A New Hybrid Architecture for the Discovery and Compaction of Knowledge: Breast Cancer Datasets Case Study

Faten Kharbat[1], Mohammed Odeh[2], and Larry Bull[2]
[1]College of Business Administration, Al-Ain University of Science and Technology, UAE
[2]Department of Computer Science & Creative Technologies, University of the West of England, UK

Abstract: *This paper reports on the development of a new hybrid architecture that integrates Learning Classifier Systems (LCS) with Rete-based production systems inference engine to improve the performance of the process of compacting LCS generated rules. While LCS is responsible for generating a complete ruleset from a given breast cancer pathological data-set, an adapted Rete-based inference engine has been integrated for the efficient extraction of a minimal and representative ruleset from the original generated ruleset. This has resulted in an architecture that is hybrid, efficient, component-based, elegant, and extensible. Also, this has demonstrated significant savings in computing the match phase when building on the two main features of the Rete match algorithm, namely structural similarity and temporal redundancy. Finally, this architecture may be considered as a new platform for research on compaction of LCS rules using Rete-based inference engines.*

Keywords: *Hybrid architecture, LCS, rete algorithm, production systems.*

Received June 6, 2012; accepted October 29, 2012; published online January 29, 2013

1. Introduction

Learning Classifier Systems (LCS) [13] and Production Systems (PS) were two of the main issues that have been investigated in Artificial Intelligence (AI) over the last three decades. PS is a model of knowledge representation [18] which was applied on many real expert system applications, and used to build expert system shells, such as CLIPS [10] and Jess [8].

LCS is a rule-based system which uses evolutionary algorithms to facilitate rule discovery [5]. It has been applied to different data mining problems and shown effectiveness in both predicting and describing evolving phenomenon [15]. However, in the real-domain environments, having generated describable rules, LCS needs further step in which a subset of minimal number of rules is to be found that still can describe the environment. In other words, a compaction process is required over the rules generated as an output of the classifier system.

A number of approaches have been attempted to develop a sufficient compaction algorithm where a minimal subset of rules can be extracted with minimal run time required. However, these attempts suffer from the same deficiency in terms of poor performance. This work discusses how production systems cycles can be utilized to improve the performance of the compaction process. A three-phased hybrid architecture has been developed that utilizes the Recognize-Act-Cycle (RAC) used by PS inference engines, and in particular adapting the Rete match algorithm to improve the previous compaction algorithms resulting in an elegant and promising approach to compacting LCS generated rules. A brief introduction to PS is introduced in section 2, followed in section 3 by a brief description of learning classifier systems' structure, and XCS in particular. A brief description of the main compaction algorithms cited in the literature is described in section 4. Finally, section 5 reports on the implementation of the new approach followed by critical evaluation of the results achieved.

2. Production Systems

Over the last three decades, AI has been the aim of many researchers particularly in the field of expert systems or knowledge-based systems. The production system represents a model of knowledge representation that is mostly applied in real applications of expert systems; e.g., R1/XCON in configuring computer systems [21], and MYCIN [20] in diagnosing bacterial infections of the blood.

In general, the architecture of a production system consists of three components: working memory, production memory, and the inference engine. The working memory (also called fact base) is the collection of facts (cases), which could be any information collected by the knowledge engineer or extracted from information systems.

Production Memory (or Rule-Base) consists of a set of rules (productions) that represent domain-specific and problem-solving knowledge Gonzalez and Dankel

[11]. Each rule has a name and can be expressed by If-Then statement. The if-part is the Left Hand Side (LHS), which is also called condition part, or the antecedent. It consists of one or more of condition elements. The then-part, which is called the Right Hand Side (RHS), consequent or action- consists of number of actions. The action behavior may affect the working memory by inserting, deleting, or modifying any of its elements.

The inference engine is the production system interpreter that executes rules. Two well-known inferencing methods are used in production systems: forward and backward chaining. Forward chaining is reasoning from facts to conclusion, whereas the backward chaining is the reverse reasoning i.e., from the hypotheses to their supporting facts [10]. The approach adopted in this paper is based on the forward reasoning model which will be explained in the next section.

2.1. The Recognize-Act-Cycle

In the forward reasoning, the inference engine execution model is a three-phase cycle of: Match; conflict resolution, and; act phases [11] as shown in Figure 1. This cycle is commonly referred to as the RAC, select-executed-cycle, or situation-action-cycle [10].

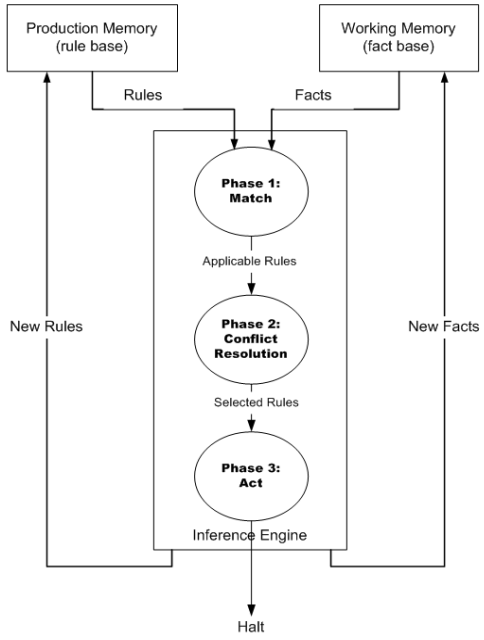


Figure 1. The RAC.

In the match phase, the inference engine evaluates the condition elements of all production rules againstthe current Working Memory elements according to a predefined match algorithm, for example the Rete algorithm. In other words, the rule base is compared to the fact base to determine which rules are applicable. The applicable rules are instantiated (activated) and grouped to form the conflict set.

Rete algorithm was introduced by [7] as an efficient match algorithm and since then it has been used as the match algorithm in many production systems and expert systems shells; e.g., CLIPS, Jess and others. The main advantage of Rete algorithm is its performance gain over previous match algorithms as it avoids redundant match iterations between working memory (facts) and production memory (rules) as fully explained in [7]. More precisely, it exploits two important characteristics of the rules in most production system programs.

Temporal redundancy: this refers to the fact that executing rules would change only few working memory elements (facts) (3-4 on average). When a new working memory element is matched against the rule-base, match results are stored and subsequent changes take place only when this working memory element gets changed. Hence, this results in performance gains when few working memory changes take place and match results of previous cycles have been saved.

Structural similarity: this refers to the structural similarity between patterns in the LHS of more than one production (rule). Rete makes the most of this feature by indexing the patterns in a tree-structured data-flow network [7].

The output of the match phase is the input to the conflict resolution phase, where the conflict set is sorted in descending order by the rule's priority which is assigned for each rule according to predefined conflict-resolution strategy (for example depth strategy). The agenda (which is the ordered conflict set) is used in a similar way to a stack, where the top instantiated rule is the first one executed, where only one rule is selected for execution per production system cycle. This happens in the third phase, namely the Act phase, where the RHS of the selected rule instantiation is executed. Each action in the RHS is one of the following four possibilities:

1. Adding a new fact to the fact-base and/or a new rule(s) to the knowledge base;
2. Modifying an existing fact;
3. Explicit deletion of a working memory element, or;
4. Halting the inference engine. The inference engine also halts if the conflict set is empty, i.e., no more rule instantiations exist to be selected for firing.

3. Learning Classifier Systems

LCS is a rule-based learning system that evolves evolutionary computation techniques, reinforcement learning, and other heuristics to generate an optimal set of rules for a given environment [5]. The first LCS architecture, namely CS-1, was introduced by [12]. However, this system was both complex and difficult to predict its behavior [28]. In [23], built on his Animate, in which Boole [24], New Boole [4], ZCS [26], and XCS [27] were refined. However, it may be

said that most current LCS research has made a shift away from Holland's original formalism after Wilson introduced XCS.

3.1. XCS

XCS uses the accuracy of rules' predictions of expected payoff as their fitness. In addition, XCS uses Genetic Algorithms (GA) [14] to evolve generalizations over the space of possible state-action pairs of a reinforcement learning task with the aim of easing the use of such approaches in large problems, i.e., those with state-action combinations that are too numerous for an explicit entry for each. It can also avoid problematic 'overgeneral' rules that receive a high optimal payoff for some inputs, but are sub-optimal to other lower payoff inputs. Further details on XCS can be found in [27].

XCS consists of a limited size population [P] of classifiers (rules). Each classifier is in the form of "IF condition THEN action" and has a number of associated parameters. The condition may consist of binary representation for simple problems, integer intervals [25], real values [29], or combination of these for more complex one. One of the main measurements in LCS is the performance which is the percentage of the correct classifications that the classifier system performs during its testing phase.

LCS in general and XCS in particular were applied to different data mining problems. It was shown that LCS could be effective for predicting and describing evolving phenomenon, in addition to its modeling ability [15]. In [19], applied XCS to the Monk datasets showing that, with an appropriate representation, XCS was able to solve these problems as accurate as or better than other competitive machine learning algorithms. Similar task was done in [1, 2, 6] on further data mining tasks drawn from the UCI Repository [3].

Moreover, Wilson [22, 25] applied XCS to a medical dataset, namely the WBC Dataset, and showed that XCS can tackle real complex learning problems, in addition to its capability to deal with different representations. Also, XCS was tested on other datasets in [2] and showed to have a competitive performance in both training and testing phases.

3.2. Case Study: The Wisconsin Breast Cancer Dataset

Wisconsin datasets are three well-known breast cancer datasets from the UCI Machine Learning Repository [3]:

1. Wisconsin Breast Cancer (WBC) Dataset which describes clinical images taken from fine needle biopsies of breast masses.
2. Wisconsin Diagnostic Breast Cancer Dataset (WDBC) which describes 30 characteristics of the cell nuclei present in each image.
3. Wisconsin Prognostic Breast Cancer Dataset (WPBC) which has follow-up data on breast cancer cases.

The development of WBC started in 1989 in Wisconsin University Hospitals by Dr. William Wolberg, and since then it has been heavily used as a test bed for machine learning techniques [17]. It consists of 699 test cases, in which 16 cases have a missing value. Every case has nine integer attributes associated with the diagnosis. Also, each attribute ranges between 1 and 10 while the diagnostic parameter (action) has binary possibilities as either malignant (34.5%), or benign (65.5%). The attributes are: Clump Thickness, Uniformity of Cell Size, Uniformity of Cell Shape, Marginal Adhesion, Single Epithelial Cell Size, Bare Nuclei, Bland Chromatin, Normal Nucleoli and Mitoses. Figure 2 shows the prediction accuracy of XCS over the WBC (average and standard deviation) compared to other learning algorithms showing the efficiency and ability of XCS to tackle real complex problems. In this research, WBC dataset has been used as the test bed to study and evaluate the outcomes of the new LCS compaction approach, namely Compaction using Recognise-Act Cycles (CRAC) [2].

DS	0-R	IB1	IBK	NaiveBayes	C4.5r8	PART	SMO	XCS	GALE
bps	51.6±0.6	83.2±3.2	82.8±4.3	78.6±5.5	80.1±4.8	79.0±3.3	86.4±3.0	83.2±3.1	83.7±3.8②
bre	65.5±1.1	96.0±1.5	96.7±1.4	96.0±2.3	95.4±1.6	95.3±2.2	96.7±1.7	96.4±2.5	95.7±2.2
bpa	58.0±1.4	63.5±6.6	60.6±6.6	54.3±2.8	65.8±6.9	65.8±10.0	58.0±1.4	65.4±6.9	68.4±6.7†

Figure 2. Prediction accuracy of XCS and other learning algorithms on the WBC.

4. Approaches to LCS Rule Compaction

- *Goals of LCS Rule Compaction*

XCS has been showing encouraging results in different domains in terms of its capability to produce a maximal, general, correct solution for a given environment. The huge size of the generated solution, however, may still be considered as a barrier to exploit its entire knowledge. For example, more than 1100 rules were generated when WBC dataset was applied to XCS [25].

The main objective of applying real-domain problems to LCS is to provide the domain experts with a complete, minimal, readable solution with an organized underlying knowledge that have the ability to describe the given environment. "Complete" is one of the proved characteristics connected to XCS [16] which means that XCS is able to describe all regions of the input/action space (complete map) for the given environment. However, by increasing the number of rules describing the environment, overlapped patterns are allowed to exist, which conflict with the second term: "minimal". In other words, there will be some regions in the environment that are described and

covered by more than one rule (or pattern). Actually, some of the real-domain problems require an overlapping solution by their nature of complexity, but the point is with the unnecessary overlapping that could be avoided.

One of the other main problems caused by the huge number of rules is presenting these rules to an expert. This violates the third term: "readable" due to the over expected number of rules that make it impossible to comprehend them smoothly or make the maximum benefit of them. For example, providing a breast cancer specialist with 1100 rules describing the 700 WBC cases may not be easily comprehendible to make use of the underlying hidden knowledge for better understanding and enrichment of breast cancer knowledge.

Therefore, developing a compaction algorithm that addresses the above dimensions is essential to increase the level of rules readability, interpretation, and organization of the underlying knowledge held in them. The main algorithms attempted to compact LCS rules are: Wilson's [22], Dixon, Wolfe and Oates [6], Fu and Davis [9], and Wyatt, Bull and Parmee [30]. In general, the main observation on these previous attempts is that they all aim to select the minimum number of rules that cover the dataset to produce a minimal subset from the generated rules that describe the original dataset on which LCS was initially trained and further tested.

In summary, the importance of the compaction step has been addressed as an essential post-phase in LCS computations. The simplest algorithm was of Dixon, Wolfe, and Oates which has a polynomial run-time complexity rather than exponential as in the algorithms of Wilson, and Fu *et al.* ones. Wyatt, Bull and Parmee modification considered to be a performance improvement to the latter ones. But, since the above algorithms use a simple match algorithm (mainly the XCS one), the acceptance of these algorithms is expected to be severely affected by the excessive low performance of matching. The next section addresses a solution to the shortcomings outlined above building on the well-know Rete match algorithm.

5. Compaction Using Recognize Act Cycle

As mentioned above, one of the main disadvantages of the previous compaction algorithms is their dependency on the simple classical match algorithm (i.e., the classifier system's match algorithm) which implies that each fact is tested against all the ruleset whenever performance of this ruleset is calculated to result in a redundant match procedure without saving any predecessor previous and similar matches. As a consequence, the run-time cost for these algorithms grows massively. This can be easily observed, for example, in Wilson's algorithm to require more than two hours running over 1100 rules and 700 facts.

Figure 3 illustrates the general architecture of the proposed approach which integrates the two systems:

1. Learning classifier system to generate a maximally general ruleset describing the underlying problem;
2. Production System Inference Engine, enabled by its RAC to compact LCS generated rules using one of the algorithms described in the previous section.

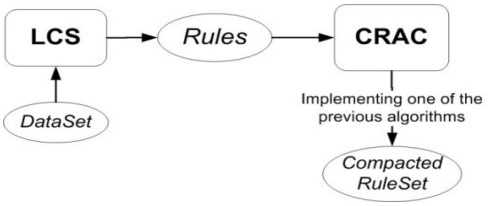


Figure 3. Architecture of the new CRAC approach.

This new approach, CRAC replaces the match algorithm used in the above compaction algorithms by Rete, the highly efficient match algorithm described in section 2, using the LCS rules to generate a minimal set of organized and representative knowledge. In implementing the CRAC approach, XCS is used as the LCS, and Jess shell as the production system shell with one of the previous compaction algorithm specifies the conflict resolution strategy.

Java expert system shell (Jess) [8] is one of the developed expert system shells, which was written in java and developed in the late 1990s. Since then a number of AI applications have been written in Jess considering the support of the forward and backward reasoning by its inference engine. Also, having additional functionalities than other shells, the ability to control Jess reasoning engine thro java and to integrate with other java programs via its well defined API are considered to be a very strong features.

The implementation of the hybrid architecture consists of three phases shown in Figure 4. The first stage starts by converting the existing rules, which were generated by LCS (XCS), to match the PS (Jess) syntax as well as translating the WBC dataset cases into its equivalent Jess format, based on a predefined and customized template. This is done by a transformation engine especially designed and developed for this purpose as described in phase 1 of Figure 4.

Having converted the syntax of the rules and facts to their equivalent Jess format, Jess's inference engine starts, in the second phase, where the adapted LCS rules are matched against the facts to build a data-flow network in which each node represents a condition element of the LHS (conditions) of each rule. Successive match results of facts against condition elements of the LHS of rules are stored saving a significant re-computation match time in subsequent inference engine cycles (i.e., RAC). In other words, the output of this phase is a data-flow network in which

the condition elements of the rules are indexed along with associated matched facts.

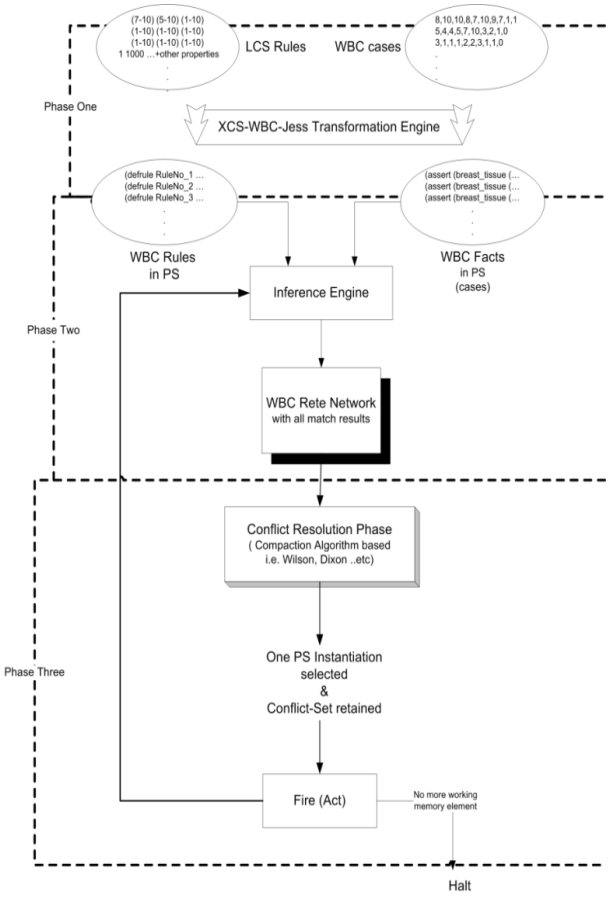


Figure 4. CRAC: Three phase hybrid architecture.

Based on one of the previous compaction algorithms, the conflict resolution phase sorts out its conflict set to determine the rules' execution order as shown in Figure 4-phase three. Each execution affects the Rete tree structure which forces the inference engine to reconstruct the affected part and not the whole tree since it is an indexed tree. In fact, this can be explained by the Rete exploitation to the characteristic of the temporal redundancy as explained before.

Typically, the system halts if the conflict set is empty and the compacted ruleset should be extracted. Bellow is the description of the previous compaction algorithms when they are implemented by the new approach CRAC. In brief, storing successive matchresults of facts against condition elements of rules save a significant re-computation match time in subsequent inference engine cycles where redundant match computations are performed and wasted from cycle to cycle in the previous match algorithm.

- *Knowledge Representation of LCS Rules using Jess*

In Jess, the deftemplate construct is used to specify the structure of data and its properties. Each template has a name and number of slots, where each represent a certain property. Jess does not seem to have a limit on the number of templates and associated slots. The general syntax of deftemplate in Jess is:

(*deftemplate* <*template name*> (*slot* <*slot name*>) +)

To add new data element into the working memory, the assert construct is used with the required template name and asserted attribute values as follows:

(*assert*(<*template name*> (<*slot name*> <*slot value*>)+))

The two templates used in the CRAC implementation are:

1. Breast-tissue in which the properties of the WBC cases are included. Figure 5 shows the structure of the breast-tissue template and an associated assert statement.
2. Rule-property in which the XCS rules' properties are described to be used within the PS. PN is the payoff numerosity product for the rule. It is used instead of calculating it each time as shown in Figure 6.

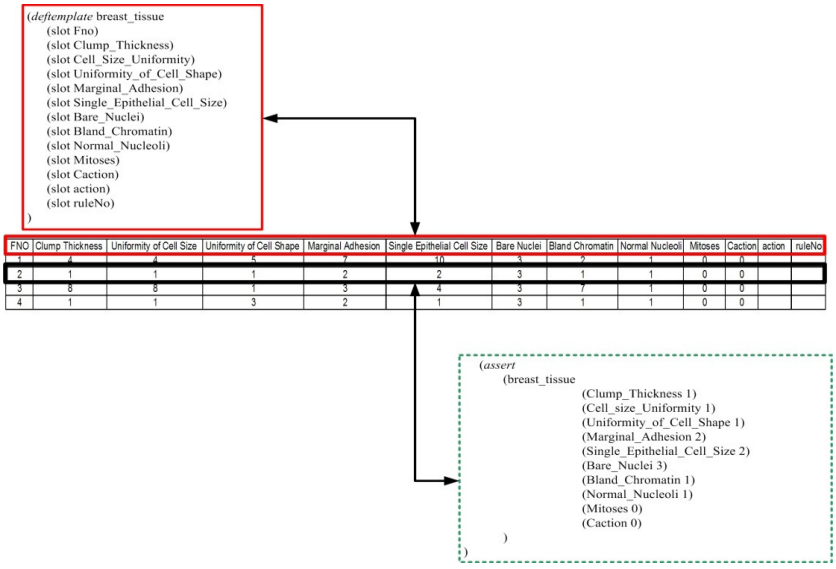


Figure 5. The breast-tissue template structure and an assert example.

```
(deftemplate rule_property
(slot ruleNo) (slot action) (slot payoff)
(slot error) (slot fitness)(slot numerosity)
(slot experience)  (slot PN))
```

Figure 6. The structure of rule-property template.

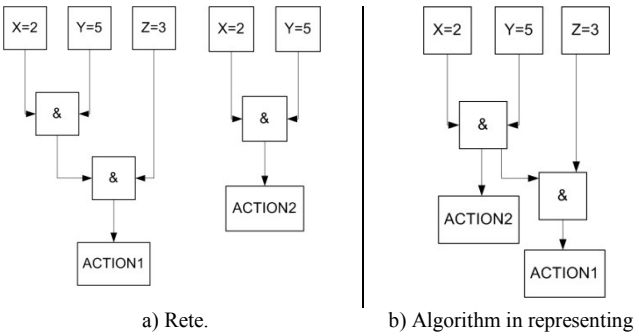
The formal way to express rules in Jess is by using defrule construct. Each rule has a name and body where the if-then block is specified.

6. Evaluation and Analysis of CRAC Implementation

In CRAC and when using the Rete match algorithm to replace the match computations in each of the previously discussed compaction algorithms results in a number of advantages ranging from significant performance gain to better explanation and readability of generated LCS rules. To study the effect of exploiting Rete algorithm compared to the simple classical match algorithm used in the above compaction algorithms, we use the following simple rules:

1. *If X=2 & Y=5 & Z=3 then Action1*
2. *If X=2 & Y=5 then Action2*

Figures 7-a and 7-b illustrate the application of the two match algorithms when matching rules 1 and 2. Although, the two rules have only two similar condition elements (*X=2* and *Y=5*), the simple classical match algorithm handles them separately so that each fact should be matched to five condition elements (*X=2*, *Y=5*, *Z=3*, *X=2*, and *Y=5*) as shown in Figure 7-a. On the other hand, Rete match algorithm exploits the LHS structural similarity of rules and builds its tree with only three condition elements reducing the cost of initial matching of condition elements by a factor of 3/5 without considering the most common cases of rules having complex condition elements with multiple features (or tests) that are similar across the ruleset as can be strictly observed in LCS generated rules over the breast cancer dataset. And, exploiting Rete match algorithm in CRAC has shown some significant gain in performance compared to the simple classical match algorithm used within the space of applying compaction. This can be evident when the matching phase gets repeated using high number of cycles, where redundant match computations are performed and wasted from cycle to cycle. For example if ten facts are matched to the above two rules, the simple classical match algorithm will have to perform 50 matching tests, whereas Rete algorithm performs only 30. In fact, in the case of the WBC generated ruleset, there are more than 361,000 similar patterns in the 1100 rules. While this paper is not about analyzing the complexity of Rete match algorithm, the simple classical match algorithm requires excessive computation time compared to Rete and this is expected to rise exponentially with the increasing number of facts and rules. In addition to the similarity structural advantage, the proposed approach makes use of Rete network in storing information about both the rules and their associated satisfied facts. Therefore, the time needed in the match process is reduced significantly since the stored information is used in the subsequent matches instead of re-matching the patterns with facts again.



a) Rete. b) Algorithm in representing rules.

Figure 7. The behavior of the simple classical match.

As explained in the previous section, each production system cycle execution (RAC) in Jess affects some minimal part of the corresponding Rete tree of the LHSs of the ruleset. This leads the inference engine to update only the affected part rather than the whole inference tree. This is because matching the facts to the patterns in rules is done only once during initial stage when the indexed tree is built followed by few changes to the memory nodes in the Rete tree as a result of adding new facts, and/or modifying or deleting existing matched facts.

The performance of the compaction algorithms discussed earlier has been compared both when using the CRAC (Rete-based) and the stand-alone implementation of these algorithms based on the simple classical match algorithm. Table 1 presents the execution results of both approaches when compacting the generated rules from running XCS over the original WBC dataset.

Table 1. The execution time for the compaction algorithms.

Compaction Algorithms	**Execution Time (in Minutes)**		**Speed up Factor(T_2/T_1)**
	T_1: CRAC (Rete Based)	**T_2: Simple Match**	
Wilson	~36	~180	5
Dixon *et al.*	~35	~2	.06
Fu & Davis	~36	~150	4.17

Table 1 provides some strong indication on the possible speed up factors that can be obtained from implementing the compaction algorithms of Wilson, Dixon *et al.* and Fu & Davis using the new CRAC approach. In particular, the speed up factor obtained reached is 5 times as can be observed in the case of Wilson CRAC based implemented. Moreover, this speed up is expected much higher if larger number of rules/facts are used given that the cost of matching using the simple match algorithm expected to rise

exponentially. Furthermore, the cost of computing match consumes 75% of the total execution time. In the case of the CRAC approach, the majority time of the time is spent in building the Rete tree for the first time. For example, building the Rete tree in Wilson algorithm took about 30 minutes where the rest of computations accounted only for only 6 minutes. This simply implies that the cost of constructing the Rete match tree in the first execution consumes most of the total execution time.

Furthermore, simple and classical match algorithms spend more than 90% of their total run-time in matching patterns [7] since the whole matching process gets repeated because of redundant match computations. However, Dixon *et al.* algorithm is a special case as shown in Table 1 since the matching phase occurs only once and this explains the efficient performance of this algorithm compared to the CRAC implementation. We simply attribute the higher computation cost in the Dixon *et al.* CRAC based implementation to the cost of constructing the Rete match tree.

7. Conclusions and Future Work

In summary, integrating LCS with CRAC results in an efficient hybrid architecture to compact knowledge discovery that is component-based, elegant, and extensible. Also, this architecture acts as bridge between LCS research and efficient execution of production systems. While LCS is responsible for generating a complete map solution for a given environment, CRAC works on the efficient extraction of a minimal ruleset from the original LCS generated ruleset. The results, so far, show that the new hybrid architecture was able to achieve competitive results in terms of run-time complexity.

In addition, integrating Rete match algorithm in the compaction process has shown significant run-time results in the form of significant savings in computing the match stage. This is because Rete builds on the two main characteristics, structural similarity and temporal redundancy. Finally, this architecture may be considered as a new platform for research on compaction of LCS rules using Rete-based inference engines.

8. References

[1] Bernado E., Llorà X., and Garrell J., "XCS and GALE: A Comparative Study of Two Classifier Systems with Six other Learning Algorithms on Classification Tasks," *in Proceedings of the International Workshop on Learning Classifier System*, London, pp. 115-132, 2001.

[2] Bernado E., Llora X., and Garrell J., "XCS and GALE: A Comparative Study of Two Learning Classifier Systems on Data Mining," *in Proceedings of the 4th International Workshop, Advances in Learning Classifier Systems*, Berlin, Germany, vol. 2321 pp. 115-132, 2002.

[3] Blake C. and Merz C., "UCI Repository of Machine Learning Databases," available at: http://www.ics.uci.edu/~mlearn/MLRepository.html, last visited 1998.

[4] Bonelli P., Parodi A., Sen S., and Wilson S., "NEWBOOLE: A Fast GBML System," *in Proceedings of International Conference on Machine Learning*, USA, pp. 153-159, 1990.

[5] Bull L., *Applications of Learning Classifier Systems*, Springer, 2004.

[6] Dixon P., Corne D., and Oates M., "A Ruleset Reduction Algorithm for the XCS Learning Classifier System," *in Proceedings of the 5th International Workshop, Learning Classifiers Systems*, Spain, pp. 20-29, 2003.

[7] Forgy C., "Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem," *Artificial Intelligence*, vol. 19, no. 1, pp. 17-37, 1982.

[8] Friedman-Hill E., *Jess in Action Java Rule-Based Systems*, USA, 2008.

[9] Fu C. and Davis L., "A Modified Classifier System Compaction Algorithm," *in Proceedings of the Genetic and Evolutionary Computation Conference*, USA, pp. 920-925, 2002.

[10] Giarratano J. and Riley G., *Expert Systems: Principles and Programming*, Course Technology, USA, 1998.

[11] Gonzalez A. and Dankel D., *The Engineering of Knowledge-Based Systems: Theory and Practice*, Prentice-Hall, USA, 1993.

[12] Holland J. and Reitman J., "Cognitive Systems Based on Adaptive Algorithms," *in Proceedings of ACM SIGART Bulletin*, USA, pp. 49-49, 1977.

[13] Holland J., *Progress in Theoretical Biology IV*, Academic Press, pp. 263-93, 1976.

[14] Holland J., *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*, Bradford Book, 1992.

[15] Holmes J., Lanzi P., Stolzmann W., and Wilson S., "Learning Classifier Systems: New Models, Successful Applications," *Information Processing Letters*, vol. 82, no. 1, pp. 23-30, 2002.

[16] Kovacs T., "XCS Classifier System Reliably Evolves Accurate, Complete, and Minimal Representations for Boolean Functions," *in Proceedings of Soft Computing in Engineering Design and Manufacturing*, England, UK, pp. 59-68, 1997.

[17] Mangasarian O. and Wolberg H., "Cancer Diagnosis via Linear-Programming," *SIAM News*, vol. 23, no. 5, pp. 1-18, 1990.

[18] Odeh M., *Concurrent Object-Oriented Execution of OPS5 Production Systems*, University of Bath, UK, 1993.

[19] Saxon S. and Barry A., "XCS and the Monk's Problems," *in Proceedings of Learning Classifier Systems*, Berlin, Germany, vol. 1813, pp. 223-242, 2000.

[20] Shortliffe E., *Computer-Based Medical Consultations: MYCIN*, Elsevier, USA, 1976.

[21] Soloway E., Bachant J., and Jensen K., "Assessing the Maintainability of XCON-IN-RIME: Coping with the Problems of a VERY Large Rule-Base," *in Proceedings of the International Conference on Artificial Intelligence*, USA, pp. 825-829, 1987.

[22] Wilson S., "Compact Rulesets from XCSI," *in Proceedings of the 4th International Workshop on Learning Classifier Systems*, London, UK, pp. 197-210, 2001.

[23] Wilson S., "Knowledge Growth in an Artificial Animal," *in Proceedings of the 4th Yale Workshop on Applications of Adaptive Systems Theory*, USA, pp. 98-104, 1985.

[24] Wilson S., "Quasi-Darwinian Learning in a Classifier System," *in Proceedings of the 4th International Workshop on Machine Learning*, USA, pp. 59-65, 1987.

[25] Wilson S., "Mining Oblique Data with XCS," *in Proceedings of the 3rd International Workshop, Learning Classifier Systems*, France, vol. 1996, pp. 158-174, 2001.

[26] Wilson S., "ZCS: A Zeroth Level Classifier System," *Evolutionary Computation*, vol. 2, no. 1, pp. 1-18, 1994.

[27] Wilson S., "Classifier Fitness Based on Accuracy," *Evolutionary Computation*, vol. 3, no. 2, pp. 149-175, 1995.

[28] Wilson S. and Goldberg D., "A Critical Review of Classifier Systems," *in Proceedings of the 3rd International Conference on Genetic Algorithms*, USA, pp. 244-255, 1989.

[29] Wilson S., "Get Real! XCS with Continuous-Valued Inputs," *in Proceedings of Learning Classifier Systems*, Berlin, Germany, vol. 1813, pp. 209-219, 2000.

[30] Wyatt D., Bull L., and Parmee I., "Building Compact Rulesets for Describing Continuous-Valued Problem Spaces using a Learning Classifier System," *in Proceedings of Adaptive Computing in Design and Manufacture VI*, London, UK, pp. 235-248, 2004.

Faten Kharbat is an assistant professor in Artificial Intelligence at the Al-Ain University, Abu Dhabi Campus, UAE. She holds PhD degree in computer science from the University of the West of England, UK, in 2006. Her main research interest are learning classifier systems, applying ontology into translation engines, knowledge based systems, applying data mining techniques to marketing, and recently is involved in quality of higher education.

Mohammed Odeh is senior lecturer in software engineering and associate of the Complex Cooperative Systems Centre at the University of West of England, Bristol, UK. He holds PhD degree in computer science from the University of Bath, 1993 in addition to PG Cert in Higher Education and membership of ACM and ILT. He has more than 20 years of experience including extensive project management experience in planning and leading a range of IT related projects in addition to management posts. He is the UWE principal investigator on the Onto REM knowledge exchange partnership with Airbus, and the SoAgile project being reviewed. His main research interests are bridging the gap between business process models and system models, ontology-driven requirements engineering, semantic and service-oriented software engineering, software cost estimation, grid computing, and knowledge management. His applied software engineering experience has been associated with banking, aerospace manufacturing, and medical informatics.

Larry Bull is a professor of artificial intelligence and based in the Department of Computer Science & Creative Technologies at UWE. His research interests are in intelligent and unconventional systems, with an emphasis on evolution. He is the founding Editor-in-Chief of the Springer journal Evolutionary Intelligence.