Multiple Classification Ripple Down Rules : Evaluation and Possibilities

Byeong Ho Kang, Paul Compton and Phil Preston
School of Computer Science and Engineering
University of New South Wales,
Sydney NSW 2052, Australia
email kang@cse.unsw.edu.au

Abstract

Ripple Down Rules (RDR) is a knowledge acquisition method which constrains the interactions between the expert and a shell to acquire only correct knowledge. Although RDR works well, it is only suitable for the problem of providing a single classification for a set of data. Multiple Classification Ripple Down Rules (MCRDR) is an extension of RDR which allows multiple independent classifications. The approach has been evaluated in simulation studies where the human expert is replaced by a simulated expert. MCRDR may provide a basis for building a general problem solver for a range of problems beyond classification.

1. Introduction

In the area of knowledge based systems (KBS), tools to solve problems can be usually classified into four categories: "Universal programming language", "General expert system tool", "Problem-specific expert system tool (shell)" and "domain-specific expert system tool" (Puppe 1993). These tools can not be used directly to acquire knowledge. Acquiring knowledge depends on the available techniques as understood by a knowledge engineer. Sometimes the knowledge acquisition process will result in a problem specific tool being built by the knowledge engineer for use by the expert (Puerta, Egar et al. 1992). In general, acquiring knowledge depends on the availability of the problem solving methods for that problem class (Kleer 1985) which leads to the question of identifying problem types.

A well-known classification of the types of problems that can be solved by expert systems is that of Hayes-Roth (Hayes-Roth 1983). He identified 10 categories: interpretation, prediction, diagnosis, design, planning, monitoring, debugging, repair, instruction and control. However, these classes do not have strong relationships to methods of building expert systems. Chandrasekaran provides a more systematic approach to problem classification(Chandrasekaran 1983; Chandrasekaran 1986; Chandrasekaran 1987), classifying problems into several generic tasks at high level and applying these tasks to system building. His approach focuses on dividing the task into sub-tasks in a high level abstraction and he considered that each task can be solved individually.

Probably KADS provides the most systematic approach of this kind to building expert systems (Wielinga, Schreiber et al. 1992). In essence KADS is concerned with a top down software engineering approach to building systems and one proceeds through various layers of analysis resulting in selection of an appropriate problem solving method and domain model. There are important differences between KADS, Chandrasekaran's Generic Tasks and other related methods, however, in essence all these methods attempt some sort of software engineering to identify or build a problem solver closely matched to the problem at hand prior to populating the knowledge base with knowledge.

However, despite the systematisation brought about through KADS and similar projects it is clear that making appropriate distinctions between problem solving methods although highly useful in building systems is a difficult and perhaps arbitrary task. Because of these difficulties, Breuker, one of the KADS originators, suggests that problems are better understood in terms of their interrelationships rather than their properties and so finds a *suite* a more useful notion than a *taxonomy* (Breuker 1994). For example, some sort of design must exist before it makes sense to consider the issue of assignment and the assignment, that is a completed system, must exist before it makes sense to consider diagnosis etc. This analysis seems superior to earlier approaches, but it also makes clear that there is probably no end to the possibilities of analysis. The one distinction the seems to remain through all analyses, including being central to Breuker's, is Clancey's distinction between classification and construction (Clancey 1985),

In Clancey's approach (1985) the task of a KBS system is to produce an appropriate answer from a given set of inputs. The system may also generate some intermediate data that is used to produce the final output. A classification system matches some important features of the input to select a solution from a pre-defined set of outputs. A heuristic classification system works based on heuristic matching methods. On the other hand, construction methods construct solutions based on the given input. The central point for the discussion here is that in a construction system, there are constraints between outputs and the construction system has to ensure that the constraints are satisfied by the proposed outputs. For example, if the outputs does not satisfy the constraints, the system should be able to change the proposed outputs, previous inputs or outputs to satisfy constraints. Another way at looking at the same distinction is that in classification systems there is a distinction between input and output, while in construction systems the input may include things that are also output. The constraints are not from one class of thing to another, inputs to outputs, but between things which may be input or output. Construction may thus be seen as a generalisation of classification.

For our concerns here the central issue is that all problem solving is in getting appropriate outputs given certain inputs. The distinction between classification and construction identifies the importance of constraints between components of the output, things that can also be input. This leads to the notion that rather than go more or less directly from inputs to outputs as in classification, it is appropriate in construction to propose a solution and then test it, or propose part of a solution which in become a further input constrains further choices for the solution etc. The importance of constraints between components of a solution underlies notions such as propose and revise or generate and test. However, although such problem solving methods facilitate solving certain types of problem, they are not essential e.g. (Gaines 1992). The central issue is that the system gives the right inputs for outputs, that the appropriate pathways from inputs to outputs can be found. The issue of the path connection between inputs and outputs can be analysed further and the reasoning between inputs and outputs may well be part of the solution. Clancey has developed the notion of "situation specific model" (Clancey 1992) to express such notions. Breuker identifies related ideas (Breuker 1994). For the discussion here, the output is whatever is desired from the system and which can be judged as right or wrong given certain inputs.

The approaches noted so far for producing appropriate outputs for inputs aim to design a problem solver very closely matched to the problem, but providing no constraints on the links and parts of links between input and output provided by the expert. Another approach is to provide a very simple mechanism for linking inputs and outputs but highly constrain the expert so that new links are added and corrections made in a way that does not degrade the performance of the knowledge base. The goal is to incrementally build up a set of appropriate and validated links (a knowledge base) regardless of the problem type. Ripple down rules (RDR) uses this approach but can only produce a single output for a given input. Obviously a single output does not provide a rich input for construction tasks. The goal of the present work is to extend the RDR approach to producing multiple conclusions for a given input (Multiple classification Ripple Down Rules (MCRDR)). Once the system is able to produce multiple conclusions (outputs/inputs) it has a basis for dealing with construction tasks, and the constraints between the outputs/inputs of a solution. The actual inference method is optional and is decided on efficiency issues, whether one proceeds in a single pass from input to output, or whether one incrementally adds to a solution and the developing solution becomes part of the input for the next cycle. Many variants are possible.

We may seem that we are suggesting a return to general purpose expert system shells (e.g. CLIPS). However, the normal use of these systems in entirely unconstrained. Knowledge engineering skill is essential in using these tools, whether they are used in an ad hoc fashion, or a systematic fashion as described above, where the whole process of selecting (or building) a tool is part of the knowledge engineering exercise. In contrast in the RDR approach here, the expert (or knowledge engineer) have no say in the details of knowledge organisation, the system is responsible for knowledge organisation, and by constraining the knowledge organisation the system is able to provide the expert with a list of conditions from which to select which will guarantee a validated rule and an incremental change to the knowledge base. A knowledge engineer is thus not required.

It should be noted that the RDR approach has much in common with methods based on Personal Construct Psychology such as repertory grids (Gaines and Shaw 1990). These methods do not constrain the expert as much as the RDR approach, they rather allow the expert to readily see the interrelationships between knowledge as it is added to the KB. They are similarly based on the idea that experts will select more valid knowledge if asked to deal with differences between cases. Similarly to the original RDR these techniques tend to deal with single classification tasks, but related to the claim here, there is a suggestion that much more can be achieved with these techniques (Bradshaw et al).

The remainder of the paper describes the various algorithms and principles underlying MCRDR (Kang and Compton 1992) and demonstrates through simulation studies that the method is viable.

2. Ripple Down Rules

The RDR approach does not use any notion of extracting or mining the expert's knowledge. RDR grew specifically from the experience gained in maintaining an early medical expert system, the GARVAN-ES1, for a number of years (Compton, Horn et al. 1989; Compton and Jansen 1990; Callan, Fawcett et al. 1991). Observation of experts during maintenance suggests that experts never provide information on how they reach a specific judgment. Rather the expert provides a justification that their judgement is correct. The justification they provide varies with the context in which they are asked to provide it (Compton, Horn et al. 1989; Compton and Jansen 1990; Callan, Fawcett et al. 1991). The context in RDR is defined as the sequence of rules that were evaluated leading to a wrong conclusion (or no conclusion). When the rule producing a new conclusion is added, this rule is evaluated only after the same sequence of rules is evaluated. The resulting structure is a set of ordered rules (if .. elsif rules) with exceptions which can themselves be ordered rules and so on (Catlett 1992). If a rule is satisfied by the data then its conclusion will be asserted unless any of its exception rules are satisfied and so on with the exceptions to the exceptions. The expert need have no knowledge about this structure and how the system appends the rule to the KB. As far as the expert is concerned he or she composes a rule of whatever generality is preferred and the system handles this rule. All rule addition is prompted by the system misclassifying or failing to classify a case.

A second advantage of the approach is that the expert can be constrained to add only valid rules. Each rule is added to the system to deal with a specific case. These cases are stored in conjunction with the rules and are called cornerstone cases. When a new rule is added, the cornerstone case associated with the rule that gave the wrong classification may be misclassified by the new rule, as any case satisfying the parent rule is passed along to the new exception rule. Therefore a new rule should be satisfied by the new case but not by the previous cornerstone case. This can be achieved by requiring the expert to select conditions for the new rule from a list of differences between the case for which the rule is added and the previous cornerstone.

For example:

old case	**new case**
TSH high	TSH high
T3 low	T3 low
FTI normal	
	TT4 high

The expert must choose either or both the conditions

FTI NOT normal
TT4 high

as conditions in the rule and can optionally chose any of the common conditions to make the rule intelligible. Such a rule is guaranteed to work on the new case but not the old case, so no further checking is required or relevant. Note that for a case that has not satisfied any rule, the difference list includes all the conditions that are true for the case and the rule can use any of these conditions, however, the rule is not evaluated until all previous rules have been evaluated.

RDR also shift the development emphasis to maintenance by blurring the distinction between initial development and maintenance. The difficulty of adding a rule to an RDR system is the same regardless of how long a KB has been under development and how large it is. This feature allows a KB to evolve along with the gradual development of domain expertise.

The major success with RDR is PIERS, an expert system used to add clinical interpretations to chemical pathology laboratory reports (Compton, Edwards et al. 1992; Edwards, Compton et al. 1993). PEIRS now has about 2000 rules, covers 25% of chemical pathology (i.e., 100 out of 500 reports per day issued by the laboratory) and is 95% accurate. Rules can deal with temporal data, allow mathematical expressions to be included and new attributes can be added at any time. PEIRS went into routine use with about 200 rules with the rest of the rules added while in routine use. Rule addition is a trivial task taking about 3 minutes per rule, so that knowledge addition for the whole system has taken about 100 hours. Most importantly, all rules have been added by an expert without any knowledge engineering or programming assistance or skill. Rule addition takes about 15 minutes per day and is a trivial extension to an expert pathologist's normal duties. A knowledge engineer/programmer was required only for the initial data modelling.

2.1. RDR Limitations

A problem with PEIRS is that a patient may have multiple independent diseases. PEIRS currently handles this problem by treating such situations as compound diseases. However, this could exponentially increase the knowledge acquisition task. A possible solution would be to separate domains and build independent KBs for each domain. However, in many domains it is not easy to separate sub-domains and a clumsy work-around would be required. The MCRDR system described below deals with this problem

A further apparent limitation of RDR is that the KB may be ill structured and considerable repetition of knowledge may result. This is not a major problem in practice, largely because of the decision list nature of the representation and because experts tend to produce very general rules (Mansuri, Compton et al. 1991; Compton, Preston et al. 1994). If the problem is significant, Gaines has proposed using the existing RDR KBS together with a data base to produce a well classified set of training cases which can then be used with the INDUCT/RDR algorithm to build a more compact RDR KBS to again be maintained by hand (Gaines 1991). We expect the likelihood of this problem occurring to be further reduced with MCRDR, as all pathways through the KBS are explored.

3. MCRDR

The aim of MCRDR is to preserve the advantages and essential strategy of RDR in dealing with multiple independent classifications. MCRDR, like RDR, is based on the assumption that the knowledge an expert provides is essentially a justification for a conclusion in a particular context. A major component of the context is the case which has been given a wrong classification and how this differs from other cases for which the classification is correct. As we shall see, the context in MCRDR is preserved differently and only includes rules that have been satisfied by the data and validation extends to differentiating the new case from a range of different cases.

3.1. Inference

The RDR inference operation is based on searching the KB represented as a decision list with each decision possibly refined again by another decision list.

Once a rule is satisfied no rules below it are evaluated. In contrast MCRDR evaluates all the rules in the first level of the KB. It then evaluates the rules at the next level of refinement for each rule that was satisfied at the top level and so on. The process stops when there are no more children to evaluate or when none of these rules can be satisfied by the case in hand. It thus ends up with multiple paths, with each path representing a particular refinement sequence and hence multiple conclusions.

The structure of an MCRDR knowledge base can be drawn as an n-ary tree with each node representing a rule. Fig 1 shows such a structure and also shows the inference for a particular case.

The inference process can be understood in terms of capturing 'paths', as shown below in Fig 2. When paths are produced there are a number of questions about whether the path produces a classification, whether the classification is redundant because it is produced elsewhere etc.

3.2. Knowledge Acquisition

When a case has been classified incorrectly or is missing a classification, knowledge acquisition is required and can be divided into three parts. Firstly, the system acquires the correct classifications from the expert. Secondly, the system decides on the new rules' location. Thirdly, the system acquires new rules from the expert and adds them to correct the knowledge base.

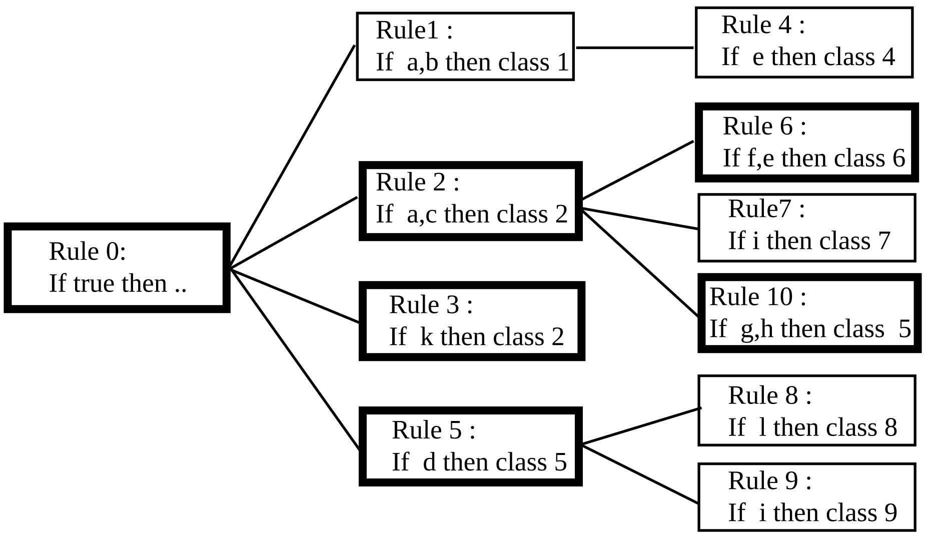


Fig 1. An MCRDR KBS. The highlighted boxes represent rules that are satisfied for the case {a,c,d,e,f,h,k}.

Path 1 [(Rule 0, ...), (Rule 2, Class 2), **(Rule 6, Class 6)**]	Info 1

Path 2 [(Rule 0, ...), (Rule 2, Class 2), **(Rule 10, Class 5)**]	Info 2 **[4]**
Path 3 [(Rule 0, ...), **(Rule 3, Class 2)**]	Info 3
Path 4 [(Rule 0, ...), **(Rule 5, Class 5)**]	Info 4 **[2]**

Fig 2. Pathways through the knowledge base from Fig 1. The rules producing conclusions are highlighted. 'Info n [...]' indicates other rule numbers with the same classification.

It is likely that experts may find the system more natural if the order of steps two and three are reversed, thereby better hiding the implicit knowledge engineering that is going on. However, the order is not crucial in terms of the algorithm.

3.2.1. Acquiring New Classifications

Acquiring new classifications is trivial, the expert simply needs to state them. For example, if the system produces classifications **class 2**, **class 5**, **class 6** for a given problem , the expert may decide that **class 6** does not need to be changed but **class 2** and **class 5** should be deleted and **class 7** and **class 9** added.

3.2.2. Locating Rules

The system should find the locations for the new rules that will provide these classifications. It cannot be assumed that the correct location for a new rule is as a refinement for one of the rules giving one of the wrong classifications. It may be a quite independent classification and the wrong classification is simply wrong. The possibilities are shown in Table 1. Note the idea of stopping rules, rules that make no conclusion, or rather give a null classification. Stopping rules play a major role in MCRDR in preventing wrong classifications being given for a case.

As well as attempting to decide whether a classification is best seen as a refinement or an independent classification, we note that in some ways it does not matter - both are workable solutions for any classification. The key effect of the rule location is to effect the evolution and maintenance of the knowledge base. If there is a strategy that tends to add rules at the top level, the knowledge base will cover the domain more rapidly but with a greater likelihood of error. If the strategy tends to add new rules at the end of paths, domain coverage will be slower but with less errors from the new rules, simply because they see less cases . This is shown in Fig 3. A rule can also be placed at appropriate intermediate levels. One can also change these strategies as the system develops. These decisions are a new type of knowledge engineering consideration - the issue is what type of development is appropriate for a particular domain, rather than the structure of the knowledge.

Wrong classifications	**To correct the KB**
Wrong classification to be stopped	Add a rule (stopping rule) at the end of path to prevent the classification
Wrong classification replaced by new classification	Add a rule at the end of path to give the new classification
A new independent classification	Add a rule at a higher level to give the new classification

Table 1. The three ways in which new rules correct a knowledge base.

Decisions about the evolution of the knowledge base do not have to be knowledge engineering decisions that override any preferences of the expert. Rather, the expert can be free to make any type of decision, but the interface can be designed so that the expert is more likely to add rules towards the top or the bottom. For example, two alternative strategies might be to ask the expert to select the important data used in reaching the conclusion or to delete the irrelevant data. If selected data satisfies a pathway the rule is placed at the bottom of the pathway or at whatever level the conditions selected reach down to. One may expect that the expert's behaviour will be conservative, he or she will either select few conditions or remove few conditions. The method used here is for the (simulated) expert to select important data in the case. We call the selected conditions a "mini-case".

Other strategies may include asking the expert to make the normal difference list selection (see below) and then asking the expert to select from a list which includes all the conditions for all the possible pathways where the rule may go. Again the expert's behaviour can probably be biased by asking for conditions to be removed or alternatively selected. In this paper we have not explored these strategies further as they do not determine whether or not the method works, they merely help tune it to a particular domain problem and application.

3.2.3. Acquiring Rule Conditions - Rule Validation

Verification and validation are concerned with ensuring that a KBS system performs as it is meant to. Verification research is normally concerned with ensuring the internal consistency of a knowledge base The normal approach in verification is to attempt to reduce the KB to pathways from data to conclusions and then look at the relationships between these pathways, the data they use, the intermediate conclusion they establish etc. (Preece, Shinghal et al. 1992). Validation, in terms of maintenance or incremental acquisition, is concerned with testing whether other cases previously correctly classified will be misclassified by a new rule, as well as ensuring the new rule covers the new case. We are mainly interested in validation, in particular in validating a KBS by testing it on cases.

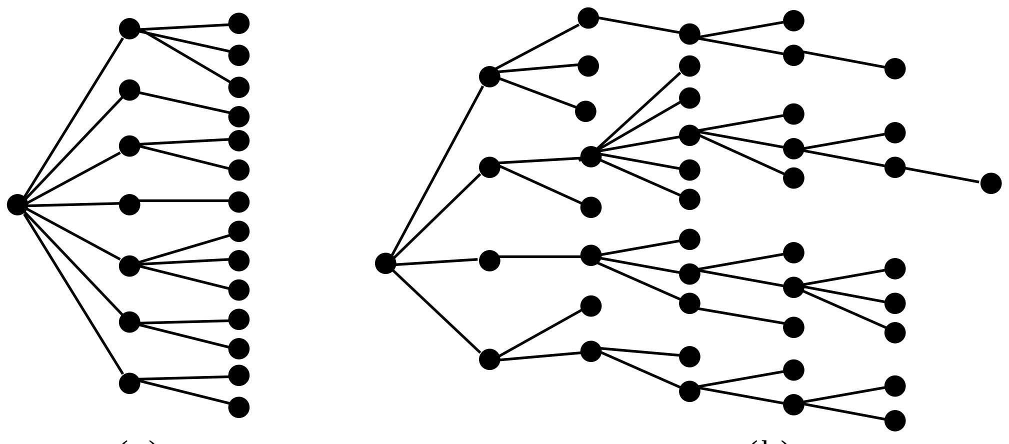


Fig 3. Structure of the MCRDR tree if rules are added mainly at the top (a) or mainly as refinements (b).

A standard technique is to use a database (Buchanan, Barstow et al. 1983) of standard cases. In this situation one depends on the cases being representative of the cases the system is meant to cover. With RDR, a case is associated with a rule because the rule is added to deal with that particular case. A new rule must distinguish between the case that caused its creation and the case associated with the rule that gave the previous incorrect classification. With MCRDR, a number of cases (cornerstone cases) can reach a new rule and the higher in the tree the more cases can reach the rule. The new rule should distinguish between the new case and all of these cornerstone cases. That is, MCRDR has multiple cornerstone cases for a rule, compared to RDR where there is a one per rule.

A rule at some level can be hit by all the cases associated with its siblings at the same level and their children lower in the system. So, the rule has to be made sufficiently specific so that none of these other cases satisfy the rule. However, it does not matter if other cases which include the same classification reach this particular rule. If a rule is added at a level below the top level, only cases which satisfy the parent rule above need to be considered as cornerstone cases. Note that as the system develops, cases may arise which correctly satisfy a rule, but may be added to the system because a rule is needed elsewhere to add a further classification. Such a case will become a cornerstone case for new rules below the rule it satisfies and for which the classification is correct. As the tree develops, the rules lower down will naturally have less cornerstone cases associated with them.

The aim then is make a new rule sufficiently precise so that it satisfies only the case it is being added for and no other stored cases, except that it does not matter if it happens to satisfy cases which include the same classification. The algorithm for selecting conditions to make the rule sufficiently precise is very simple and some discussion is needed why a more sophisticated approach was not chosen. Consider a new case A and two cornerstone cases B and C. In creating a new rule, one may imagine that expert should choose at least one of conditions from

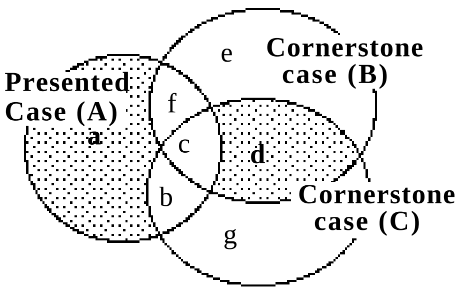
(Case A - (Case B   Case C))

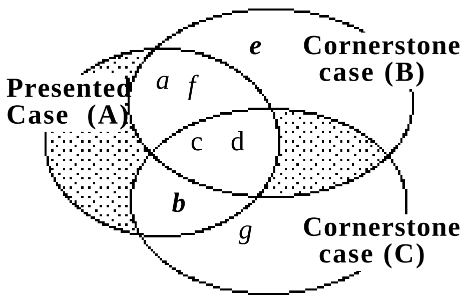
or

negated conditions from the ((Case B ∩ Case C) - Case A).

However, as seen in Fig 4, these may be empty - leading to the situation where no rule conditions can be found. Alternatively the difference list may contain only trivial conditions that are irrelevant. In other words there are no common conditions that distinguish the presented case from all the cornerstone cases, but a number of different conditions distinguish different cases and these conditions must all be included in the new rules.

The algorithm we use is as follows. First, the system can form a cornerstone case list which can reach the new rule and should be distinguished from the presented case. The expert is asked to select from a difference list between the presented case and one of the cornerstone cases in the list of cornerstone cases to be considered. The system then tests all cornerstone cases in the list against the conditions selected and deletes cornerstone cases from the list that do not satisfy the condition selected. The expert is then asked to choose conditions from a difference list between the current case and one of remaining cornerstone cases in the list. The conditions selected are added as a conjunction to the rule. The system repeats this process until there is no cornerstone case in the list which satisfies the rule. A crucial question for the following evaluation is whether this process requires too many cycles of adding conditions to rules.





(a)

Difference list between A and (B and C)	***a* *NOT d***

Difference list between A and (B∩ C)	**empty**
Difference list between A and B	***b* *NOT e***
Difference list between A and C	***a, f* *NOT g***

(b)

Fig 4. Two similar cases in which difference lists are to be found to distinguish the presented case A from two cornerstone cases, B and C. Note in case (a) the difference between the presented case and the intersection of the other cases provides a suitable condition. For case (b) the new rule has to include conditions both from the difference list between And B and from the difference list between B and C.

After the system adds a rule with the selected conditions, it tests the remaining cornerstone cases associated with the parent rule and any cases which can satisfy the new rule are saved as a cornerstone case of the new rule. Note again that it is permissible for cases which include the classification given by the rule to satisfy the rule, and the saved cornerstone cases include such cases.

Finally the new case is added to the cornerstone case data base. The lists of cornerstone cases for the other rules correctly satisfied by the case (ie. giving a correct classification for the case) are also updated to include the new case. The system is now ready to run another case and, if the classifications provided are incorrect, for more knowledge acquisition.

It should be noted that MCRDR systems may be developed for a whole domain or incrementally, a sub-domain at a time to avoid demands on the expert. That is, cases which have more than one classification may only be given one classification initially. In dealing with sub-domains incrementally, the algorithm is the same except for the extra requirement of consulting the expert as to whether the new classification may apply to the each of the cornerstone cases which can satisfy the rule. However, the feature remains that as the rule becomes more precise to exclude a specific case, the other cases that are also excluded are not considered further.

3.3. Evaluation

Evaluation of knowledge acquisition techniques is a new area of study because of the difficulty of dealing with experts in a way that provides reproducible and comparable data, as well as the obvious problem of finding experts with the time available to participate in such studies. One approach has been in the Sisyphus projects (Linster 1992; Gaines and Musen 1994). However, in the Sisyphus projects there is no actual knowledge acquisition from experts, the aim of Sisyphus is to evaluate approaches to developing a domain model and a problem solving method. The issues concern how easily and how well a given methodology or tool can produce an appropriate problem solver for the given Sisyphus problems. The actual population of the knowledge base with knowledge is of little concern, the knowledge is given in the specifications. Modelling is the central issue. In contrast, with machine learning evaluation the issues are the performance of a KBS built from test cases. The initial modelling is of little concern.

In earlier studies we have developed a machine learning style evaluation for RDR (Mansuri, Compton et al. 1991; Compton, Preston et al. 1994). In essence, these studies assess an incrementally developed RDR KBS. The KBS is built by sequentially evaluating a large data base of cases with the developing KBS and passing any cases that are misclassified by the system to an expert for an new rule to be added. In these studies the expert is a simulated expert. We have used the same approach to evaluate MCRDR.

The critical questions with MCRDR are the performance of the KBS - how well does it work on test cases; secondly, the size of the KBS, as there is no attempt to make the KB compact as with inductive machine learning methods; and thirdly the complexity of the KA task. It has been shown in the PEIRS studies that the KA task in RDR is very small and easily managed by experts as a small extension to their normal duties. In MCRDR, the expert has essentially the same task of selecting from a difference list but this may be repeated a number of times to deal with the various cornerstone cases and may become tedious.

The simulated expert should perform similarly to a human expert. The role of the expert is to identify important features in the case which justify why an MCRDR classification should be changed. A simulated expert able to identify important features is essentially a rule trace of the same case run through another KBS with a high level of expertise in the domain. In these studies the other KBS has been built by applying the INDUCT/RDR machine learning program to the total set of cases available (Gaines and Compton 1992). INDUCT./RDR is the machine learning algorithm which uses the INDUCT algorithm and can produce the RDR format knowledge base (Gaines and Compton 1992). The reason for choosing Induct is that not only does it perform similarly to other machine learning techniques such as C4.5 (Gaines 1989), but the INDUCT/RDR tends to produce knowledge bases which are much smaller than those produced by other machine learning approaches (Gaines and Compton 1992). The size of the INDUCT/RDR KB is thus a "gold standard" for how compact a knowledge base should be.

We can set several levels of expertise by manipulating different strategies to choose conditions. One would expect the best performance from a system where conditions are selected from the intersection of the difference list and the INDUCT rule trace (recalling that the difference list is a set of valid conditions from which to construct a rule). The worst performance will come from a system where a condition is randomly selected from the difference list. Intermediate performance will come from a mixture of the two. Note that we do not expect the best simulated exert to out-perform a human expert. We are unable to tell which of the conditions in the rule trace are the most important, while one assumes that a human expert knows what are the most important features in a case leading to a conclusion. Secondly, we include only satisfied rules in the rule trace, although the actual sequence of rules evaluated in RDR is important in determining what conditions are necessary in rules lower down the pathway. Further, as will be seen below, even with the best simulated expert it was frequently necessary to select conditions at random from difference lists.

3.3.1. Experimental design

Two different domains are used for the evaluation (Table 2). One of data used were thyroid cases that had been run through the GARVAN-ES1 KBS to ensure a consistent classification. The other is "tic-tac-toe" data from Irvine machine learning data set. We tested the system on these single classification domains as all the problems the system may have in terms of size and repeated knowledge acquisition would show up more in a single classification domain, where the system only has disadvantages and no advantages. These particular domains have been well studied and in particular were used to evaluate RDR (Mansuri, Compton et al. 1991; Compton, Preston et al. 1994) so that comparisons between the two systems could be made. The GARVAN-ES1 cases cover a fairly consistent period in a larger 10 year data base of 45,000 cases that has been studied (Gaines and Compton 1994). A smaller subset has been available as part of the UC Irvine machine learning database. "Tic tac toe" data set is randomized to get rid of any artificial effect (it is a collection of all possible result in the game of tic-tac-toe).

Domains	**Total cases**	**Test cases**	**Type of collections**

GARVAN-ES1	21822	Last 6822	historical collection of natural data
Tic-tac-toe	1000	Last 300	randomized data

Table 2 Systems built using these cases should have some resemblance to a real system built because both data set are in fact randomized data (GARVAN-ES1 is collection of historical data). The last 6822 cases and 300 cases in both domains are used as a test case data set.

An INDUCT/RDR KBSs were built using the entire 21822 cases in GARVAN-ES1 and 1000 cases in the 'tic tac toe" domain so that they presumably had total expertise for the domain. Note that because the classifications for the cases in GARVAN-ES1 were produced by running the cases through the GARVAN-ES1 KBS the classification were consistent and no pruning of the inductive KB was required. In the case of tic-tac-toe the data set covered all solutions and was also consistent.

In the experiments, the system starts with an empty KB and it tests each case from case 1 to case 15,000 in GARVAN-ES1 and case 1 to case 700 in tic-tac-toe. The last 6822 cases in GARVAN-ES1 and 300 cases in tic-tac toe are test data. Note that this does not follow the protocols used in machine learning evaluations for randomising training and test data, however it does correspond to the historical reality of building a system over a period of time and testing it in the next period.

Note that the data here differ from (Mansuri, Compton et al. 1991; Compton, Preston et al. 1994). The data includes many boolean attributes, most of which will always be false. e.g. *pregnant, on-t4, following surgery, I-131 treatment* etc. These have normally been derived from the clinical notes on the request form from the referring clinician. The clinician may sometimes include one, or at most two, of these as likely key factors in explaining any results. In the earlier study, both Induct and the random selection of conditions used in some of the simulated expert studies used such attributes only when they were TRUE. From past experience rules almost never use the FALSE value of such attributes. This strategy was followed to ensure that the simulated expert was reasonable and INDUCT produced a small knowledge base even with a small training set. In these later studies attributes with the value FALSE have not been excluded, firstly because the results are adequate and secondly to provide better comparisons with further studies on other machine learning data sets. Note that the existence of the FALSE valued attributes only affects MCRDR when rules are placed at the top of the tree and there is no difference list. This was in fact the most common strategy used, so that the results presented are "worst possible".

To simplify the experimental design we chose to put new rules at the either the top or bottom of rule pathways. We imagined that putting rules at the top would be a worse situation with respect to the number of cornerstone cases to be seen. The data presented here covers only rules added at the top of pathways.

In the study the following levels of expertise are used:

Clever expert	selects all conditions from the Induct rule trace.
Moderate expert	selects one condition from the intersection of the difference list and Induct rule trace
Stupid expert	selects one condition from the difference list
Clever expert (RDR)	selects 4 conditions from the intersection of the difference list and Induct rule trace.

Note that we are not concerned with the precise types of expertise involved in these studies, the results are meant to be a qualitative indicator of the size and performance of the knowledge base changes with expertise. Note that a different strategy was used for the RDR clever expert and MCRDR clever expert and further studies will be carried out using the same protocol. However, it will make no difference to the results as the MCRDR moderate expert with a lower level of expertise than either of the clever experts produces comparable results.

A further difficulty arises with the "stopping rules", rules that give a null classification and whose purpose is to prevent a classification being made. Some 80% of the Garvan cases used do not have any thyroid abnormality and should return the null classification. If at any stage these are given a classification by the MCRDR system a stopping rule is required. The null classification is also the default classification given by Induct (because it is the majority classification in the data base) so that there is no Induct rule trace for such cases. To use the simulated expert in this situation, the cornerstone case from the rule giving the wrong classification is run on Induct. Negated conditions are selected from the difference list which are also in the the Induct trace. That is, a condition which Induct considers important in reaching the conclusion for the cornerstone case also discriminates the cornerstone case from the case requiring the classification to be stopped. This condition is used in its negated form. This strategy was used where possible for the moderate and clever experts but frequently such a condition did not exist and a condition had to be randomly selected from the difference list.

3.3.2. Results

Fig 5 show the error rates of various MCRDR systems as they develop. The error data are given by testing the cases in the data base. Note that the default error rate is 22.7% as the null classification is correct for 77.3% of cases in GARVAN-ES1, and are 62.6% for the 'tic tac toe' data set. The moderate expert, stupid expert and RDR clever expert all have higher error rates than the Induct or the clever MCRDR at the beginning. While the stupid expert remains at a high error rate, the other methods are all comparable to Induct once sufficient cases are seen. Note that, for Induct as well as the other methods, the error rate continues to fall as more cases are added to the training set. As shown by Catlett, this is a common phenomenon with induction applied to very large training sets (Catlett 1992).

Although the error rates for all but the stupid expert are reasonable, the critical question is whether these results are achieved at the cost of increased knowledge acquisition. Fig 6 shows the amount of knowledge acquisition required to achieve the error rates in Fig 5. Note that the Y axis indicates the number of knowledge acquisition incidents or cases that had to be dealt with as being misclassified. For any given case more than one rule may be added. In contrast in the RDR system each case is dealt with by a single rule so that the number of cases seen is the same as the number of rules. Note that the performance of the MCRDR systems is at least as good as an RDR system. A moderate expert RDR system is not shown here but in the earlier studies the final moderate expert KB was 50% bigger than the clever expert KB in the GARVAN-ES1 domain. Here the sizes are comparable. The final sizes are shown in Table 3.

Another crucial consideration with MCRDR is the complexity of the KA task, i.e. the number of difference lists the expert must select from to make a sufficiently precise rule. This is shown in Table 4. If we exclude the stupid expert as not being representative of a human expert, then on average the expert only has to see 2-3 difference lists. If we exclude the lower level stopping rules and consider only top level rules reached by all cornerstone cases in the system, an average of 5.5 difference lists have to be dealt with. Note that in this study all rules giving conclusions were put at the top level.

The number of cases to be dealt with increases as the knowledge base increases in size and more cornerstone cases are stored (Fig 7). Note that Fig 7 is again a worst case because the graph covers only rules added of the top level. It can be seen with a clever expert that the worst case is an average of 3 cases per rule, and with a moderate expert an average of 4 or 5 cases per rule in both domains.

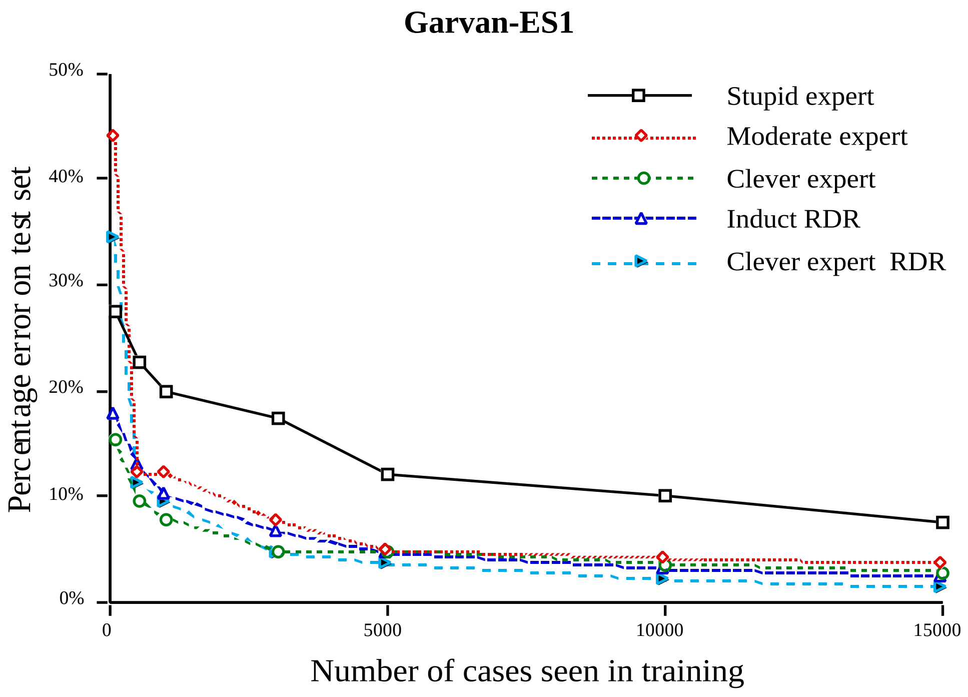
4. Discussion

4.1. Size and Performance of MCRDR

These results demonstrate that a manually built MCRDR expert system performs similarly to an RDR system and an inductively built expert system. The error rate versus number of training cases for induction or number of cases evaluated by the manual MCRDR system are very similar. It should be noted that we have not yet carried out tests on randomised data to get confidence limits. However, from results elsewhere in these Proceedings on randomised data with single classification RDR, we would expect little effect on the conclusions here.

It should be noted that we would expect a human expert to do better than the synthetic expert or induction for small training sets. In a previous study a human expert built an RDR system with a 12% error rate from 291 training cases versus 74% for C4.5 for the same training cases (Mansuri, Compton et al. 1991) (As noted above, Induct performs similarly to C4.5). This test set was 9514 Garvan thyroid cases, but importantly the training set was 291 unusual cases which the original Garvan-ES1 expert system had used as test cases or which the system had misinterpreted at some stage (Horn, Compton et al. 1985). Because the cases were diverse this training set was a challenge for an inductive approach but not for an expert.

It should be noted that the clever and moderate expert produce similar sized knowledge bases and have a similar performance. This is to be expected. The use of at least one condition from the Induct trace ensures a rule has some relevance, while the stopping rule approach ensures that a rule that is too general is made sufficiently narrow as required. These simulated experts produced a knowledge base that was only about three times the Induct KB size. When we consider that inductive methods aim at producing compact knowledge bases and that Induct/RDR tend to produce considerably smaller KBs than other methods (Gaines and Compton 1992), this is a very good result. Note that we propose both RDR and MCRDR for domains where large data bases of well classified cases are not available for induction. In many domains large data bases are available, but without suitable and consistent classification information to enable them to be used for induction. Once a manual expert system has been built there is the opportunity to use this to provide well classified cases which can then be passed to a machine learning system to produce a more compact KB (Gaines 1991). However, we suggest that with a KB approximately double the smallest possible size this is probably not necessary and certainly the size is not a significant problem during development.



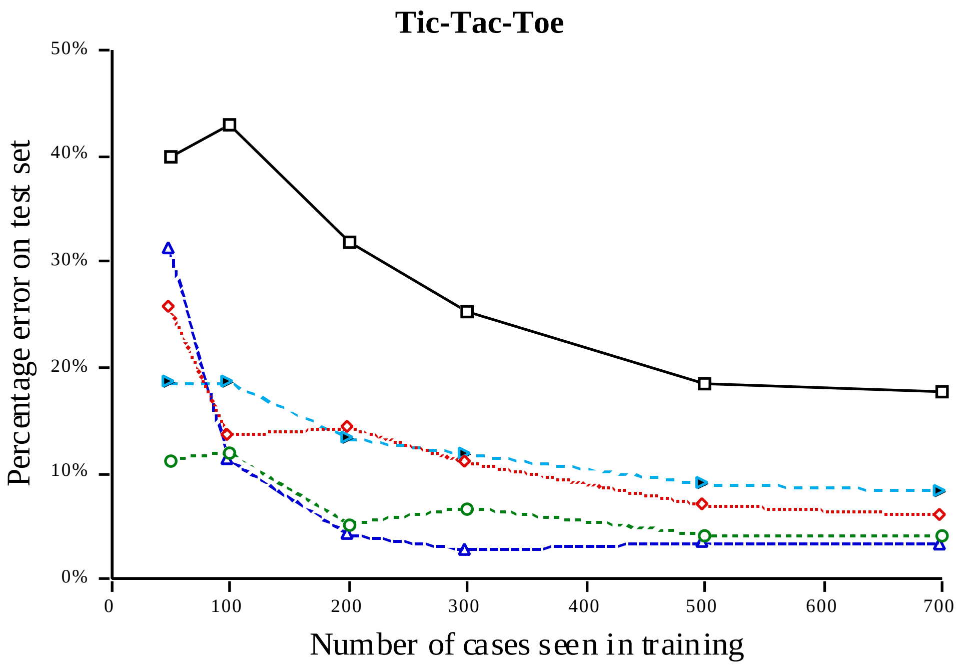
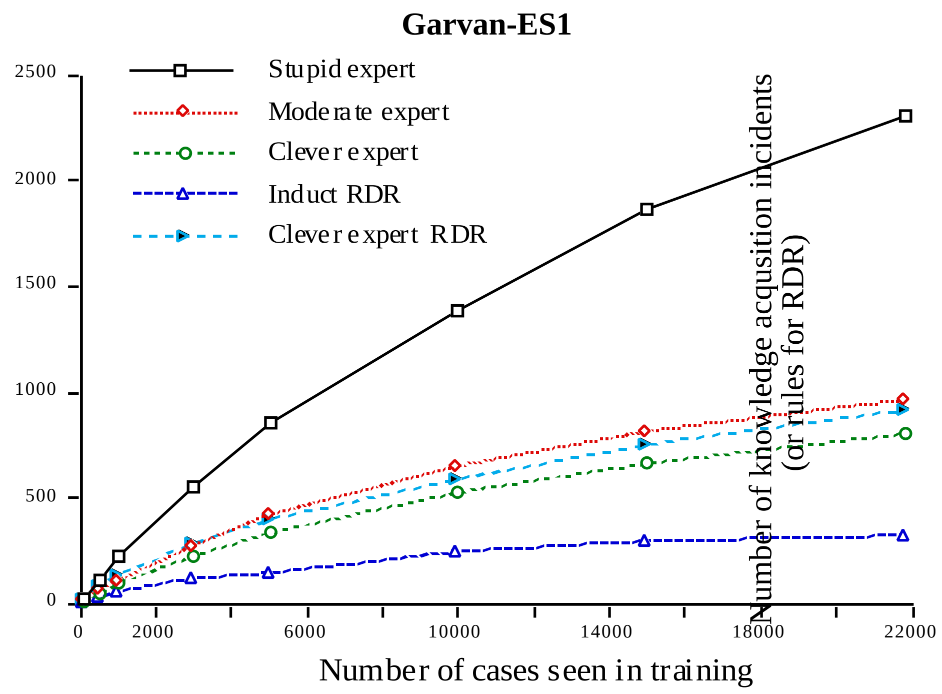


Fig 5. Error rates on a test set (6822 cases) with different methods of building a KBS. The x axis indicates the number of cases used as a training set for Induct or evaluated in building an RDR or MCRDR system and the y axis is percentage errors for all the cases in the test set for this sized KB



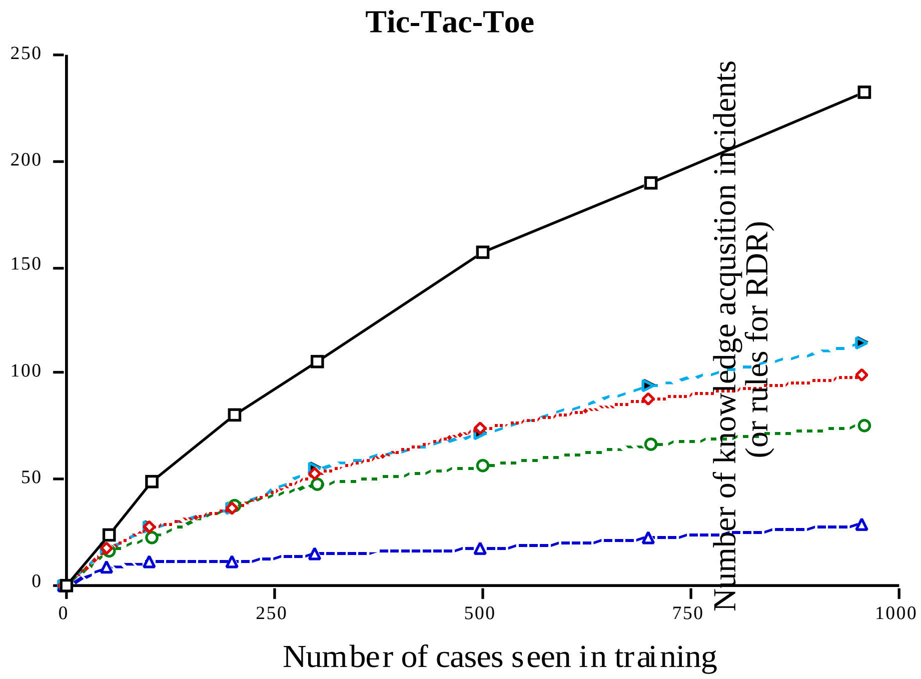
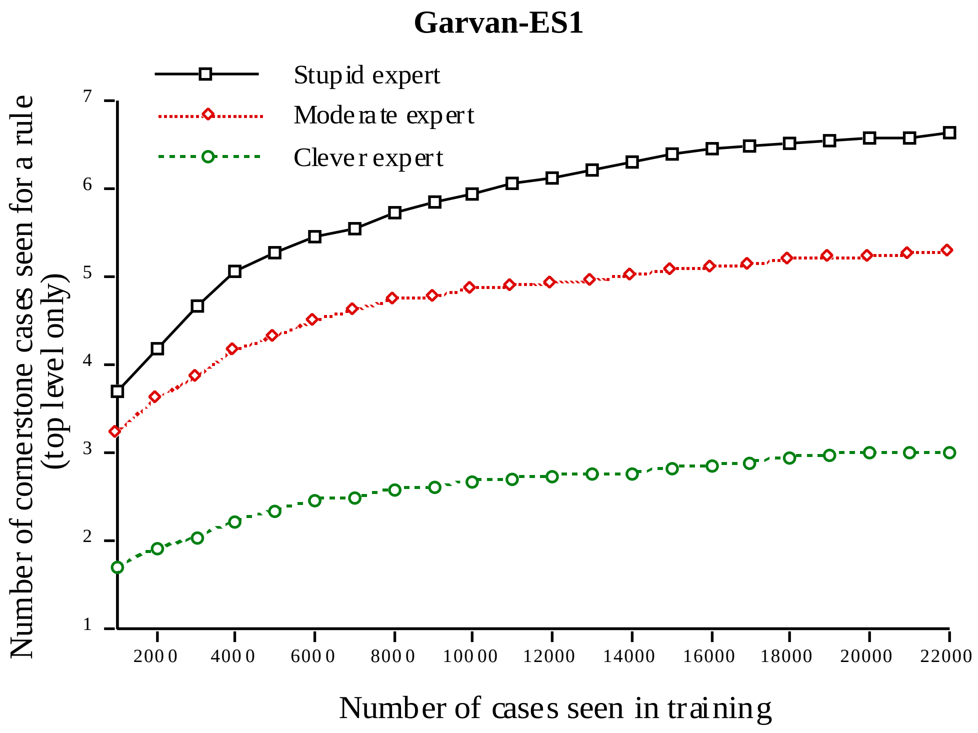
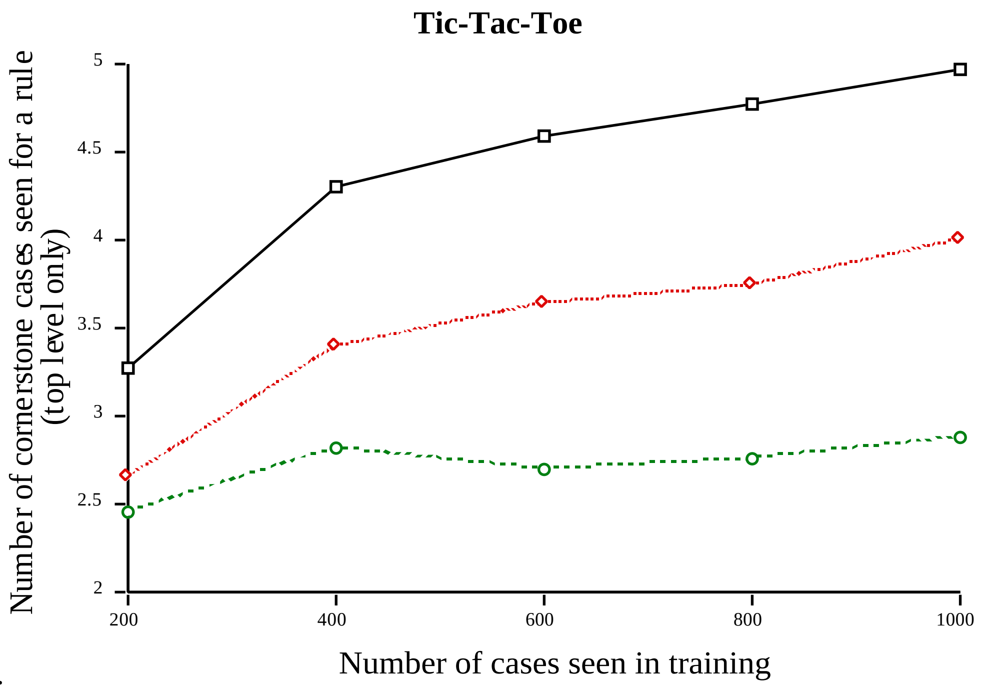


Fig 6. This shows the number of knowledge acquisition incidents versus developing size of the KB. Alternatively, it is the number of corrections made versus total number of cases seen. For the RDR clever expert a single case produces a single rule so that the Y axis is also the number of rules. For the Induct RDR KB the Y axis is again the number of rules.





s.

Fig 7 The average number of cornerstone cases seen by the expert as the knowledge base develops. Note that the data include only the worst case of rules added at the top level.

Domain	Stupid Expert	Moderate Expert	Clever Expert	Clever Expert RDR	Induct RDR
GARVAN-ES1	2300	974	805	924	332
Tic-Tac-Toe	233	99	75	114	29

Table 3. The number of rules made by an MCRR system..

TIC TAC TOE	Average cases/rule	Maximum cases/rule	Average cases/rule *top level rules*
Moderate expert	2.55	13	5.45
Clever expert	1.84	14	3.01
Stupid expert	3.22	17	7.13

TIC TAC TOE	Average cases/rule	Maximum cases/rule	Average cases/rule *top level rules*
Moderate expert	2.25	8	4
Clever expert	1.87	7	2.86
Stupid expert	2.45	10	4.96

Table 4. The first column indicates the average number of cornerstone cases or difference lists seen by the expert in adding rules. The second column indicates the maximum number of cornerstone cases or difference lists seen by the expert in adding rules. The third column indicates the number of cases seen by rules added at the top level, i.e. rules which can be reached by every cornerstone case in the system. Note that the only one difference list is seen by the expert in RDR because RDR handles only a single cornerstone case for each rule.

Finally the size of an individual knowledge acquisition task is on average about 2-4 times as great as the RDR knowledge acquisition task because of the requirement to deal with extra cornerstone cases (Fig 7 and Table 4). In the worst case the expert has to consider 17 cases, but this is very rare. It is likely to be due to the random selection of conditions that sometimes occurs with the simulated expert and is unlikely to occur with a real expert. The expert maintaining PIERS takes about three minutes per case to add a new rule. Expert's report that the major part of this goes into thinking about the case and deciding on the classification and its appropriate wording Even if we double this time for MCRDR, it is more than reasonable.

We anticipate better results with a human expert as opposed to simulated experts. With RDR the actual sequence of rules that fail in a pathway is critical as the rule above a rule in RDR removes cases before they reach the later rule allowing the later rule to be quite general. We only used satisfied rules in the Induct rule pathways. Further it was frequently found that there was no intersection between the difference list and the Induct rule trace so that a condition had to be randomly selected from the difference list. In this study this is a major problem with stopping rules. Because all pathways are explored with MCRDR we expect repetition to be less of a problem with MCRDR than RDR so less knowledge acquisition incidents are required. The only case where this shows up clearly in this study is with the moderate expert.

Regardless of whether or not MCRDR will produce a more compact KB than RDR, this study clearly demonstrates that it does not greatly increase the knowledge acquisition required in a single classification domain. This implies that in a multiple classification problem domain it will provide a viable solution for the incremental development of KBS.

We propose to use the same experimental design on other datasets used in the machine learning literature, to confirm that these results are domain independent. We also propose to randomise the data and repeat these experiments. However this may not be entirely appropriate with the GARVAN-ES1 data as it is taken from a real domain and the cases are in the same order as they occurred in reality. Building a system by taking sequential cases and testing it on cases that occur after the system is assumed complete (here at 15,000 cases) is exactly the scenario that occurs in the real world where one expects the system to apply to future cases.

4.2. Implications from MCRDR

As a first minor point we should note that MCRDR hold out some promise of reducing the brittleness of KBS. An MCRDR system, unlike a conventional expert system, keeps a history of corrections to rules. A case may follow a certain pathway and reach conclusion X, but elsewhere in the KB it may also be able to satisfy a rule which changes X to Y but without satisfying the whole pathway. We have carried out preliminary studies investigating whether such relationships could be used to enable a system to warn when its conclusion may be erroneous and have found that the system gave useful warnings (Kang and Compton 1992). We propose to expand these studies now that the MCRDR system has been validated. We speculate on the relationship between keeping and using a history of corrections to a knowledge base and human learning from experience.

The major issue is the application of MCRDR to construction. It seems likely to be inefficient to have a single pass from data to conclusions. With any RDR system one has to wait till suitable cases occur (or generate synthetic cases) to add the appropriate knowledge. In a construction system for a specific case there may not be sufficient input to exercise the knowledge that is already included, however a partial solution may well then match against other knowledge in the system, suggesting that is more efficient to keep adding conclusions to the input and repeating the inference process rather than having a single pass. Similarly it is likely to be more efficient to assume various conditions in rules are satisfied if they are as yet unknown, but represent possible output from the system. All this is perfectly conventional inferencing. We have previously developed a system for configuring ion chromatography equipment (Mulholland, Preston et al. 1993) but using multiple single RDR KBS which had been developed by induction. The inference mechanism assumed unknown conditions were true and if any individual conclusion attributes had the same value after an inference cycle this was added to working memory and the inference repeated. Various criteria again, perfectly conventional, can be used terminate the inference.

Obviously there are many variants of the same process that can be identified. The software engineering approach to knowledge acquisition attempts to identify the best for the particular problem. In the RDR framework, it doesn't matter whether or not the best approach is adopted. As long as the approach will work, as long as there is a framework for presenting the expert with case differences it appears the problem solver will gradually learn how to deal with the problem. The task in any knowledge acquisition incident remains the same, selecting from a difference list, but the speed with which the system converges to a solution may vary and may be improved with a different inference strategy.

Knowledge acquisition for MCRDR for construction as well as classification is concerned with constraining the expert in the choice of conditions to go in a rule and locating where the rule should go in the KB. There are a number of ways this can be achieved. The simplest is that the new rule must go at the top of the tree or at a level where the rules above use only to input data not to any conclusions. Case difference refer only to the actual input. The case is constantly re-run till a complete solution is developed. This is obviously a very crude approach. A second approach considers the whole case including the input and output components. The case is run and one or more components of the solution is incorrect. To produce one of these, a new rule has to be added at the bottom of the path or at some level higher and a stopping rule added at the bottom. The new conclusion to be reached is added to the input and the case re-run. Only those previously identified locations which are still reachable by the case are suitable for a rule to be added that concludes this particular output.

The second issue is the case differences to be presented to the expert. The system records the sequence of rules that were satisfied in reaching the conclusions including the repeated cycles. RDR does the same the but the sequence comes from a single pass. One can identify when in this sequence the new rule will be reached. All conclusions following this rule are deleted from the case. Case differences are then used in the same way as multiple classifications, except that the current case includes the partial solutions so far generated. It is also stored in this way as a cornerstone case. The case is re-run after each new rule is added till a complete solution is built up.

This is a fairly simple solution based closely on the notion of a context pathway which is the basis of RDR so that one would expect that it should work. However, it remains to be investigated. There are also other possible methods that could be used. The central issue is that as long as these methods fulfil the requirement of validated rules added in context, they are likely to provide for incremental knowledge acquisition without the assistance of a knowledge engineer. The choice between methods is concerned with matching the knowledge acquisition performance to the requirements of the domain e.g. rapid coverage with many error to be corrected, or gradual incremental coverage. The demands on the expert, will also be a consideration. This appears to be a new type of knowledge engineering decision.

We have suggested that it may be possible to use machine learning to help improve an evolving knowledge base (Compton, Kang et al. 1993). An apparent disadvantage of RDR style KBSs, the repetition, may in fact provide a basis for improving the organisation. A simple example is that in a classification or multiple classification system no organisation of the conclusions is required. However in a configuration system where one wishes to assume unknown conditions may be true, it is imperative that the conclusions be organised as attribute values pairs so that arrival at a specific attribute value pair for a conclusion means other conditions using the same attribute are no longer unknown. It seems that the non co-occurrence of conditions in rule pathways could be used to suggest that these may be different values of the same attribute as the knowledge base develops and so improve performance.

Our conclusion is to suggest is that the effort required by the software engineering approach to knowledge acquisition may be unnecessary. We have shown here that an MCRDR system will happily handle both single and multiple classification tasks. We hypothesise that with some extensions the same approach will be able to handle construction tasks as well, initially at least configuration. Of course the conclusion with respect to construction tasks is speculative. However in the limited classification studies so far we have been able to actually build systems without knowledge engineering support apart from development of the domain model. There does not seem to be clear evidence yet that the software engineering approaches lead to knowledge bases that can be actually populated with knowledge by an expert without knowledge engineering support.

With respect to the necessity of building an initial domain model, we have suggested that an RDR knowledge base may provide sufficient information to be able to use machine learning to adapt the model as the KB develops. In the meanwhile we have argued elsewhere that local refinement allows an inadequate domain model to be refined in context (Compton, Edwards et al. 1992; Edwards, Compton et al. 1993) and that with this facility an expert can define a domain model for a sophisticated domain unaided (Preston, Edwards et al. 1994).

In conclusion it seems that despite the efforts of AI to develop methods which will give optimal models perhaps testing and fixing is perhaps a an easier way to build useful systems than investing the effort demanded in trying to develop the "right" model (Menzies and Compton 1994)

5. Acknowledgment

This work has been funded by the Australian Research Council. The authors are grateful to Brian Gaines for making available the Induct software and his encouragement in these studies.

6. References

Breuker, J. (1994). Components of Problem Solving and Types of Problems. A Future for Knowledge Acqusition: Proceedings of EKAW'94 Eds. L. Steels, G. Schreiber and W. Van de Velde. Berlin, Springer-Verlag. 118-136.

Buchanan, B., D. Barstow, et al. (1983). Constructing an expert system. Building expert systems Eds. F. Hayes-Roth, D. Waterman and D. Lenat. Reading Massachusetts, Addison Wesley. 127-67.

Callan, J. P., T. E. Fawcett, et al. (1991). CABOT: an adaptive approach to case-based search. The Twelfth International Joint Conference on Artificial Intelligence, Sydney, Morgan Kaufmann.

Catlett, J. (1992). Ripple-Down-Rules as a mediating representation in interactive induction. Proceedings of the Second Japanese Knowlege Acquisition for Knowledge-Based Systems Workshop, Kobe, Japan,

Chandrasekaran, B. (1983). “Towards a taxonomy of problem solving types.” AI Magazine **4, No1**(Winter-Spring): 9 - 17.

Chandrasekaran, B. (1986). “Generic tasks in knowledge based reasoning: high-level building blocks for expert system design.” IEEE (FALL): 23-30.

Chandrasekaran, B. (1987). Towards a Functional Architecture for Intelligence Based on Generic Information Processing Tasks. The Twelfth International Joint Conference on Artificial Intelligence, Milan, Morgan Kaufmann.

Clancey, W. J. (1985). “Heuristic classification.” Artificial Intelligence **27**(3): 289-350.

Clancey, W. J. (1992). “Model construction operatiors.” Artificial Intelligence **53**: 1-115.

Compton, P., G. Edwards, et al. (1992). “Ripple down rules: turning knowledge acquisition into knowledge maintenance.” Artificial Intelligence in Medicine **4**: 47-59.

Compton, P., K. Horn, et al. (1989). Maintaining an expert system. Application of Expert Systems Ed. J. R. Quinlan. London, Addison Wesley. 366-385.

Compton, P. and R. Jansen (1990). “A philosophical basis for knowledge acquisition.” Knowledge acquisition **2**: 241-257.

Compton, P., B. H. Kang, et al. (1993). Knowledge acquisition without analysis. Knowledge Acquisition for Knowledge Based Systems, Lectures Notes in AI (723) Eds. G. Boy and B. Gaines. Berlin, Springer Verlag. 278-299.

Compton, P., P. Preston, et al. (1994). Local patching produces compact knoweldge bases. A Future for Knowledge Acquisition Eds. L. Steels, G. Schreiber and W. V. d. Velde. Berlin, German, Springer-Verlag. 104-117.

Edwards, G., P. Compton, et al. (1993). “PEIRS: a pathologist maintained expert system for the interpretation of chemical pathology reports.” Pathology **25**: 27-34.

Gaines, B. (1989). Knowledge acquisition: the continuum linking machine learning and expertise transfer. The Third European Workshop on Knowledge Acquisition for Knowledge-Based Systems, Paris,

Gaines, B. (1992). The Sisyphus problem-solving example through a visual language with KL-ONE-like knowledge representation. Sysyphus'91: Models of Problem Solving, GMD (Gesellschaft fur Mathematik und Datenverarbeitung mbH).

Gaines, B. and M. Musen, Eds. (1994). Proceedings of the 8th AAAI-Sponsored Banff Knowledge Acquisition for Knowledge-Based Systems Workshop. Banff, Canada,

Gaines, B. and M. Shaw (1990). Cognitive and logical foundations of knowledge acquisition. 5th Banff AAAI Knowledge Acquisition for Knowledge Based Systems Workshop, Bannf, Canada,

Gaines, B. R. (1991). The sisyphus problem solving example through a visual language with KL-ONE-like knowledge representation. Sisyphus'91: Models of Problem Solving, Gaines, B. R. and P. Compton (1994). “Induction of meta-knoweldge about knowledge discovery.” IEEE Transactions on Knowledge and Data Engineering **5**(6): 990-992.

Gaines, B. R. and P. J. Compton (1992). Induction of Ripple Down Rules. AI '92. Proceedings of the 5th Australian Joint Conference on Artificial Intelligence, Hobart, Australia, World Scientific, Singapore.

Hayes-Roth, F. (1983). Building expert systems. London, Addison-Wesley.

Horn, K., P. J. Compton, et al. (1985). “An expert system for the interpretation of thyroid assays in a clinical laboratory.” Aust Comput J **17**(1): 7-11.

Kang, B. H. and P. Compton (1992). Knowledge acquisition in context : the multiple classification problem. Proceedings of the 2nd Pacific Rim International Conference on Artificial Intelligence, Seoul, Korea,

Kleer, J. d. (1985). “Book review : building expert systems by F. Hayes-Roth, D. A. Waterman and D. B. Lenat.” Aritificial Intelligence **25**(1): 105-107.

Linster, M., Ed. (1992). Sisyphus'91: Models of Problem Solving. GMD (Gesellschaft fur Mathematik und Datenverarbeitung mbH).

Mansuri, Y., P. Compton, et al. (1991). A comparison of a manual knowledge acquisition method and an inductive learning method. Australian workshop on knowledge acquisition for knowledge based systems, Pokolbin, Australia,

Menzies, T. and P. Compton (1994). Knowledge Acquisition for Performance Systems; or: When can "tests" replace "tasks"? Proceedings of the 8th AAAI-Sponsored Banff Knowledge Acquisition for Knowledge-Based Systems Workshop, Banff, Canada,

Mulholland, M., P. Preston, et al. (1993). An expert system for ion chromatography developed using machine learning and knowledge in context. The Sixth International Conference on Industrial & Engineering Applications of Artificial Intelligence and Expert Systems, Endinburgh, Scotland,

Preece, A. D., R. Shinghal, et al. (1992). “Principles and practice in verifying rule-based systems.” The Knowledge Engineering Review **7**(2): 115 - 141.

Preston, P., G. Edwards, et al. (1994). A 1600 rule expert systems without knowledge engineer. World Congress on Expert Systems, Lisbon, Portugal, Macmillan New Media License.

Puerta, A., J. Egar, et al. (1992). “A multiple method knowledge acqusition shell for the automatic generation of knowledge acquisition tools.” Knowledge Acquisition **4**: 171-197.

Puppe, F. (1993). Systematic introduction to expert systems : knoweldge representations and problem-solving methods. Berlin, Springer-Verlag.

Wielinga, B. J., A. T. Schreiber, et al. (1992). “KADS: a modellingg approach to knowledge engineering.” Knowledge Acquisition **4**: 5-54.