] URBANOWICZ, R., GRANIZO-MACKENZIE, A., AND MOORE, J. Instance-linked attribute tracking and feedback for michigan-style supervised learning classifier systems. In *Proceedings of the 14th annual conference on Genetic and evolutionary computation* (2012), pp. 927–934.

[88] URBANOWICZ, R., AND MOORE, J. Retooling fitness for noisy problems in a supervised michigan-style learning classifier system. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation* (2015), pp. 591–598.

[89] URBANOWICZ, R., RAMANAND, N., AND MOORE, J. Continuous endpoint data mining with exstracs: A supervised learning classifier system. In *Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation* (2015), pp. 1029–1036.

[90] URBANOWICZ, R. J., ANDREW, A. S., KARAGAS, M. R., AND MOORE, J. H. Role of genetic heterogeneity and epistasis in bladder cancer susceptibility and outcome: a learning classifier system approach. *Journal of the American Medical Informatics Association 20*, 4 (2013), 603–612.

[91] URBANOWICZ, R. J., BERTASIUS, G., AND MOORE, J. H. An extended michigan-style learning classifier system for flexible supervised learning, classification, and data mining. In *International Conference on Parallel Problem Solving from Nature* (2014), Springer, pp. 211–221.

[92] URBANOWICZ, R. J., GRANIZO-MACKENZIE, A., AND MOORE, J. H. An analysis pipeline with statistical and visualization-guided knowledge discovery for michigan-style learning classifier systems. *IEEE computational intelligence magazine 7*, 4 (2012), 35–45.

[93] URBANOWICZ, R. J., AND MOORE, J. H. Learning classifier systems: a complete introduction, review, and roadmap. *Journal of Artificial Evolution and Applications 2009* (2009).

[94] URBANOWICZ, R. J., AND MOORE, J. H. The application of michigan-style learning classifiersystems to address genetic heterogeneity and epistasisin association studies. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation* (2010), pp. 195–202.

[95] URBANOWICZ, R. J., AND VARGAS, D. V. Introducing learning classifier systems: rules that capture complexity. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion* (2018), pp. 619–648.

[96] VIJAYAKUMAR, S., AND SCHAAL, S. Locally weighted projection regression: An o (n) algorithm for incremental real time learning in high dimensional space. In *Proceedings of the Seventeenth International Conference on Machine Learning (ICML 2000)* (2000), vol. 1, pp. 288–293.

[97] WADA, A., TAKADAMA, K., SHIMOHARA, K., AND KATAI, O. Learning classifier system with convergence and generalization. In *Foundations of Learning Classifier Systems*. Springer, 2005, pp. 285–304.

[98] WILSON, S. Generalization in the xcs classifier system. *Citeseer* (1998).

[99] WILSON, S. W. Classifier fitness based on accuracy. *Evolutionary computation 3*, 2 (1995), 149–175.

[100] WILSON, S. W. Get real! xcs with continuous-valued inputs. In *International Workshop on Learning Classifier Systems* (1999), Springer, pp. 209–219.

[101] WILSON, S. W. Compact rulesets from xcsi. In *International Workshop on Learning Classifier Systems* (2001), Springer, pp. 197–208.

[102] WILSON, S. W. Classifiers that approximate functions. *Natural Computing 1*, 2 (2002), 211–234.

[103] XIAO, J., XIE, L., HE, C., AND JIANG, X. Dynamic classifier ensemble model for customer classification with imbalanced class distribution. *Expert Systems with Applications 39*, 3 (2012), 3668–3675.

[104] ZHONG, P., AND FUKUSHIMA, M. Regularized nonsmooth newton method for multi-class support vector machines. *Optimisation Methods and Software 22*, 1 (2007), 225–236.

[105] ZHOU, Z.-H., AND JIANG, Y. Nec4. 5: neural ensemble based c4. 5. *IEEE Transactions on Knowledge and Data Engineering 16*, 6 (2004), 770–773.

[106] ZINTGRAF, L. M., COHEN, T. S., ADEL, T., AND WELLING, M. Visualizing deep neural network decisions: Prediction difference analysis. *arXiv preprint arXiv:1702.04595* (2017).