Visualisation and Optimisation of Learning Classifier Systems for Multiple Domain Learning

Yi Liu(✉), Bing Xue, and Will N. Browne

Victoria University of Wellington, Wellington, New Zealand 6014
liuyi4@myvuw.ac.nz

Abstract. Learning classifier system (LCSs) have the ability to solve many difficult benchmark problems, but they have to be applied individually to each separate problem. Moreover, the solutions produced, although accurate, are not compact such that important knowledge is obscured. Recently a multi-agent system has been introduced that enables multiple, different LCSs to address multiple different problems simultaneously, which reduces the need for human system set-up, recognises existing solutions and assigns a suitable LCS to a new problem. However, the LCSs do not collaborate to solve a problem in a compact or human observable manner. Hence the aim is to extract knowledge from problems by combining solutions from multiple LCSs in a compact manner that enables patterns in the data to be visualised. Results show the successful compaction of multiple solutions to a single, optimum solution, which shows important feature knowledge that would otherwise have been hidden.

Keywords: Learning classifier systems · Multiple domain learning

1 Introduction

Learning Classifier Systems (LCSs) are a family of evolutionary computation techniques that evolve a population of 'condition-action' heuristics that cooperate to address classification tasks. They have the ability to provide solutions in domains exhibiting epistasis (e.g. non-linear feature interaction) and heterogeneity (e.g. different combinations of features producing the same class). The transparent nature of the heuristic (rule) format enables humans to interrogate the knowledge discovered.

LCSs have been shown to be accomplished at solving many different and difficult problem domains [11], such as salient object detection, game policy design, patient care and so forth [16]. However, as with many artificial intelligence approaches, they require humans to carefully set the parameters for each pre-selected problem domain. Furthermore, the solutions produced, albeit accurate, often require post-processing to obtain the most compact (condensed) solution. This reduces the transparency of knowledge to human users, although the additional time for processing and memory storage for the non-condensed solution is minimal nowadays.

Recently, the advances in fast and large memory computation have enabled system comprising of multiple LCSs to be realised [13]. This has the advantage of allowing multiple LCS setups to coexist so that one suited to an individual problem can be selected autonomously and quickly. Moreover, problem types can be identified, best matches to past solutions determined and new problems assigned to appropriate LCS setups. However, the problem remains of storing the best solution to a problem in a compact and human understandable manner.

The research question addressed here is whether each solution to one problem from multiple different setup LCSs can be combined into a single solution, improving both the understanding (i.e. visualisation) and compactness (i.e. condensation). The objectives are to generate novel algorithms/methods to combine multiple LCS populations together into a single solution for the first time. Secondly, to design methods for human understanding of the knowledge from multiple LCS solutions in order to aid transparency of the knowledge discovered.

LCSs have been applied to many types of problem including Boolean, integer, real-valued and categorical encoded domains. This can be coupled with either supervised or reinforcement learning and instantaneous or delayed reward environmental interaction. As an initial investigation of the novel methods, Boolean domains are selected as the solutions are well understood, varied and straightforward to visually judge the worth of evolved solutions. Stimulus-response (i.e. instantaneous) learning is often used in knowledge discovery tasks so is appropriate here. As the multi-agent system can contain both supervised and reinforcement learning based LCSs, reinforcement learning was selected as arguably the more complex problems due to the absence of a known class. It is worth noting that although the system has been tested up to and including 70-bit problems, only 6-bit results are presented due to space restrictions (2^6 vs. 2^{70} long optimal solutions) and (6 feature wide graphs vs. 70 wide graphs).

2 Background

Learning Classifier Systems (LCSs) produce cooperative solutions of 'if condition then action' rules as specified by the Michigan approach adopted here [16]. Evolutionary search is used for global exploration for the best forms of solution for an environmental niche (an area of solution space that has common characteristics). While reinforcement learning is used to fine tune the parameter values, such as rule fitness, in order to guide this local search. When an environmental state is passed to an LCS it searches its population of rules to determine while rules match this input in order to form the match set $[M]$. Initially, $[M]$ may be empty, such that the *covering* method is called to generate a rule with generality to cover the input space.

Not all rules will match an input, hence the need for a cooperating population. Alternating between deterministic and guided stochastic selection an action (e.g. proposed class of data) will be selected from those advocated in $[M]$. All rules (termed classifiers when supporting parameters are included) advocating this action are placed in the action set $[A]$. Once the action is effected on the environment (e.g. the proposed class is checked for appropriateness) then the parameters of classifiers in $[A]$ are updated through reinforcement learning (e.g. a recency-weighted filter or a Q-learning like policy update for multistep problems). Once the classifiers in a niche have been tested sufficiently to obtain their relative worth to the system an evolutionary search algorithm is used, often in this niche (e.g. only classifiers currently in $[A]$ are evolved).

This project is based on two assumptions. Firstly, LCS can extract information from Boolean problems [9], which is plausible from previous results [12]. According to Iqbal [6–8,10], reinforcement based LCS [2], e.g. XCS [18], can solve many complex Boolean problems, but they often express the target solution in a rich way making knowledge hard to observe. Secondly, the solutions can be non-optimal if the parameters are poorly set-up, such that different partially correct information is discovered in different runs [5].

Recently, different LCSs have been combined into a multi-agent system, affectionately known as Original Intelligence System (OIS) as it approaches the task of learning multiple problems in a novel manner [13]. This is different to the GALE system [1] for individual problems as OIS seeks continuous learning using multiple agents for multiple problems, see Fig. 1.

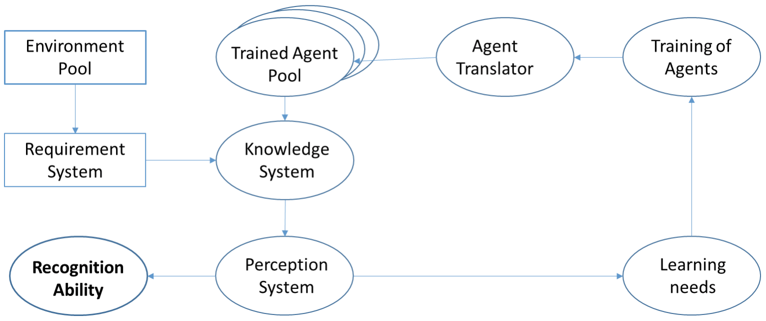


Fig. 1. Original Intelligence System [13]

OIS forms the base of the novel methods, which are related to compaction [3,4] and visualisation [15,17] of extracted knowledge, although unusually as a combination of multiple agents rather than one. These techniques are also not competitors as they can be used within OIS without loss of benefits.

3 Method

The novel contribution is split into three main parts. The first part uses different versions of standard LCSs (in this case different XCSs) to learn a benchmark problem several times. The second part selects reasonably high fitness rules from all the trained agents. The last part is using our novel search technique, termed *attribute-search* to discover the unique best classifier set among the whole good performance rules.

To collect the candidate classifiers, four strategies are employed in sequence. 1. Elite selection, 2. Experience assessment 3. Consistency assessment and 4. Correctness Partition. Therefore, the acronym for these four strategies is EECC, see Fig. 2.

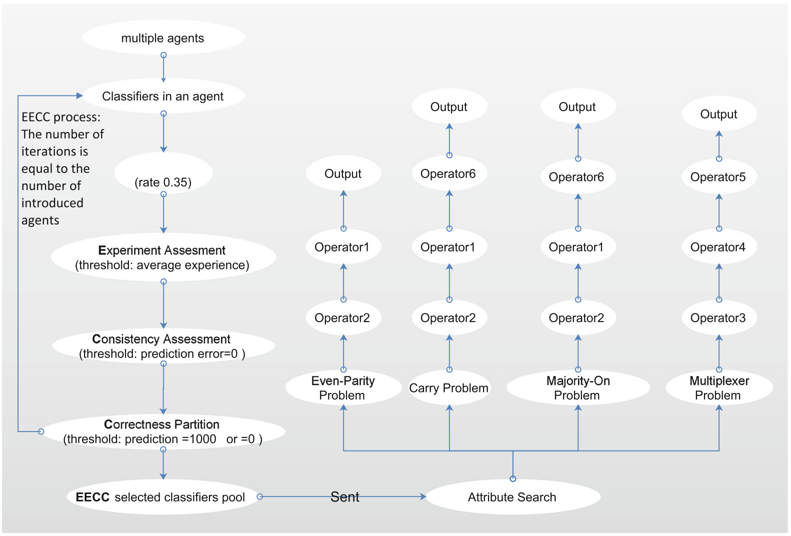

multiple agents
Classifiers in an agent
EECC process: The number of iterations is equal to the number of introduced agents
(rate 0.35)
Experiment Assesment (threshold: average experience)
Consistency Assessment (threshold: prediction error=0)
Correctness Partition (threshold: prediction =1000 or =0)
EECC selected classifiers pool
Sent
Attribute Search
Even-Parity Problem
Carry Problem
Majority-On Problem
Multiplexer Problem
Operator1
Operator2
Operator3
Operator4
Operator5
Operator6
Output

Fig. 2. EECC and Attribute search: firstly, multiple agents produce final classifiers which EECC sub-selects to send to attribute search. Currently, the best operation combination for a specific problem domain is set manually, e.g. for the even-parity problem, all the EECC selected classifiers will undergo operator2 and operator1 in turn.

Elite Selection. The elite selection algorithm firstly ranks all classifiers in an agent according to their value of numerosity (a parameter that stores the number of duplicates of a unique classifier) from high to low. The classifiers are selected according to their rank above a threshold based on population size and training number.

Experience Assessment. In XCS, the parameter experience indicates the number of training instances for every classifier. We only consider classifiers with an experience greater than the average to remove poorly tested rules.

Consistency Assessment. We delete any classifiers whose prediction error is higher than 0 as they are not considered accurate. This could be tuned for noisy domains.

Correctness Partition. In XCS, both completely correct and incorrect classifiers can reduce their prediction error to 0. We employ the best action map strategy to only use correct classifiers [14].

Attribute-Search: In Boolean problems, every bit can be considered as an attribute according to their location. Therefore, an m-bit Boolean problem can be considered as [Attribute0, Attribute1, …Attribute$_m$]. The number of specified (not don't care) attributes ranges from 0 (general) to m (specific). The number of unique classifiers is $2 * 3^m$ for binary classification. The number of optimum classifiers varies with the problem domain.

The aim is to reduce the amount of the introduced unique classifiers that are redundant to solving the problem. Since the classifiers are distinguished by the introduced attributes, we cluster the EECC collected classifiers according to the number of specific attributes they contain. We group the whole corresponding classifiers space into $\mathrm{m}+1$ sub-spaces. For example, sub-space 0 means all the classifiers in this space contain no specified attribute, e.g. in 6-bit Boolean problems ######:0 is a typical member in that space.

Algorithm 1. Find the best none-overlap combination of classifiers from the best subspace

```
1  begin
2      Input: $S$ ← a set of classifier $n$ rules;
3      Output: $BestSubset$ ← a subset of classifiers;
4      $BestSubset$ ← an empty classifier set;
5      $MaximumNumerosity$ ← 0;
6      Rank the $n$ rules in $S$ in a descending order, according to their numerosity
       values;
7      foreach $s \in S$ do
8          $S_{temp}$ ← $S$;
9          $Subset$ ← an empty rule set;
10         Add $s$ to $Subset$; // every rule has a chance to be a priority
              rule to be returned
11         Remove $s$ from $S_{temp}$;
12         foreach $s' \in S_{temp}$ do
13             if $s'$ does not have overlap with any rule in Subset then
14                 Add $s'$ to $Subset$;
15         $TotalNumerosity$ ← calculate the total Numerosity of rules in $Subset$;
16         if $TotalNumerosity > MaximumNumerosity$ then
17             Update $MaximumNumerosity$ ← $TotalNumerosity$;
18             Update $BestSubset$ ← $Subset$;
19     Return $BestSubset$;
```

Attribute-search is an umbrella name for a set of simple operations, which focus on search a reasonable classifiers combination among the EECC selected classifiers. This search technique includes 6 operations:

1. Error detection operation
2. Correct and incorrect group merge operation
3. Most reasonable sub-search space detection operation
4. Best none-overlap classifiers combination detection
5. Removal of over-general and over-specific rules
6. Subsumption operation.

Error Detection Operation. This operation focus on detecting the potential over-general classifiers in EECC selected classifiers. The basic idea of this operation is to compare each classifier (termed the target) with the classifiers in the higher sub-search space. The sum of the numerosity of any contrasting classifiers in the higher sub-search space that have an overlapping condition, but different action, with the target classifier is termed the conflict value. When the conflict value surpasses the target classifier's numerosity, we define the target classifier as an over-general classifier and delete this classifier.

Correct and Incorrect Group Merge Operation. A best action map is formed by flipping the action of any completely incorrect classifier (i.e. 0 prediction) then merge into the correct classifier set. Any classifiers having the same condition and action are combined into one classifier with summed numerosity.

Most Reasonable Sub-space Detection. The process of this strategy is to firstly cluster the EECC selected classifiers according to the number of attributes contained in the condition part. To solve an m-bit Boolean problem, we create m+1 sub-search spaces from minimum 0 to maximum m. For example, in 6-bit multiplexer problem, the attribute interval we obtained is [3, 5]. Then we calculate the total numerosity value for every remaining sub-space in the attribute interval, e.g. [5832, 17, 36] corresponding to sub-spaces [3, 4, 5]. We select the most important sub-spaces, e.g. sub-space 3, by a percentage threshold.

Best None-Overlap Classifiers Combination Detection. In string based condition, the overlap space between two classifiers can be calculated. Therefore, the overlapping classifiers can be detected. We search each rank from top to bottom and delete the classifiers that have overlap between the previous kept classifiers. Then we calculate the total numerosity value of each ranked combination of the remaining classifiers. The ranked combination with the highest numerosity is returned.

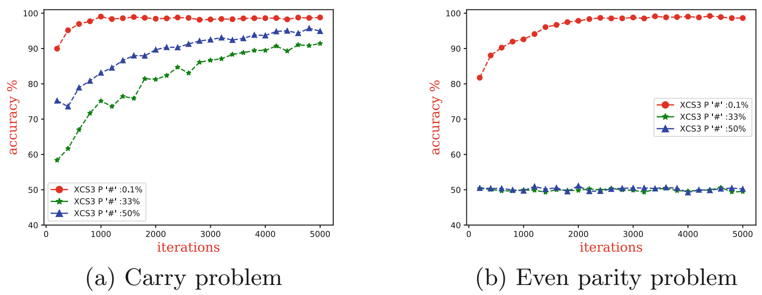
Removal of Over-General and Over-Specific Rules. The next step is to remove the over-general classifiers in the lower sub-search space, and the over-specific classifiers in the higher sub-search space. We can calculate the over-lap space between classifiers and detect a non-overlapping classifier in the best sub-space. Therefore, we detect whether a classifier-rule in the lower sub-search space can be expressed in the best sub-space.

Any classifier that has over-lap between a classifier in the best non-overlap classifier combination in the condition part and their action is different will be considered as an over-general classifier and it will be deleted. For example, in Table 3 the number 3 classifier (i.e. 000###:0) which belongs to the best sub-search space 3. We detect a classifier in the lower sub-search space, which is 00####:1 where its condition over-laps and its action is different so it will be removed.

Subsumption Operation. The subsumption operation in Attribute-search is the same as the subsumption in the XCS - if we can find a more general accurate classifier in a lower sub-space then we can directly delete a matching higher sub-space classifier.

4 Results and Discussion

In our experiment, standard XCSs are employed to generate the 360 agents for 6-bit problems, i.e. 30 runs (to statistically reduce variability) of three different XCSs were used for four problems. The three different versions of standard XCS only differed by the probability of generating don‘t care symbol, $P_{\#}$, see Fig. 3. Importantly, the training runs had a limited number of iterations in order to generate a variety of populations for the EECC system to merge, see Fig. 4.



(a) Carry problem (b) Even parity problem

Fig. 3. Example training plots showing deliberately compromised performance

Imperfect agents were introduced to test whether our strategy can select the optimal classifiers when there are poor classifiers in the whole classifier set. This is because in real problems there is no guarantee that every trained agent can completely solve the target problem.

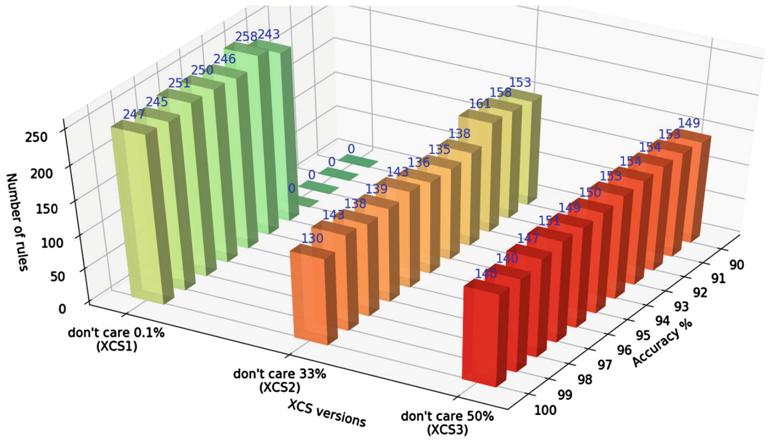


Fig. 4. The distribution of global agents' accuracy.

Table 1. XCS (accuracy)

Problems	XCS1	XCS2	XCS3	EECC and attribute-search
Even parity	[94.5%, 100%]	[43.8%, 54.5%]	[46.4%, 56.3%]	[100%, 100%]
Carry	[95.5%, 100%]	[76.3%, 98.7%]	[89.4%, 100%]	[100%, 100%]
Majority on	[95.9%, 100%]	[66.9%, 94.2%]	[74.3%, 96.2%]	[100%, 100%]
Multiplexer	[97.2%, 100%]	[72.7%, 100%]	[94.5%, 100%]	[100%, 100%]

Table 2. XCS (classifiers)

Problems	XCS1	XCS2	XCS3	EECC and attribute-search
Even parity	[845, 913]	[215, 288]	[322, 455]	64
Carry	[214, 262]	[135, 196]	[168, 223]	18
Majority on	[421, 480]	[209, 264]	[235, 332]	35
Multiplexer	[243, 258]	[130, 161]	[140, 154]	8

The performance results in terms of training accuracy and number of rules produced are shown in Tables 1 and 2 respectively. Note that as the purpose of LCSs is extraction of patterns through classification that a separate test set is not used as no claims on predictive ability to unseen data are being made. The tables show that non-optimal knowledge has been combined to generate optimal and compact rule sets, which can be interrogated, using the introduced methods.

4.1 Attribute Importance

The Attribute intervals following the sub-space clustering of the EECC collected classifiers are shown in Fig. 5 for the 90 agents of each problem. Essentially, for a given attribute level, how likely is a given attribute to be specified. This clearly visualises the different underlying patterns in each problem domain.

Multiplexer Problem. We cannot find any classifiers in the sub-space 0, 1, 2, 4, 5 and 6. Thus, the number of attributes in a single classifier is limited to [3]. It is well-known that the global optimal classifiers are indeed in the sub-space3. When we combine EECC and sub-space clustering together, the system can automatically find the best attribute interval for us.

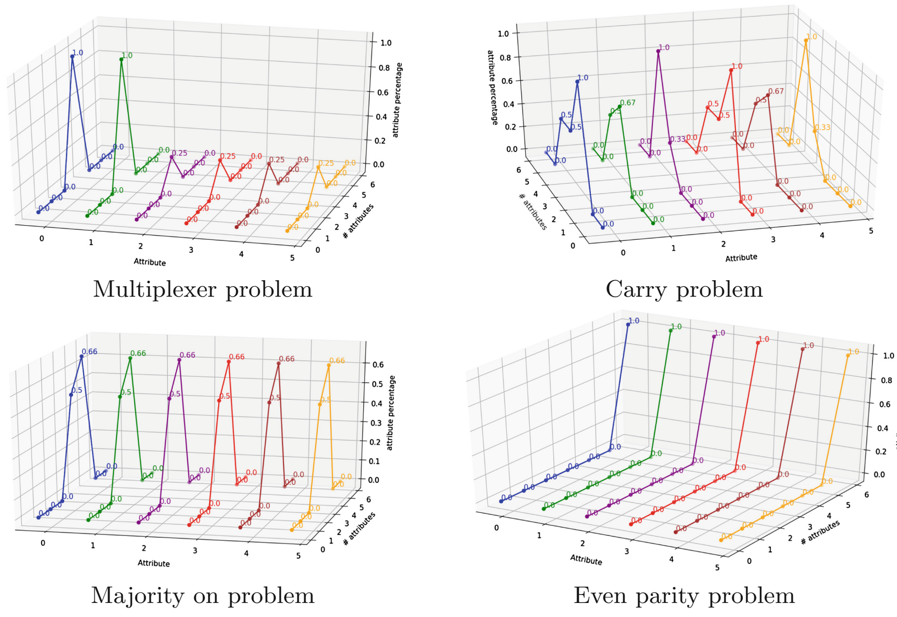


Fig. 5. The attribute importance change according to different attribute space in attribute-search selected classifiers for certain problem

Carry Problem. The parameter settings for EECC used to analyze the Carry problem were the same as the setting in the multiplexer problem. Firstly, the methods reduced the number of introduced classifiers from 150 to 18. Secondly, according to the results in Table 6, the attribute interval of Carry problem is [2, 4]. Thus the attribute-space for the Carry problem is narrowed by the EECC. Thirdly, the attribute importance distribution for the most reasonable sub-search space, i.e. based on highest numerosity, was [1 0 0 1 0 0]. This shows the importance of the most significant bits in the Carry problem, which autonomously shows the nature of the problem.

Majority on Problem. A sample of the rule set produced, which is optimal, is shown in Table 5. Again the attribute interval, Fig. 5, shows the distribution of attribute importance, where no rules with attribute [0, 1, 2, 5, 6] are present. This demonstrates that both over-general and over-specific rules have been eliminated.

Even Parity Problem. When generating the result for the Even parity problem, the EECC needed three adjustments. Firstly, the Elite selection threshold was increased to 0.8 from the standard 0.35 as low numerosity classifiers were being produced by the agents. Moreover, if we set the threshold to 0.3, none of the classifiers were selected from EECC indicating a too strict threshold that needed relaxing. Secondly, in the Experience assessment part, we need to discount the experience threshold (or run the experiments for more iterations). Thirdly, in the Consistency assessment part, the prediction error was set raised to 1 as the lack of experience meant the parameter value had not converged.

The correct total of 64 classifiers was discovered where, as necessary, all are in sub-search space6, i.e. fully specified (table not shown due to length as fully enumerated 6-bit Boolean domain). Therefore the attribute interval for the Even parity problem is [6], see Fig. 5. This discovery supports that LCS can not generate any general classifiers suitable for this problem. EECC can detect the nature of the Even parity problem.

4.2 Specific Search

Since XCS generated classifiers can be grouped based on the number of attributes contained by the condition. Then we create the sub-search space for each XCS. In a m-bit Boolean problem, each of the $m + 1$ sub-space has the ability to express the whole problem domain. The extended OIS can detect which sub-space is more reasonable for solving the target problem. Also which sub-spaces are too over-general and which sub-spaces are too specific. Thus, it is practical for extended OIS to limit the search space. Thus, three strategies are introduced to complete the attribute-search. First one is the most reasonable attribute-space detection, we achieved this goal by finding the sub-space which has the highest totally numerosity value among all the sub-spaces detected in the attribute interval. Then we implement a strategy named as best non-overlap classifiers combination detection. Since in a string-based condition, we can calculate the overlap between classifiers, if no overlap classifiers combination can be detected, among all the non-overlap classifiers combination, we select the best one means the one has the highest total numerosity value. After this step, all the classifiers in the higher search space (greater number of attributes) we implement the decomposed operation, then any of the classifiers, which can be expressed in the best classifiers combination will be deleted. For any classifiers in the lower sub-search space (less number of attributes) we implement the conflict operation if a conflict occurs this classifier will be deleted (Table 4).

Table 3. Raw classifiers after EECC is applied in 6-bit multiplexer problem

ID	Condition	Action	Numerosity
0	01#1##	1	624
1	001###	1	593
2	10##1#	1	526
3	000###	0	486
4	11###0	0	452
...	...	...	...
21	110#00	0	7
22	01#010	0	6
23	#00101	0	6
24	#00001	0	5

Table 4. The final result for the Multiplexer problem

ID	Condition	Action	Numerosity
0	01#1##	1	624
1	001###	1	593
2	10##1#	1	526
3	000###	0	486
4	11###0	0	452
5	11###1	1	451
6	01#0##	0	394
7	10##0#	0	311

Time. Using XCS to train one 6-bit Boolean problem with maximum iterations as 3000 and maximum population 500 needs 1 to 2 min. By utilizing GRID computing, training 90 agents only take us 5 to 10 min. EECC and attribute-specific only spend 40 s to finish their tasks. Compared with the previous XCSs, we only introduce 4 to 8 addition minutes in the training step, but we can reduce more than 100 classifiers in our final result. In addition, our agent is much more compressed than the agents trained from previously XCSs. Therefore, when we reuse them to solve problems, they compute much more quickly than the previous agents speeding-up OIS.

5 Discussion

The Attribute-search for the Multiplexer problem is operation 3, operation 4 and operation 5. Firstly, we find the most reasonable sub-search space. Secondly, we find the best unique classifiers combination to obtain a unique classifiers combination in the best sub-search space. The reason is that the Multiplexer problem does not need any overlapping classifiers to solve the problem. Lastly, we run the decompose operation to delete all the classifiers in the higher sub-search space, which can be expressed totally in the most-reasonable sub-search space.

The Attribute-search for the Carry problem and the Majority on problem is operation 2, operation 1, and operation 6. Firstly, we merged the incorrect classifier set and correct classifier set from EECC selected classifiers. Then implement the error detect to delete the noise classifiers. Lastly, we run the subsumption operation to delete any classifiers in the higher sub-search space that can be subsumed by the classifiers that are in the lower sub-search space. The reason for the Carry problem and the Majority on problem use the same Attribute-search process is that they all need overlapping classifiers to address the target problem.

Table 5. Sample rules for the Majority on problem

Condition	Action	Numerosity
0	00###0	22
0	#0#00#	22
0	#000##	13
0	##0#00	8
1	#1#111	63
1	1#11#1	63
1	#11#11	59
1	#111#1	55

Table 6. The final result of the Carry problem, the attribute interval is [2, 4]

Condition	Action	Numerosity
1	1##1##	1312
0	0##0##	923
1	11##1#	215
1	#1#11#	185
0	0###00	156
0	0#0#0#	124
0	00##0#	75
0	###000	61
0	00###0	59
0	#0#0#0	58
0	000###	58
0	#000##	57
0	#0#00#	46
0	##000#	28
1	##1111	29
1	#111#1	23
1	1#1#11	21
1	111##1	13

The Attribute-search for the Even parity problem only uses the operation 1 and operation 2. We merge the correct and incorrect classifier sets and then detect the erroneous classifiers. A very interesting phenomenon is that almost all the EECC selected classifiers are in the incorrect classifier set. It appears that the system tends to solve the Even parity problem by combining over-general classifiers and incorrect specific classifiers.

6 Conclusion

The novel methods demonstrate that it is both practical and beneficial to combine the results of multiple LCSs as they have differing viewpoints into the same problem. This is related to ensembles of 'weak classifiers', albeit each LCS may be a collection of powerful classifiers in its own right. Results show the successful compaction of multiple solutions to a single, optimum solution, which highlights important feature knowledge that would otherwise have been hidden.

Future work will introduce more styles of LCS, e.g. supervised learners, and different types of problem, e.g. real-valued, into OIS in order to seamlessly improve its capabilities.

References

1. Bernadó, E., Llorà, X., Garrell, J.M.: XCS and GALE: a comparative study of two learning classifier systems on data mining. In: Lanzi, P.L., Stolzmann, W., Wilson, S.W. (eds.) IWLCS 2001. LNCS, vol. 2321, pp. 115–132. Springer, Heidelberg (2002). doi:10.1007/3-540-48104-4_8
2. Browne, W., Scott, D.: An abstraction algorithm for genetics-based reinforcement learning. In: Proceedings of the 7th annual conference on Genetic and evolutionary computation, pp. 1875–1882. ACM (2005)
3. Butz, M.V., Lanzi, P.L., Wilson, S.W.: Function approximation with XCS: Hyperellipsoidal conditions, recursive least squares, and compaction. Trans. Evol. Comput. **3**(12), 355–376 (2008)
4. Dixon, P.W., Corne, D.W., Oates, M.J.: A preliminary investigation of modified XCS as a generic data mining tool. In: Lanzi, P.L., Stolzmann, W., Wilson, S.W. (eds.) IWLCS 2001. LNCS, vol. 2321, pp. 133–150. Springer, Heidelberg (2002). doi:10.1007/3-540-48104-4_9
5. Ioannides, C., Browne, W.: Investigating scaling of an abstracted LCS utilising ternary and s-expression alphabets. In: Bacardit, J., Bernadó-Mansilla, E., Butz, M.V., Kovacs, T., Llorà, X., Takadama, K. (eds.) IWLCS 2006-2007. LNCS, vol. 4998, pp. 46–56. Springer, Heidelberg (2008). doi:10.1007/978-3-540-88138-4_3
6. Iqbal, M., Browne, W.N., 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, pp. 863–870. ACM (2012)
7. Iqbal, M., Browne, W.N., Zhang, M.: Evolving optimum populations with XCS classifier systems. Soft. Comput. **17**(3), 503–518 (2013)
8. Iqbal, M., Browne, W.N., 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, pp. 1045–1052. ACM (2013)
9. Iqbal, M., Browne, W.N., Zhang, M.: Learning overlapping natured and niche imbalance boolean problems using XCS classifier systems. In: 2013 IEEE Congress on Evolutionary Computation (CEC), pp. 1818–1825. IEEE (2013)
10. Iqbal, M., Browne, W.N., Zhang, M.: Reusing building blocks of extracted knowledge to solve complex, large-scale boolean problems. IEEE Trans. Evol. Comput. **18**(4), 465–480 (2014)
11. Iqbal, M., Naqvi, S.S., Browne, W.N., Hollitt, C., Zhang, M.: Salient object detection using learning classifiersystems that compute action mappings. In: Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation GECCO 2014, pp. 525–532 (2014)
12. Lanzi, P.L.: Mining interesting knowledge from data with the XCS classifier system. In: Proceedings of the 3rd Annual Conference on Genetic and Evolutionary Computation, pp. 958–965. Morgan Kaufmann Publishers Inc. (2001)
13. Liu, Y., Iqbal, M., Alvarez, I., Browne, W.N.: Integration of code-fragment based learning classifier systems for multiple domain perception and learning. In: 2016 IEEE Congress on Evolutionary Computation (CEC), pp. 2177–2184 (2016)
14. Orriols-Puig, A., Bernadó-Mansilla, E.: A further look at UCS classifier system. In: GECCO06, pp. 8–12 (2006)
15. Urbanowicz, R.J., Granizo-Mackenzie, A., Moore, J.H.: An analysis pipeline with statistical and visualization-guided knowledge discovery for Michigan-style learning classifier systems. Comput. Intell. Mag. **7**(4), 35–45 (2012)

16. Urbanowicz, R.J., Browne, W.N.: Introduction to Learning Classifier Systems. Springer, Heidelberg (2017)
17. Urbanowicz, R.J., Moore, J.H.: Exstracs 2.0: description and evaluation of a scalable learning classifier system. Evol. Intell. **8**(2–3), 89–116 (2015)
18. Wilson, S.W.: Classifier fitness based on accuracy. Evol. Comput. **3**(2), 149–175 (1995)