D. R. S. Michalski 2

In Proceedings of The 4th European Working Session on Learning
December, 1989

RULE OPTIMIZATION VIA SG-TRUNC METHOD

Jianping Zhang & Ryszard S. Michalski
Artificial Intelligence Center
George Mason University
Fairfax, VA 22030, USA

ABSTRACT

Most inductive learning systems generate complete and consistent descriptions. In order to achieve completeness and consistency in the presence of noise or imprecision, one may generate overly complex and detailed descriptions. Such descriptions, however, may not perform well in future cases and suffer the disadvantage of excessive complexity. This is the well known phenomenon of overfitting. In this paper, a rule optimization method called SG-TRUNC is described and evaluated experimentally. SG-TRUNC improves previous TRUNC methods and has been implemented in a more efficient way. In the method, an optimized description is obtained through a sequence of generalization and/or specialization operations performed on a complete and consistent concept description. The operations applied always simplify a description.

The method has been implemented in AQ16 that has been applied to two domains: a designed testing problem and "multiplexer" F_{11}. The experimental results have shown that both simplicity and performance improvements can be gained in the domains where noise is present.

1. INTRODUCTION

Most methods of machine learning research assume that concepts are precise entities, represented by a single symbolic description. In such a representation, the boundaries of a concept are well-defined. All instances of a concept are assumed to be equally representative. If an instance satisfies the given concept description, then it belongs to the concept, otherwise it does not. However, most human concepts have a context-dependent meaning and lack precisely defined boundaries. Examples of human concepts are usually not all equivalent. They may have different degrees of typicality in representing the concept. Thus, human concepts are flexible, as they adapt to the context in which they are used.

Most learning systems make the "noise free domain" assumption that the examples presented to the systems do not contain errors and the description language is complete, and consequently constrain their searches only for rules that are both consistent and complete. The requirement of "noise free domain" is often too hard to be satisfied in the real world. The flexibility of concepts and the noise in training sets prevent most learning systems from being applied to many real world learning tasks.

In order to handle flexible concepts, Michalski (1987) proposed a two-tiered concept representation. In the representation, a complete concept description is split into two-parts: the Base Concept Representation (BCR) and the Inferential Concept Interpretation (ICI). The BCR describes the concept simply and explicitly by characterizing the typical and easy-to-define concept meaning. The prototypical instances of the concept can therefore be classified by simple matching with the BCR. The ICI specifies allowable modifications of typical meaning, concept matching procedures, special cases and context-dependency, and thus implicitly defines concept boundaries.

The TRUNC method was first introduced by Michalski et. al (1986) to generate a simple two-tiered concept description. In the TRUNC method, BCR is obtained through removing

some components (conjunctions or disjunctions) from a complete, consistent and more complex concept description generated by AQ15. ICI is simply a predefined flexible matching function. In this paper, we will present a new TRUNC method called SG-TRUNC that has been implemented in AQ16. AQ16 was developed from AQ15 by incorporating SG-TRUNC method. Both AQ15 and the idea of the TRUNC method will be discussed in next two sections. Section 4 presents SG-TRUNC method. Section 5 shows the experimental results from two problems: a designed testing problem and the "Multiplexer" F_{11} problem. Section 6 discusses the comparisons with related work. Section 7 contains the conclusion and suggestions for future research.

2. INDUCTIVE LEARNING PROGRAM: AQ15

AQ15 is a program that incrementally learns decision rules from examples and counterexamples of decisions, and possibly previously learned rules. The rules learned are complete and consistent with the training examples. In learning the rules, the program uses (i) background knowledge that consists of rules and concepts the program already knows, (ii) the definition of descriptors and their types, and (iii) a preference criterion that evaluates competing candidate hypotheses. Each training example characterizes an object (situation, process, etc.) and specifies the correct decision associated with that object. The generated decision rules are optimized according to a flexible criterion selected by the user. The criterion measures the quality of the rules from the viewpoint of the specific problem under consideration. The user may also specify the initial decision hypotheses to be used for incremental learning. In this case, the program will improve them until they are consistent with all available facts. The AQ15 program also has the capability of constructive induction. It can use new descriptions as input data to simplify previously generated decision rules. Finally, the AQ15 program includes a decision rule testing utility. A more detailed presentation about AQ15 is in (Hong, Mozetic & Michalski, 1986).

The concept descriptions learned by AQ15 are represented in VL_1 which is a simplified version of Variable-valued Logic System VL and used to represent attributional concept descriptions. In VL_1, a description of a concept is a disjunctive normal form which is called a cover. A cover is represented as a disjunction of complexes. A complex is a conjunction of selectors. A selector is a form:

$$[L \# R]$$

where
L is called the referee. It is an attribute;
R is called the referent. It is a set of values in the domain of the attribute in L;
is one the following relational symbols: $=, <, >, \leq, \geq, \neq$.

3. THE TRUNC METHOD AND FLEXIBLE MATCHING

This section contains a discussion of the basic idea behind the TRUNC method and a description of the idea of flexible matching.

The TRUNC method was first discussed in (Michalski, et al., 1986). The idea behind the method is to determine the optimal distribution of the concept description between the explicit base concept representation (BCR) and the inferential concept interpretation (ICI) (Michalski, 1987). One of the simple realization of the idea is described below.

In AQ15, each complex generated is associated with a pair of weights: *total* (t-weight) and *unique* (u-weight). The t-weight of a complex is the number of positive examples covered by the complex and the u-weight is the number of the positive examples uniquely covered by the complex. The complexes are ordered according to decreasing values of the t-weight. The t-weight may be interpreted as a measure of the typicality or the representativeness of a complex as a concept description. The complex with the highest t-weight may be viewed as describing the most typical concept examples, and thus serves as its prototypical description. The u-weight may be interpreted as a measure of importance of the complex.

We distinguish between two methods for recognizing the concept membership of an instance: the strict match and the flexible match. In the strict match, one tests whether an instance strictly satisfies the condition part of a rule (a

complex). In the flexible match, one determines the degree of closeness between the instance and the condition part.

Using the strict match, one can recognize a concept without checking other candidate concepts. In the flexible match, one needs to determine the most closely related concept. Such a match can be accomplished in a variety of ways, ranging from approximate matching of features to conceptual cohesiveness (Michalski & Stepp, 1983).

The weight-ordering of complexes described above suggests an interesting possibility. Suppose we have a t-weight ordered disjunction of complexes, and we remove from it the "lightest" complex that is the complex with the lowest u-weight. Such a truncated description will not strictly match events that uniquely satisfy the removed complex. However, by applying a flexible match, these events may still be most closely related to the correct concept, and thus be correctly recognized. A truncated description is, of course, simpler but carries a potentially higher risk of recognition error, and requires a more sophisticated evaluation. We can proceed further and remove the second "lightest" complex from the cover, and observe the performance. Each such step produces a different trade-off between the complexity of the description on the one hand, and the risk factor and the evaluation complexity on the other. At some step the best overall result may be achieved for a given application domain. This method of knowledge reduction by truncating ordered covers and applying a flexible matching is called TRUNC. The truncated description serves as the BCR of the two-tiered concept description. A flexible matching function is the ICI of the two-tiered concept description. See (Michalski, et al., 1986) for further details of the TRUNC method and flexible matching.

Bergadano et, al. (1988a, 1988b) applied the TRUNC method to generate two-tiered concept descriptions in a more sophisticated way. In their system, the TRUNC method is seen as a state space search. The process of truncation is guided by a general description quality, and is implemented as a best first search, i.e., the descriptions of better quality are considered first. The operations used in the search are both generalization (selector removal) and specialization (complex removal). The algorithm is computationally expensive when the initial descriptions contain many complexes.

4. THE SG-TRUNC METHOD

4.1 Discussion

In the SG-TRUNC method, both Specialization and Generalization operations are applied to a consistent and complete description represented as a disjunction of complexes. The generalization is implemented as selector removal, and specialization is implemented as complex removal. Both operations simplify a description. After a selector is removed from a complex, the complex is more general and covers more examples. A consistent complex may become inconsistent, after some of its selectors are removed. After a complex is removed from a description, the description is less general and covers less examples. Therefore the description may no longer be complete.

The examples of a flexible concept can be divided into three types: typical examples, non-typical examples and exceptions (rare cases). Noisy data can be treated as exceptions or non-typical examples depending on the degree of the noise. Typical examples share many common attributes with each other and few attributes with the examples of the other concepts. Non-typical examples share fewer attributes with typical examples. Exceptions share few attributes with typical examples, they may share many attributes with negative examples and look like negative examples.

As mentioned in the previous section, complexes generated by AQ15 can be ordered according to their t-weights or u-weights. A t-weight may be interpreted as a measure of the representativeness or typicality of a complex as a concept description. Complexes with high t-weights may be viewed as describing many typical examples, and thus serve as prototypical descriptions. The examples covered by complexes with low t-weights are often far from the other positive examples, thus these complexes describe rare, exceptional cases. If the learning examples from which complexes are derived are noisy, a complex with low t-weight may also be indicative of errors in the data.

Figure 1 illustrates this situation. "+" and "-" represent a positive example and a negative example, respectively. CPX1 and CPX2 are two complexes that cover all positive examples. The t-weights of CPX1 and CPX2 are 14 and 2, respectively. CPX1 is a representative description of the concept. The two examples covered by CPX2 are considered as exceptions.

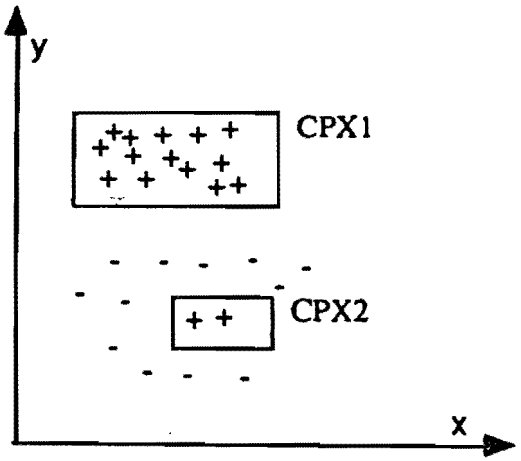

Figure 1: Complex with low t-weight

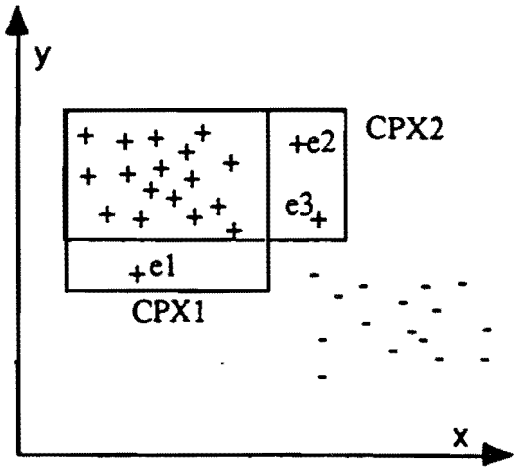

Figure 2: Complexes with low u-weights

A u-weight may be interpreted as a measure of the importance of the complex. The higher the u-weigh, the more important the complex. A representative complex may not be important. It may carry a lot of redundant information, therefore it is not important. The complexes most interesting to the SG-TRUNC method are those with high t-weights and low u-weights. These complexes are representative but not important. Let CPX be a complex with a high t-weight and a low u-weight. Many of the examples covered by CPX are also covered by other complexes. These multiply covered examples can be treated as typical examples. The examples uniquely covered by CPX are considered as non-typical examples. In Figure 2, both CPX1 and CPX2 have high t-weights and low u-weights. Thus, both of them are representatives and not important. e1, e2 and e3 are less typical examples and the others are typical examples.

From the above discussion, one can see that two kinds of complexes, the complexes with low t-weights and the complexes with low u-weights and high t-weights, can be removed. The t-weight of a complex never changes. The u-weight of a complex may increase when other complexes are removed. Therefore, removing complexes with low t-weights is simple. They can be removed independent of the others. But removing complexes with high t-weights and low u-weights is more complicated. Only some complexes with low u-weights can be removed. For example, in Figure 2 only one of the two complexes CPX1 and CPX2 can be removed, although the u-weights of both are small. We thus have to decide which one of these two complexes should be removed.

The loss of coverage resulting from removing complexes with high t-weights and low u-weights can be restored by flexible matching. After a complex is removed, only those examples uniquely covered by it are no longer covered. Since these uncovered examples are close to many typical examples which are still covered by some other complexes. That is, these uncovered examples are close to some remaining complexes, so they can be easily matched with these complexes by flexible matching. Suppose CPX1 in Figure 2 is removed, e1 is no longer covered strictly, but it can be matched with CPX2 by flexible matching. The loss of coverage that results from removing the complexes with low t-weights is not easily restored by flexible matching. In this case, the examples that are no longer covered may not be close to any remaining complexes. In Figure 1, after CPX2 is removed, the two examples covered by it are not close to CPX2, so they cannot be matched with CPX2 by flexible

matching. In order to restore the loss of coverage resulting from removing such complexes, some extra rules are needed (see Bergadano et al., 1988a & 1988b).

The exceptions from other concepts break one complex into several. In Figure 3a, the exception e breaks one complex into four. If we generalize one of the four complexes to allow it to cover the exception, the four complexes merge into one larger complex, see Figure 3b. One way to generalize a complex is to remove some of the selectors of the complex.

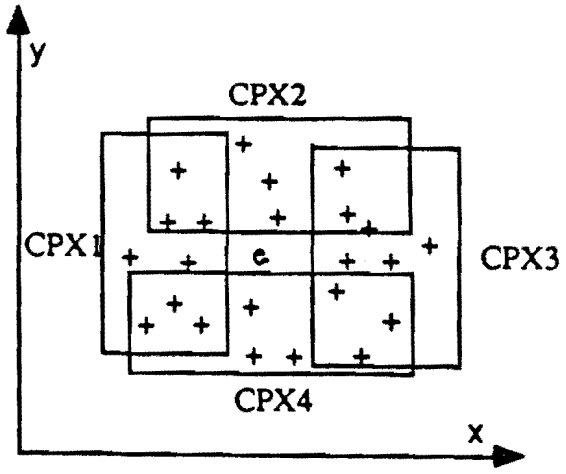

Figure 3a: Before selector removal

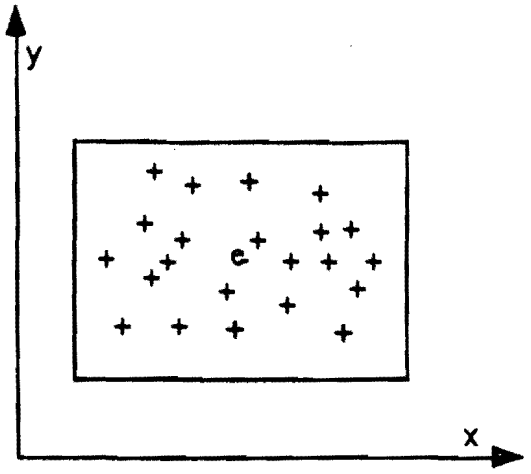

Figure 3b: After selector removal

In AQ learning algorithm, no backtracking is allowed during the process of generating a complex. Some selectors selected at the beginning of the process may become irrelevant after more selectors are selected. The complexes with such irrelevant selectors are often too specific, so these irrelevant selectors must be removed.

The main goal of selector removal is to merge complexes. The way to achieve this is to reduce the u-weights of some complexes so that they can be removed. After a selector is removed from a complex, its t-weight increases, and the u-weights of other complexes decreases. It is desirable that removal of selectors from a complex maximize the increase of the t-weight of the complex, and the decreases of the u-weights of the others, while the complex does not cover many negative examples.

4.2 The SG-TRUNC Method

The method we developed works in two stages, selector removal and complex removal. Two parameters, INCONS and INCOMP, are used to control selector removal and complex removal respectively. INCONS is used to restrict the degree of inconsistency produced by selector removal. INCOMP is used to restrict the degree of incompleteness produced by complex removal.

Selector removal. As discussed above, the goal of removing selectors is to merge complexes, and at the same time, to simplify the complex from which selectors are removed. In order to merge complexes, the u-weights of merging complexes need to be reduced so that these complexes can be removed during the complex removal stage. After selectors are removed from a complex, the t-weight of the complex often increases. This may cause the decreases in the u-weights of other complexes. The more the t-weight of a complex increases, the more the u-weights of other complexes decrease, and the greater the chance of merging complexes. On the other hand, removal of selectors from a complex may result in an inconsistent complex. The inconsistency needs to be restricted in some range. For these reasons, removing a selector from a complex should maximize the increase of the number in positive examples covered and minimize the increase in the number of negative examples covered.

Based on the ideas discussed above, we developed a very simple selector removal algorithm that works as follows.

1. Choose a complex that has not been processed before.
2. Choose a selector from the complex which has the smallest ratio #SNI/#SPI and NCVD/PCVD ≤ INCONS, where #SPI (#SNI) is the number of additional positive (negative) examples covered by the new complex after having removed the selector, PCVD (NCVD) is the number of positive (negative) examples covered by the new complex after having removed the selector. INCONS is the parameter to control the degree of inconsistency provided by the user, and $0 \leq$ INCONS ≤ 1.
3. If a selector is chosen in step 2, then remove it from the current complex and repeat step 2 and 3. Otherwise, go step 4.
4. Repeat step 1,2 3, 4 until all complexes are processed.

In the algorithm, the parameter INCONS is used to control the the degree of inconsistency of the truncated complex. The larger INCONS, the more selectors that can be removed. When INCONS is set to 1, the complex, after selector truncation, may cover the same number of negative examples and positive examples. When INCONS is set to 0, only those selectors whose removal does not result in inconsistency are removed.

Complex removal. We have said that complexes with low t-weights and complexes with low u-weights and high t-weights are candidates for being removed. All complexes with low t-weights can be removed immediately. The removal of complexes with low u-weights is more complicated. Only a subset of the complexes with low u-weights can be removed. Thus we need to select the subset of the complexes with low u-weights to remove.

We sum up the ideas behind our complex removal algorithm as follows:

1. All complexes with low t-weights should be removed.
2. The complexes with high u-weights cannot be removed.
3. Remove some of the complexes with low u-weights and retain the others.
4. Remove complexes as much as possible.
5. The algorithm is not forced to remove any complexes when they all have about the same t-weight.

The algorithm is described as follows:

Input: a concept description D which consists of a disjunction of a set C of complexes.

Output: a concept description D' which consists of a disjunction of a set C' of complexes, where C' is a subset of set C.

1. Initialize set C' and set R to the null set, where R is the set of complexes that are removed.
2. Set LGT to the largest t-weight among all complexes in C and C', and recompute the u-weights for the complexes in C. (In computing the u-weights, the complexes in C' need to be considered.)
3. Determine a set of complexes in set C, and move these complexes from the set C to set C'. A complex cpx is moved to C' if
   cpx(u-weight)/LGT ≥ INCOMP
   where INCOMP is the parameter used to control complex removal and $0 \leq$ INCOMP ≤ 1.
4. If no complex is moved from C to C' in step 3, the complex with the largest t-weight in C is selected and removed from C to C'.
5. If C is not empty, determine a set of complexes in C, and move them from set C to set R. A complex cpx is moved to R if
   cpx(u'-wieght)/LGT < INCOMP
   where u'-weight is the number of positive examples covered by cpx and not covered by any complex in C'.
6. Repeat step 2 to step 6 until C becomes empty.
7. If all complexes in C' satisfy the following condition:
   u-weight/LGT ≥ INCOMP
   then stop and output set C' as the new concept description D'. Otherwise, set C to C' and C' to null, and repeat step 2 to 7.

In the algorithm, the parameter INCOMP is used

to control the the degree of incompleteness. The larger INCOMP is, the more complexes can be removed. When INCOMP is set to 1, only complexes with the largest t-weights are not removed. When INCOMP is set to 0, no complex is removed.

In step 3, all complexes with high u-weight (u-weight/LGT ≥ INCOMP) are selected to retain. All complexes with low t-weights (t-weight/LGT ≤ INCOMP) are removed in step 5, because the u'-weight of a complex is less or equal to the t-weight of the complex. It is obvious that all complexes remaining have high u-weight (u-weight/LGT > INCOMP), after the algorithm terminates. If the t-weights of all complexes are low, no complex is removed.

Finally the algorithm always terminates. The inner loop from step 2 to step 6 terminates if C becomes empty. The size of C reduces at least by 1 for each execution of the loop. As the loop executes, at least one complexes are moved to C' in either step 3 or step 4. The loop from step 2 to 7 terminates if all complexes in C' satisfy the condition u-weight/LGT ≥ INCOMP. If some of the complexes in C' do not satisfy the condition, the loop step 2 to 7 executes again. During each execution of the loop, at lease one complex is moved to R.

5 EXPERIMENTAL RESULTS

To evaluate the SG-TRUNC method, we ran experiments in two problems, a designed testing problem and the "Multiplexer" F_{11} problem. We chose these two problems because they allow us to control the environments of the experiments. The experimental results show the performance of the SG-TRUNC method under varying noise levels. We tested the method under noisy environments instead of imprecise environments, because noisy environments are easier to create.

5.1 A Designed Testing Problem

The designed problem has two concepts represented by 4 attributes each of which has 6 values. The first target concept description is represented by one complex with two selectors and the second is represented by two complexes, each of which has only one selector.

Table 1: Experimental Results of Artificial Domain

Noise Levels	No Truncation			INCOMP = 0.1 INCONS = 0.1			INCOMP = 0.2 INCONS = 0.2			INCOMP = 0.3 INCONS = 0.3			INCOMP = 0.4 INCONS = 0.4		
	ACC %	CPX	SEL	ACC %	CPX	SEL	ACC %	CPX	SEL	ACC %	CPX	SEL	ACC %	CPX	SEL
0%	100	3	4	100	3	4	100	3	4	100	3	4	100	3	4
5%	87.5	9	27	98	3	5	100	3	4	100	3	4	100	3	4
10%	71	12	38	92.5	4	10	100	3	4	100	3	4	100	3	4
15%	74	13	42	76.5	11	35	98	3	6	100	3	4	100	3	4
20%	71	15	49	76.5	11	35	78	7	18	84	3	7	83.5	4	2

The size of the instance space of the problem is 1296. 100 examples are used for learning, 50 for each concept. They are generated randomly. The SG-TRUNC method is tested under different levels of noise and the truncation control parameters, INCOMP and INCONS, are set to different values. Table 1 is the results of the experiments, in which the rules are tested on all instances.

In the table, ACC is the accuracy of the learned rules over all instances. CPX (SEL) is the number of complexes (selectors) in the generated rules. INCOMP and INCONS are parameters used to control the complex and selector removals in AQ16. Noise is created by switching the class of the examples with the given probability.

5.2 The Multiplexer F_{11} Problem

The "Multiplexer" problems were first used by Wilson (1987a, b) to test his system Boole which learns concepts expressed as classification rules. Later, this family of tasks was used by Quinlan (1988) to compare genetic classifiers with decision-tree classifiers.

The "Multiplexer" is a family of tasks in which an object consists of n "address" bits and 2^n "data" bits. An object belongs to the positive concept if the particular data bit indicated by the address bits is "on". A member of this family of tasks is named by the total number of bits (number of "address" + number of "data") involved. For the task F_6 the address bits are x_0 and x_1 and the data bits x_2 through x_5. If x_0 and x_1 are both off, for instance, the address is 0 and the object is a member of the positive class iff the 0th data bit (x_2) is on. The rule for the positive concept is:

$$[x_0 = 0]\,[x_1 = 0]\,[x_2 = 1] \vee$$
$$[x_0 = 0]\,[x_1 = 1]\,[x_3 = 1] \vee$$
$$[x_0 = 1]\,[x_1 = 0]\,[x_4 = 1] \vee$$
$$[x_0 = 1]\,[x_1 = 1]\,[x_5 = 1]$$

The rule for negative concept is:

$$[x_0 = 0]\,[x_1 = 0]\,[x_2 = 0] \vee$$
$$[x_0 = 0]\,[x_1 = 1]\,[x_3 = 0] \vee$$
$$[x_0 = 1]\,[x_1 = 0]\,[x_4 = 0] \vee$$
$$[x_0 = 1]\,[x_1 = 1]\,[x_5 = 0]$$

Table 2: Experimental Results of F_{11}

Noise Levels	No Truncation			INCOMP = 0.1 INCONS = 0.1			INCOMP = 0.2 INCONS = 0.2			INCOMP = 0.3 INCONS = 0.3			INCOMP = 0.4 INCONS = 0.4		
	ACC %	CPX	SEL	ACC %	CPX	SEL	ACC %	CPX	SEL	ACC %	CPX	SEL	ACC %	CPX	SEL
0%	100	3	4	100	3	4	100	3	4	100	3	4	100	3	4
5%	87.5	9	27	98	3	5	100	3	4	100	3	4	100	3	4
10%	71	12	38	92.5	4	10	100	3	4	100	3	4	100	3	4
15%	74	13	42	76.5	11	35	98	3	6	100	3	4	100	3	4
20%	71	15	49	76.5	11	35	78	7	18	84	3	7	83.5	4	2

F_{11} is used for our experiments. F_{11} has 3 "address" bits and 8 "data" bits. The size of the instance space is 2048. Unlike the artificial domain in the previous section, each target concept description here is more complicated and represented by 8 complexes (32 selectors). It is a challenge for our truncation algorithm. It is much harder to select a set of complexes to retain than to select one or two. 400 examples are generated randomly as a training set (200 for each concept). The algorithm is tested under different noise levels. The truncation control parameters INCOMP and INCONS are set to different values. Table 2 shows the experimental results for the domain F_{11}. In the experiments, the rules generated are tested on all instances.

The contents of the table are the same as for Table 1. The time spent in learning the rules ranges from 1 second to 10 second on a Sun 3/50.

5.3 Discussion of The Experimental Results

In the previous two subsections, we have shown the experimental results with the AQ16 on two problems, an designed problem and the "Multiplexer" F_{11} problem. In this subsection, we discuss the experimental results.

The results from both problems showed that not only were the rules simplified, but that the performance of the rules was improved on noisy data. In the domain of F_{11}, when INCOMP and INCONS become too large, the performance begins to drop. Each of the target concept descriptions has 8 complexes. When INCOMP is too large, the descriptions were over-truncated. Thus the performance drops. When a target concept description has many complexes, it is safe not to set INCOMP too large.

It is not necessary to set INCOMP and INCONS to the same value. In fact, better results may be obtained from other combinations of INCOMP and INCONS values. For example, with 20% noise data of the designed problem, INCOMP set to 0.3 and INCONS set to 0.35, the accuracy is 98% which is much better than the results shown in Table 1. It is not a easy task to set proper values for INCOMP and INCONS. Generally speaking, the noisier the data, the larger values INCOMP and INCONS must be set to. As we mentioned above, these two parameters also depend on the target concept descriptions. In the experiments, the best results are obtained when INCOMP and INCONS are set to the values around 0.2. To set proper values to INCOMP and INCONS, one needs the knowledge about the data and the target concept descriptions in the given domain.

The results confirm the argument that the greater the noise, the greater the number of complexes and selectors are generated. Some of the complexes are removed immediately and others are removed once selectors are removed. This is proved by the following experiment. We first set INCONS to 0 and INCOMP to 0.2. Some of the complexes are removed. Then both INCONS and INCOMP are set to 0.2. More complexes are removed.

6. RELATED WORK

The research presented here is related to recent and important work in machine learning that investigates the effects of simplifying concept descriptions (e.g. Fisher & Schlimmer; 1988; Iba et al., 1988; Quinlan, 1987; Clark & Niblett, 1989). An advantage of the presented method is that it may not experience any major loss of coverage as a result of description simplification because of flexible matching.

Closer relevant work is concerned with the problem of pruning decision trees (Quinlan, 1987; Cestnik, Kononenko, and Bratko, 1987). The two methods are similar in the sense that both methods remove components from complete and consistent concept descriptions. The removal often results in incomplete and inconsistent descriptions. Pruning removes subtrees from decision trees, whereas truncation removes selectors and complexes from covers. Removing a subtree can be viewed as removing complexes from one concept and selectors from another.

Now we discuss three important differences between the approach presented here and

pruning of decision trees. First is the lack of constraints on the part of the representation that is removed when applying SG-TRUNC method. In pruning of decision trees, only paths ending in leaves may be pruned. During the generalization of decision trees, each attribute is selected independent of the remaining attributes. Thus, the attributes near the root may be less relevant and the attributes near the leaves may be more relevant. The tree pruning has nothing to do with this problem. Second, pruning will always specialize one concept and generalize the other, while truncation of rules can perform generalization and specialization independently. Third, pruning has no way to restore the loss of coverage resulting from the pruning. The loss of coverage resulting from truncation may be restored by flexible matching.

In table 3, we sum up the results of the experiments performed on "multiplexer" with the decision tree building system ASSISTANT (Cestnik et al., 1987). This system was developed from Quinlan's ID3; the basic algorithm was improved to handle incomplete and noisy data, continuous and multivalued attributes. This system also supports tree-pruning mechanisms. The same training and testing sets were used for ASSISTANT that were used for the previous experiments.

Table 3: Experimental results on Multiplexer F_{11} with ASSISTANT

noise level	no pruning			pruning		
	ACC %	NDS	LVS	ACC %	NDS	LVS
0%	92	183	92	86.5	91	46
3%	84.5	201	101	78	95	48
5%	85.5	203	102	86	119	60
10%	80	259	130	76	79	40
15%	70	279	140	63	57	29
20%	73.5	295	148	57	39	20

In Table 3, ACC is the percentage of accuracy on testing data, NDS and LVS are the number of nodes and leaves in the decision tree generated respectively. From the result, it is obvious that AQ16 outperforms ASSISTANT in the domain of "multiplexer" F_{11} on both accuracy and complexity. The pruned trees almost always degrade accuracy on the testing data. The more noisy the data, the simpler the pruned trees. As indicated in (Quinlan, 1988), the most relevant attributes in the domain are all address bits, but such bits themselves provide no information about the class membership. Instead each of the data bits independently provides more information. Thus all generated decision trees began with a test on a data bit. As we discussed above, pruning only removes the paths ending in leaves. In the case here those most relevant attributes, address bits, are very possible in these paths and pruned so that the quality of the pruned trees decrease.

In (Quinlan, 1987) a method for transforming decision trees into rules and then performing truncation is presented. The method presented in this paper works on the rules generated by AQ algorithm instead of decision trees. Also our algorithm is more sophisticated especially for complex removal.

CN2 induction algorithm (Clark & Niblett, 1989) is another related work. CN2 uses a heuristic function to terminate search during rule construction, based on an estimate of the noise present in the data. This results in rules that may not classify all the training examples correctly, but that perform well on new data. CN2 can be viewed as an induction algorithm that includes pre-truncation, while the algorithm reported here is post-truncation. CN2 applies truncation during the rule generation and AQ16 applies truncation after the rule generation. The pre-truncation is more efficient, but it fails to remove the irrelevant selectors generated first and redundant complexes generated first.

7. CONCLUSION

In this paper, the SG-TRUNC method has been described, in which, both selector and complex removals are applied. Selector removal tries to reduce the impact produced by noise or

exceptions from other concepts which break a few complexes into many. For this purpose, selector removal attempts to increase the intersection among complexes so that some of the complexes are redundant and can be removed, while introducing minimum inconsistency. Complex removal tries to reduce the impact produced by noise or exceptions as well as nontypical instances of the concept to be learned. To achieve this, all complexes with low t-weights and some of the complexes with low u-weights are removed. The goal of the method is to obtain simpler descriptions which remain accurate. In this way, the comprehensibility and the predictive power of the acquired knowledge are improved.

The SG-TRUNC method has been implemented in a learning system AQ16, that is a new member of the AQ family of learning systems. The system has been applied to two problems, a designed testing problem and the "Multiplexer" F_{11} problem. Experimental results have shown that both simplicity and performance improvements can be gained. The system has been also applied to some real world domains, such as vision (Pachowicz, 1989) and US airplane classification (Janikow, 1989). The results obtained from these domains are promising.

A number of problems remain to be addressed in the future. First, a more advanced selector removal algorithm needs to be designed. Second, a method needs to be designed to pass information concerning the importance of attributes to the flexible matching function. Finally, the effects of the two parameters INCONS and INCOMP on real world domains should be explored.

ACKNOWLEDGEMENTS

This research was done in the Artificial Intelligence Center of George Mason University. The activities of the Center are supported in part by the Defence Advanced Research Projects Agency under grant No. N00014-87-K-0874, administered by the Office of Naval Research, and in part by the Office of Naval Research under grant No. N00014-88-K-0226 and N00014-88-K-0397. The authors thank Hugo de Garis for useful comments and suggestions and Janet Holmel for proof reading.

REFERENCES

Bergadano, F., Matwin, S., Michalski, R. S. and Zhang, J., "Learning Flexible Concept Descriptions Using a Two-tiered Knowledge Representation: Ideas and a Method", Reports of the Machine Learning and Inference Laboratory, No 88-4, Artificial Intelligence Center, George Mason University, 1988a.

Bergadano, F., Matwin, S., Michalski, R. S. and Zhang, J., "Learning Flexible Concept Descriptions Using a Two-tiered Knowledge Representation: Implementation and Experiments", Reports of the Machine Learning and Inference Laboratory, No 88-5, Artificial Intelligence Center, George Mason University, 1988b.

Cestnik, B., Kononenko, I., and Bratko, I., "ASSISTANT 86: A Knowledge-elicitation Tool for Sophisticated Users", Procs. of the 2nd European Workshop on Learning, pp. 31-45 (1987).

Clark, P. and Niblett, T., "The CN2 Induction Algorithm", Machine Learning 3, pp 261-183, 1989

Fisher, D. H.and Schlimmer, J. C., "Concept Simplification and Prediction Accuracy", Procs. of the Fifth Int'l. Conf. On Machine Learning, Ann Arbor, pp. 22-28 , 1988.

Hong, J., Mozetic, I, and Michalski, R. S., "AQ15: Incremental Learning of Attribute-Based Descriptions from Examples, The Method and User's Guide", Report ISG 86-5, UIUCDCS-F-86-949, Dept of Computer Science, University of Illinois, Urbana.

Iba, W., Wogulis, J., and Langley, P., "Trading off Simplicity and Coverage in Incremental Concept Learning", Procs. of the Fifth Int'l. Conf. On Machine Learning, Ann Arbor, pp. 73-79 (1988).

Janikow, C.Z. "The AQ16 Inductive Learning Program: Some Experimental Results with

AQ16 and Other Symbolic and Nonsymbolic Programs", Reports of the Machine Learning and Inference Laboratory, Artificial Intelligence Center, George Mason University, 1989

Michalski, R. S., "Two-Tiered Concept Meaning, Inferential Matching and Conceptual Cohesiveness", Chapter in the Book "Similarity and Analogy", Stella Vosniadou and A. Orton, (Eds), 1987.

Michalski, R. S. and Stepp, R. E., "Learning from Observations: Conceptual Clustering. " In Machine Learning - An Artificial Intelligence Approach, R. S. Michalski, J. G. Carbonell, T. M. Mitchell (Eds.), 1983.

Michalski, R. S., Mozetic, I., Hong, J., and Lavrac, N., "The Multi-purpose Incremental Learning System AQ15 and its Testing Application to Three Medical Domains", Proc. 5th AAAI, pp. 1041-1045, 1986.

Pachowicz, P. W., "Low-level Numerical Characteristics and Inductive Learning Methodology in Texture Recognition", Proc. of IEEE Second Workshop on Tools for Artificial Intelligence, Washington D.C., 1989.

Quinlan, J. R., "Induction of Decision Trees" Machine Learning 1, 1986.

Quinlan, J. R., "Simplifying Decision Trees", International Journal of Man-Machine Studies, 1987

Quinlan, J. R., "An Empirical Comparison of Genetic and Decision-Tree Classifiers", Procs. of the Fifth Int'l. Conf. On Machine Learning, Ann Arbor, pp. 135-141 , 1988.

Wilson, S. W., "Quasi-Darwinian Learning in a Classifier System", in Proceedings of Fourth International Machine Learning Workshop, 1987a

Wilson, S. W., "Classifier Systems and the Animat Problem", Machine Learning 2,4., 1987b