Machine Learning, 21, 235–267 (1995)
© 1995 Kluwer Academic Publishers, Boston. Manufactured in The Netherlands.

An Integration of Rule Induction and Exemplar-Based Learning for Graded Concepts

JIANPING ZHANG jianping@zhang.cs.usu.edu
Department of Computer Science, Utah State University, Logan, Utah 84322-4205

RYSZARD S. MICHALSKI michalski@aic.gmu.edu
Artificial Intelligence Center, George Mason University, Fairfax, VA 22030

Editor: John Grefenstette

Abstract. This paper presents a method for learning *graded* concepts. Our method uses a hybrid concept representation that integrates numeric weights and thresholds with rules and combines rules with exemplars. Concepts are learned by constructing general descriptions to represent common cases. These general descriptions are in the form of decision rules with weights on conditions, interpreted by a similarity measure and numeric thresholds. The exceptional cases are represented as exemplars. This method was implemented in the Flexible Concept Learning System (FCLS) and tested on a variety of problems. The testing problems included practical concepts, concepts with graded structures, and concepts that can be defined in the classic view. For comparison, a decision tree learning system, an instance-based learning system, and the basic rule learning variant of FCLS were tested on the same problems. The results have shown a statistically meaningful advantage of the proposed method over others both in terms of classification accuracy and description simplicity on several problems.

Keywords: learning from examples, graded concepts, exemplar-based learning, concept learning

1. Introduction

In real world applications, many concepts are defined in an inherently imprecise manner. Such concepts are referred to as *flexible* concepts (Michalski, 1990). The imprecision can be due to undefined boundaries (as in prototype representations), boundaries defined only within some range, *graded* boundaries (as in fuzzy sets), context-dependent boundaries, or a combination of the above. This paper concentrates on the representation of flexible concepts with graded boundaries or graded degrees of membership (Smith & Medin, 1981; Barsalou, 1985). Such concepts are called *graded* concepts in the rest of this paper. The basic ideas of our method have a direct link to the original work and existing papers on flexible concepts (Michalski, 1990; Bergadano, Matwin, Michalski, & Zhang, 1992). This paper presents a new approach and a significant extension of earlier ideas.

Examples of graded concepts are usually not all equivalent. They may be characterized by a degree of typicality in representing the concepts, which can be viewed as the degree to which an example shares the common concept properties (Rosch and Mervis, 1975). Concept representations used in many learning systems, e.g. decision trees (Quinlan, 1986) and logic-type representations such as decision rules (Michalski, 1983), are not appropriate for describing these concepts. This is partly because they represent a concept through a single symbolic description. Using such a single description makes it difficult to capture the *graded* nature of a concept. This problem was well recognized by Michalski, Mozetic, Hong, & Lavrac (1986) and Quinlan (1987). This paper presents a method for learning graded concepts. This method was implemented in the Flexible Concept Learning System (FCLS).

FCLS employs a novel hybrid representation for describing graded concepts. The hybrid representation is a simple but powerful form of a *two-tiered concept representation* (Michalski, 1987; 1990). This representation integrates weights and thresholds into rules and combines rules with exemplars. Each rule consists of a conjunction of weighted conditions and a threshold. A rule in FCLS is called a *Weighted Threshold Rule*, or *WTR*. The conditions of a WTR explicitly describe the central tendency of a graded concept, while a partial matching method and the threshold extend the WTR to describe less typical cases of the concept. WTRs can be viewed as generalized exemplars that cover common cases of a concept, whereas specific examples may be stored as exemplars to describe exceptions. FCLS can estimate a concept member's degree of typicality. In general, our hybrid representation consists of three elements: a symbolic element (conditions), a numeric element (weights and thresholds), and an exemplar element.

The FCLS inductive learning algorithm generates a concept description as a set of WTRs and/or a set of zero or more exemplars. It learns in batch mode. In the process of learning it adjusts both the symbolic (conditions) and numeric (weights and thresholds) aspects of the hybrid representation to achieve the best fit between a concept description and given concept examples. It also adjusts the distribution between WTRs and exemplars.

The ideas in FCLS were developed from POSEIDON (Bergadano, et al., 1992), a system that learns flexible concepts, and exemplar-based learning methods (Aha, Kibler, & Albert, 1991; Salzberg, 1991). Both FCLS and POSEIDON utilize two-tiered concept representations and perform partial matching. FCLS departs from POSEIDON in two aspects. First, rules in POSEIDON do not have weights and thresholds, and its partial matching procedure is predefined. In FCLS, each rule is associated with a set of weights and a threshold, and the partial matching procedure is adjustable during learning. Second, learning in POSEIDON is divided into two steps. The first step applies AQ15 (Michalski, et al., 1986) to generate a complete and consistent concept description. The second step optimizes the complete and consistent description generated in the first step by removing some of its components (disjuncts or conjuncts). Thus, the final concept descriptions generated by POSEIDON depend on the descriptions generated by AQ15. In FCLS, each time a rule is generated, the rule is optimized by calculating weights of conditions and the threshold. Details and experimental results of the comparison of these two approaches are discussed in Section 7.

Exemplar-based learning (Smith & Medin, 1980; Bareiss, 1989; Aha, et al., 1991; Salzberg, 1991; Zhang, 1992) was proposed to learn graded concepts. FCLS can be viewed as an extension of exemplar-based learning. First, a WTR actually is a generalized exemplar. Second, specific exemplars may be stored as a part of a concept description.

FCLS was tested on a variety of problems. These problems included learning practical concepts such as congressional voting and lymphatic cancer, graded concepts such as n-of-m concepts, and concepts that are represented as DNF functions. To see how the method compares with others, the decision tree learning system C4.5 (Quinlan, 1987), an instance-based approach (Aha, et al., 1991), and an AQ-like rule learning system (Michalski, et al., 1986; Clark & Niblett, 1989) were applied on the same problems. FCLS was also empirically compared with POSEIDON and NGE, an exemplar-based learning method (Salzberg, 1991). The results have shown a statistically significant advantage of FCLS over the other methods both in terms of classification accuracy and description simplicity in graded concepts. Improvements have also been achieved on real problems.

2. Concept Representation

The hybrid concept representation used in FCLS consists of a set of decision rules, a set of exemplars, and a similarity measure. Each decision rule is represented as a WTR, which consists of both a symbolic and a numeric element. The symbolic element is a conjunction of conditions that explicitly describes the central tendency of a graded concept. The numeric element consists of a set of weights and a threshold. The weight of a condition reflects its degree of necessity, while the threshold defines the boundary of the WTR. Exemplars are specific examples. The similarity measure determines the similarity between an example and a WTR (or an exemplar). The following subsections describe each of the components in depth.

2.1. *Weighted Threshold Rule (WTR)*

A WTR is composed of a conjunction of conditions called a disjunct, a set of weights and a threshold. A disjunct is represented as a VL1 *complex* (Michalski, 1983). Each complex is a conjunction of *selectors* (conditions), each of which is a relational expression:

$$[A = V]$$

where A is an attribute, and V is a value or a disjunction of values from the domain of A.

Each condition is associated with a weight. Its value ranges from 0 to ∞. A larger weight means a more necessary condition. A condition with a 0 weight is irrelevant and can be ignored, while a condition with a ∞ weight is a necessary condition. Any value other than 0 and infinity reflects the relative necessity of the condition in comparison with other conditions in the same WTR. For example, if the weights of all conditions of a WTR are equal, then all conditions are equally important regardless of the value of these weights. Weights are computed during learning.

In addition to weights, each WTR has a threshold that is a real number between 0 and 1 inclusive. A threshold defines the boundary of a WTR. An example is *covered* by a WTR if its similarity to the WTR is larger than or equal to its threshold. When the threshold is equal to 1, all the conditions of the WTR must be satisfied in order to match the WTR. Thresholds are adjusted during learning.

2.2. *Similarity Measure*

The similarity measure determines the similarity between an example and a WTR (or an exemplar). It maps an example and a WTR to a real value between 0 and 1. The similarity of an example e and a WTR is defined as the inverse function of their distance normalized by the largest possible distance between an example in the example space and the WTR. Specifically, it is calculated as:

$$\text{Similarity}(e, \text{WTR}) = 1 - \frac{\text{Distance}(e, \text{WTR})}{\text{MAX}_{i=1\ldots n}\{\text{Distance}(e_i, \text{WTR})\}},$$

where $e_1, \ldots, e_n$ are all the examples in the example space. Distance (e, WTR) is a weighted Manhattan distance between e and WTR:

$$\text{Distance}(e, \text{WTR}) = \sum_i \text{weight}(\text{WTR\_}c_i) * \text{Distance}(e, \text{WTR\_}c_i),$$

where WTR_c_i is a condition of WTR, weight(WTR_c_i) is the weight of the condition WTR_c_i, and Distance(e, WTR_c_i) is the distance between e and WTR_c_i. Conditions with a ∞ weight are ignored when calculating Distance(e, WTR). Similarity(e, WTR) is set to 0 if there exists some ∞ weighted condition that is not satisfied by e.

The distance between an example and a condition depends on the type of the attribute involved in the condition. An attribute can be either nominal or linear. A nominal condition relates a nominal attribute to a single or an internal disjunction of values. This distance is 0 when the example's attribute value matches one of these values and is otherwise 1. A linear condition relates a linear attribute to a range of values or an internal disjunction of ranges (e.g., [height = 1...3 v 6...9]). A satisfied condition returns the value of distance 0. The distance between an example and an unsatisfied condition is the difference between the example's attribute value and the nearest end-point of the interval of the condition, normalized by the largest possible distance between the attribute values and the condition. For example, if the domain of x is $[0\ldots10]$, the value of x for the example e is 4, and the condition c is $[x = 7\ldots9]$, then Distance$(e, c) = \frac{7-4}{7-0} = \frac{3}{7}$.

We use a different similarity measure from the one used in AQ15 (Michalski, et al., 1986) for two reasons. First, the measure in AQ15 does not calculate distances. Second, selectors are not weighted in AQ15.

2.3. *Exemplars*

An exemplar is an example of the concept to be learned, and can be represented as a WTR. The disjunct of the WTR is the example itself, and the threshold and weights are set to 1. An exemplar can be partially matched (see Section 3).

2.4. *Examples of the Hybrid Representation*

To illustrate the idea of the hybrid representation, let us consider a simple imaginary concept *R-ball* (Michalski, 1990). The meaning of the concept R-ball is defined as three disjuncts:

(SHAPE = round) & (BOUNCES = yes) or
(SHAPE = round) & (SIZE = medium v large) or
(BOUNCES = yes) & (SIZE = medium v large)

By using the hybrid representation, these three disjuncts merge into one WTR:

[SHAPE = round : 1]
[BOUNCES = yes : 1]
[SIZE = medium v large : 1]
Threshold = $\frac{2}{3}$ = 0.67

The number following a condition is its weight. This WTR includes three conditions:

$$[\text{SHAPE} = \text{round}]\ \&\ [\text{BOUNCES} = \text{yes}]\ \&\ [\text{SIZE} = \text{medium v large}],$$

which represent the central tendency of the concept R-ball. Each condition is equally important. The meaning defined by the WTR is that an object that satisfies any two or more of the three conditions is a R-ball, otherwise it is not. Balls that satisfy all three conditions are the typical ones, while those that only satisfy two of the three conditions are less typical.

Now suppose the meaning of the concept R-ball changes a bit and all R-balls must be round. The new meaning of the concept is now defined by two disjuncts:

$$\begin{gathered}(\text{SHAPE} = \text{round})\ \&\ (\text{BOUNCES} = \text{yes})\ \text{or}\\ (\text{SHAPE} = \text{round})\ \&\ (\text{SIZE} = \text{medium v large})\end{gathered}$$

These two disjuncts are combined into one WTR:

$$\begin{aligned}&[\text{SHAPE} = \text{round} : \infty]\\ &[\text{BOUNCES} = \text{yes} : 1]\\ &[\text{SIZE} = \text{medium v large} : 1]\\ &\text{Threshold} = \frac{1}{2} = 0.5\end{aligned}$$

In this WTR, the condition [SHAPE = round] is a necessary condition and must be satisfied by all R-balls. The other two conditions are not necessary conditions; only one of them must be satisfied.

Suppose that the attribute *SIZE* is linear and the order of its values is *small, medium*, and *large*. The WTR representing R-ball becomes:

$$\begin{aligned}&[\text{SHAPE} = \text{round} : 0]\\ &[\text{BOUNCES} = \text{yes} : 1]\\ &[\text{SIZE} = \text{large} : 1]\\ &\text{Threshold} = \frac{1}{4} = 0.25\end{aligned}$$

An R-ball with a large size is more typical than an R-ball with a medium size. For example, consider four R-balls: Rb1 (round yes large), Rb2 (round yes medium), Rb3 (round yes small), and Rb4 (round no medium). Their similarities to this WTR are:

$$\begin{aligned}\text{Similarity}(\text{Rb1}, \text{WTR}) &= 1 - \frac{0+0+0}{2} = 1,\\ \text{Similarity}(\text{Rb2}, \text{WTR}) &= 1 - \frac{0+0+0.5}{2} = \frac{3}{4},\\ \text{Similarity}(\text{Rb3}, \text{WTR}) &= 1 - \frac{0+0+1}{2} = \frac{1}{2},\\ \text{Similarity}(\text{Rb4}, \text{WTR}) &= 1 - \frac{0+1+0.5}{2} = \frac{1}{4}.\end{aligned}$$

so Rb1 is more typical than Rb2, which is more typical than Rb3, which is more typical than Rb4.

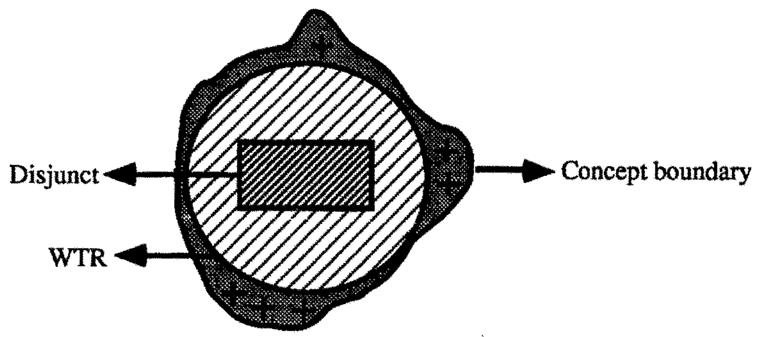

Figure 1. An illustration of the hybrid concept representation.

Figure 1 graphically shows how a graded concept is described by this concept representation. In this figure, the area inside the irregular shape is the space covered by the graded concept that is described by one WTR (the circle) and 6 exemplars (+). The rectangle inside the circle is the disjunct of the WTR that describes the central tendency of the concept.

3. Partial Matching

A two-step partial matching method was implemented in FCLS to match an example with a WTR. In the first step, an example is covered by a WTR if its similarity to the WTR is not smaller than the WTR's threshold. The examples covered by no WTR are referred to as *no-match* examples, while the examples covered by more than one WTR are referred to as *multiple-matched* examples. The first step of the partial matching method fails to classify no-match and multiple-matched examples. To classify no-match and multiple-matched examples, the second step was proposed. In the second step, the classification of a no-match or a multiple-matched example is determined by its relative similarity to WTRs of all concept descriptions. An example is classified to a concept if its relative similarity to one of the WTRs of the concept description is the largest among WTRs of all concept descriptions. The relative similarity between a WTR and an example e is defined as follows:

$$\text{Relative\_Similarity}(e, \text{WTR}) = \begin{cases} 1 & e \in \text{covered(WTR)} \wedge \text{WTR\_}t = 1 \\ \frac{\text{Similarity}(e,\text{WTR}) - \text{WTR\_}t}{1 - \text{WTR\_}t} & e \in \text{covered(WTR)} \wedge \text{WTR\_}t \neq 1 \\ \frac{\text{Similarity}(e,\text{WTR}) - \text{WTR\_}t}{\text{WTR\_}t} & e \notin \text{covered(WTR)} \end{cases}$$

where WTR_t is WTR's threshold and covered(WTR) is the set of all examples covered by WTR. When WTR_t $= 1$ and e is covered by WTR, e satisfies all WTR's conditions so Relative_Similarity (e, WTR) $= 1$. The relative similarity of an example to a WTR that covers it is between 0 and 1 inclusive. The relative similarity of an example to a WTR that does not cover it is between 0 (exclusive) and -1 (inclusive). Therefore, a covered example always has a larger relative similarity than an uncovered example.

Another important reason for proposing the two-step partial matching method is that exemplars can be partially matched. Because the threshold of an exemplar is 1, an exemplar cannot be partially matched if only the first step is applied. When the second step is applied, an exemplar can be partially matched.

4. The FCLS Learning Algorithm

The learning task of FCLS is to generate a concept description for each given concept from a set of examples. Each concept description is a disjunction of WTRs and/or a set of exemplars. This section describes the FCLS learning algorithm.

4.1. The FCLS Learning Algorithm

Table 1 shows the FCLS learning algorithm. The input of FCLS includes a set of examples with their concept memberships and a number of parameter values. The output is a disjunction of WTRs and/or a set of exemplars for each given concept.

The parameter *Max_Err_Rate* is the maximum error rate that a WTR (or a description) is allowed to have. The error rate of a WTR (or a description) is the ratio of the number of negative examples covered to the total number of examples covered by the WTR (or the description). The parameter *Min_Coverage* is the minimum fraction of all positive examples that a WTR must cover. A *Max_Err_Rate* of 0 forces FCLS to produce consistent and complete descriptions. *Max_Err_Rate* is useful for tolerating noise. *Min_Coverage* controls the distribution of a concept description between WTRs and exemplars. A larger value of *Min_Coverage* favors exemplars, while a smaller one favors WTRs. When *Min_Coverage* $= 0$, no exemplar is generated and concept descriptions include only generalized WTRs. When *Min_Coverage* $= 1$, the concept description includes only exemplars so that the learning algorithm becomes pure exemplar-based learning. The parameters *Beam_Width* and *Max_Tries* are discussed in Sections 4.2 and 4.3, respectively. We now define *acceptable* WTRs.

Table 1. The FCLS learning algorithm.

```
FCLS(Examples, Max_Err_Rate, Min_Coverage, Beam_Width, Max_Tries)
    1. Descriptions <- Empty
    2. Repeat
      2.1 Current_Concept <- Select_Current_Concept(Descriptions, Examples)
      2.2 WTRS <- WTR_Generating(Current_Concept, Examples, Max_Err_Rate,
              Min_Coverage, Beam_Width).
      2.3 WTR <- WTR_Optimizing(Current_Concept, Examples, WTRS, Max_Err_Rate,
              Min_Coverage, Max_Tries, Beam_Width).
      2.4 If WTR <> NULL then Descriptions <- Descriptions + WTR
      Until Error_Rate(Descriptions) ≤ Max_Err_Rate or WTR = NULL
    3. While Error_Rate(Descriptions) > Max_Err_Rate
      3.1 Current_Concept <- Select_Current_Concept(Descriptions, Examples)
      3.2 Descriptions <- Descriptions + Select_One_Exemplar(Current_Concept, Examples).
    4. Return Descriptions
```

A WTR is acceptable if:

(1) $\frac{p}{p_{\text{total}}} >$ Min_Coverage and

(2) $\frac{n}{p+n} \leq$ Max_Err_Rate,

where $p(n)$ is the number of positive (negative) training examples covered by the WTR, and p_{total} is the total number of positive training examples.

FCLS works in an iterative fashion. In each iteration the concept description with the largest error omission is selected by the function Select_Current_Concept as the current concept. The current concept is then generalized. Generalization in FCLS consists of two iterative processes: WTR generation and exemplar selection. The WTR generation process generates a disjunction of acceptable WTRs for each concept, while the exemplar selection process selects a set of exemplars. Each iteration of the WTR generation process tries to generate an acceptable WTR for the selected concept: *Current_Concept.* The function *Select_Current_Concept* selects the concept with the largest error of omission (i.e., the percentage of uncovered positive examples) as *Current_Concept.* The WTR generation process is composed of two algorithms: *WTR generating* and *WTR optimizing*. The WTR generating algorithm generates a set of WTRs with unitary weights and thresholds, and performs neither weight learning nor threshold adjusting. The WTR optimizing algorithm optimizes the WTRs generated by the WTR generating algorithm through learning weights and adjusting thresholds. If an acceptable WTR is generated, then the WTR optimizing algorithm returns it, and otherwise it returns NULL.

WTR generation is an iterative process that is repeated until either no acceptable WTR can be generated or the error rate of the descriptions generated is not larger than *Max_Err_Rate*. The error rate of descriptions is the fraction of all training examples that are not correctly classified by the two-step partial matching method.

If the error rate of generated descriptions is larger than *Max_Err_Rate*, then FCLS selects a set of exemplars to reduce the error rate. The algorithm for selecting exemplars is similar to IB2 (Aha, et al., 1991). Each iteration of the exemplar selection algorithm selects an incorrectly classified example as an exemplar of the selected concept *Current_Concept*, then reclassifies all remaining incorrectly classified examples. This process is repeated until the error rate is no longer larger than *Max_Err_Rate*.

4.2. *The WTR Generating Algorithm*

The WTR Generating algorithm first generates a set of most general disjuncts whose error rate is not larger than *Max_Err_Rate*. These disjuncts are then converted to WTRs by setting all weights and thresholds to 1. Table 2 summarizes the WTR generating algorithm.

This algorithm is similar to the AQ algorithm (Michalski, 1983) and performs a general-to-specific beam search. The beam width is specified by *Beam_Width*. *Current_Disjuncts* stores *Beam_Width* disjuncts that have the highest potential for improvement and is initialized to the most general disjunct, which covers the entire example space. During each cycle, the error rate of each disjunct in *Current_Disjuncts* is tested. If the error rate of a disjunct is not higher than *Max_Err_Rate*, then the disjunct is added into *Consistent_Disjuncts*. Otherwise, the disjunct is specialized by removing a value from one of its conditions. This specialization is repeated for each condition of the disjunct. The value to be removed from a condition is chosen to maximize the number of negative examples and minimize the number of positive examples excluded from the disjunct. This value is chosen by the function

Table 2. The WTR generating algorithm.

```
WTR_Generating(Current_Concept, Examples, Max_Err_Rate, Min_Coverage, Beam_Width)
    1. Current_Disjuncts <- {Most_General_Disjunct}
    2. Consistent_Disjuncts <- empty.
    3. **Repeat**
       3.1 New_Disjuncts <- Empty
       3.2 **For** each Disjunct in Current_Disjuncts
             **if** error_rate(Disjunct) $\leq$ *Max_Err_Rate*,
             **then** Consistent_Disjuncts <- Consistent_Disjuncts + Disjunct
             **else** For each condition $[A = v_1 \cdots v_n]$ of Disjunct, if $n > 1$ then
                  i. V <- Select_Value($\{v_1, \ldots, v_n\}$, Disjunct, Current_Concept, Examples)
                 ii. **if** V <> NULL
                     **then** Disjunct <- Remove V from $[A = v_1 \cdots v_n]$ of Disjunct
                          Disjunct_Potential_Quality (Disjunct)
                          New_Disjuncts <- New_Disjuncts + Disjunct
       3.3 Current_Disjuncts <- Select_Best_Disjuncts(New_Disjuncts, Beam_Width)
       **Until** Current_Disjuncts = empty
    4. Consistent_Disjuncts <- Select_Best_Disjuncts(Consistent_Disjuncts, Beam_Width)
    5. Return Convert_Disjuncts_To_WTRs(Consistent_Disjuncts)
```

Select_Value. If only one value is involved in a condition, then *Select_Value* returns NULL. This iteration yields several new disjuncts, each of which covers fewer negative examples. Each new disjunct is evaluated for its potential quality by the Disjunct Potential Quality Evaluation Function. *Select_Best_Disjuncts* selects *Beam_Width* new disjuncts with the highest potential qualities and stores them in *Current_Disjuncts*. When *Current_Disjuncts* is empty, the *Beam_Width* disjuncts with the highest potential quality in *Consistent_Disjuncts* are converted to WTRs by setting their weights and thresholds to 1 and are returned.

The potential quality of a disjunct consists of two parts: *current quality* and *potential improvement*. Current quality is computed based on the completeness and consistency of a disjunct, while potential improvement is an estimate of how much improvement can be achieved on the basis of the disjunct's current quality. A disjunct with a low current quality is probably not worth being improved, even though it has a high potential improvement. A disjunct with a low potential improvement has little chance for further improvement.

The current quality of a disjunct is computed based on the number of positive and negative examples covered by the disjunct. The potential improvement of a disjunct is computed based on the distribution of the covered positive examples and negative examples. Some distributions make a disjunct much easier to be improved than the others. For example, Fig. 2 shows two disjuncts DNT1 and DNT2 that cover the same number of positive and negative examples; they have the same current quality. However, DNT1 is much easier to improve than DNT2 because the positive examples covered by DNT2 are scattered, while the positive examples covered by DNT1 are concentrated. A disjunct with dispersed covered positive examples is hard to improve, so it has a low potehtial improvement. A disjunct with concentrated covered positive examples can be easily specialized to a consistent disjunct, so it has a high potential improvement.

The Disjunct Potential Quality evaluation function is defined as a product of two parts:

$$\text{Disjunct\_Potential\_Quality}(d) = \frac{p}{p_{\text{total}}} * \text{Disjunct\_Potential\_Improvement}(d)$$

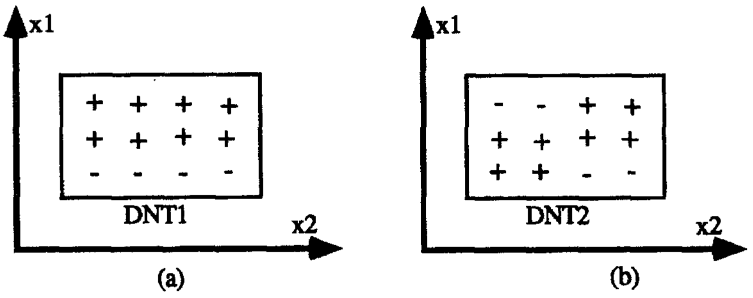

Figure 2. Illustration of the difference between the quality and potential improvement of disjuncts: DNT1 and DNT2 have the same quality, but DNT1 has a higher potential improvement than DNT2.

where d is the disjunct to be evaluated, p is the number of positive examples covered by d, and p_{total} is the total number of positive examples. Disjunct_Potential_Improvement(d) includes two aspects: consistency and the distribution of the covered positive and negative examples.

Assume that the disjunct d is represented as:

$$d = [A_1 = v_{11} V \dots V v_{1m_1}] \& \dots \& [A_n = v_{n1} V \dots V v_{nm_n}]$$

where A_1 to A_n are all attributes. For each attribute A_i $(1 \leq i \leq n)$, its importance to d Importance(A_i, d) is computed as follows:

$$\text{Importance}(A_i, d) = \sum_{k=1\dots m_i} (p_{ik} * q_{ik})$$

$$p_{ik} = \frac{|p\_cvd(d) \cap E_{ik}|}{|p\_cvd(d)|}$$

$$q_{ik} = \frac{|p\_cvd(d) \cap E_{ik}|}{|(cvd(d) \cap E_{ik}|)}$$

where $p\_cvd(d)$ is the set of all positive examples covered by d, $cvd(d)$ is the set of all examples covered by d, and E_{ik} is the set of all examples whose value of A_i is v_{ik}. $\frac{|p\_cvd(d)|}{|cvd(d)|} \leq \text{Importance}(A_i, d) \leq 1$. When $q_{ik} = \frac{|p\_cvd(d)|}{|cvd(d)|}$ for $k = 1, \dots, i_m$, covered positive and negative examples are equally distributed over all values of the attribute A_i. Thus, Importance(A_i, d) takes the smallest value: $\frac{|p\_cvd(d)|}{|cvd(d)|}$. When q_{ik} is either 1 or 0 for $k = 1, \dots, i_m$, both covered positive and negative examples are highly concentrated so Importance(A_i, d) $= 1$. In such a case, d can be specialized to a consistent disjunct by removing all values of A_i with $q_{ik} = 0$.

Disjunct_Potential_Improvement(d) is defined as the probabilistic sum of Importance-(A_i, d) for $(1 \leq i \leq m)$. The probabilistic sum of Importance(A_1, d) and Importance(A_2, d) is defined as follows:

$$\text{Disjunct\_Potential\_Improvement}(d) = \text{Importance}(A_1, d) + \text{Importance}(A_2, d) \\ -\text{Importance}(A_1, d) * \text{Importance}(A_2, d)$$

One characteristic of the probabilistic sum is that if one of Importance(A_i, d) $(1 \leq i \leq m)$ is equal to 1, then Disjunct_Potential_Improvement(d) $= 1$.

4.3. The WTR Optimizing Algorithm

The WTR optimizing algorithm optimizes WTRs by adjusting their weights and thresholds to best fit training examples. The boundary of a WTR is defined by its threshold. Decreasing the threshold increases the WTR's coverage. To decrease the threshold of a WTR without increasing its error rate, the similarities of *nearly* covered negative examples must be reduced. A *nearly covered example* of a WTR is an example whose similarity to the WTR is in the range of WTR_t (the threshold of WTR) and WTR_t $- \Delta$. Δ is a function of all the weights of WTR[1]. One way to reduce the similarities of nearly covered negative examples is to specialize the disjunct of a WTR by removing some values of a condition. The value chosen for specialization occurs on many nearly covered negative examples and few nearly covered positive examples. Thus, a WTR is optimized by specializing its disjunct and decreasing its threshold.

The WTR optimizing algorithm is similar to the WTR generating algorithm in that it also performs a general-to-specific beam search. In this algorithm, the threshold is decreased, while the disjunct is specialized. Thus, a WTR is often generalized although its disjunct is specialized. Another major difference involves the different negative examples that these two algorithms try to exclude. The WTR generating algorithm reduces the number of covered negative examples, whereas the WTR optimizing algorithm reduces the number of *nearly* covered negative examples. This difference is reflected in their different potential quality evaluation functions and their methods for selecting values of a condition for specialization. The potential quality of a disjunct is computed solely based on the covered examples, while the potential quality of a WTR is computed based on covered and nearly covered examples. In the WTR generating algorithm, the value of a condition that occurs most frequently on covered negative examples and least frequently on covered positive examples is selected for specialization, while in the WTR optimizing algorithm, the value of a condition that occurs most frequently on nearly covered negative examples and least frequently on nearly covered positive examples is selected for specialization.

After a threshold decreases, some nearly covered examples may become covered. Therefore, a WTR with many nearly covered positive examples has a higher improvement potential than a WTR with many nearly covered negative examples. In Fig. 3, the two circles are the boundaries of two distinct WTRs, and examples (+ for positive and − for negative) inside the boundary of each WTR are its covered examples, while examples outside and near the boundary of each WTR are its nearly covered examples. WTR1 has a quality higher than WTR2, because WTR1 covers no negative examples. However, WTR1 has little potential

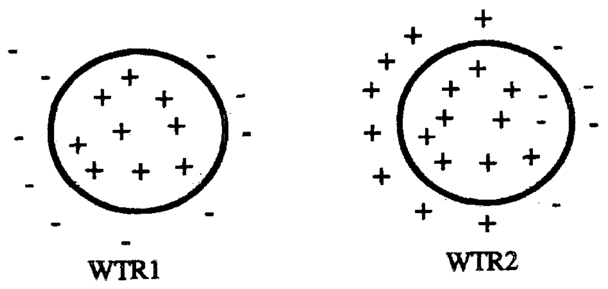

Figure 3. Illustration of the potential improvement of a weighted threshold rule (WTR): WTR2 has a lower quality than WTR1, but a higher potential improvement.

improvement, because all its nearly covered examples are negative examples. In contrast, WTR2 has a lower quality, but a larger potential improvement.

The potential quality of a WTR is computed based on both covered and nearly covered examples.

$$\text{WTR\_Potential\_Quality(WTR)} = \frac{\text{WTR}_{\text{pos}}}{\text{pos}} * \frac{\text{WTR}_{\text{pos}}}{\text{WTR}_{\text{pos}} + \text{WTR}_{\text{neg}}}$$

$$\text{WTR}_{\text{pos}} = \sum_{e \in \text{WTR}_{\text{pos\_covered}} \cup \text{WTR}_{\text{pos\_nearly-covered}}} C\text{\_Value}(e, \text{WTR})$$

$$\text{WTR}_{\text{neg}} = \sum_{e \in \text{WTR}_{\text{neg\_covered}} \cup \text{WTR}_{\text{neg\_nearly-covered}}} C\text{\_Value}(e, \text{WTR})$$

$$C\text{\_Value}(e, \text{WTR}) = \begin{cases} 1 & e \in \text{WTR}_{\text{covered}} \\ 1 - 0.5 * \frac{\text{WTR\_t} - \text{Similarity}(e,\text{WTR})}{\text{WTR\_t} - (\text{WTR\_t} - \Delta)} & e \in \text{WTR}_{\text{nearly-covered}} \end{cases}$$

WTR_{pos} (WTR_{neg}) is the sum of C_Values (Coverage Values) of positive examples (negative examples). The C_Value of a covered example is 1. The C_Value of a nearly covered example ranges from 0.5 to 1, depending on the distance to the threshold. The closer to the threshold it is, the higher its C_Value. $\text{WTR}_{\text{pos\_covered}}$ ($\text{WTR}_{\text{neg\_covered}}$) is the set of all positive (negative) examples covered WTR. $\text{WTR}_{\text{pos\_nearly-covered}}$ ($\text{WTR}_{\text{neg\_nearly-covered}}$) is the set of all nearly covered positive (negative) examples. $\text{WTR}_{\text{covered}}$ is all examples covered and $\text{WTR}_{\text{nearly-covered}}$ are all examples nearly covered.

Table 3 shows the WTR optimizing algorithm. *Current_WTRs* is a list of length *BEAM_WIDTH* of WTRs to be improved and is initialized to the WTRs generated by the WTR generating algorithm. These initial WTRs are optimized by computing their weights and thresholds. *Best_WTR* stores the WTR with the highest quality and is initialized to the acceptable WTR with the highest quality in *Current_WTRs*.

As in most inductive learning systems, the quality of a WTR is evaluated based on its completeness and consistency with regard to the training examples. Generally, one can gain completeness at the expense of consistency, or one can gain consistency by sacrificing completeness. They are two competing criteria. For this reason, the quality evaluation function is defined as the product of these two parts: normalized completeness and normalized consistency.

Quality(WTR) = Normalized_Completeness(WTR) * Normalized_Consistency(WTR)

$$\text{Normalized\_Consistency(WTR)} = \begin{cases} 1 & \text{Max\_Err\_Rate} = 0 \wedge n = 0 \\ 0 & \text{Max\_Err\_Rate} = 0 \wedge n > 0 \\ \max\left\{\frac{\text{Max\_Err\_Rate}, -\frac{n}{p+n}}{\text{Max\_Err\_Rate}}, 0\right\} & \text{otherwise} \end{cases}$$

$$\text{Normalized\_Completeness(WTR)} = \begin{cases} 0 & \text{Min\_Coverage} = 1 \\ \max\left\{\frac{\frac{p}{p_{\text{total}}} - \text{Min\_Coverage}}{1 - \text{Min\_Coverage}}, 0\right\} & \text{otherwise} \end{cases}$$

where p_{total} represents the total number of positive examples, and p and n are the number of positive and negative examples covered by WTR, respectively. When the completeness of a WTR is not larger than *Min_Coverage*, the WTR is not acceptable. In such cases,

Table 3. The WTR optimizing algorithm.

```
WTR_Optimizing(Current_Concept, Examples, Initial_WTRs, Max_Err_Rate,
                    Min_Coverage, Beam_Width, MAX_TRIES)
    1. Current_WTRs <- Initial_WTRs
       Best_WTR <- Empty
    2. For each WTR in Current_WTRs
            WTR <- Weight_Learning(WTR)
            WTR <- Threshold_Adjusting(WTR)
            if Acceptable(WTR) and (Best_WTR = Empty or Quality(WTR) > Quality(Best_WTR))
            then Best_WTR <- WTR
    3. Repeat
       3.1. NEW_WTRs <- empty
       3.2. For each WTR in Current_WTRs
            For each condition $[A = v_1 \dots v_n]$ of WTR, if $n > 1$ then
              i. V <- Select_Value($\{v_1, \dots, v_n\}$,WTR, Current_Concept, Examples)
              ii. if V <> NULL
                 then WTR <- Remove V from $[A = v_1 \dots v_n]$ of WTR
                      WTR <- Weight_Learning(WTR)
                      WTR <- Threshold_Adjusting(WTR)
                      WTR_Potential_Quality (WTR)
                      if Acceptable(WTR, Max_Err_Rate, Min_Coverage) and
                          Quality(WTR) $\geq$ Quality(Best_WTR)
                      then Best_WTR <- WTR
                          No_Improvement <- 0
                      else No_Improvement <- No_Improvement + 1
                      New_WTRs <- New_WTRs + WTR
       3.3. Current_WTRs <- Select_Best_WTRs(New_WTRs, Beam_Width)
       Until No_Improvement > MAX_TRIES or Current_WTRs = empty
    4. Return Best_WTR
```

Normalized_Completeness(WTR) = 0 and Quality(WTR) = 0. When the inconsistency of a WTR is larger than or equal to *Max_Err_Rate*, the WTR is not acceptable. In such cases, Normalized_Consistency(WTR) = 0 and Quality(WTR) = 0. This means that the quality of an unacceptable WTR is equal to 0.

Like the WTR generating algorithm, this algorithm repeats the beam search until the stop condition is satisfied. In each cycle of the loop, a set of new WTRs is generated. Each newly generated WTR is evaluated by two functions: the Quality Evaluation Function and the WTR Potential Quality Evaluation Function. The acceptable WTR with the highest quality replaces *Best_WTR*, if its quality is higher than or equal to *Best_WTR*'s. The *Beam_Width* new WTRs with the highest potential qualities are selected for further improvement. *MAX_TRIES* is an integer parameter that controls the execution of the loop. If Best_WTR has not been improved in *MAX_TRIES* steps, then the algorithm stops.

4.4. Weight Learning and Threshold Adjustment

In computing the weight of a condition, the algorithm counts the number of positive and negative examples that do not match the condition. This weight learning algorithm is similar to the one used in STAGGER (Schlimmer, 1987). Specifically, the weight of a condition *c* is calculated as:

$$\text{weight}(c) = \frac{p(\text{unmatched} \mid \text{NEG})}{p(\text{unmatched} \mid \text{POS})}$$

where p(unmatched | NEG) and p(unmatched | POS) are the fraction of negative and positive examples that do not match with the condition c. When c is satisfied by all positive examples, p(unmatched | POS) $= 0$ so weight(c) $= \infty$ and c is necessary. When c is satisfied by all negative examples, p(unmatched | NEG) $= 0$ so weight(c) $= 0$. This case seldom occurs, because such a condition is rarely generated by the learning algorithm. The fewer negative examples that satisfy c, the larger p(unmatched | NEG) and weight(c). The more positive examples that satisfy c, the smaller p(unmatched | POS), therefore the larger weight(c). When both p(unmatched | NEG) and p(unmatched | POS) are equal to 0, weight(c) is set to 1.

The decrease in a threshold is divided into a number of steps; the threshold is decreased by a fixed quantity δ in each step. After each decrease, the coverage of the WTR is checked to see if it covers more examples. If not, its threshold is again decreased by δ. This process is repeated until the WTR covers more examples. Afterwards, the WTR is evaluated to check if it has improved. If it has not improved in *MAX_TRIES* times, then the adjustment of the threshold stops, and the threshold on which the WTR achieves the highest quality is the threshold of the WTR. The quantity δ decreased in each step for WTR is determined as follows:

$$\delta = \frac{1 - \text{MIN}_{i=1\ldots n}\{\text{Similarity}(e_i, \text{WTR})\}}{100},$$

where $e_1, \ldots, e_n$ are the training examples.

4.5. *Time Complexity of FCLS*

FCLS consists of two iterations: WTR generation and exemplar selection. WTR generation is much more time consuming than exemplar selection, so we ignore the complexity of exemplar selection. WTR generation repeats the WTR generating and the WTR optimizing algorithms. Each iteration generates one WTR. The following subsections discuss the time complexities for the WTR generating and WTR optimizing algorithms. Let n be the size of the training set, a be the number of attributes, v be the maximum number of values of an attribute, and b be *Beam_Width*.

4.5.1. Time Complexity of the WTR Generating Algorithm. Because each iteration of the WTR generating algorithm removes a value from a condition of a disjunct, the maximum number of iterations (the Repeat Loop in Step 3 in Table 2) is $a \cdot v$. The For Loop in Step 3.2 in Table 2 repeats b times. The else part of Step 3.2 in Table 2 executes at most a times. The time taken to select a value of an attribute for specialization (the function Select_Value) is $O(n \cdot v)$, plus the time taken to evaluate the potential quality of the disjunct, which is $O(a \cdot n \cdot v)$. Finally the time taken to select b best disjuncts from $b \cdot a$ disjuncts (the function Select_Best_Disjuncts) is $O(a \cdot b \cdot \log(a \cdot b))$. Therefore, the overall time for the WTR generating algorithm is $O(a^3 \cdot b \cdot n \cdot v^2)$.

4.5.2. Time Complexity of the WTR Optimizing Algorithm. Similar to the WTR generating algorithm, the maximum number of iterations of the WTR optimizing algorithm (the Repeat Loop of Step 3 in Table 3) is $a \cdot v$. The outside For Loop of Step 3.2 in Table 3 repeats b times and the inside For Loop of Step 3.2 executes at most a times. The time taken to generate a WTR (3.2.i and 3.2.ii in Table 3) is the sum of the time taken to select a value of an attribute for specialization, to evaluate the potential quality and quality of the WTR, to calculate weights, and to adjust the threshold. The time to select a value of an attribute is $O(v \cdot n)$. Both the time to evaluate the potential quality and the time to evaluate the quality are $O(a \cdot n \cdot v)$. The time for weight learning is $O(a \cdot n \cdot v)$. Each time the threshold is adjusted, the potential quality and quality of the WTR are reevaluated. The time to match a WTR and an example is the time to calculate the distance between the example and the WTR. Its time complexity is $O(a)$. The distance between an example and the WTR is fixed during threshold adjusting. Therefore, we can cache all distances between all examples and the WTR, so that we do not have to measure the distances each time the threshold is adjusted. The time for threshold adjusting is $O(n)$. The time taken to select b best WTRs from $b \cdot a$ WTRs is $O(b \cdot a \cdot \log(b \cdot a))$. Finally, the overall time for the WTR optimizing algorithm is $O(a^3 \cdot b \cdot n \cdot v^2)$.

4.5.3. Summary. We now give the time complexity of the FCLS algorithm. The Repeat Loop of Step 2 in Table 1 executes at most n times, because each WTR covers at least one example that is not covered by other WTRs. Therefore, the time complexity of FCLS is $O(a^3 \cdot b \cdot n^2 \cdot v^2)$.

4.5.4. Comparison with CN2 and C4.5. When all attributes are binary, the time complexity[2] of CN2 is $O(a^3 \cdot b \cdot n^2)$. For binary attributes, the overall time for FCLS is the same as that of CN2. According to (Utgoff, 1989), the time for constructing a tree (no pruning) is $O(a^2 \cdot n + 2^a)$ for binary attributes. The actual run time of FCLS is larger than C4.5 and CN2. This is caused by the weight learning, threshold adjusting, distance measuring, and evaluation functions.

5. An Example Illustrating the FCLS Algorithm

Consider again the concept R-ball used in Section 2.4. In addition to the three attributes: SHAPE, BOUNCE and SIZE, an irrelevant attribute COLOR that takes the values *white* and *black* is added to the problem. Each object in the domain is now described by four attributes. Table 4 shows all training examples. Examples 1 to 6 are positive examples, and examples 7 to 12 are negative examples.

In this example, weight learning is ignored. *Beam-Width*, Max_Err_Rate, and Min_Coverage are set to 1, 0 and 0 respectively. First, the FCLS algorithm calls the WTR generating algorithm, which starts with the most general disjunct:

$$[\text{SHAPE} = \text{round} \vee \text{square}][\text{BOUNCE} = \text{yes} \vee \text{no}]$$
$$[\text{SIZE} = \text{large} \vee \text{medium} \vee \text{small}][\text{COLOR} = \text{white} \vee \text{black}]$$

For each of the four conditions, the WTR generating algorithm chooses one value to remove and generates four more specific disjuncts. For example, it chooses the value

Table 4. Positive and negative examples of the concept R-ball.

#	SHAPE	BOUNCE	SIZE	COLOR	CLASS
1	round	yes	large	white	positive
2	round	yes	small	black	positive
3	square	yes	medium	black	positive
4	round	no	large	white	positive
5	round	yes	small	white	positive
6	square	yes	medium	black	positive
7	square	no	small	black	negative
8	round	no	small	black	negative
9	square	yes	small	white	negative
10	square	no	large	white	negative
11	round	no	small	white	negative
12	square	yes	small	black	negative

square to remove for the first condition, because removing the value *square* excludes more negative examples (7, 9, 10 and 12) and fewer positive examples (3 and 6) from the disjunct than removing the value *round.* This yields the following four new disjuncts:

[SHAPE = round][BOUNCE = yes ∨ no][SIZE = large ∨ medium ∨ small]
[COLOR = white ∨ black]
Examples Covered: 1, 2, 4, 5, 8, 11
[SHAPE = round ∨ square][BOUNCE = yes][SIZE = large ∨ medium ∨ small]
[COLOR = white ∨ black]
Examples Covered: 1, 2, 3, 5, 6, 9, 12
[SHAPE = round ∨ square][BOUNCE = yes ∨ no][SIZE = large ∨ medium]
[COLOR = white ∨ black]
Examples Covered: 1, 3, 4, 6, 10
[SHAPE = round ∨ square][BOUNCE = yes ∨ no][SIZE = large ∨ medium ∨ small]
[COLOR = black]
Examples Covered: 2, 3, 6, 7, 8, 12

Assuming a beam width of one, one of the four disjuncts is selected based on its potential quality for further improvement. For simplicity, we choose the most consistent disjunct:

[SHAPE = round ∨ square][BOUNCE = yes ∨ no]
[SIZE = large ∨ medium][COLOR = white ∨ black]

This disjunct covers four positive examples and one negative example, and has higher consistency than the other three disjuncts. Repeating the same process, the following four new disjuncts are generated:

[SHAPE = round][BOUNCE = yes ∨ no][SIZE = large ∨ medium]
[COLOR = white ∨ black]

Examples Covered: 1, 4

[SHAPE = round ∨ square][BOUNCE = yes][SIZE = large ∨ medium]
[COLOR = white ∨ black]

Examples Covered: 1, 3, 6

[SHAPE = round ∨ square][BOUNCE = yes ∨ no][SIZE = medium]
[COLOR = white ∨ black]

Examples Covered: 3, 6

[SHAPE = round ∨ square][BOUNCE = yes ∨ no][SIZE = large ∨ medium]
[COLOR = black]

Examples Covered: 3, 6

All of these four disjuncts cover no negative examples, but they cover different numbers of positive examples. The second disjunct, which covers the largest number of positive examples, is chosen as the output of the WTR generating algorithm. The following disjunct:

[BOUNCE = yes][SIZE = large ∨ medium]
Threshold = 1

serves as the initial WTR for the WTR optimizing algorithm. This initial WTR is first optimized by decreasing its threshold. Table 5 shows the distances and similarities of all examples to the above WTR. δ, the quantity to decrease in the threshold, is 0.01. The threshold adjustment algorithm continues to reduce the threshold by 0.01 until more examples are covered (i.e., Threshold = 0.5). The 0.5 threshold does not improve the quality of the WTR, so the threshold remains 1.

The WTR optimizing algorithm optimizes this WTR by specializing its disjunct and decreasing its threshold. The way to specialize its disjunct is the same as in the WTR generating algorithm (i.e., by removing a value from a condition). Because no value can be removed from the condition [BOUNCE = yes], only three new WTRs are generated. The

Table 5. Distances and similarities of examples to [BOUNCE = yes][SIZE = large ∨ medium].

Examples	Distance	Similarity
1	0	1
2	1	0.5
3	0	1
4	1	0.5
5	1	0.5
6	0	1
7	2	0
8	2	0
9	1	0.5
10	1	0.5
11	2	0
12	1	0.5

following WTR is the best one of the three new WTRs:

[SHAPE = round][BOUNCE = yes][SIZE = large $\vee$ medium]
Threshold = 1

This WTR is then optimized by reducing its threshold. Finally, the following WTR is returned.

[SHAPE = round][BOUNCE = yes][SIZE = large $\vee$ medium]
Threshold = 0.67

This WTR covers all six positive examples, none of the negative examples and is the final description for the concept R-ball.

6. Empirical Evaluation

To evaluate FCLS, a number of experiments were conducted on various problems with FCLS, C4.5 (Quinlan, 1993), and IB3 (Aha, et al. 1991). Seven problems were used in the evaluation. Three of them were designed to involve graded concepts so that FCLS was expected to perform well on them. The remaining four problems were 11-multiplexor, 4-term 3DNF, congressional voting records, and lymphatic cancer and used to test the FCLS's applicability. In addition to the above experiments, some experiments were conducted to evaluate the performance when exemplars were used in the hybrid representation and when data contained noise.

6.1. *Design of Experiments*

In our experiments, we ran FCLS under three modes: learning DNF (DNF mode), learning threshold rules (without weight learning, TR mode), and learning weighted threshold rules (with weight learning, WTR mode). The algorithm for learning DNF is the WTR generating algorithm described in Section 4.2. This algorithm generates a set of disjuncts as a concept description, and it provides the performance baseline for TR and WTR modes. The TR mode is the FCLS algorithm with only threshold adjusting but without weight learning. The WTR mode is the FCLS algorithm with both threshold adjusting and weight learning. In our experiments, we varied values of several parameters of C4.5 and the best results were presented. We used rules generated by C4.5. C4.5 provides another performance baseline for the FCLS method.

The performance was evaluated for two dependent variables: classification accuracy and description complexity. Classification accuracy was measured as the percentage of correct classifications made by the concept description on a set of test examples. The testing set was either the entire space of examples or a set of randomly selected examples that were not in the training set. Description complexity was measured by the numbers of rules (or WTRs) and conditions involved in a description.

In all experiments, FCLS was run on training sets of various sizes. For each training set size, FCLS was run on ten different randomly generated training sets. The final descriptions produced from these ten runs were tested for accuracy on a set of test examples, and measured for complexity respectively. The results reported are the average of the ten runs. *Beam_Width* was set to three for all experiments. *Max_Tries* was set to five. Both *Min_Coverage* and *Max_Err_Rate* were set to 0 for the experiments reported in Sections 6.2, 6.3 and 6.4, so the descriptions generated by FCLS were complete and consistent. The experimental results with varying values of *Min_Coverage* are reported in Section 6.5.

6.2. Experiments on Designed Problems

The experiments described in this subsection were performed on three specially designed problems, called *designed problem I to III*. These problems were designed to test the novel features in FCLS. All test sets included 1000 randomly selected examples and were disjoint with all training sets.

Designed Problem I is an m-of-n concept and contains ten nominal attributes with four values each: 0, 1, 2, and 3. The target concept has the general form of "at least k of n conditions are satisfied." Specifically, the concept is "if and only if any five or more of the first seven attributes of an example have value 0 or 1, then the example belongs to the concept." This description can be compactly represented by one WTR:

$$\begin{aligned}&[x_1 = 0 \vee 1 : 1] \,\&\, [x_2 = 0 \vee 1 : 1] \,\&\, [x_3 = 0 \vee 1 : 1] \,\&\, [x_4 = 0 \vee 1 : 1]\\&\quad \&\, [x_5 = 0 \vee 1 : 1] \,\&\, [x_6 = 0 \vee 1 : 1] \,\&\, [x_7 = 0 \vee 1 : 1]\\&\text{Threshold} = 5/7 = 0.71\end{aligned}$$

where the number following a ":" is the weight of the condition. In a DNF representation, 21 disjuncts are needed to represent the concept.

Designed Problem II consists of eight linear attributes each of which has four values: 0, 1, 2 and 3. The target concept is described by six conditions, two of which are twice as important as the other four conditions. Specifically, the concept is expressed by one WTR:

$$\begin{aligned}&[x_1 = 0 \vee 1 : 2] \,\&\, [x_2 = 0 \vee 1 : 2] \,\&\, [x_3 = 0 \vee 1 : 1] \,\&\, [x_4 = 0 \vee 1 : 1]\\&\quad \&\, [x_5 = 0 \vee 1 : 1] \,\&\, [x_6 = 0 \vee 1 : 1]\\&\text{Threshold} = 5/8 = 0.625\end{aligned}$$

This problem is more complicated than designed problem I because all attributes are linear and the weights of all conditions are not equal.

Designed Problem III contains 15 binary attributes. The target concept can be described by two WTRs, each of which consists of 6 conditions, two of which are twice as important as the other four. The target concept is described by the disjunction of the following two WTRs:

$$\begin{array}{ll}\text{WTR1:} & \text{WTR2:}\\ \quad [x_1 = 0 : 2] \,\&\, [x_2 = 0 : 2] & \quad [x_7 = 0 : 2] \,\&\, [x_8 = 0 : 2]\\ \qquad \&\, [x_3 = 0 : 1] & \qquad \&\, [x_9 = 0 : 1]\\ \quad [x_4 = 0 : 1] \,\&\, [x_5 = 0 : 1] & \quad [x_{10} = 0 : 1] \,\&\, [x_{11} = 0 : 1]\\ \qquad \&\, [x_6 = 0 : 1] & \qquad \&\, [x_{12} = 0 : 1]\\ \quad \text{Threshold} = 5/8 = 0.625 & \quad \text{Threshold} = 5/8 = 0.625\end{array}$$

Tables 6, 7, and 8 show the detailed results of the experiments with the three designed problems, respectively. Each table contains five columns: *Training Set Size*, *Learning*

Table 6. Results from Designed Problem I.

Training set size	Learning method		Accuracy	#Rules	#Conds
100	C4.5	RULE	74% ± 2%	6.2 ± 0.7	8.9 ± 1.6
	FCLS	DNF	78% ± 2%	8.0 ± 0.5	40.3 ± 4.5
		TR	89% ± 4%	3.5 ± 0.7	23.6 ± 4.3
		WTR	91% ± 4%	2.7 ± 0.5	20.2 ± 2.1
200	C4.5	RULE	76% ± 3%	8.6 ± 1.1	14.8 ± 3.0
	FCLS	DNF	80% ± 1%	12.5 ± 0.6	74.8 ± 6.0
		TR	99% ± 2%	2.6 ± 0.7	19.5 ± 6.5
		WTR	98% ± 2%	2.8 ± 0.7	21.5 ± 4.9
300	C4.5	RULE	79% ± 2%	10.9 ± 1.0	20.8 ± 3.2
	FCLS	DNF	82% ± 1%	16.7 ± 1.3	104.4 ± 10.9
		TR	99% ± 1%	2.6 ± 0.8	19.4 ± 6.6
		WTR	99% ± 1%	2.4 ± 0.4	18.8 ± 3.7
400	C4.5	RULE	78% ± 2%	12.3 ± 1.1	25.0 ± 3.1
	FCLS	DNF	84% ± 1%	20.9 ± 1.3	133.5 ± 11.1
		TR	99% ± 2%	3.7 ± 1.9	29.1 ± 16.5
		WTR	98% ± 2%	3.4 ± 1.1	27.7 ± 10.1

Table 7. Results from Designed Problem II.

Training set size	Learning method		Accuracy	#Rules	#Conds
100	C4.5	RULE	79% ± 1%	4.1 ± 0.8	9.5 ± 2.4
	FCLS	DNF	77% ± 2%	9.2 ± 0.6	46.3 ± 3.3
		TR	82% ± 2%	5.7 ± 0.5	35.5 ± 3.9
		WTR	83% ± 2%	5.0 ± 0.4	32.3 ± 3.2
200	C4.5	RULE	82% ± 2%	5.5 ± 0.9	12.9 ± 3.1
	FCLS	DNF	79% ± 2%	15.9 ± 0.7	89.7 ± 5.3
		TR	85% ± 1%	9.8 ± 1.1	62.7 ± 7.2
		WTR	88% ± 2%	7.8 ± 1.2	52.6 ± 7.1
	C4.5	RULE	82% ± 1%	5.2 ± 1.0	12.1 ± 3.0
	FCLS	DNF	82% ± 2%	20.8 ± 1.5	120.7 ± 9.6
		TR	86% ± 2%	12.0 ± 1.4	87.4 ± 10.1
		WTR	91% ± 1%	9.0 ± 1.2	69.3 ± 9.3
400	C4.5	RULE	82% ± 1%	5.6 ± 0.5	13.0 ± 3.6
	FCLS	DNF	82% ± 1%	26.3 ± 1.7	160.1 ± 11.6
		TR	87% ± 1%	17.8 ± 0.8	122.4 ± 6.5
		WTR	93% ± 2%	8.9 ± 2.0	75.5 ± 12.8

Method, *Accuracy*, *#rules*, and *#conds*. *Training Set Size* is the size of a training set. *Accuracy* is the percentage of correctly classified test examples. *#rules* and *#conds* are the number of rules (WTRs) and conditions involved in concept descriptions generated, respectively. All results are accompanied by a 95% confidence interval calculated using a Student t-test.

In all three problems, significant improvements were achieved on both accuracy and complexity by the TR and WTR modes over the DNF mode at all training set sizes. The TR and WTR modes obtained significantly higher accuracies than C4.5. C4.5 generated fewer rules and conditions than the TR and WTR modes in Designed Problems II and III. This is

Table 8. Results from Designed Problem III.

Training set size	Learning method		Accuracy	#Rules	#Conds
100	C4.5	RULE	80% ± 3%	4.6 ± 0.8	9.9 ± 3.2
	FCLS	DNF	79% ± 2%	13.5 ± 1.0	59.2 ± 5.8
		TR	82% ± 2%	7.9 ± 0.7	57.9 ± 5.3
		WTR	85% ± 2%	6.1 ± 1.0	50.7 ± 8.9
200	C4.5	RULE	80% ± 2%	5.4 ± 1.3	12.7 ± 4.2
	FCLS	DNF	81% ± 1%	22.9 ± 0.8	112.4 ± 7.0
		TR	84% ± 2%	13.5 ± 1.1	103.0 ± 7.8
		WTR	86% ± 1%	10.1 ± 1.4	85.4 ± 10.1
300	C4.5	RULE	82% ± 2%	6.4 ± 1.2	16.5 ± 4.5
	FCLS	DNF	83% ± 1%	28.6 ± 1.4	151.3 ± 10.5
		TR	86% ± 2%	18.3 ± 1.5	140.7 ± 12.5
		WTR	90% ± 2%	13.2 ± 1.5	120.8 ± 13.7
400	C4.5	RULE	82% ± 2%	5.8 ± 1.3	13.8 ± 3.8
	FCLS	DNF	84% ± 1%	37.4 ± 2.3	210.5 ± 18.4
		TR	87% ± 1%	23.6 ± 1.6	187.9 ± 18.0
		WTR	92% ± 2%	15.5 ± 1.0	144.6 ± 9.8

because C4.5 simplified the rules generated and allowed inconsistency and incompleteness in rules. Rules generated in the TR and WTR modes were consistent and complete. The majority of these rules cover only a small number of examples and may be removed without significantly degrading accuracy. It is almost always true that the average number of conditions in a rule generated by the TR or WTR mode is larger than the average number of conditions in a rule generated by the DNF mode. This result is due to the fact that a TR or WTR often contains a highly specialized disjunct. The results from Designed Problem I show that the TR and WTR modes have very similar performances. This was expected because all conditions of the target concept are equally important, and weights play no role.

The WTR mode outperformed the TR mode in Designed Problem II and Designed Problem III. These improvements are due to the weight learning in the WTR mode. Conditions of target concepts in these two problems are weighted differently. The TR mode does not perform weight learning, so it fails to capture different weights of conditions. The accuracies in these two problems are not as high as those in Designed Problem I. This can be explained as follows. In the WTR mode, weights are adjusted based on training examples. It is almost impossible to learn the exact weights of target concepts, thus the descriptions generated in these two problems are only approximations of the target concept descriptions. Linear attributes in Designed Problem II and disjunction in Designed Problem III increase learning difficulty.

FCLS generated descriptions for both concepts, one for the positive concept and one for the negative concept. In Designed Problems I and II, FCLS generated one WTR for the positive concept in the most cases, but generated more than one WTR for the negative concept. Most errors were introduced by the description of the negative concept. FCLS needs to be improved so that it generates rules for positive examples only.

We claimed that FCLS can estimate the degree of typicality of members of concepts. This claim is consistent with the experimental results that show a more typical example gets a higher relative similarity to the concept description. For instance, in our experiments, the relative similarity of the example (0 0 1 0 0 0 1 1 0 3) in the Designed Problem I is 1, while the relative similarity of the example (1 1 2 3 1 1 0 2 3 3) is 0.05. (0 0 1 0 0 0 1 1 0 3) is a typical example, because its first seven values are either 0 or 1. (1 1 2 3 1 1 0 2 3 3) is a boundary example, because only five of its first seven values are either 0 or 1.

6.3. Experiments on 11-Multiplexor and 4-Term 3DNF

This section describes the experiments on the 11-multiplexor and a 4-term 3DNF. The hybrid representation has no advantage over logic representations in representing the concepts involved in these two problems. Adversely, the hybrid representation complicates these tasks because it necessitates searching a larger hypothesis space. During testing, the entire event space was used as the test set in the 11-multiplexor problem, and 1000 testing events were used for the 4-term 3DNF problems.

11-Multiplexor. The *Multiplexor* 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* bits + number of *data* bits) involved. F_{11} has three *address* bits and eight *data* bits. The size of the instance space is 2048. Each of the two target concept descriptions (positive and negative) consists of eight disjuncts and 32 conjuncts.

4-Term 3DNF Boolean Function. A l-term kDNF boolean function (Pagallo and Haussler, 1988) consists of l disjunctive terms of at most k conjuncts each. The 4-term 3DNF boolean function used in the experiments consists of four disjuncts, each of which has exactly three conjuncts. No attribute is shared by two disjuncts, and the total number of attributes in this problem is 16.

The results from these two problems are reported in Tables 9 and 10, respectively. In the 11-multiplexor problem, the DNF mode performed better than the TR mode when the training set size was 100. It achieved about the same performance as the TR mode when the training set sizes were 200, 300 and 400. Both the DNF and the TR modes significantly outperformed the WTR mode when the training set sizes were 100, 200 and 300. All three modes performed at about the same level when the training set size was 400. The target of 11-multiplexor is a DNF expression, so the DNF representation is the most appropriate representation. The TR representation enforces less representational bias than the DNF representation, while the WTR representation enforces less representational bias than the TR representation. Therefore, more examples are needed for the WTR mode to converge to the target concept than for the TR and DNF modes. C4.5 achieved higher accuracy than the WTR mode, but lower than the DNF and TR modes. The results achieved by C4.5 were similar to the results reported in (Quinlan, 1993).

As shown in Table 10, the accuracy of the WTR mode is worse than that of the DNF and TR modes. Similar to 11-multiplexor, the TR mode achieved about the same accuracies as the DNF mode when the training set sizes were 200, 300, and 400. Actually, the TR mode obtained slightly higher accuracies than the DNF mode when the training set sizes were 300 and 400. In some trials, the DNF and TR modes generated exactly the same descriptions. This interesting result shows that the TR mode succeeds in adjusting its representation for a given problem, but the WTR mode does not.

Table 9. Results from 11-multiplexor.

Training set size	Learning method		Accuracy	#Rules	#Conds
100	C4.5	RULE	67% ± 3%	12.4 ± 1.5	42.9 ± 6.1
	FCLS	DNF	77% ± 5%	19.3 ± 0.8	93.4 ± 4.4
		TR	69% ± 4%	17.2 ± 0.9	112.9 ± 7.1
		WTR	71% ± 2%	13.5 ± 2.5	104.2 ± 6.2
200	C4.5	RULE	88% ± 4%	18.6 ± 1.3	74.1 ± 6.0
	FCLS	DNF	96% ± 2%	20.9 ± 1.2	91.7 ± 10.3
		TR	95% ± 2%	18.5 ± 1.5	112.9 ± 7.1
		WTR	82% ± 3%	23.9 ± 1.0	157.3 ± 10.9
300	C4.5	RULE	96% ± 4%	18.3 ± 0.9	74.6 ± 4.1
	FCLS	DNF	99% ± 1%	21.5 ± 1.2	95.8 ± 8.0
		TR	99% ± 1%	22.0 ± 1.2	99.6 ± 8.3
		WTR	93% ± 2%	24.2 ± 1.9	143.1 ± 127
400	C4.5	RULE	100% ± 0%	16.8 ± 0.7	67.2 ± 0.7
	FCLS	DNF	100% ± 0%	20.8 ± 1.1	90.8 ± 7.3
		TR	100% ± 0%	20.8 ± 1.2	89.4 ± 6.5
		WTR	99% ± 1%	21.5 ± 1.4	112.4 ± 10.0

Table 10. Results from the 4-term 3DNF boolean function.

Training set size	Learning method		Accuracy	#Rules	#Conds
100	C4.5	RULE	73% ± 4%	9.7 ± 1.8	29.9 ± 7.8
	FCLS	DNF	89% ± 4%	11.6 ± 0.6	44.3 ± 3.5
		TR	74% ± 3%	9.9 ± 1.3	78.7 ± 9.2
		WTR	75% ± 2%	9.3 ± 1.0	75.9 ± 6.2
200	C4.5	RULE	94% ± 6%	10.6 ± 1.0	32.6 ± 1.9
	FCLS	DNF	95% ± 1%	14.5 ± 0.4	59.0 ± 5.0
		TR	93% ± 3%	14.5 ± 1.0	102.9 ± 8.7
		WTR	85% ± 2%	13.3 ± 0.4	103.5 ± 2.3
300	C4.5	RULE	98% ± 1%	12.8 ± 0.9	40.3 ± 4.0
	FCLS	DNF	96% ± 1%	15.3 ± 1.3	63.3 ± 8.0
		TR	99% ± 1%	15.3 ± 1.9	111.5 ± 12.7
		WTR	88% ± 2%	17.6 ± 1.0	143.1 ± 10.6
400	C4.5	RULE	100% ± 0%	15.5 ± 1.5	51.8 ± 7.3
	FCLS	DNF	98% ± 1%	16.7 ± 1.0	69.8 ± 6.0
		TR	100% ± 0%	14.7 ± 2.3	107.4 ± 21.6
		WTR	92% ± 1%	20.1 ± 0.8	160.0 ± 6.5

6.4. Experiments on Two Practical Problems

In addition to artificial domains, two practical domains, congressional voting records and lymphatic cancer, were also used to test FCLS. The data regarding the U.S. Congressional voting records were the same as the ones used by Lebowitz (1987) in his experiments on conceptual clustering. The data represent the 1981 voting records of 100 selected representatives, each of which is characterized by 19 attributes. The problem was to learn descriptions discriminating between the voting records of Democrats and Republicans. Ten training sets for each of the four training sizes (20, 40, 60, 80) were formed by randomly drawing examples from the 100 examples. All test sets were the remaining examples. The

Table 11. Results from the congressional voting records.

Training set size	Learning method		Accuracy	#Rules	#Conds
20	C4.5	RULE	79% ± 2%	2.9 ± 0.6	3.3 ± 0.9
	FCLS	DNF	78% ± 5%	2.7 ± 0.6	7.6 ± 3.2
		TR	79% ± 3%	2.0 ± 0.0	7.5 ± 2.8
		WTR	77% ± 3%	2.4 ± 0.5	8.3 ± 3.4
40	C4.5	RULE	80% ± 3%	3.5 ± 0.6	5.0 ± 1.7
	FCLS	DNF	79% ± 4%	4.5 ± 1.0	22.5 ± 5.4
		TR	85% ± 3%	2.3 ± 0.5	17.4 ± 4.6
		WTR	82% ± 5%	3.1 ± 0.8	19.2 ± 5.5
60	C4.5	RULE	81% ± 5%	3.7 ± 0.4	5.4 ± 0.9
	FCLS	DNF	81% ± 4%	6.4 ± 0.6	36.8 ± 4.8
		TR	86% ± 4%	3.2 ± 0.9	27.9 ± 5.9
		WTR	86% ± 3%	4.0 ± 0.5	33.0 ± 3.9
80	C4.5	RULE	76% ± 6%	3.4 ± 0.7	5.8 ± 1.6
	FCLS	DNF	80% ± 5%	8.1 ± 1.0	52.0 ± 8.5
		TR	87% ± 2%	4.1 ± 1.0	38.9 ± 7.7
		WTR	86% ± 4%	5.3 ± 0.6	47.6 ± 7.8

Table 12. Results of the lymphatic cancer.

Training set size	Learning method		Accuracy	#Rules	#Conds
25	C4.5	RULE	70% ± 5%	6.0 ± 1.1	10.0 ± 2.2
	FCLS	DNF	71% ± 5%	6.0 ± 0.5	19.2 ± 2.0
		TR	68% ± 5%	4.0 ± 0.0	20.4 ± 3.5
		WTR	70% ± 6%	4.8 ± 0.6	18.6 ± 1.7
50	C4.5	RULE	74% ± 4%	8.1 ± 1.5	14.7 ± 3.7
	FCLS	DNF	76% ± 3%	7.9 ± 0.8	32.1 ± 3.8
		TR	80% ± 3%	5.3 ± 0.6	32.8 ± 5.4
		WTR	81% ± 3%	5.7 ± 0.7	31.6 ± 4.8
75	C4.5	RULE	76% ± 4%	8.5 ± 1.4	15.8 ± 3.6
	FCLS	DNF	79% ± 4%	9.3 ± 0.6	45.2 ± 6.3
		TR	83% ± 4%	6.9 ± 0.6	46.3 ± 3.4
		WTR	82% ± 3%	6.9 ± 0.4	46.9 ± 3.2
100	C4.5	RULE	78% ± 5%		15.2 ± 3.0
	FCLS	DNF	82% ± 4%	10.3 ± 0.7	55.0 ± 3.8
		TR	86% ± 4%	7.1 ± 0.7	54.9 ± 6.2
		WTR	82% ± 3%	7.0 ± 0.6	53.4 ± 6.2

results are reported in Table 11. Both the WTR and the TR modes significantly improved the accuracy over the DNF mode at the training set sizes 20, 40, and 60. The WTR and TR modes performed similarly.

The lymphatic cancer data is characterized by 18 attributes and 4 diagnostic classes. Data of 148 patients were available. 25, 50, 75, and 100 examples were randomly selected for learning respectively, the remaining 48 examples were used as the test set. The results are shown in Table 12. Both the WTR and TR methods attained higher accuracy except at the size of 25. The description generated by the WTR and TR methods are simpler than the DNF mode.

Table 13. Results from the congressional voting records for varying values of *MIN_COVERAGE.*

MIN_COVERAGE	Learning method	Accuracy	#Rule	#Exemplar
0.00	TR	87% ± 2%	4.1± 1.0	0 ± 0
	WTR	86% ± 4%	5.3± 0.6	0 ± 0
0.05	TR	88% ± 4%	4.0± 0.7	1.0 ± 0.4
	WTR	85% ± 3%	4.5± 0.6	1.0 ± 0.7
0.10	TR	90% ± 3%	3.4± 0.4	1.7 ± 1.1
	WTR	87% ± 4%	3.5± 0.5	3.2 ± 1.3
0.15	TR	87% ± 5%	2.8± 0.4	2.9 ± 1.0
	WTR	88% ± 4%	2.8± 0.3	3.5 ± 1.0
0.20	TR	85% ± 4%	2.5± 0.4	3.2 ± 1.6
	WTR	86% ± 6%	2.6± 0.0	3.9 ± 1.3
	Exemplar-based	84% ± 7%	0.0± 0.0	17.0 ± 1.5

Table 14. Results of the lymphatic cancer for varying of *MIN_COVERAGE.*

MIN_COVERAGE	Learning method	Accuracy	#Rule	#Exemplar
0.00	TR	86% ± 4%	7.1 ± 0.7	0 ± 0
	WTR	82% ± 3%	7.0 ± 0.6	0 ± 0
0.05	TR	88% ± 3%	6.4 ± 0.5	0.3 ± 0.4
	WTR	84% ± 4%	6.6 ± 0.5	0.8 ± 0.8
0.10	TR	90% ± 4%	5.2 ± 0.3	4.0 ± 1.2
	WTR	85% ± 5%	5.3 ± 0.5	4.0 ± 1.7
0.15	TR	89% ± 3%	4.4 ± 0.4	5.8 ± 1.9
	WTR	83% ± 4%	4.5 ± 0.4	6.7 ± 2.5
0.20	TR	88% ± 2%	4.3 ± 0.4	6.3 ± 1.6
	WTR	83% ± 4%	4.2 ± 0.3	7.6 ± 2.9
	Exemplar-based	78% ± 4%	0.0 ± 0.0	26.8 ± 2.6

6.5. *Experiments with Varying MIN_COVERAGE VALUES*

We also conducted some preliminary experiments to evaluate the use of exemplars in FCLS on these two practical problems. We ran FCLS in the TR and WTR modes with varying values of the parameter *MIN_COVERAGE.* The larger the value of *MIN_COVERAGE*, the more exemplars may be included in the final descriptions. Tables 13 and 14 show the results from congressional voting records and lymphatic cancer, respectively. The results obtained from these two problems are very similar. When some exemplars were included to replace some small TRs or WTRs, the accuracies increased. After too many TRs or WTRs were replaced by exemplars, the accuracies decreased. More experiments need to be conducted to thoroughly evaluate the use of exemplars.

6.6. *Experiments on Instance-Based Learning*

Instance-based learning (IBL) algorithms are closely related to the FCLS algorithm and are used to learn concepts with graded structures. The major difference between IBL and FCLS is that FCLS generates generalized instances. This section discusses the experiments conducted to empirically compare IBL with FCLS. IB3 (provided by D. Aha) was

Table 15. Results of IB3, the TR mode, and the WTR mode.

	Training set	IB3		TR		WTR	
Problem	size	Accuracy	#ins	Accuracy	#Rules	Accuracy	#Rule
DPI	200	79% ± 1	180 ± 8	99% ± 2	3 ± 1	98% ± 2	3 ± 1
	400	80% ± 1	372 ± 10	99% ± 2	4 ± 2	98% ± 2	3 ± 1
DPII	200	77% ± 3	160 ± 16	85% ± 1	10 ± 1	88% ± 2	8 ± 1
	400	79% ± 4	330 ± 26	87% ± 1	18 ± 1	93% ± 1	9 ± 2
DPIII	200	77% ± 2	151 ± 18	84% ± 2	14 ± 1	86% ± 1	10 ± 1
	400	78% ± 3	321 ± 33	87% ± 1	24 ± 2	92% ± 2	16 ± 1

run on the three designed problems (DP I, DP II, and DP III) and the results are shown in Table 15. We varied values of several important parameters (Suggested by Dr. Aha) in these experiments and selected the best results. FCLS performed significantly better than IB3 on all the three problems. The significant improvement achieved by FCLS is partly due to the generalization. When a graded concept is highly disjunctive, IBL may perform better than FCLS. For example, the least improvement achieved by FCLS on these problems was on the Designed Problem III in which the concept is described by two WTRs.

6.7. *Experiments on Noisy Data*

We have not yet addressed the issue of handling noisy data. FCLS can be extended to be noise tolerant to some degree. First, the parameter *Max_Err_Rate* provides a means of coping with noisy data. When *Max_Err_Rate* is larger than 0, the description generated by FCLS is allowed to be inconsistent. That is, it can cover some noisy negative examples. Second, exceptional cases in FCLS are represented as exemplars. Noisy examples are similar to exceptional cases. Therefore when the data is noisy, we can turn off the exemplar selection module so that no exceptional cases are stored as exemplars. This is controlled by the parameter *Min_Coverage*. This section describes the experiments conducted to evaluate the noise handling ability of FCLS.

The problem used in these experiments was the Designed Problem I with 200 training examples and 1000 test examples. We ran FCLS on the 200 training examples containing different levels of noise with varying values of *Max_Err_Rate* and *Min_Coverage*. Noisy data were generated by switching the class membership of randomly selected examples. For simplicity, *Max_Err_Rate* and *Min_Coverage* were always set to the same value. It is not necessary to set *Max_Err_Rate* and *Min_Coverage* to the same value. In fact, better results may be obtained from other combinations of *Max_Err_Rate* and *Min_Coverage* values. FCLS was run under the TR mode. For comparison, C4.5 and IB3 were also run on the same data sets with different settings of parameters. Table 16 shows the experimental results, where *Acc* represents accuracy.

The results were consistent with what we expected. Accuracy and simplicity of concept descriptions decreased as the noise level increased. Accuracy increased as the values of the two parameters increased. At the noise levels of 5% and 10%, accuracy began to drop when *Max_Err_Rate* and *Min_Coverage* became too large. Simplicity of concept descriptions was improved with increasing noise levels. One difficulty with this noise handling method is

Table 16. Experimental results on noisy data.

	FCLS(TR mode)													
Noise	0.00		0.05		0.10		0.15		0.20		C4.5		IB3	
level	Acc%	#Rules	Acc%	#Rules	Acc%	#Rules	Acc%	#Rules	Acc%	#Rules	Acc%	#Rules	Acc%	#ins
0%	99 ± 2	3 ± 1									76 ± 3	9 ± 1	79 ± 1	180 ± 8
5%	81 ± 2	10 ± 1	89 ± 3	6 ± 1	91 ± 2	4 ± 1	87 ± 3	3 ± 1	87 ± 3	2 ± 0	74 ± 4	10 ± 2	76 ± 1	13 ± 1
10%	76 ± 2	10 ± 1	83 ± 2	9 ± 1	86 ± 2	5 ± 1	88 ± 2	3 ± 1	86 ± 2	3 ± 0	76 ± 3	11 ± 1	76 ± 1	13 ± 1
15%	71 ± 2	15 ± 1	78 ± 3	11 ± 1	79 ± 2	7 ± 1	82 ± 5	4 ± 1	85 ± 2	2 ± 0	72 ± 5	12 ± 2	77 ± 1	13 ± 2
20%	68 ± 2	16 ± 1	73 ± 3	12 ± 1	73 ± 2	8 ± 1	77 ± 3	5 ± 1	80 ± 4	3 ± 1	68 ± 8	24 ± 2	76 ± 1	13 ± 2

the determination of *Max_Err_Rate* and *Min_Coverage* values. Generally speaking, higher values are required for *Max_Err_Rate* and *Min_Coverage* for higher noise levels.

From Table 16, it can be observed that both C4.5 and IB3 are highly noise tolerant, since their accuracies did not degrade much with noisier data. For noisy data, the highest accuracy for IB3 was obtained when the parameter *storeall*, which determines the number of instances stored, was set to *off*. For nonnoisy data, IB3 achieved the highest accuracy when *storeall* was set to *on*.

6.8. *Summary*

FCLS worked significantly better on graded concepts than other learning methods, such as C4.5, IB3, and AQ-like methods. The WTR mode achieved accuracies higher than the TR mode on concepts with weighted conditions, but the differences were not substantial. The TR mode performed as well as the DNF mode and C4.5 on concepts such as DNF and Multiplexor, when enough training examples were available. This occurred because the TR mode can adjust its representation to adapt concepts to be learned. In contrast, the WTR mode failed to adjust its representation, so that it did not perform well on these classically defined concepts. Improvements were achieved if exemplars were added to the hybrid representation. More experiments need be conducted in order to reach a conclusion. Finally, the noise handling capabilities of FCLS have been empirically evaluated. The results show that this method did improve the performance on a noisy domain, but the problem of how to specify the values of those two parameters exists. The noise handling method of FCLS needs to be improved.

7. Related Work

This section relates FCLS to some other works, which include Schlimmer's STAGGER (1987), Bergadano, et al.'s POSEIDON (1992), Salzberg's NGE (1991), and Murphy and Pazzani's GS (1991).

FCLS is similar to STAGGER in that they both utilize hybrid representations that combine a numeric representation with a symbolic representation, and they both perform partial matching to classify examples. In STAGGER, each pattern is associated with a pair of weights that capture the relative importance of the pattern to the overall description of the concept. Patterns can consist of Boolean functions of attribute values. Weights in FCLS are associated only with conditions (attribute values) of a WTR and reflect the relative importance of the conditions to the WTR. If a pattern in STAGGER is a boolean function of attribute values, then no weight is associated with the attribute values of the pattern. The degree of match between an example and a concept description in STAGGER is measured using a Bayesian approach, whereas it is evaluated by a distance measure and an adjustable threshold in FCLS. When attributes are nominal, the difference between these two measures is minor.

In addition to the differences of representations and matching procedures, the FCLS learning algorithm differs from STAGGER's. In STAGGER, the weights of the current patterns are first computed from the training examples, and then are used to classify the examples in an approach similar to a Bayesian classifier. If this does not work well, then new patterns are constructed and the weights of these new patterns are computed. STAGGER begins with a simple concept description and performs a search towards more complex descriptions, which can take a great deal of time. FCLS instead uses a simpler general-to-specific beam search that should allow it to learn many complex concepts more quickly. STAGGER performs constructive induction and learns incrementally, while FCLS cannot. In general, FCLS combines exemplar-based learning and rule learning, while STAGGER combines a statistical learning approach and a rule-learning approach.

POSEIDON (Bergadano, et al., 1992) generates two-tiered concept descriptions. The first tier, called *Base Concept Representation*, explicitly captures basic concept properties and is created in two phases. In phase 1, AQ15 is used to induce a complete and consistent concept description from supplied examples. In phase 2, this description is optimized by removing some disjuncts and conjuncts. This optimization process is guided by a domain-dependent quality criterion. The second tier, called *Inferential Concept Interpretation* (ICI), characterizes allowable concept modifications and context dependency, and consists of a procedure for partial matching and a set of inference rules. The partial matching procedure is predefined and the inference rules are supplied by human experts based on the explanation of exceptional cases (which are generated by the system).

In FCLS, a disjunct of a concept description is generated in one step by performing a general-to-specific beam search. In the second phase of POSEIDON, a disjunct can never be specialized. Therefore, if AQ15 generates some overgeneralized disjuncts, then POSEIDON has no way to improve them. For this reason, POSEIDON cannot learn n-of-m concepts. In POSEIDON, the partial matching procedure is predefined, while the partial matching procedure in FCLS is adjustable during learning by modifying its weights and the threshold. Moreover, FCLS's partial matching procedure is different among disjuncts. FCLS does not include inference rules in its partial matching procedure. In general, FCLS representation reduces the number of rules used to represent concepts. However, its rules are more complex and the evaluation of rules is more complex. POSEIDON should be applied when conceptual interpretation of concepts is important. FCLS should be applied when a bias for graded behavior is useful.

POSEIDON was experimentally applied on two different problems: labor management contracts and congressional voting data (Bergadano, et al., 1992). To empirically compare FCLS with POSEIDON, FCLS was run on these two problems with the same training and testing data. Table 17 shows the results which were the average of two runs. The training set and test set of the labor contract data consisted of 27 and 30 examples, respectively. Both the training set and test set of the congressional voting data included 50 examples. In Table 17, #rules (#conds) is the number of disjuncts (conjuncts) involved in the descriptions generated.

POSEIDON achieved higher accuracy in the problem of labor management contracts, while FCLS attained higher accuracy than POSEIDON in the problem of congressional

Table 17. Results from two domains used in POSEIDON.

	Labor contract			Congressional voting		
	Accuracy	#Rules	#Conds	Accuracy	#Rules	#Conds
POSEIDON	90%	9	12	92%	10	21
FCLS						
DNT	83%	7	19	89%	5	29
TR	85%	2	13	96%	2	26
WTR	86%	2	13	93%	3	20

voting records. The higher accuracy achieved by POSEIDON in the problem of labor management contracts is because it was given the ICI rules by a human expert. POSEIDON's descriptions included three rules provided by human experts. Without these rules, POSEIDON's accuracy was only 83%, which is lower than FCLS's. The major reason for the higher accuracy obtained by FCLS is that FCLS can capture the graded structure of concepts. In the domain of congressional voting records, most of the sixteen attributes are relevant to distinguish republicans from democrats, but only some of the conditions involved in these relevant attributes need to be satisfied in order to distinguish a republican congressman from a democratic congressman. In other words, the target concept in this problem is similar to an n-of-m concept and has a graded structure. In both problems (except the WTR mode in Congressional Voting), FCLS generated only one WTR for each concept.

Salzberg (1991) described an exemplar-based learning approach called nested generalized exemplar (NGE) approach. There are several similarities between FCLS and NGE. First, both algorithms allow examples to be generalized. Second, both algorithms combine the uses of rules (generalized examples) with specific examples. Third, both algorithms dynamically adjust their distance functions by modifying the weights of attributes (conditions in FCLS). FCLS and NGE also differ on several aspects. First, NGE uses a single set of weights on its attributes, whereas FCLS learns a different weight for a condition depending on which rule it is in. Second, each WTR in FCLS is associated with a threshold that defines the boundary of the WTR, whereas each hyperrectangle in NGE is associated with a weight that changes the boundary of the hyperrectangle. Third, the learning algorithms of the two approaches are different. A hyperrectangle in NGE is generalized from specific examples, whereas a WTR in FCLS is generated by performing a general-to-specific beam search. NGE learns incrementally, whereas FCLS does not. NGE is strongly dependent on the order of examples presented. Finally, FCLS can estimate the typicality of concept members, but NGE cannot. In general, NGE was not designed for concepts with graded structures, therefore it is not appropriate for tasks involving graded concepts. To empirically demonstrate this claim, we ran NGE (provided by D. Aha) on the designed problems I and II. Aha (1995) helped us in running this experiment and we varied values of important parameters of NGE. Table 18 shows the results. NGE stored more instances than FCLS did; this result is due to the greater degree of generalization performed by FCLS. Because of the generalization performed by NGE, it stored fewer instances than IB3. NGE achieved about the same accuracy as IB3 on designed problem I, but much lower accuracy than IB3 on designed problem II.

GS was designed to learn m-of-n concepts. In GS, m-of-n concepts are learned by creating new terms corresponding to m-of-n concepts during induction of decision trees.

Table 18. Results of NGE, IB3, and the WTR mode.

Problem	Training set size	NGE		IB3		WTR	
		Accuracy	#Ins	Accuracy	#Ins	Accuracy	#Rule
DPI	200	78% ± 1	10 ± 1	79% ± 1	180 ± 8	98% ± 2	3 ± 1
	400	78% ± 1	12 ± 2	80% ± 1	372 ± 10	98% ± 2	3 ± 1
DPII	200	59% ± 3	99 ± 6	77% ± 3	160 ± 16	88% ± 2	8 ± 1
	400	61% ± 6	154 ± 18	79% ± 4	330 ± 26	93% ± 1	9 ± 2

A m-of-n term is represented as a list of relevant attributes and their values and an integer threshold. Each m-of-n term is generated by conducting a hill-climbing search. Two operators are applied to generate new m-of-n hypotheses. The first one adds an attribute-value pair to the relevant attribute list without increasing the threshold, and the second one adds an attribute-value pair to the relevant attribute list and increases the threshold by 1. The evaluation function of m-of-n hypotheses was not clearly given in (Murphy & Pazzini, 1991). The m-of-n terms generated are embedded in a decision tree as nodes, therefore GS can generate a disjunct of m-of-n terms as a concept description. GS and FCLS are similar in that both algorithms use a similar search method (beam search in FCLS and hill-climbing search in GS) to generate a m-of-n terms. Also, similar operators (adding attribute-value pair and adjusting the threshold in GS and removing attribute-value pair and adjusting the threshold) are used in search. The major difference between GS and FCLS is that the threshold in GS is an integer between 1 and n, while the threshold in FCLS is a real value between 0 and 1. Because of this difference, FCLS's representation is more powerful than that of GS and can represent different kind of graded concepts. While FCLS can process numeric attributes, GS was designed for use with only nominal attributes. GS does not have attribute weights, and it can only learn standard n-of-m concepts. The advantage of GS over FCLS is its simpler learning algorithm.

8. Conclusion and Future Work

The experiments presented in Section 6 demonstrate that FCLS can effectively learn concepts with graded structures. This result is first attributed to the hybrid concept representation used in FCLS. The hybrid representation combines a symbolic representation with a numeric one. The symbolic part of the representation is composed of rules, and the numerical part consists of weights on conditions and thresholds of rules. Unlike NGE, weights are assigned to conditions rather than attributes. That is, an attribute may have different weights depending on conditions in a given rule. The central tendency or the basic principle of a graded concept is explicitly described by the symbolic part, whereas the numerical part, together with a similarity measure, extends the symbolic part to describe less typical cases of graded concepts.

The hybrid representation can also be viewed as an extension of the exemplar-based representation. First, a WTR can be considered as a generalized exemplar. Second, specific exemplars may be stored as a part of a concept description to describe exceptions. Because a WTR is often a highly generalized exemplar, concept descriptions represented by the hybrid representation are useful for understanding concepts, comparing different concepts, identifying exceptions, and efficiently storing and using concept descriptions.

Although partial matching methods have been known for several decades, the two-step partial matching method used in FCLS is novel. The first step is a partial matching between an example and a WTR defined as a function of both their similarity and the WTR's threshold. FCLS dynamically adjusts the first step partial matching through weight learning and threshold adjustment. The second step measures the relative similarity between an example and a WTR. The first step of the partial matching method decides the coverage of a WTR, while the second step allows us to classify examples that do not match with any WTR and examples that match more than one WTR. The experimental results show that this two-step partial matching significantly improved classification accuracy in most problems that we tested.

Concept descriptions represented in our hybrid representation capture typicality information. An example with a larger relative similarity to a WTR is more typical than an example with a smaller relative similarity to a WTR.

Another accomplishment is the development of a technique for generating concept descriptions represented in the hybrid representation. This technique generates a concept description by adjusting the distribution between the symbolic and numeric representations, and between the generalized descriptions and the exemplars to achieve the 'best' performance for a given problem.

A number of problems need to be addressed in the future. The similarity measure in FCLS only measures the syntactic similarity of an example to a concept description. An interesting problem is to augment the current syntactic similarity measure with a knowledge-based semantic similarity measure. The semantic similarity measure will include a set of inference rules and defines the similarity between an example and a WTR based on the semantics. An approach similar to those used in POSEIDON (Bergadano, et al., 1992) and Protos (Bareiss, 1989) may be used as the augmented similarity measure.

Another related future research topic is constructive induction. In general, constructive induction can produce descriptions that are easier to understand, and capture the salient features of concepts. It would be useful to apply constructive induction to generate rules that capture the principles of concepts.

One of the limitations of FCLS is that it cannot learn incrementally. Incremental learning in FCLS involves not only generalizing and/or specializing current descriptions, but also adjusting the distribution between the symbolic and numeric representation and the distribution between the generalized descriptions and specific exemplars. When a new example is not correctly classified by the current description, some WTRs need to be generalized or specialized. For instance, if the new example is not covered by any WTR, then the weights and thresholds of the WTRs that are close to the new example are adjusted so that the new example can be covered by one of the WTRs. If the new example is still not covered after the weights and thresholds of some WTRs are adjusted, then the new example is stored as an exemplar. After a certain number of exemplars are accumulated, some new WTRs may be generated. WTRs surrounded by exemplars need to be relearned. The problem of incremental learning in FCLS merits further investigation.

Finally, the FCLS algorithm, especially the quality functions, is too complicated. We believe that it can be simplified without significantly degrading its performance. Currently, we are designing a simpler algorithm in which some of the features mentioned above are being incorporated.

Acknowledgments

Part of this research was done in the Center for Artificial Intelligence at George Mason University. The activities of the Center are supported in part by the Defense Advanced Research Projects Agency under the grants administered by the Office of Naval Research, No. N00014-91-J-1854, in part by the Office of Naval Research under No. N00014-91-J-1351. The first author was also supported by a Faculty Research Grant of Utah State University. The authors are grateful to Dr. David Aha from Naval Research Lab. for providing us IB3 and NGE and for helping us run experiments with IB3 and NGE. The authors thank Michael Hieb and Gheorghe Tecuci for many useful comments and suggestions. The authors also thank John Grefenstette and the reviewers for their detailed comments and suggestions.

Notes

1. For a detailed definition of nearly covered examples, see (Zhang, 1990).
2. Clark & Niblett (1989) reported that the time complexity of CN2 is $O(a^2 \cdot b \cdot n^2)$. They claimed that the overall time for a single specialization step is $O(a \cdot b \cdot n)$, but we think that it should be $O(a^2 \cdot b \cdot n)$ because each step generates $a \cdot b$ new complexes, each of which needs to be evaluated and evaluation of each complex takes time $O(a \cdot n)$.

References

Aha, D. (1995). *An implementation and experiment with the nested generalized exemplars algorithm*, Technical Report AIC-95-003, Naval Research Laboratory, Navy Center for Applied Research in Artificial Intelligence.

Aha, D., Kibler, D., & Albert, M. (1991). Instance-based learning algorithm, *Machine Learning*, 6:37–66.

Bareiss, E.R. (1989). *An exemplar-based knowledge acquisition*, Academic Press.

Barsalou, L. (1985). Ideals, central tendency, and frequency of instantiation as determinants of graded structure in categories, *Journal of Experimental Psychology: Learning, Memory and Cognition*, 11:629–654.

Bergadano, F., Matwin, S., Michalski, R.S., & Zhang, J. (1992). Learning two-tiered descriptions of flexible concepts, *Machine Learning*, 8:5–43.

Clark, P., & Niblett, T. (1989). The CN2 induction algorithm, *Machine Learning*, 3:261–283.

Lebowitz, M. (1987). Experiments with incremental concept formation: UNIMEN, *Machine Learning Journal*, 2:103–138.

Michalski, R.S. (1983). A theory and methodology of inductive learning, in R.S. Michalski, J.G. Carbonell, & T.M. Mitchell (Eds.), *Machine Learning: An Artificial Intelligence Approach* (Vol. 1) San Mateo, CA: Morgan Kaufmann.

Michalski, R.S. (1987). How to learn imprecise concepts: a method employing a two-tiered representation for learning, *Proceedings of the Fourth International Workshop on Machine Learning* (pp. 50–58), Irvine, CA.

Michalski, R.S. (1990). Learning flexible concepts: fundamental ideas and a method Bases on two-tiered representation, in Y. Kodratoff, & R.S. Michalski (Eds.), *Machine Learning: An Artificial Intelligence Approach* (Vol. 3), San Mateo, CA: Morgan Kaufmann.

Michalski, R.S., Mozetic, I., Hong, J., & Lavrac, N. (1986). The multi-purpose incremental learning system AQ15 and its testing application to three medical domains, *Proceedings of the Fifth National Conference on Artificial Intelligence* (pp. 1041–1045).

Murphy, P.M., & Pazzani, M.J. (1991). ID2-of-3: Constructive Induction of M-of-N Concepts for Discriminators in Decision Trees, *Proceedings of the Eighth International Workshop on Machine Learning* (pp. 183–187).

Pagallo, G., & Haussler, D. (1988). *Feature discovery in empirical learning*, Technical Report UCSC-CRL-88-08, University of California at Santa Cruz, Santa Cruz, CA.

Quinlan, J.R. (1986). Induction of decision trees, *Machine Learning*, 1:81–106.

Quinlan, J.R. (1987). Simplifying decision trees, *International Journal of Man-Machine Studies*, 27:221–234.

Quinlan, J.R. (1993). *C4.5 Programs for Machine Learning*, San Mateo, CA: Morgan Kaufmann.

Rosch, E.H., & Mervis, C.B. (1975). Family resemblances: studies in the internal structure of categories. *Cognitive Psychology*, 7:573–605.

Salzberg, S. (1991). A nearest hyperrectangle learning method, *Machine Learning*, 6:251–276.

Schlimmer, J.C. (1987). *Concept acquisition through representational adjustment*, Ph.D. thesis, Department of Information and Computer Science, University of California, Irvine.

Smith, E.E., & Medin, D.L. (1981). *Categories and Concepts*, Harvard University Press.

Utgoff, P.E. (1989). Incremental Induction of Decision Trees, Machine Learning, 4:161–186.

Zhang, J. (1990). *Learning flexible concepts from examples: employing the ideas of two-tiered concept representation*, Ph.D. Thesis, Department of Computer Science, University of Illinois at Urbana-Champaign.

Zhang, J. (1991). Learning flexible concepts with integrating symbolic and subsymbolic approaches, *Proceedings of International Workshop on Multistrategy Learning* (pp. 289–304), George Mason University.

Zhang, J. (1992). Selecting Typical Instances in Instance-Based Learning, *Proceedings of the Ninth International Conference on Machine Learning* (pp. 470–479), Aberdeen Scotland, Morgan Kaufmann.

Received November 19, 1991
Final Manuscript May 12, 1995