Soft Comput (2015) 19:1863–1880
DOI 10.1007/s00500-014-1369-7

METHODOLOGIES AND APPLICATION

Improving genetic search in XCS-based classifier systems through understanding the evolvability of classifier rules

Muhammad Iqbal · Will N. Browne · Mengjie Zhang

Published online: 11 July 2014

Abstract Learning classifier systems (LCSs), an established evolutionary computation technique, are over 30 years old with much empirical testing and foundations of theoretical understanding. XCS is a well-tested LCS model that generates optimal (i.e., maximally general and accurate) classifier rules in the final solution. Previous work has hypothesized the evolution mechanisms in XCS by identifying the bounds of learning and population requirements. However, no work has shown exactly how an optimum rule is evolved or especially identifies whether the methods within an LCS are being utilized effectively. In this paper, we introduce a method to trace the evolution of classifier rules generated in an XCS-based classifier system. Specifically, we introduce the concept of a family tree, termed *parent-tree*, for each individual classifier rule generated in the system during training, which describes the whole generational process for that classifier. Experiments are conducted on two sample Boolean problem domains, i.e., multiplexer and count ones problems using two XCS-based systems, i.e., standard XCS and XCS with code-fragment actions. The analysis of parent-trees reveals, for the first time in XCS, that no matter how specific or general the initial classifiers are, all the optimal classifiers are converged through the mechanism 'be specific then generalize' near the final stages of evolution. Populations where the initial classifiers were slightly more specific than the known 'ideal' specificity in the target solutions evolve faster than either very specific, ideal or more general starting classifier populations. Consequently introducing the 'flip mutation' method and reverting the conventional wisdom back to apply rule discovery in the match set has demonstrated benefits in binary classification problems, which has implications in using XCS for knowledge discovery tasks. It is further concluded that XCS does not directly utilize all relevant information or all breeding strategies to evolve the optimum solution, indicating areas for performance and efficiency improvement in XCS-based systems.

Keywords Learning classifier systems · XCS · XCSCFA · Genetic algorithms · Evolvability

Communicated by V. Loia.

M. Iqbal (✉) · W. N. Browne · M. Zhang
Evolutionary Computation Research Group,
School of Engineering and Computer Science,
Victoria University of Wellington,
Wellington 6140, New Zealand
e-mail: muhammad.iqbal@ecs.vuw.ac.nz

W. N. Browne
e-mail: will.browne@ecs.vuw.ac.nz

M. Zhang
e-mail: mengjie.zhang@ecs.vuw.ac.nz

1 Introduction

A learning classifier system (LCS) is a rule-based online learning system that adaptively learns a task by interacting with an unknown environment and uses evolutionary computing to evolve the rules according to the reinforcement received from the environment. The LCS technique is over 30 years old with much empirical understanding and foundations of theoretical understanding having been developed (Butz 2006; Butz et al. 2004; Drugowitsch 2008). Specifically, Butz et al. (2004) analyzed different evolutionary pressures in XCS and provided guidelines to set two main parameters in XCS (i.e., the population size and the covering probability) by considering the covering challenge and the schema challenge. All the existing LCS literature, including Butz et al. (2004), used performance curves as a measure to validate a proposed theoretical model. To the best

of our knowledge, there is no published work in the literature showing the *evolution* of an individual classifier rule generated in an LCS. In this paper, we introduce the concept of a family tree, termed *parent-tree*, to trace the evolution of classifier rules generated in an XCS-based classifier system.

XCS (Wilson 1955) is a well-tested formulation of LCS that uses accuracy-based fitness to learn the problem. Each rule in XCS is of the form 'if *condition* then *action*', having two parts: a condition and the corresponding action. Commonly, the condition is represented by a fixed-length bit-string defined over the ternary alphabet {0, 1, #} where '#' is the 'don't care' symbol which can be either 0 or 1; and the action is represented by a numeric constant. For example, in the rule '001### : 1' the condition is represented as '001###' and the action as '1'. XCS generates maximally general and accurate classifier rules in the final solution due to different evolutionary pressures (Butz et al. 2004).

XCS keeps both correct and incorrect classifiers, provided they are accurate, i.e., accurately predict the reward received from the environment. It is noted that the building blocks of information in the condition of an incorrect classifier are exactly the same as in the counterpart correct classifier. For example, '000### : 1 → 0' (where the ending '0' is the predicted payoff of the rule) is an accurate incorrect classifier which has the same condition as that in the counterpart accurate correct classifier '000### : 0 → 1000' (where the ending '1000' is the predicted payoff of the rule). The rule discovery operation is applied to the action set, which is formed by the classifiers advocating a *certain* action, commonly selected at random, and covering the currently observed environmental input. As all the classifiers in an action set advocate the same action, the correct and incorrect classifiers cannot occur in the same action set, and thus cannot be simultaneously used in breeding of the new classifiers. This means, in an XCS system, that although both correct and incorrect classifiers are kept throughout the learning of the system, the building blocks of information in them are not efficiently exploited as they are not allowed to take part in the same breeding operation.

Previously, we implemented code-fragment based XCS systems, where a code fragment is a tree-expression similar to a tree generated in genetic programming (GP) (Poli et al. 2008). The main aim of using code fragments was to identify and extract building blocks of knowledge from smaller problems in a domain and to reuse the extracted knowledge to solve complex, large-scale problems in the domain. First, we introduced code fragments in the condition of a classifier rule in XCSCFC and solved problems of a scale that existing LCS and GP approaches cannot, e.g., the 135-bit MUX problem (Iqbal et al. 2013d). Later on, we used code fragments in the action of a classifier rule in XCSCFA, which solved various complex Boolean problems (Iqbal et al. 2013c).[1] These have included the 7-bit even-parity problem, the 7-bit majority-on problem, the 4+4 bit carry problem and the DV1 problem, which are difficult to be learned using standard XCS. Investigation showed that both the rich encoding scheme and ability to breed correct/incorrect classifiers were important to the success of XCSCFA compared with XCS, but not how or to what extent each was important.

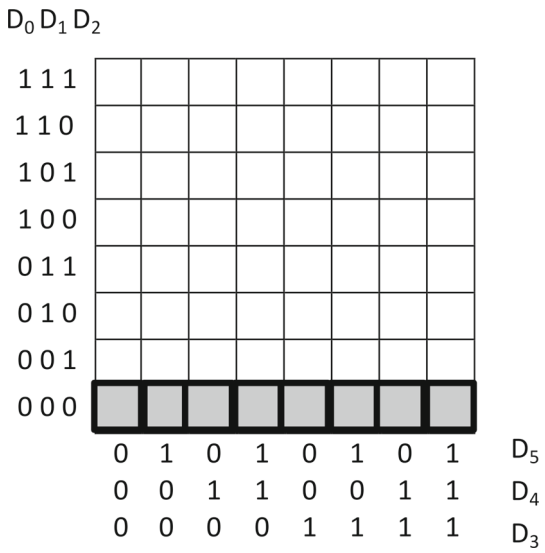


Fig. 1 The search space of a 6-bit binary problem represented in two-dimensional grid form. An exemplar target search space area is denoted by the *grey-filled cells* which is equivalent to '000###' in the ternary encoding scheme

This work aims to especially understand the evolution of optimal classifiers in both XCS and XCSCFA such that methods found to be missing in the base XCS can be added for performance improvements.

1.1 Evolving classifier rules

To aid in the visual interpretation of the evolved classifier rules and the parent-trees, a Boolean problem search space can be represented in a two-dimensional grid form. For example, the search space of a 6-bit binary problem is shown in Fig. 1. Here, D0, D1,…, D5 denote the problem input values and an exemplar target search space area is denoted by the grey-filled cells which is equivalent to '000###' in the ternary encoding scheme.

It is hypothesized that the classifier rules in an XCS are generated using different evolutionary mechanisms, which

[1] We also adopted the code-fragment action-based approach in XCSRCFA (Iqbal et al. 2012) to compute continuous actions. Further, in XCSSMA (Iqbal et al. 2013b), cyclic graphs have been incorporated in the action of a classifier rule to evolve compact solutions that could solve any size problems in a number of important domains, such as parity problems.

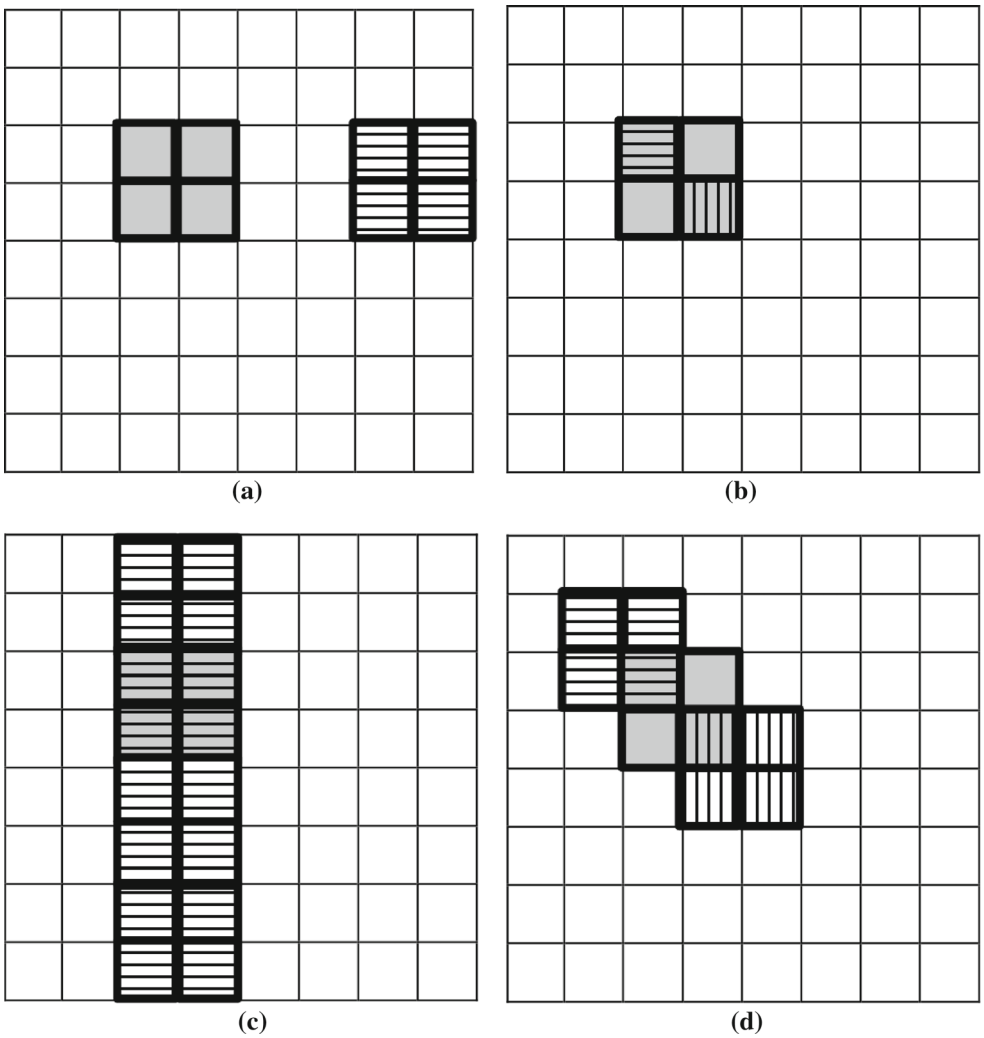


Fig. 2 The different possible mechanisms to evolve the hypothetical target classifier rule, denoted by *grey-filled cells*, in an XCS-based system. The two hypothetical parent classifiers are denoted by the cells filled with *horizontal lines* and *vertical lines*, respectively

are described as 'close-by then mutate', 'be specific then generalize', 'be general then specify', and 'overlap then recombine', see Fig. 2. In 'close-by then mutate', during the evolution of the system, a classifier rule may be evolved close to an optimum classifier, i.e., a single mutation in one dimension may produce the optimum classifier, see Fig. 2a. It is noted that such classifiers although 'close' to optimum receive zero environmental reward. In 'be specific then generalize', during the evolution of the system, some evolved classifier rules cover a sub-part of the search space covered by an optimum classifier. These rules are always accurate and correct. Then, a mutation or crossover operation on two such classifiers can make them general enough to optimally cover the whole space of the niche, see Fig. 2b. In 'be general then specify', during the evolution of the system, some evolved classifier rules cover the target search space plus extra undesired parts of the search space. These rules are not maximally accurate as they are only partially correct. These are termed over-general rules which were considered a problem in early LCSs. The mutation operation can make such classifiers specific enough to cover only the target search space, see Fig. 2c. In 'overlap then recombine', during the evolution of the system classifier rules may evolve to a form such that each of them covers a part of the search space covered by an optimum classifier. Similar to the over-general classifiers in Fig. 2c, these rules are not maximally accurate. These classifiers can be recombined by the crossover operation resulting in the optimum rule, see Fig. 2d.

1.2 Specific objectives

As stated above, the aim of the work presented here is to explore evolvability of classifier rules, specifically the optimal rules in the final solution obtained using XCS and XCSCFA such that any identified improvements in XCS can be included. The specific objectives are:

1. Verify that 'be specific then generalize' is the most common mechanism and determine what proportion alternative mechanisms take.
2. Explicitly show how an optimal rule evolves.
3. Investigate the methods used to produce these rules with various learning parameters in XCS and XCSCFA.
4. Transfer, create or discover new methods for XCS from XCSCFA.

To achieve the above contributions, it was required to analyze the evolvability of optimal classifiers. The evolution and analysis of the individual members in a population have been studied in various evolutionary computation techniques, such as canonical genetic programming (Altenberg 1994; Hemberg et al. 2013; Xie and Zhang 2013; Xie et al. 2006), linear genetic programming (Hu et al. 2012, 2013), genetic algorithms (Galván-López and Poli 2006; Hart and Ross 2001), neural networks (Stanley and Miikkulainen 2002; Wagner 2008), and gene networks (Izquierdo and Fernando 2008). Specifically, Xie et al. (2006) have used parentage information to locate ancestors of the best program generated in a GP run to reduce the fitness evaluation cost in tree-based GP systems. Stanley and Miikkulainen (2002) have used historical information for each gene in a genome to evolve neural network topologies along with weights. The resulting system, known as NeuroEvolution of Augmenting Topologies (NEAT), outperformed the best fixed-topology method by utilizing the history of genes evolution. However, to the best of our knowledge, there has been no published work in the literature to analyze and investigate the evolvability of classifier rules in an LCS, which have high elitism (steady-state) and cooperative populations.

The classifier rules in an XCS-based system routinely keep statistics regarding their performance history, which allows insight into behaviors of the classifiers. We introduce the concept of a family tree, termed *parent-tree*, for each individual classifier rule generated in the training process. The parent-tree of a classifier rule describes the whole generational history for that classifier. We also compute various statistics at each level in parent-trees to analyze the evolution of classifier rules from different aspects.

1.3 Organization

The rest of the paper is organized as follows. Section 2 describes the necessary background in XCS and XCSCFA. In Sect. 3, the concept of parent-trees is explained, which provides the platform for the novel contributions of this paper. Section 4 introduces the problem domains and experimental setup. In Sect. 5, the evolution of classifier rules produced in XCS and XCSCFA is analyzed from different aspects using the novel concept of parent-trees. In the last section, this work is concluded and the future work is outlined.

2 Learning classifier systems

Traditionally, an LCS represents a rule-based agent that incorporates evolutionary computing and machine learning to solve a given task by interacting with an unknown environment via a set of sensors for input and a set of effectors for actions (Bull and Kovacs 2005; Holland et al. 2000). After observing the current state of the environment, the agent performs an action, and the environment provides a reward. The goal of an LCS is to evolve a set of classifier rules that collectively solve the problem. The generalization property in LCS allows a single rule to cover more than one environmental state provided that the action-reward mapping is similar. Traditionally, generalization in an LCS is achieved using a special 'don't care' symbol (#) in classifier conditions, which matches any value of a specified attribute in the vector describing the environmental state. LCS can be applied to a wide range of problems including data mining, control, modeling and optimization problems (Behdad et al. 2012; Bull 2004; Shafi et al. 2009).

2.1 XCS

XCS (Wilson 1955, 1998) is a formulation of LCS that uses accuracy-based fitness to learn the problem by forming a complete mapping of states and actions to rewards. In XCS, the learning agent evolves a population $[P]$ of classifiers, where each classifier consists of a rule and a set of associated parameters estimating the quality of the rule.

Each rule in XCS is of the form 'if *condition* then *action*', having two parts: a condition and the corresponding action. Commonly, the condition is represented by a fixed-length bitstring defined over the ternary alphabet $\{0, 1, \#\}$ where '#' is the 'don't care' symbol which can be either 0 or 1; and the action is represented by a numeric constant.

Each classifier has three main parameters: (1) prediction p, an estimate of the payoff that the classifier will receive if its action is selected, (2) prediction error ϵ, which estimates the error between the classifier's prediction and the received payoff, and (3) fitness F, computed as an inverse function of the prediction error. In addition, each classifier keeps an experience parameter exp, which is a count of the number of times it has been updated, a numerosity parameter n, which is a count of the number of copies of each unique classifier, and a time stamp ts, which keeps the time-step of the last invocation of a GA on a set of classifier rules to which this classifier belonged.

XCS operates in two modes, explore (training) and exploit (application). In the following, XCS operations are concisely described. For a complete description, the interested reader is referred to the original XCS papers by Wilson (1955, 1998), and to the algorithmic details by Butz and Wilson (2002).

In the explore mode, the agent attempts to obtain information about the environment and describes it by creating the decision rules, using the following steps:

- observes the current state of the environment, $s \in S$, where S is the set of all possible environmental states. The current state s is usually represented by a fixed-length bitstring defined over the binary alphabet {0, 1}.
- selects classifiers from the classifier population $[P]$ that have conditions matching the state s, to form the match set $[M]$.
- performs covering: for every action $a_i \in A$ in the set of all possible actions, if a_i is not represented in $[M]$, a random classifier is generated with a given generalization probability $P_\#$ such that it matches s and advocates a_i, and added to the population (termed covering).[2]
- forms a system prediction array, $P(a_i)$ for every $a_i \in A$ that represents the system's best estimate of the payoff should the action a_i be performed in the current state s. Commonly, $P(a_i)$ is a fitness weighted average of the payoff predictions of all classifiers advocating a_i.
- selects an action a to explore (probabilistically or randomly) and selects all the classifiers in $[M]$ that advocated a to form the action set $[A]$.
- performs the action a, records the reward r from the environment, and uses r to update the associated parameters of all classifiers in $[A]$.
- if the average time period since the last rule discovery operation applied is greater than a preset threshold θ_{GA}, then applies an evolutionary mechanism (commonly a GA) in the action set $[A]$, to introduce new classifiers to the population. First of all, two parent classifiers are selected from $[A]$ based on fitness and the offspring are created from them. Next, the conditions of the offspring are crossed with probability χ and then each bit in the conditions is mutated with probability μ such that both offspring match the currently observed state s. After that, the actions of the offspring are mutated with probability μ.

  It is to be noted that in XCS, only two children are produced by an evolutionary operation, as opposed to typical (generational) GA and GP evolution where the whole population is replaced by the newly generated individuals. In XCS, the genetic operations are applied in sequence on two selected parent classifiers to produce two offspring, whereas in the GA and GP the genetic operations are applied in parallel on the whole population of individuals to produce the new generation of individuals that replace all the current generation.

[2] If the classifier population size grows larger than the specified limit, then one of the classifier rules has to be deleted so that the new rule can be inserted.

In addition, the explore mode may perform subsumption to merge specific classifiers into any more general and accurate classifiers. There are two subsumption procedures in XCS: (a) GA subsumption, and (b) action set subsumption. If *GA subsumption* is being used and an offspring generated by the GA has the same action as that of the parents, then its parents are examined to see if either of them: (1) has an experience value greater than a threshold, (2) is accurate, and (3) is more general than the offspring, i.e., has a set of the matching environmental inputs that is a proper superset of the inputs matched by the offspring. If this test is satisfied, the offspring is discarded and the numerosity of the subsuming parent is incremented by one. If the offspring is not subsumed by its parents, then it can be checked whether it is subsumed by other classifiers in the action set. In *action set subsumption*, any less general classifiers in an action set $[A]$ are subsumed by the most general subsumer (i.e., accurate and sufficiently experienced) classifier in the set $[A]$. Subsumption deletion is a way of biasing the genetic search towards more general, but still accurate, classifiers (Butz et al. 2004; Wilson 1998). It also effectively reduces the number of classifier rules in the final population (Kovacs 1996).

In contrast to the explore mode, in the exploit mode the agent does not attempt to discover new information and simply performs the action with the best predicted payoff. The exploit mode is also used to test learning performance of the agent in the application.

2.2 XCS with code-fragment actions (XCSCFA)

In XCSCFA (Iqbal et al. 2013a), the typically used numeric action in XCS is replaced by a GP-tree like code fragment. For simplicity, each code fragment is a binary tree and to limit the tree size a code fragment can have maximum seven nodes. The function set for the tree is problem dependent such as {AND, OR, NOT ...} for binary classification problems and $\{+, -, *, / \ldots\}$ for symbolic regression problems. The terminal set is {D0, D1, D2, …, Dn − 1} where n is the length of an environmental input message. A population of classifiers having code-fragment actions is illustrated in Fig. 3. The symbols &, |, ∼, d, and r denote AND, OR, NOT, NAND, and NOR operators, respectively. The code-fragment trees are shown in postfix form.

There are two ways to compute the action value of a classifier in XCSCFA (Iqbal et al. 2013c): (1) by loading the terminal symbols in the action tree with the corresponding binary values from the condition in the classifier rule, and (2) by loading the terminal symbols with the corresponding binary values from the environmental input; where the former is used here because it produces easily interpretable optimal rules as in standard XCS. To compute the action value of a classifier using classifier condition, a 'don't care' symbol ('#') in the condition is randomly treated as 0 or 1. Therefore,

Sr. No.	Condition						Action
	D0	D1	D2	D3	D4	D5	
1	0	0	1	#	#	0	D4D0&D2|
2	0	1	#	0	0	#	D2D5&D0D3|d
3	0	0	#	1	0	1	D0D1|D2D5&|
4	0	#	1	0	1	0	D2D0d
5	1	0	0	#	1	1	D5∼D1r
6	0	0	1	0	#	#	D3D1rD0D3&d
...	...						...

Fig. 3 Classifier population using code-fragment actions. Here &, |, d, ∼, and r denote AND, OR, NAND, NOT, and NOR operators, respectively. The code-fragment actions are shown in postfix form

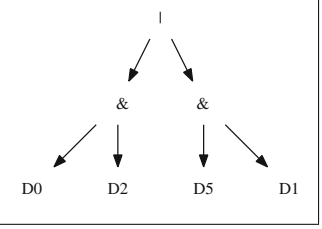
Condition						Action
D0	D1	D2	D3	D4	D5	
1	0	#	0	1	#	

Fig. 4 A classifier rule with code-fragment action

a classifier rule in XCSCFA may output different values as its action at different times, even for the same environmental input. This is termed *inconsistency* of the action value in a classifier rule, and that classifier is termed inconsistent classifier. Enabling the system to consider the inconsistency in a GP-like representation has been shown to be beneficial in certain domains (Iqbal et al. 2013c).

For example, consider the classifier rule shown in Fig. 4 and the environmental input message '101011'. In the condition of this classifier rule, D2 is a '#' symbol that will be randomly considered as 0 or 1 to compute the action value. Now, if D2 is considered as 1 then the action value will be 1 and if D2 is taken as 0 then the action value will be 0. Hence, the action value in XCSCFA is not necessarily consistent unlike standard XCS.

The XCS system keeps a complete map, i.e., the classifiers advocating consistently correct classification as well as the classifiers advocating consistently incorrect classification. A consistently incorrect classifier contains the same building blocks of information as in the counterpart correct classifier, e.g., '000### : 1' and '000### : 0', but in standard XCS these classifiers cannot occur in the same action set, and thus cannot be simultaneously used in breeding of the new classifiers. The XCS ability to keep a complete map combined with the inconsistent actions may preserve important building blocks of information in XCSCFA. Due to inconsistent action values, the incorrect classifiers can occur in the same action set as correct classifiers in XCSCFA and thus can be used for the production of good classifiers.

In XCSCFA, when the rule discovery mechanism is applied to the action set $[A]$ to produce two offspring, conditions and action trees of the offspring are created by applying GA- and GP-based genetic operations, respectively. First of all, two parent classifiers are selected from $[A]$ based on fitness and the offspring are created from them. Next, the conditions and action trees of the offspring are crossed with probability χ by applying GA- and GP-crossover operations, respectively. After that, the conditions of the resulted children by crossover are mutated with probability μ, such that both children match the currently observed state s. Then, the action trees of the children are mutated with probability μ, to replace a subtree of the action with a randomly generated subtree of depth up to 1. The interested reader is referred to Iqbal et al. (2013a) for further details of the rule discovery operation in XCSCFA.

It is to be noted that due to the multiple genotypes to a single phenotype mapping of code-fragment actions in XCSCFA, subsumption deletion is less likely to occur. Subsumption deletion is still made possible by matching the code-fragment actions on a character by character basis.

3 Parent-trees

The main purpose of the work presented here is to investigate how the optimal classifiers in the final solutions are evolved in two XCS-based classifier systems, i.e., standard XCS and XCSCFA. We introduce the concept of a parent-tree, for the first time in the field of LCS, for each classifier in the final solution, which describes the whole generational process for that classifier. In addition to the construction of parent-trees, we also compute various statistics at each level in parent-trees to analyze the evolution of classifier rules from different aspects, as will be described in Sect. 5. This section describes the concept of parent-trees using two optimal classifier rules generated in learning the 6-bit multiplexer problem.

To trace the family hierarchy of an evolved classifier rule using its parent-tree, the following additional attributes have been maintained for each classifier:

- id, which is a unique identification number for each classifier.
- gen, which is the generation number of the classifier.
- pid_1 and pid_2, which are the identification numbers of its parents (if any).
- d, which is the depth of its parent-tree.

In all the results presented here, a parent-tree of depth 0 consists of only a single node. The root (i.e., an optimal classifier) is considered at the top level in the parent-tree and

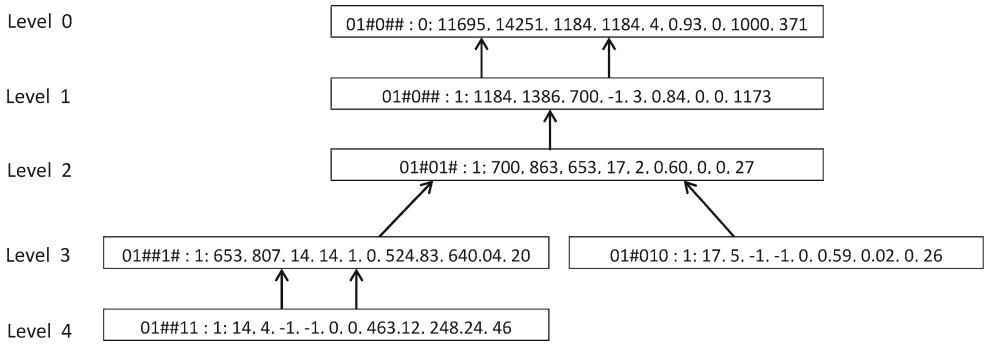


Fig. 5 A sample parent-tree for the classifier rule '01#0## : 0' generated in learning the 6-bit multiplexer problem using XCS

numbered level 0, i.e., a smaller level number represents a higher level in the parent-tree. The classifiers at the lowest level (i.e., greatest numbered parent-tree level) are the originating classifiers.

If the condition of a classifier is represented using the ternary alphabet {0, 1, #}, then an optimal classifier rule for the 6-bit multiplexer problem will contain three '#' symbols in the condition part, see Sect. 4.1 for details. A parent-tree for the optimal classifier rule '01#0## : 0' generated in learning the 6-bit multiplexer problem using XCS is depicted in Fig. 5. Each classifier rule in this parent-tree is encoded as 'condition : action; id, gen, pid_1, pid_2, d, F, ϵ, p, exp', where id, gen, pid_1, pid_2, d, F, ϵ, p, exp denote identification number, generation number, identification numbers of the parents (if any), depth, fitness, prediction error, prediction, and experience of the classifier, respectively. It is possible that the same classifier is selected as the both parents or only one parent classifier is mutated to create an offspring classifier. The identification number "-1" denotes no parent. If identification numbers for both parents of a classifier are -1, then that classifier is a newly created one during the covering operation. If the identification number for one of the parents is -1 and for the other is a positive number, then the classifier is generated from a single parent classifier using the mutation operation during the rule discovering process.

The tree in Fig. 5 highlighted three very interesting properties of the evolution within XCS-based classifier systems: (1) it is not necessary for a classifier rule to be accurate to be used in evolution of the target classifiers in the final solution set, e.g., the classifier rule '01##1# : 1' at level 3 is not accurate, but is used in production of the accurate classifier '01#01# : 1'; (2) a classifier rule generated in an earlier generation during the evolutionary process can be retained in the population until a good partner classifier is mated with it to produce a better classifier rule, e.g., the classifier rule '01#010 : 1' at level 3 was generated in the 5th generation and mated with the classifier '01##1# : 1' in generation 863 and produced the classifier '01#01# : 1'; and (3) the action part of an accurate, but incorrect classifier can be mutated in the rule discovery operation to produce the accurate and correct classifier, e.g., the action of the classifier '01#0## : 1' at level 1 was mutated to produce the correct optimal classifier '01#0## : 0'.

The evolution of the classifier rule '01#0## : a' ($a = 0$ denotes the classifier action) is shown in a 2-dimensional form in Fig. 6, where the problem search space covered by each classifier rule is represented in grids. For example, the classifier rule '01#010 : a' at level 3 covers the search space areas '010010' and '011010'. The target area is denoted by grey-filled cells and the areas covered by parent classifiers are denoted by the cells filled with horizontal lines and vertical lines. If the same classifier is selected as both parents in a crossover operation, then it is denoted by a cell filled with both horizontal and vertical lines. It is observed that all the classifiers involved in the evolution of the target classifier '11###0 : a', shown at level 0, cover a part of the target search space. Two classifiers, one at level 4, i.e., '01##11 : a', and one at level 3, i.e., '10##1# : a' additionally cover parts of the search space outside the target area, but there is no over-general classifier, i.e., a classifier that covers the whole target search space plus extra undesired parts of the search space, in the whole evolutionary process for the target classifier '01#0## : a'.

The evolution of another rule '001### : a' ($a = 1$ is the desired classifier action), generated in learning the 6-bit multiplexer problem using XCSCFA, is shown in a 2-dimensional form in Fig. 7 as it highlights important properties of XCSCFA. The target area is denoted by grey- filled cells, the areas covered by consistent parent classifiers are denoted by the cells filled with horizontal lines and vertical lines, and the areas covered by inconsistent parent classifiers are denoted by the cells filled with dashed horizontal lines and dashed vertical lines. It is interesting to note that three classifiers involved in the evolution of the target classifier '001### : a', shown at level 0, do not cover any part of the target search space, see left classifier at level 4, right classifier at level 3, and left classifier at level 2. If we consider the target search space as the 'figure' and the remaining search space area as

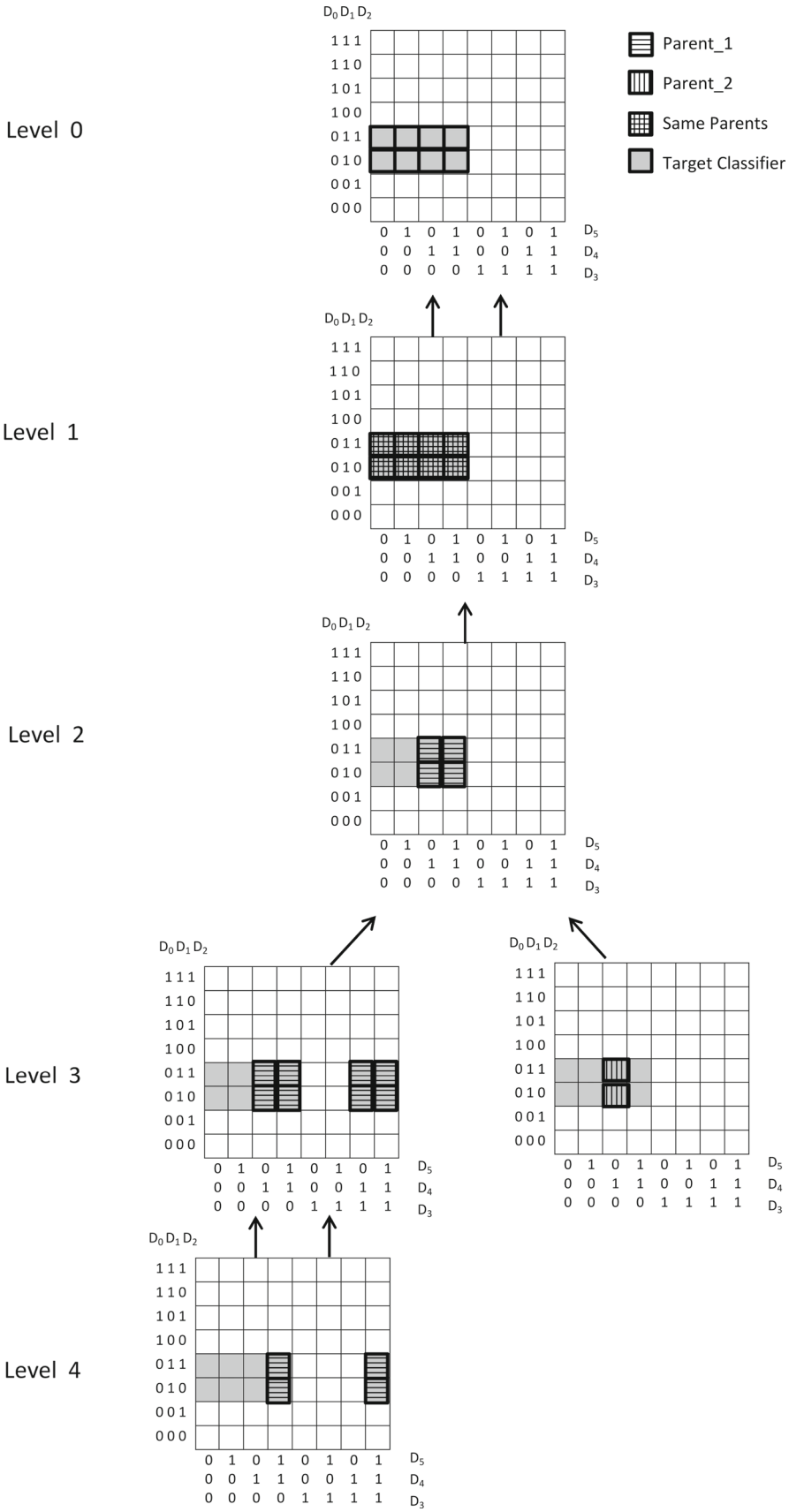


Fig. 6 A sample parent-tree, in 2-dimensional form, for the classifier rule '01#0## : *a*' generated in learning the 6-bit multiplexer problem using XCS. Here, only the conditions of classifiers are represented in a 2-dimensional form, showing the problem search space covered by each classifier; and the classifier action is denoted by *a*

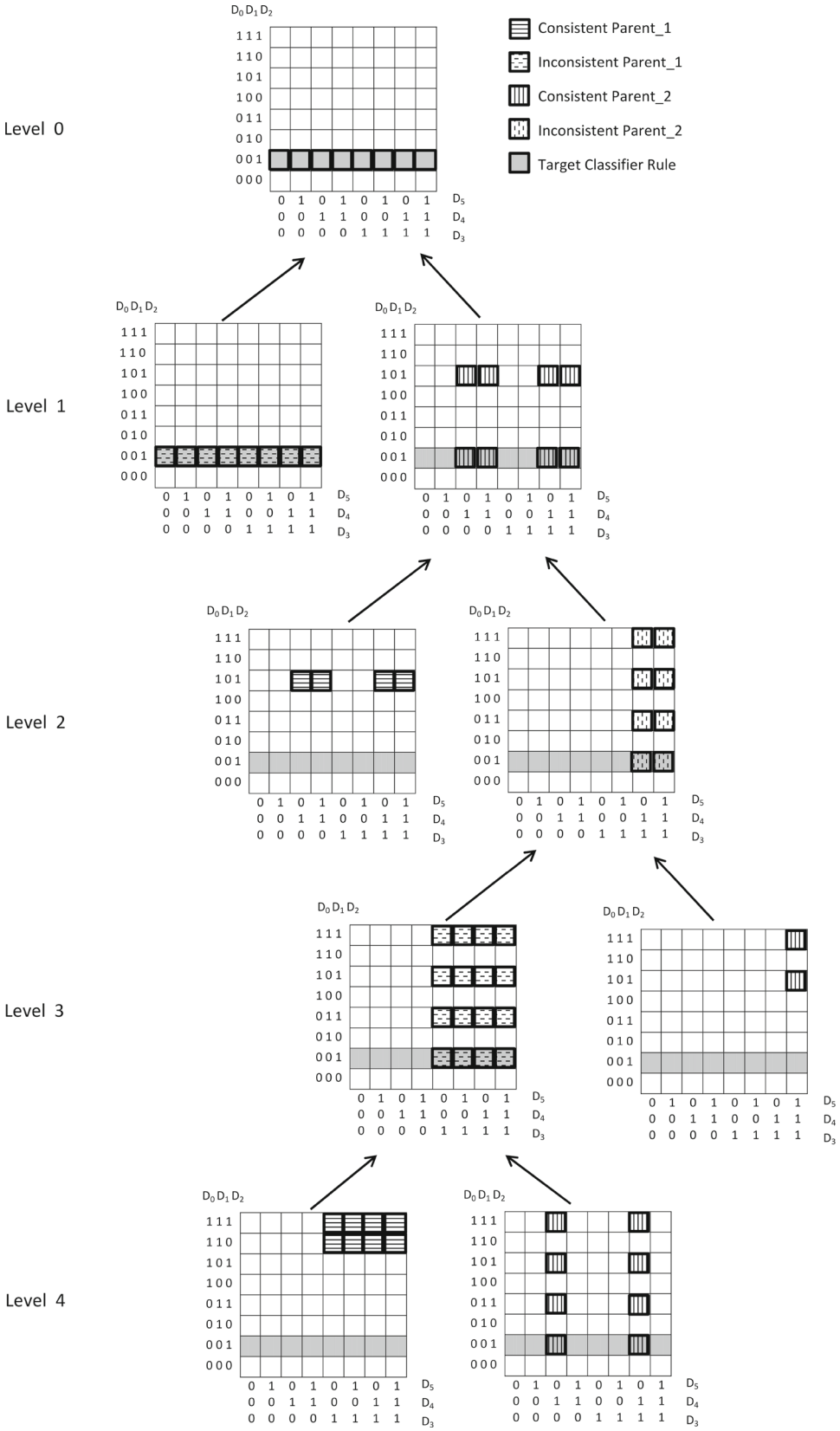


Fig. 7 A sample parent-tree, in 2-dimensional form, for the classifier rule '001### : a' generated in learning the 6-bit multiplexer problem using XCSCFA. Here, only the conditions of classifiers are represented in a 2-dimensional form, showing the problem search space covered by each classifier; and the classifier action is denoted by a

the 'ground', analogous to the 'figure and ground' terminology used in the field of object recognition (Carreira et al. 2012), then we can say that XCSCFA utilizes the 'ground' to evolve the 'figure'. For example, the two classifiers at level 4 cover only two cells out of the eight target search space cells, but the produced classifier at level 3 covers four target cells. The other property of XCSCFA highlighted in the parent-tree shown in Fig. 7 is that XCSCFA utilizes inconsistent classifiers (again the 'ground' or partial 'ground') to evolve the consistent target classifier, see left classifier at level 3, right classifier at level 2, and left classifier at level 1. It can be noted that the target classifier at level 0 is directly generated from the breeding of an inconsistent classifier with a consistent one at level 1.

4 Experimental design

This section describes the experimental design (i.e., data sets and parameter settings) to be used for the investigation of evolution and analysis of the optimal classifier rules.

4.1 The problem domains

The problems used in the experimentation are the 11-bit multiplexer problem and the 5-bit count ones problem as it is straightforward to trace the evolution of the optimal rules produced in the final solution sets in these problems. Similar trends are observed in the analysis of the much complex domains solved by these techniques, e.g., the 70-bit MUX, but the simpler domains with shorter parent-trees are clearer to present.

A multiplexer is an electronic circuit that accepts input strings of length $n = k + 2^k$, and gives one output. The value encoded by the k address bits is used to select one of the 2^k remaining data bits to be given as output. For example in the 11-bit multiplexer, if the input is 10100111010 then the output will be zero as the first three bits 101 represent the index five, which is the sixth bit following the address. Multiplexer problems are highly non-linear and have multi-modality and epistasis properties (Ioannides and Browne 2007). The optimum ternary encoded solution set for the 11-bit multiplexer problem consists of 16 maximally general, accurate and correct classifiers, shown in Table 1.

In count ones problems only k bits are relevant in an input instance of length l (Butz 2006). If the number of ones in the k relevant positions is greater than half k, the problem instance is of class one, otherwise class zero. The count ones problem used in this work is of length $l = 15$ with the first $k = 5$ relevant bits. For example, in this problem, input string '101011001011100' would be in class one, whereas input string '100101100111010' would be class zero. In the count ones problem domain, the complete solution consists of strongly overlapping classifiers, so is therefore difficult to learn (Iqbal et al. 2013c). The optimum ternary encoded solution set for the 5-bit count ones problem consists of 20

Table 1 The optimum ternary encoded rule set for the 11-bit multiplexer problem

No.	Input	Output
1	0000#######	0
2	0001#######	1
3	001#0######	0
4	001#1######	1
5	010##0#####	0
6	010##1#####	1
7	011###0####	0
8	011###1####	1
9	100####0###	0
10	100####1###	1
11	101#####0##	0
12	101#####1##	1
13	110######0#	0
14	110######1#	1
15	111#######0	0
16	111#######1	1

Table 2 The optimum ternary encoded rule set for the 5-bit count ones problem

No.	Input	Output
1	##000##########	0
2	#0#00##########	0
3	#00#0##########	0
4	#000##########	0
5	0##00##########	0
6	0#0#0##########	0
7	0#00##########	0
8	00##0##########	0
9	00#0##########	0
10	000##########	0
11	##111##########	1
12	#1#11##########	1
13	#11#1##########	1
14	#111##########	1
15	1##11##########	1
16	1#1#1##########	1
17	1#11##########	1
18	11##1##########	1
19	11#1##########	1
20	111##########	1

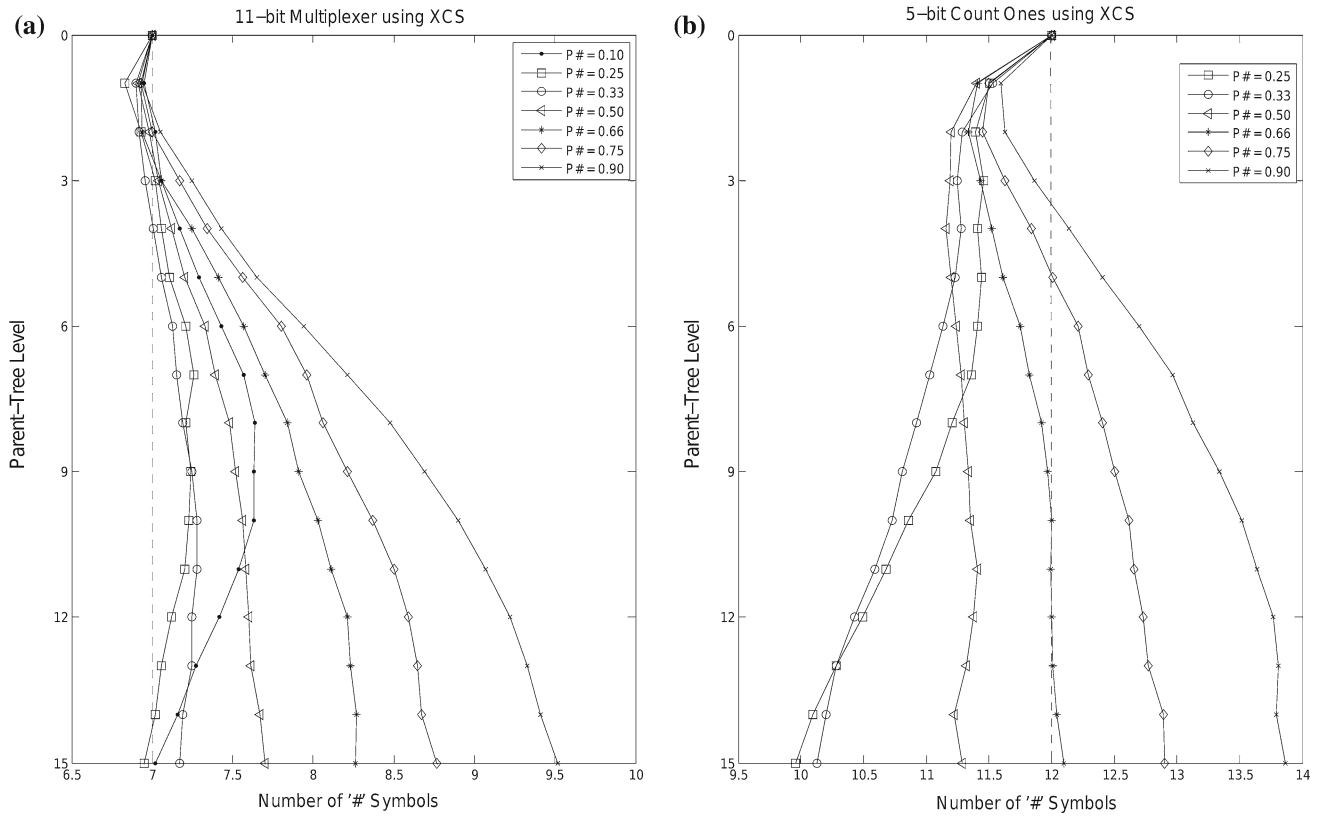


Fig. 8 The convergence of the optimal classifier rules in learning the 11-bit multiplexer problem and the 5-bit count ones problem using XCS. Only the top 15 levels in the parent-trees are shown

maximally general, accurate and correct classifiers, shown in Table 2.

4.2 Experimental setup

Unless stated otherwise, the following parameter values, commonly used in the literature, are used for the experimentation here, as suggested by Butz and Wilson (2002): learning rate $\beta = 0.2$; fitness fall-off rate $\alpha = 0.1$; fitness exponent $\nu = 5$; prediction error threshold $\epsilon_0 = 10$; two-point crossover with probability $\chi = 0.8$; mutation probability $\mu = 0.04$; fraction of mean fitness for deletion $\delta = 0.1$; experience threshold for classifier deletion $\theta_{del} = 20$; threshold for GA application in the action set $\theta_{GA} = 25$; classifier experience threshold for subsumption $\theta_{sub} = 20$; initial prediction $p_I = 10.0$; initial fitness $F_I = 0.01$; initial prediction error $\epsilon_I = 0.0$; reduction of the fitness $fitnessReduction = 0.1$; and the selection method is tournament selection with tournament size ratio 0.4. Both GA subsumption and action set subsumption are activated for the 11-bit multiplexer problem, but for the 5-bit count ones problem action set subsumption is deactivated as suggested by Iqbal et al. (2013c). The function set for the code fragments used in XCSCFA is {AND, OR, NOT, NAND, NOR}, denoted by {&, |, ∼, d, r}, for all the problem domains experimented in this work. The maximum number of classifiers used is 500 for the 11-bit multiplexer problem and 1,000 for the 5-bit count ones problem. The number of training examples used is 50,000 for all the experiments. Explore and exploit problem instances are alternated. The reward scheme used is 1,000 for a correct classification and 0 otherwise.

To get an insight of the evolvability of an evolved classifier, we have conducted seven sets of experiments with different probability values of the 'don't care' symbol '#' used in the covering operation, denoted by $P_\#$, i.e., $P_\# = 0.10, 0.25, 0.33, 0.50, 0.66, 0.75$ and 0.90. All the results presented here are average of the 30 independent runs.

5 Evolution and analysis of classifier rules

The parent-trees are analyzed to provide important insights about evolution of the optimal classifier rules. In addition to the construction of parent-trees, we also computed various statistics at each level in the parent-trees of the optimal classifier rules to analyze the evolution of rules from different aspects. The rest of this section provides detailed experiments in learning sample problems using XCS and

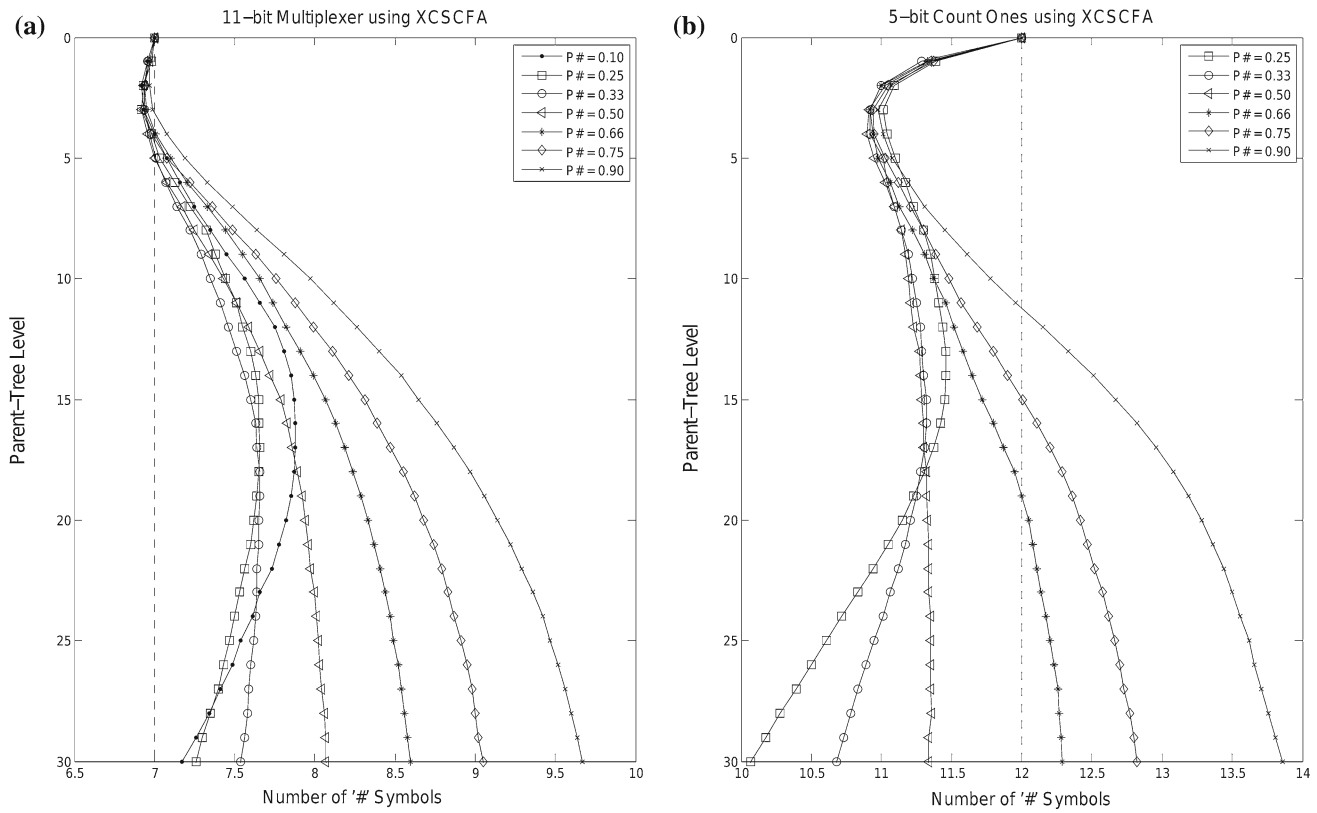


Fig. 9 The convergence of the optimal classifier rules in learning the 11-bit multiplexer problem and the 5-bit count ones problem using XCSCFA. Only the top 30 levels in the parent-trees are shown

XCSCFA that illustrate the novel analysis and rule evolution. Note that it is the *analysis* of the results that is important, not the results themselves, e.g., multiplexer problems have been solved up to 135-bits using code-fragment based classifier systems (Iqbal et al. 2013d).

5.1 The convergence of optimal classifiers

To analyze evolution of the optimal classifier rules in terms of generalization, we calculated the number of '#' symbols at each level in the parent-trees of such classifiers obtained in the final solution using XCS and XCSCFA. The convergence of the optimal classifier rules in learning the 11-bit multiplexer problem and the 5-bit count ones problem using XCS and XCSCFA is shown in Figs. 8 and 9, respectively.

It is to be noted that using $P_{\#} = 0.10$, both XCS and XCSCFA could not consistently learn the 5-bit count ones problem, therefore, the corresponding convergence curves are not presented here. It is observed that using $P_{\#} = 0.10$, XCS failed three times out of the 30 runs and XCSCFA failed 13 times, in learning the 5-bit count ones problem indicating that they were trapped in the covering/deletion loop as described by Butz et al. (2004). With $P_{\#} = 0.10$, both XCS and XCSCFA need larger populations to handle the covering challenge (Butz et al. 2004), to successfully learn the 5-bit count ones problem.

It is interesting to note that all the classifiers are converging to the final form from specific to general at the top of parent-trees, even starting with the very high generalization probability value of 0.90. This is a new discovery in XCS-based classifier systems. It is worth noting that the optimal rules are evolved from specific to general, but if the originating classifiers are too specific, i.e., $P_{\#}$ is set very small, then it may slow down the convergence of the optimal rules due to the increased parent-tree depth as shown in Figs. 10 and 11 for XCS and XCSCFA, respectively. This finding complements the theoretical models developed by Butz et al. (2004) that to converge to the optimal solution faster, it is better to use slightly specific initial classifiers than the target classifiers.

The optimal rules generated in XCSCFA have larger parent-trees than XCS, see Figs. 10 and 11, respectively, because in XCSCFA it takes longer to evolve accurate rules due to the code-fragment actions and the inconsistent classifiers. In XCSCFA, an optimal rule can directly be evolved from other optimal rules covering the same niche (i.e., having the same condition), which results in further increased parent-tree depth of the evolved optimal rule.

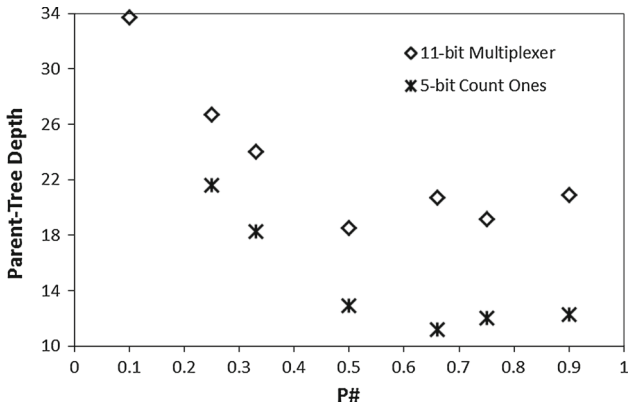


Fig. 10 The average depth of the parent-trees obtained in learning the 11-bit multiplexer problem and the 5-bit count ones problem using XCS with different values of $P_\#$

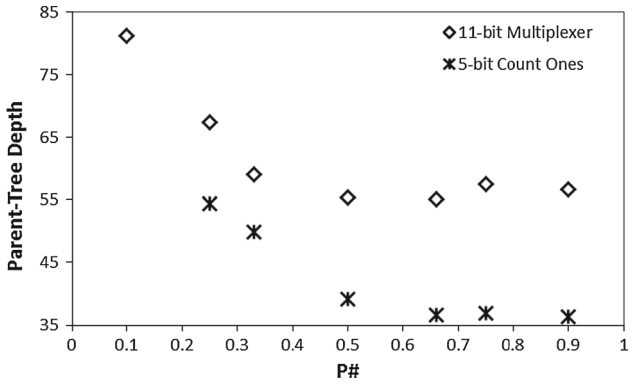


Fig. 11 The average depth of the parent-trees obtained in learning the 11-bit multiplexer problem and the 5-bit count ones problem using XCSCFA with different values of $P_\#$

5.2 The occurrence of optimal classifiers

The first occurrence of the optimal classifier rules in learning the 11-bit multiplexer problem and the 5-bit count ones problem using XCS and XCSCFA is shown in Figs. 12 and 13, respectively. All the graphs in this section are in color for better readability.

It is observed that for values of $P_\#$ near to 0, XCS was relatively slower to evolve the optimal classifier rules in learning the 11-bit multiplexer problem as well as the 5-bit count ones problem, see Fig. 12a, b, respectively. As the value of $P_\#$ increased, the optimal rules in XCS evolved more rapidly. The fastest evolution of the optimal rules in learning the 11-bit multiplexer problem and the 5-bit count ones problem using XCS achieved when $P_\#$ was 0.50 and 0.50–0.66, respectively; even though the maximally general solutions for the 11-bit multiplexer problem and the 5-bit count ones problem involve only classifiers with $7/11 \approx 0.64$ and $12/15 = 0.8$ 'don't care' symbols, respectively. When $P_\#$ was close to 1, XCS slowly evolved the optimal rules and as $P_\#$ decreased, the evolution of the optimal rules improved again. This finding is consistent with Butz et al. (2004), where it was theorized that slightly more specific classifiers than the target classifiers speed convergence.

Similar to XCS, for values of $P_\#$ near to 0, XCSCFA was relatively slower to evolve the optimal classifier rules in learning the 11-bit multiplexer problem, see Fig. 13a, and as the value of $P_\#$ increased, the optimal rules in XCSCFA evolved more rapidly. The fastest evolution of the optimal rules in learning the 11-bit multiplexer problem using XCSCFA achieved when $P_\#$ was 0.33, indicating that XCSCFA prefers slightly smaller $P_\#$ values than XCS as smaller $P_\#$ values help in generating more consistent code-fragment actions as compared to larger $P_\#$ values. Similarly, when $P_\#$ was close to 1, XCSCFA slowly evolved the optimal rules and as $P_\#$ decreased, the evolution of the optimal rules improved again. Surprisingly, in learning the 5-bit count ones problem using XCSCFA, the evolution of the optimal rules is almost similar for any $P_\#$ value, with 0.25 being slightly fast, as shown in Fig. 13b. This may be due to the overlapping property of the count ones problem domain. This will be further investigated in future work.

The comparison of Figs. 12 and 13 revealed that the evolution of the optimal rules in XCS is earlier and faster than XCSCFA. For example, with $P_\#$ equal to 0.25, the first optimal rule in learning the 5-bit count ones problem using XCS and XCSCFA occurred at generation number ≈2,000 and 17,000, respectively. Similarly, the first ten optimal rules in learning the 5-bit count ones problem, with $P_\#$ equal to 0.25, using XCS were evolved within ≈2,000 generations, whereas XCSCFA took ≈27,000 generations to evolve the first ten rules in this case. The reason for the slow evolution of the optimal rules in XCSCFA is again the code-fragment actions, the inconsistent classifiers, and the multiple genotypes to a single phenotype mapping of code-fragment actions as described in Sect. 5.1.

It is to be noted that the evolution of the optimal rules in learning the 11-bit multiplexer problem and the 5-bit count ones problem using XCS was completed in less than 40,000 generations and 14,000 generations, see Fig. 12a, b, respectively. However, in XCSCFA, the optimal rules were found to be evolved continually until 50,000 generations, i.e., at the end of the training in learning the 11-bit multiplexer problem as well as the 5-bit count ones problem, see Fig. 13a, b, respectively. This is because in XCSCFA the number of the optimal rules is greater than XCS due to the multiple genotypes to a single phenotype mapping of code-fragment actions in XCSCFA, e.g., XCSCFA evolved more than 60 genotypic optimal rules in learning the 11-bit multiplexer problem, see Fig. 13a, instead of the only 16 required phenotypic optimal rules shown in Table 1.

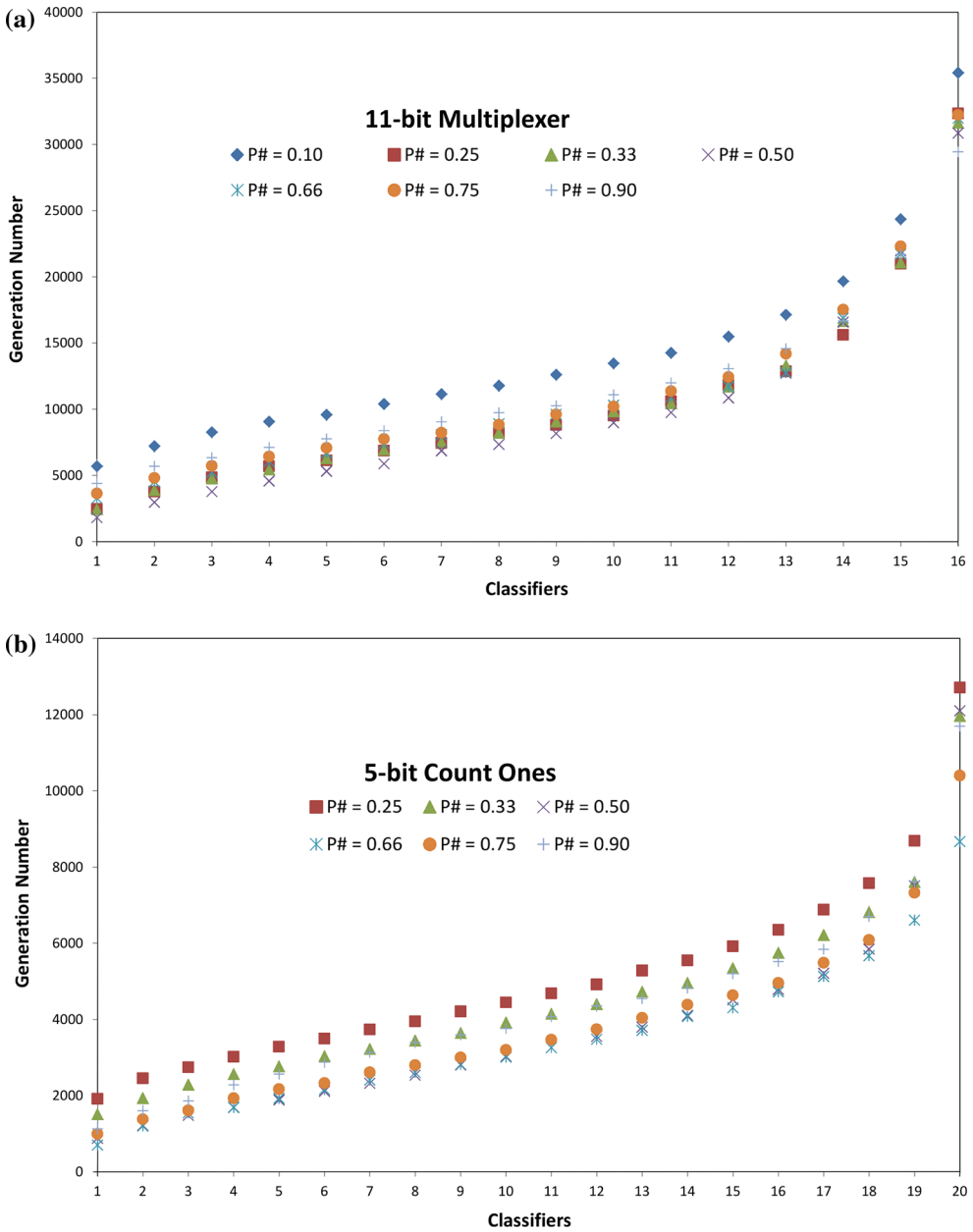


Fig. 12 The first occurrence of the optimal classifier rules in learning the 11-bit multiplexer problem and the 5-bit count ones problem using XCS

5.3 Transferring identified mechanisms to XCS

XCSCFA has classification performance improvements over XCS as the problems increase to more complex scales (Iqbal et al. 2013c), but requires more iterations and more resources than XCS due to the richness of the code fragments. The next step of the work was to consider transferring the identified mechanisms that assist XCSCFA to XCS.

In an XCS-based system, the rule discovery operation is conventionally applied to the action set that is formed by the classifiers advocating a *certain* action. In the explore mode, this action is commonly selected at random. All the classifiers in an action set advocate the same action and the mutation operation to change the action of a newly produced child classifier is applied with a probability μ, which has the commonly used value of 0.04. Therefore, 96 % of the newly produced children in standard numeric action-based XCS have the same action value as that of the parent classifiers. It means that, in an XCS-based system, although both correct and incorrect classifiers are kept throughout the learning of the system, the building blocks of information in them are not efficiently exploited as they are not allowed to take part in the same breeding operation.

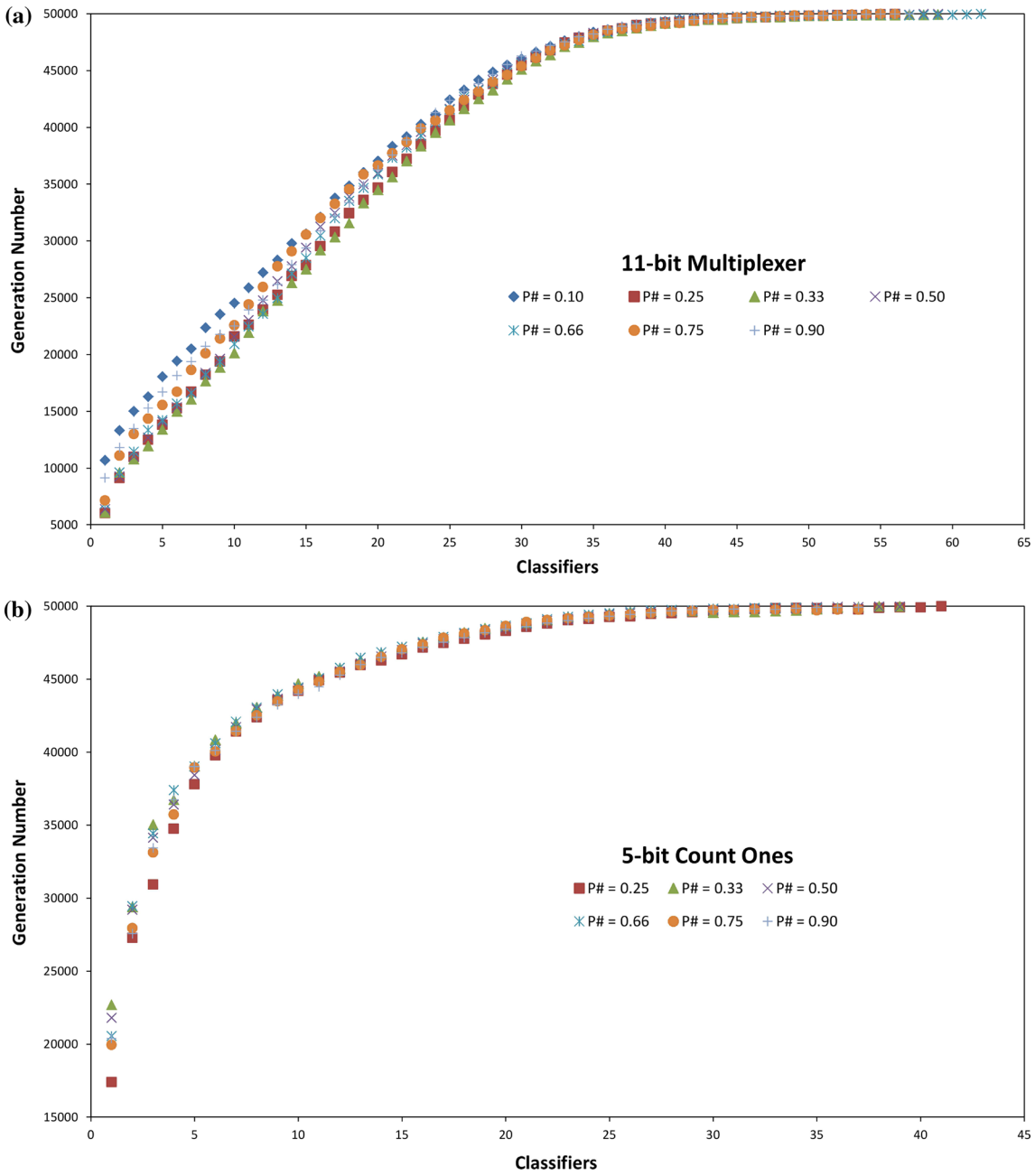


Fig. 13 The first occurrence of the optimal classifier rules in learning the 11-bit multiplexer problem and the 5-bit count ones problem using XCSCFA

In XCSCFA, due to the inconsistent action values, it is possible that an action set consists of classifiers with the potential for different actions. It can be seen from the parent-tree shown in Fig. 7 that XCSCFA enables both incorrect and correct classifiers to breed together. Therefore, the rule discovery operation in XCSCFA can exploit the building blocks of information more efficiently as compared to standard XCS. Fortunately, in an XCS-based system, both correct and incorrect classifiers exist in the same match set formed against the currently observed environmental input message. In the original XCS paper by Wilson (1955), the rule discovery was applied to the match set, but later it was moved to the action set to reduce the number of inaccurate classifiers produced by the system in certain domains (Wilson 1998).

It is hypothesized that if the rule discovery operation is applied to the match set, building blocks of information can be efficiently used for the evolution of potentially good classifiers in learning the classification problems that have the right symmetries, e.g., multiplexer problems. In addition, for binary classification problems, the action of an incorrect rule

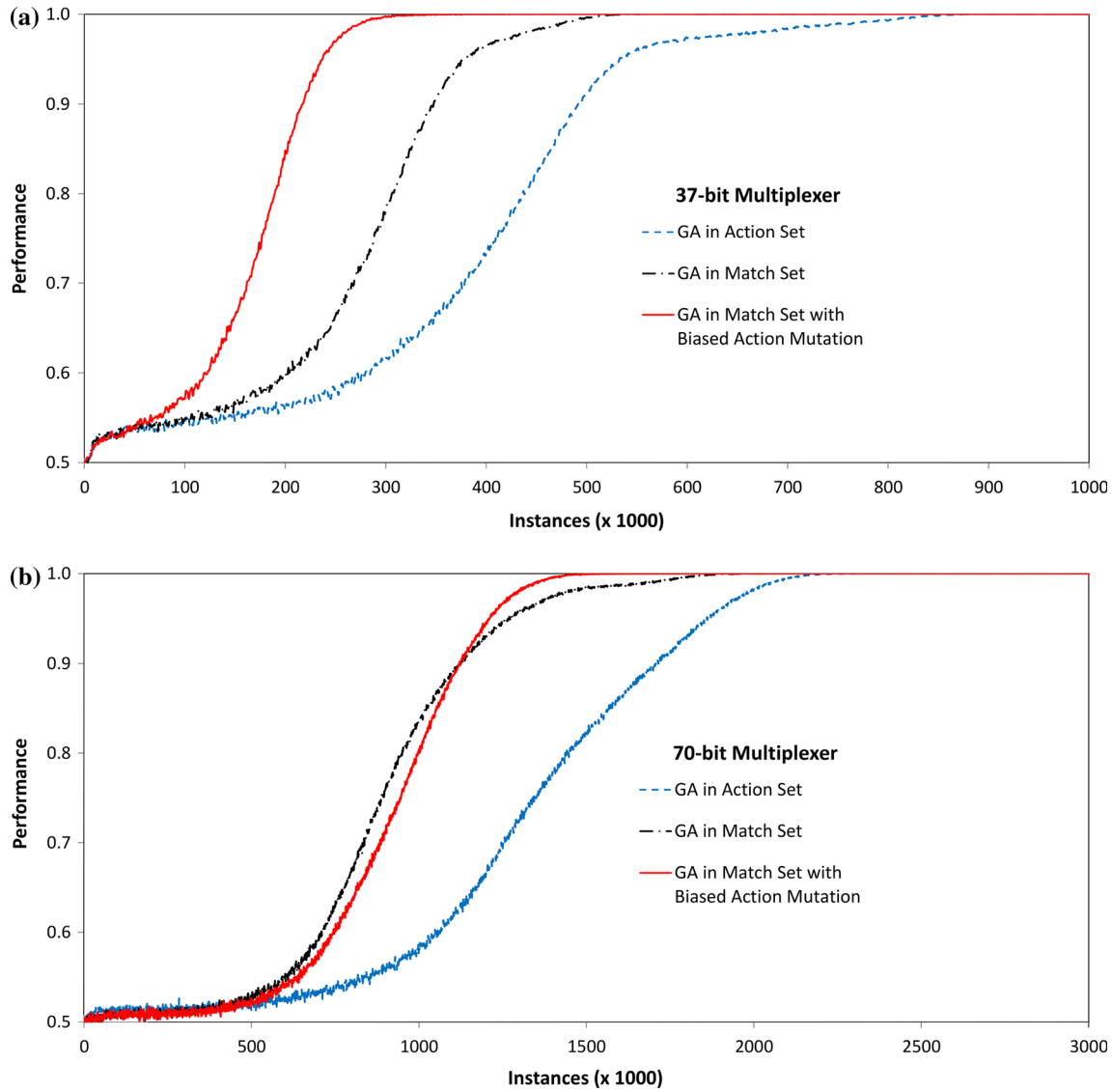


Fig. 14 Performance comparison in learning multiplexer problems using XCS. Please observe the scale differences for the figures

can be explicitly flipped in the mutation operation resulting in further improved performance.

To test this hypothesis, we conducted experiments on the 37-bit multiplexer and the 70-bit multiplexer problems using three different configurations in XCS: (1) applying GA in the action set, (2) applying GA in the match set, and (3) applying GA in the match set but using biased action mutation where the action of an incorrect but accurate rule is explicitly flipped in the mutation operation. The maximum number of classifiers used is 5,000 and 20,000 and the number of training examples used is one million and five million for the 37-bit multiplexer problem and the 70-bit multiplexer problem, respectively. The value of $P_{\#}$ used is 0.5 and 1.0 for the 37-bit multiplexer problem and for the 70-bit multiplexer problem, respectively. The mutation probability μ for the 70-bit multiplexer problem is reduced from 0.04 to 0.01 as suggested by Butz (2006). For a fair comparison, the action set subsumption is deactivated.

The performance comparison in learning the 37-bit multiplexer problem and the 70-bit multiplexer problem using three different configurations in XCS is shown in Fig. 14. As anticipated, applying GA in the match set improved the performance of XCS in terms of reducing the number of input instances required to reach the 100 % performance level. The biased action mutation further reduced the number of input instances required to reach the 100 % performance level.

Figure 15 shows that applying GA in the match set produced more condensed solutions than applying GA in the action set in learning the 37-bit multiplexer and the 70-bit multiplexer problems. The biased action mutation further reduced the population size in the final solution in learning the 37-bit multiplexer problem, see Fig. 15a.

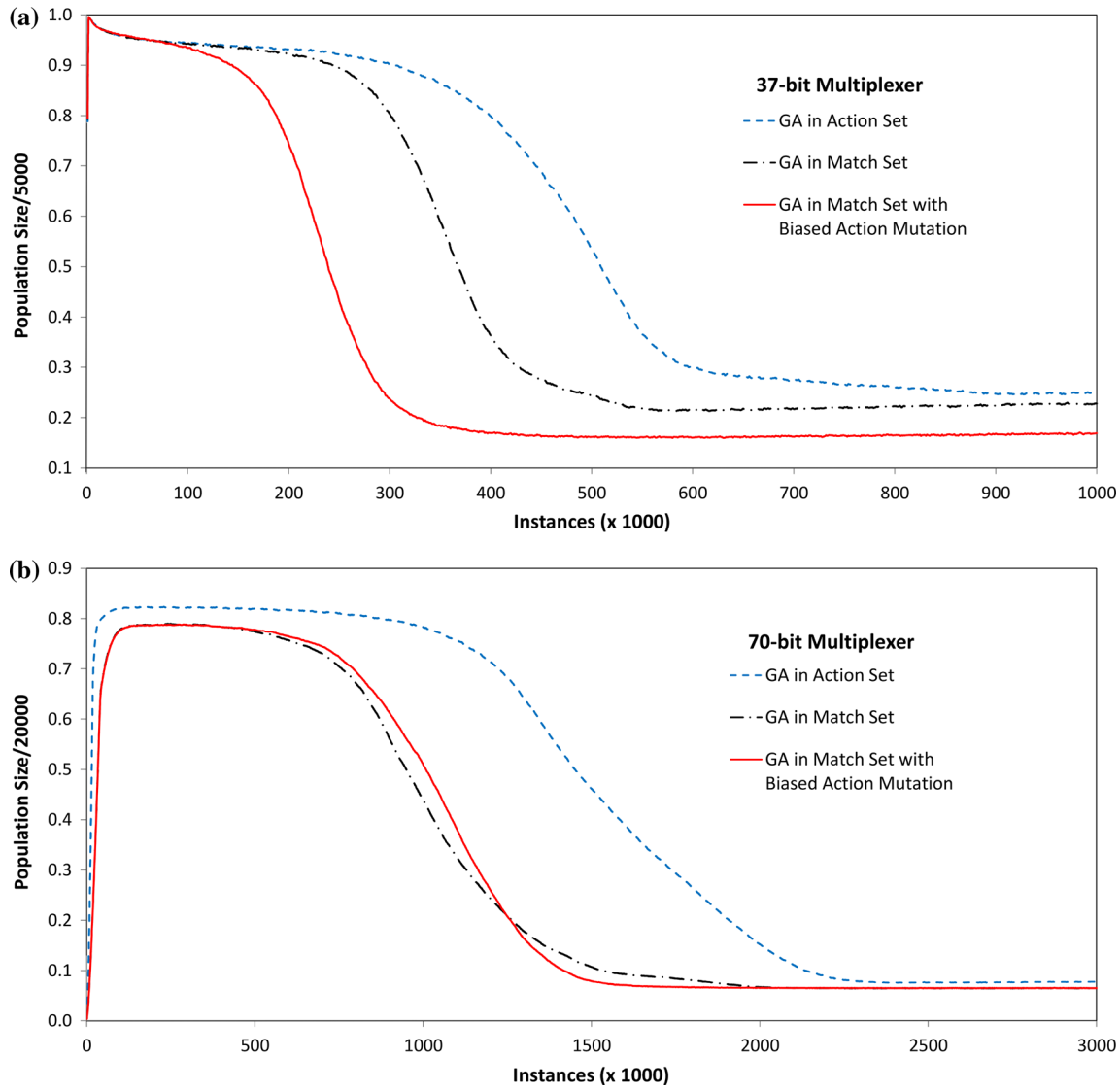


Fig. 15 Population size comparison in learning multiplexer problems using XCS. Please observe the scale differences for the figures

However, for multi-class problems and sequential problems, it is beneficial to keep the rule discovery operation in the action set because they do not have the right symmetries to exploit the building blocks of information in the complete map as described by Wilson (1998).

6 Conclusions

The aim of this work was to understand the evolution of the optimal rules in XCS-based classifier systems, in terms of evolutionary methods especially with regard to generalization (specificity) of rule coverage and how this develops. We created parent-trees for each of the optimal classifier rules in the final solution obtained using different initial generalization probability values, denoted by $P_{\#}$, and observed the following:

- The optimal classifiers are generated using different mechanisms such as 'overlap then recombine', 'close-by then mutate', but mostly 'be specific then generalize'.
- The optimal classifiers in all the experiments converged to the final form by becoming specialized at the top levels of the parent-trees, even using the very large $P_{\#}$ value of 0.9. This is a new discovery in XCS-based systems. This finding complements the theoretical models developed by Butz et al. (2004) that to converge to the optimal solution faster, it is better to use relatively specific initial classifiers as compared to anticipated generality in the final solution.
- A very small or very large $P_{\#}$ value resulted in larger parent-trees, which provides a reason for a slower rate of convergence to the optimum population as modeled by Butz et al. (2004).

- By applying the rule discovery operation to the match set, building blocks of information can be efficiently used for the evolution of potentially good classifiers in learning the classification problems that have the right symmetries. In addition, for binary classification problems, the action of an incorrect rule can be explicitly flipped in the mutation operation resulting in further improved performance.

These findings support and complement the theoretical models developed by Butz et al. (2004).

This work has identified that XCSCFA exploits the building blocks of information more efficiently than XCS by enabling both incorrect and correct classifiers to breed together. On the other hand, XCS does not directly use all relevant information or breeding strategies, which offers areas for performance and efficiency improvement in XCS-based systems. An initial step has been to introduce biased mutation to transfer incorrect to correct classifiers in binary domains that is an obvious step, but not previously adopted in the community. Similarly, the action set has been widely adapted, but for binary classification problems the match set is considered more beneficial, especially when coupled with biased mutation.

Further improvements are suggested, such as creating 'generalized to specific' operators because currently an overgeneral rule produced in an XCS-based system gets removed rather than specified by the system. This mechanism needs to be revisited. In multi-class problems, the use of 'figure and ground' needs to be considered.

The application of parent-trees to analyze the evolution of optimal solutions in alternative population-based evolutionary algorithms is considered worthy of further investigation.

References

Altenberg L (1994) The evolution of evolvability in genetic programming. In: Advances in genetic programming. MIT Press, Massachusetts, pp 47–74

Behdad M, Barone L, French T, Bennamoun M (2012) On XCSR for electronic fraud detection. Evol Intell 5(2):139–150

Bull L (2004) Applications of learning classifier systems. Springer, Heidelberg

Bull L, Kovacs T (2005) Foundations of learning classifier systems: an introduction. Springer, Berlin

Butz MV (2006) Rule-based evolutionary online learning systems: a principal approach to lcs analysis and design. Springer, Berlin

Butz MV, Kovacs T, Lanzi PL, Wilson SW (2004) Toward a theory of generalization and learning in XCS. IEEE Trans Evol Comput 8(1):28–46

Butz MV, Wilson SW (2002) An algorithmic description of XCS. Soft Comput 6(3–4):144–153

Carreira J, Li F, Sminchisescu C (2012) Object recognition by sequential figure-ground ranking. Int J Comput Vision 98(3):243–262

Drugowitsch J (2008) Design and analysis of learning classifier systems: a probabilistic approach. Springer, Berlin

Galván-López E, Poli R (2006) An empirical investigation of how and why neutrality affects evolutionary search. In: Proceedings of the genetic and evolutionary computation conference, pp 1149–1156

Hart E, Ross P (2001) GAVEL—a new tool for genetic algorithm visualization. IEEE Trans Evol Comput 5(4):335–348

Hemberg E, Berzan C, Veeramachaneni K, O'Reilly UM (2013) Introducing graphical models to analyze genetic programming dynamics. In: Proceedings of the twelfth workshop on foundations of genetic algorithms, pp 75–86

Holland JH, Booker LB, Colombetti M, Dorigo M, Goldberg DE, Forrest S, Riolo RL, Smith RE, Lanzi PL, Stolzmann W, Wilson SW (2000) What is a learning classifier system? In: Learning classifier systems, from foundations to applications. Springer, New York, pp 3–32

Hu T, Banzhaf W, Moore JH (2013) Robustness and evolvability of recombination in linear genetic programming. In: Genetic programming. Springer, New York, pp 97–108

Hu T, Payne JL, Banzhaf W, Moore JH (2012) Evolutionary dynamics on multiple scales: a quantitative analysis of the interplay between genotype, phenotype, and fitness in linear genetic programming. Genetic Program Evolv Mach 13(3):305–337

Ioannides C, Browne WN (2007) Investigating scaling of an abstracted LCS utilising ternary and S-expression alphabets. In: Proceedings of the genetic and evolutionary computation conference, pp 2759–2764

Iqbal M, Browne WN, Zhang M (2012) XCSR with computed continuous action. In: Proceedings of the Australasian joint conference on artificial intelligence, pp 350–361

Iqbal M, Browne WN, Zhang M (2013a) Evolving optimum populations with XCS classifier systems. Soft Comput 17(3):503–518

Iqbal M, Browne WN, Zhang M (2013b) Extending learning classifier system with cyclic graphs for scalability on complex, large-scale boolean problems. In: Proceedings of the genetic and evolutionary computation conference, pp 1045–1052

Iqbal M, Browne WN, Zhang M (2013c) Learning complex, overlapping and niche imbalance Boolean problems using XCS-based classifier systems. Evol Intell 6(2):73–91

Iqbal M, Browne WN, Zhang M (2013d) Reusing building blocks of extracted knowledge to solve complex, large-scale Boolean problems. IEEE Trans Evol Comput. doi:10.1109/TEVC.2013.2281537

Izquierdo EJ, Fernando CT (2008) The evolution of evolvability in gene transcription networks. Artif Life 11:265–273

Kovacs T (1996) Evolving optimal populations with XCS classifier systems. Technical Report CSR-96-17 and CSRP-9617, University of Birmingham, UK

Poli R, Langdon WB, McPhee NF (2008) A field guide to genetic programming. http://lulu.com and freely available at http://www.gp-field-guide.org.uk (with contributions by J. R. Koza)

Shafi K, Kovacs T, Abbass HA, Zhu W (2009) Intrusion detection with evolutionary learning classifier systems. Nat Comput 8(1):3–27

Stanley KO, Miikkulainen R (2002) Evolving neural networks through augmenting topologies. Evol Comput 10(2):99–127

Wagner A (2008) Robustness and evolvability: a paradox resolved. Proc R Soc B 275(1630):91–100

Wilson SW (1995) Classifier fitness based on accuracy. Evol Comput 3(2):149–175

Wilson SW (1998) Generalization in the XCS classifier system. In: Proceedings of the genetic programming conference, pp 665–674

Xie H, Zhang M (2013) Parent selection pressure auto-tuning for tournament selection in genetic programming. IEEE Trans Evol Comput 17(1):1–19

Xie H, Zhang M, Andreae P (2006) A study of good predecessor programs for reducing fitness evaluation cost in genetic programming. In: Proceedings of the IEEE congress on evolutionary computation, pp 2661–2668