Model Generation by Domain Refinement and Rule Reduction

Thomas Sudkamp, *Member, IEEE*, Aaron Knapp, and Jon Knapp

***Abstract*—The granularity and interpretability of a fuzzy model are influenced by the method used to construct the rule base. Models obtained by a heuristic assessment of the underlying system are generally highly granular with interpretable rules, while models algorithmically generated from an analysis of training data consist of a large number of rules with small granularity. This paper presents a method for increasing the granularity of rules while satisfying a prescribed precision bound on the training data. The model is generated by a two-stage process. The first step iteratively refines the partitions of the input domains until a rule base is generated that satisfies the precision bound. In this step, the antecedents of the rules are obtained from decomposable partitions of the input domains and the consequents are generated using proximity techniques. A greedy merging algorithm is then applied to increase the granularity of the rules while preserving the precision bound. To enhance the representational capabilities of a rule and reduce the number of rules required, the rules constructed by the merging procedure have multi-dimensional antecedents. A model defined with rules of this form incorporates advantageous features of both clustering and proximity methods for rule generation. Experimental results demonstrate the ability of the algorithm to reduce the number of rules in a fuzzy model with both precise and imprecise training information.**

***Index Terms*—Fuzzy system models, granularity, rule learning algorithms, rule reduction.**

I. Introduction

WHEN a system is too complex or too poorly understood to be described in precise mathematical terms, fuzzy modeling provides the ability to linguistically specify approximate relationships between the input and the desired output. The relationships are represented by a set of fuzzy if-then rules in which the antecedent is an approximate representation of the state of the underlying system and the consequent provides a range of potential responses. Traditionally, expert analysis and heuristic estimation have been used to produce fuzzy models [11]–[13]. In spite of the intuitive advantages of encapsulating expert knowledge as fuzzy rules, the ability to produce rules by a heuristic analysis of a system becomes more challenging as the relationships within the system increase in number and complexity.

The difficulties encountered in heuristic rule construction have led to the development of alternative approaches for producing fuzzy models. Automatic rule generation techniques have been introduced to generate models from training data. When sufficient training information is available, these techniques can produce highly precise models. A cost incurred in obtaining the precision is that the resulting models lack a linguistic interpretation and, in the case of rule-based models, frequently consist of a large number of rules. Algorithms for model construction generally fall within one of four categories, namely, neural-fuzzy systems [14], [15]; evolutionary rule generation [16]–[18]; clustering [19], [20]; and proximity analysis [21]–[25]. The term *proximity analysis* is used to describe a category of rule learning algorithms in which the domain decompositions are predetermined and the consequent of a rule is obtained from an analysis of the training data that occur in the support of the antecedent of the rule.

There are two often conflicting properties that are desired in a fuzzy model: interpretability and precision. The former is accomplished by having a small number of rules of large granularity while the latter frequently requires multiple rules. Rather than attempt to produce a model that exhibits minimal error on the training data, the goal of this research is to generate models that are defined by a small number of rules and satisfy a prescribed precision bound. This objective is in keeping with the philosophy of approximate reasoning and fuzzy inference that exact or optimal responses are not necessarily required to provide acceptable solutions to a problem.

One method for achieving both of the objectives is to construct a rule base using a learning algorithm and then employ a rule reduction algorithm to reduce the number and increase the granularity of the rules. There are two primary strategies for reducing the number of rules in a fuzzy rule base: 1) dimension reduction and 2) rule merging. Dimension reduction attempts to determine functional relationships between input variables or identify variables that have minimal impact on the result. When these relationships are discovered, the variables are removed decreasing the dimension of the input space. Dimension reduction is frequently employed in classification problems in which the objects are defined by a large number of attributes. The objective of reducing the number of input variables is to facilitate the system design and to reduce the computational resources required. One popular approach to dimension reduction is to use singular value decomposition to identify input variables that may be eliminated [1]–[3].

The NEFCLASS system [4], [5] provides another approach for constructing a small rule base for a classification problem. Initially, the user specifies domain partitions and a maximum number of rules. A neuro-fuzzy system modifies the parameters that define the domain partitions while generating the rule base. A statistical analysis of the impact of the resulting rules on the classification of the training data provides information for the

Manuscript received June 20, 2001; revised January 15, 2002. This paper was recommended by Associate Editor G. Skarmeta.

T. Sudkamp is with the Department of Computer Science, Wright State University, Dayton, OH 45431 USA (e-mail: tsudkamp@cs.wright.edu).

A. Knapp and J. Knapp are with TriVectus, Phoenix, AZ 85032-5908 USA.

Digital Object Identifier 10.1109/TSMCB.2003.808186

user to delete attributes, fuzzy sets from domain partitions, or rules to reduce the size of the rule base.

Rule combination and region expansion are frequently applied to models with a small number of inputs, primarily in system modeling and function approximation applications. One approach to rule combination uses a similarity analysis of the fuzzy sets in the rules to identify similar rules, which are then combined into a single rule [6]. The objective of this type of merging is principally to remove rules providing redundant information. Berthold and Huber [7] provide an region expansion strategy for constructing a model from hyper-dimensional fuzzy points. An n-dimensional fuzzy point represents the antecedent of a rule with n input variables. A training instance initiates the construction of a fuzzy point and the consequent of the associated rule is the output fuzzy set that best matches the output of training instance. The region is extended as far as possible until another training point is encountered that is inconsistent with the associated output value.

The algorithm for model construction presented in this paper combines proximity-based rule generation with an iterative refinement of the partitions of the input domains to produce a model that satisfies the precision bound. A rule reduction algorithm is then used to merge rules while preserving the precision bound [8], [9]. The merging is accomplished by combining the regions of applicability of adjacent rules and constructing an appropriate consequent for the extended rule. The objective of the merging is to increase the granularity of the rule base, where the granularity may be considered a measure of the degree to which the rule generalizes training data. The granularity of a fuzzy rule is generally obtained from the sigma-count or the support of the rule antecedent [10]. The utility of rule merging is accentuated when there are a large number of rules, as is frequently the case when rules are generated from training data.

The merging strategy produces rules whose antecedents are multi-dimensional fuzzy sets. The particular form of the rules was selected for three reasons: to increase the representational capability of the rules, to facilitate rule merging, and to ensure an efficient evaluation of the inference procedure. Models built with the type of multi-dimensional rule described in Section II using triangular partitions of the input spaces constitute a superset of models produced using Mamdani rules or Takagi–Sugeno–Kang (TSK) rules with single point consequents.

The combination of iteration and rule merging produces models that have advantages associated with both proximity and clustering-based rule learning. Proximity learning algorithms are efficient in both the generation of rules and in the run-time execution of the resulting models. The determination of the rule consequents generally requires a single pass through the training data. When the model is run, the use of parametrically defined input domain partitions ensures that the selection of rules requires no search but rather is accomplished by a direct computation. The advantage of clustering is that there are no restrictions on the form of the fuzzy sets in the antecedents of the rules. The fuzzy sets are determined by the clustering algorithm and their shape is strictly driven by the data. In our algorithm, this effect is achieved by merging areas, which produces antecedents shaped by the data rather than by *a priori* partitions of the input domains.

Throughout this paper, we will consider learning rules to model a two-input system with input domains $U = [-1, 1]$ and $V = [-1, 1]$. The output will take values from the domain $W = [-1, 1]$. The impact of increased dimensionality is discussed in Section V-A.

II. Partitions, Rules, and Models

An initial step in the construction of a fuzzy model using a proximity-based learning strategy is the selection of the antecedents of the rules in the rule base. This is accomplished by partitioning the input domain into a set of local regions that define the supports of the antecedents of a rule. A fuzzy partition [26] of a two-dimensional input space $U \times V$ consists of a set of fuzzy sets $D_1, \ldots, D_t$ over $U \times V$ with

$$\sum_{i=1}^{t} \mu_{D_t}(x, y) = 1$$

for every $x \in U$ and $y \in V$.

A partition of a multi-dimensional domain $U \times V$ is *decomposable* if there are fuzzy partitions $\{A_1, \ldots, A_n\}$ of U and $\{B_1, \ldots, B_m\}$ of V whose Cartesian products form the partition of $U \times V$. That is, every $D_k = A_i \times B_j$ for some i and j. The partitions of fuzzy rule bases with multiple inputs frequently employ decomposable partitions. In this case, the antecedent of a rule has the form "if X is A_i and Y is B_j" where A_i and B_j are fuzzy sets from the partitions of U and V, respectively.

Because of their simplicity and ease of computation, many partitions of one-dimensional spaces consist of triangular fuzzy sets [27]. In a triangular partition, the fuzzy sets $A_1, \ldots, A_n$ are completely determined by the selection of a sequence of points $a_1 = -1, a_2, \ldots, a_n = 1$, where the point a_i is the center point of the triangular fuzzy set A_i. The support of A_i is the interval (a_{i-1}, a_{i+1}) and the membership function is

$$\mu_{A_i}(x) = \begin{cases} \frac{x - a_{i-1}}{a_i - a_{i-1}}, & \text{if } a_{i-1} \le x \le a_i \\ \frac{-x + a_{i+1}}{a_{i+1} - a_i}, & \text{if } a_i < x \le a_{i+1} \\ 0, & \text{otherwise.} \end{cases}$$

Fig. 1 shows a partition into triangular fuzzy sets defined by five points evenly spaced on $[-1, 1]$.

There are two major types of fuzzy rules, which differ in the form of the consequent. A Mamdani-style rule [28], [29] for a two-input system with decomposable partitions of the input space has the form

$$\text{if } X \text{ is } A_i \text{ and } Y \text{ is } B_j \text{ then } Z \text{ is } C_{i,j}$$

where A_i and B_j are fuzzy sets from the partitions of the input domains U and V, and $C_{i,j}$ is a fuzzy set over the output domain. A TSK [12] rule has the form

$$\text{if } X \text{ is } A_i \text{ and } Y \text{ is } B_j \text{ then } z = g_{i,j}(x, y)$$

where $g_{i,j}$ is a function of the input (x, y).

A rule base consists of a rule for each pair of fuzzy sets A_i and B_j. The rules and the inference techniques combine to produce a model $\hat{f}$, a function $\hat{f} : U \times V \to W$. Consider a model

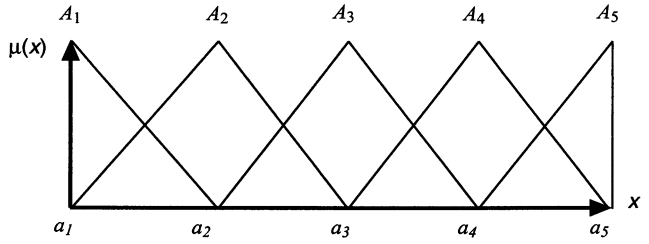


Fig. 1. Triangular decomposition.

defined by Mamdani-style rules in which the partitions consist of triangular fuzzy sets and weighted averaging is used for defuzzification and rule aggregation. The value for input (x, y) is determined by, at most, four rules. For input $x \in [a_i, a_{i+1}]$, and $y \in [b_j, b_{j+1}]$, the rules

"if X is A_i and Y is B_j then Z is $C_{i,j}$,"
'if X is A_i and Y is B_{j+1} then Z is $C_{i,j+1}$',
'if X is A_{i+1} and Y is B_j then Z is $C_{i+1,j}$',
"if X is A_{i+1} and Y is B_{j+1} then Z is $C_{i+1,j+1}$"

and weighted averaging combine to produce a function $\hat{f}_{i,j}$ over $[a_i, a_{i+1}] \times [b_j, b_{j+1}]$

$$\begin{aligned}\hat{f}_{i,j}(x,y) =& \frac{\sum_{s=i}^{i+1}\sum_{t=j}^{j+1}\mu_{A_s}(x)\mu_{Bs}(y)c_{s,t}}{\sum_{s=i}^{i+1}\sum_{t=j}^{j+1}\mu_{A_s}(x)\mu_{Bs}(y)} \\ =& \frac{(a_{i+1}-x)(b_{j+1}-y)c_{i,j}}{(a_{i+1}-a_i)(b_{j+1}-b_j)} \\ &+ \frac{(x-a_i)(b_{j+1}-y)c_{i+1,j}}{(a_{i+1}-a_i)(b_{j+1}-b_j)} \\ &+ \frac{(a_{i+1}-x)(y-b_{j+1})c_{i,i+1}}{(a_{i+1}-a_i)(b_{j+1}-b_j)} \\ &+ \frac{(x-a_i)(x-b_j)c_{i+1,j+1}}{(a_{i+1}-a_i)(b_{j+1}-b_j)} \end{aligned} \quad (1)$$

where $c_{i,j}$ is the center point of the triangular fuzzy set $C_{i,j}$. Fig. 2 shows the surface generated by the local function $f_{i,j}$. Equation (1) shows that, with triangular partitions and weighted averaging, the local approximating function $\hat{f}_{i,j}$ is completely determined by the values a_i, a_{i+1}, b_j, b_{j+1}, $c_{i,j}$, $c_{i+1,j}$, $c_{i,j+1}$, and $c_{i+1,j+1}$. The borders of the surface are simply obtained by linear interpolation between the values at the corners.

The model $\hat{f}$ is constructed from $(n-1)(m-1)$ local approximating functions $\hat{f}_{i,j}$, $1 \le i < n-1$, and $1 \le j < m-1$, where the domain of $\hat{f}_{i,j}$ is the rectangle $[a_i, a_{i+1}] \times [b_j, b_{j+1}]$. The fuzzy partitioning of the input domains provides a continuous transition between adjacent rules that guarantees that the function $\hat{f}$ obtained from the local functions $\hat{f}_{i,j}$ is continuous over $U \times V$.

A TSK rule with single point consequent has the form

$$\text{if } X \text{ is } A_i \text{ and } Y \text{ is } B_j \text{ then } z = c_{i,j}.$$

Using weighted averaging aggregation, the approximating function $\hat{f}$ defined by a TSK rule base is the same as that obtained using Mamdani rules where $c_{i,j}$ is the center point of the consequent fuzzy set $C_{i,j}$.

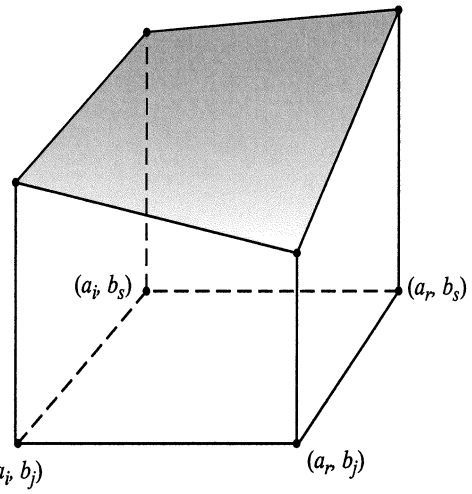


Fig. 2. Local surface over $[a_i, a_r] \times [b_j, b_s]$.

The partitions $A_1, \ldots, A_n$ of U and $B_1, \ldots, B_m$ of V divide the input domain into $(n-1)(m-1)$ rectangular regions determined by the center points of the triangular fuzzy sets. Each region has the form $[a_i, a_{i+1}] \times [b_j, b_{j+1}]$ for some $1 \le i < n$ and $1 \le j < m$. Using TSK rules with single point consequents and input $x \in [a_i, a_{i+1}]$ and $y \in [b_j, b_{j+1}]$, the value of the local approximating function $\hat{f}_{i,j}(x, y)$ is determined by values of the function at the four corners of this rectangle; that is, by the points $(a_i, b_j, c_{i,j})$, $(a_i, b_{j+1}, c_{i,j+1})$, $(a_{i+1}, b_j, c_{i+1,j})$, and $(a_{i+1}, b_{j+1}, c_{i+1,j+1})$.

Rather than having the combination of four rules define the surface, we will now define a form of TSK-style rule that permits a single rule to generate the surface $\hat{f}_{i,j}(x, y)$. Defining rule antecedents as Cartesian products of fuzzy sets from independent partitions of the input domains restricts the regions that can be the support of a rule. The projection of any two antecedents onto an input domain must be either be identical or disjoint. To add flexibility to the modeling process, we will construct models from rules with multi-dimensional antecedents. A rule have will the form

$$\begin{aligned}&\text{if } X \times Y \text{ is } D = [(a_i, b_j), (a_r, b_s)] \\ &\quad \text{then } g(x,y) = [c_{i,j}, c_{i,s}, c_{r,j}, c_{r,s}]\end{aligned}$$

where $[(a_i, b_j), (a_r, b_s)]$ denotes the rectangle with corners (a_i, b_j), (a_r, b_j), (a_i, b_s), and (a_r, b_s). The consequent of the rule specifies the values at the corners of the rectangular region; $(a_i, b_j, c_{i,j})$, $(a_i, b_s, c_{i,s})$, $(a_r, b_j, c_{r,j})$, and $(a_r, b_s, c_{r,s})$. This is a TSK-style rule where the function $g(x, y)$ is obtained from the corner points using (1) by substituting a_r for a_{i+1}, b_s for b_{j+1}, $c_{i,s}$ for $c_{i,j+1}$, $c_{r,j}$ for $c_{i+1,j}$, and $c_{r,s}$ for $c_{i+1,j+1}$. That is, (1) has been generalized to produce a local function over an arbitrary rectangular region of the input space. Thus, rule bases obtained from rules of this form will have the ability to generate all models that could be produced using Mamdani-style rules or TSK rules with single point consequents whose antecedents are obtained from triangular partitions of the input domains. However, employing the type of multi-dimensional antecedents described above allows us to produce additional models with nondecomposable partitions of the input domain that require fewer rules to satisfy a precision requirement.

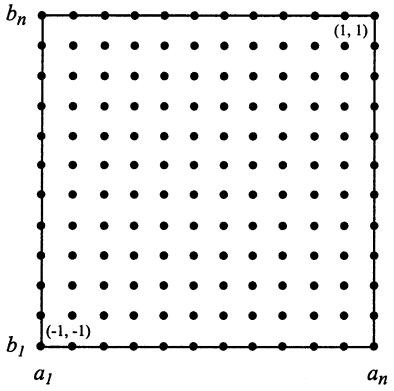


Fig. 3. Domain decomposition and grid points.

III. Rule Construction

The generation of a model begins by selecting points $\{a_1, \ldots, a_n\} \in U$ and $\{b_1, \ldots, b_m\} \in V$ that define the partitions of the input domains. The ordered pairs (a_i, b_j) form a grid over the space $U \times V$. Fig. 3 shows the domain with $n = m = 11$, which will produce a rule base with 100 rules. Throughout this paper we will use partitions with evenly spaced grid points to facilitate the rule reduction strategy and to enhance the run-time efficiency of the model. These features will be discussed in Section V.

For a rule with antecedent "if $X \times Y$ is $D = [(a_i, b_j), (a_r, b_s)]$," the learning process consists of selecting the constants $c_{i,j}$, $c_{i,s}$, $c_{r,j}$, and $c_{r,s}$ in the consequent. A set $T = \{(x_t, y_t, z_t) \mid t = 1, \ldots, N\}$ of training examples, where $x_i \in U$ and $y_i \in V$ are input values and $z_i \in W$ is the associated response, provides the information needed for the generation of the rules.

The value $c_{i,j}$ associated with grid point (a_i, b_j) is determined by the weighted average of all training points whose projection onto the input space lies within a distance q of (a_i, b_j). The initial distance q is determined from the spacing of the grid points. Let $gs = \min\{a_{i+1} - a_i, b_{i+1} - b_i\}$ be the minimum distance between grid points. Initially, q is set to three-fourths of the grid size: $q = .75(gs)$. Let (x_k, y_k, z_k) be a training point and $d_{i,j,k}$ be the distance from (x_k, y_k) to grid point (a_i, b_j). The set $T_{i,j,q}$ is the subset of the training set T consisting of all training points (x_k, y_k, z_k) with the distance $d_{i,j,k}$ less than or equal to q. The value associated with grid point (a_i, b_j) obtained from $T_{i,j,q}$ using weighted averaging is

$$c_{i,j} = \frac{\sum \left(1 - \left(\frac{d_{i,j,k}}{q}\right)\right) z_k}{\sum \left(1 - \left(\frac{d_{i,j,k}}{q}\right)\right)}$$

where the summation is taken over all training points in $T_{i,j,q}$.

Since training data may be sparsely distributed throughout the input space, it is possible that no training point lies within distance q from a grid point (a_i, b_j). When this occurs, the radius q defining the set $T_{i,j,q}$ is expanded incrementally until training information is found. The expansion of the radius is accomplished by adding $.25(gs)$ to q after each unsuccessful search for training data.

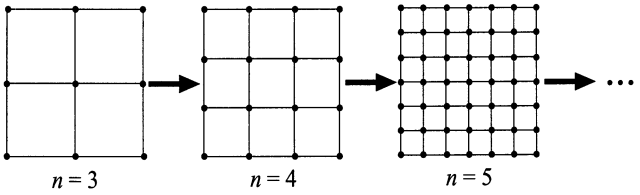


Fig. 4. Grid point refinement.

When all grid points have been assigned a value by the preceding procedure, the rule base is completely specified. Each rule has the form

$$\text{if } X \times Y \text{ is } D = [(a_i, b_j), (a_{i+1}, b_{j+1})]$$
$$\text{then } g(x, y) = [c_{i,j}, c_{i,j+1}, c_{i+1,j}, c_{i+1,j+1}]$$

where $c_{i,j}$, $c_{i,j+1}$, $c_{i+1,j}$, and $c_{i+1,j+1}$ are the values computed for grid points (a_i, b_j), (a_i, b_{j+1}), (a_{i+1}, b_j), and (a_{i+1}, b_{j+1}). The rule base defines the model $\hat{f} : U \times V \to W$ consisting of the local surfaces associated with each rule. The boundary conditions of the local surfaces ensure that $\hat{f}$ is continuous over $U \times V$.

IV. Rule Base Generation by Refinement

The construction of a rule base consists of two steps: the generation of an original rule base followed by rule reduction. Adjacent rules produced by the learning algorithm must have supports that facilitate the merging process that follows. To accomplish this, a proximity learning algorithm is incorporated into a generate-and-test domain partition refinement strategy to construct the rule base.

The objective of the refinement of the partitions is to produce a rule base that approximates the training data within a prescribed maximum error β. The generation begins by partitioning the input domains into a small number of fuzzy sets and constructing a rule base. If the resulting rule base satisfies the precision bound, the process is complete. Otherwise a new partition is selected and generate-and-test procedure is repeated.

Throughout this presentation we will assume that partitions of input spaces have the same number of fuzzy sets. Thus the grid formed by the partitions will by an $n \times n$ grid, as shown in Fig. 3. The initial partition sets n to 2 with grid points $a_1 = -1, a_2 = 1$ and $b_1 = -1, b_2 = 1$. With this partition, the only rule is

$$\text{if } X \times Y \text{ is } D = [(a_1, b_1), (a_2, b_2)]$$
$$\text{then } g_{1,1}(x, y) = [c_{1,1}, c_{1,2}, c_{2,1}, c_{2,2}].$$

The values for the $c_{i,j}$s are obtained as described in the previous section and the training data are then used to determine the error of the model. The error at training point (x_t, y_t, z_t) is $|\hat{f}(x_t, y_t) - z_t|$. The rule base generation is completed if the maximum error of the training data lies within the specified precision bound. If not, the algorithm expands the number of regions in the grid by decomposing the input domains with $(n+1)$ points, as shown in Fig. 4. This procedure will continue incrementing n until a grid of size $n \times n$ is found that satisfies the precision bound.

The procedure for generating the initial rule base can be summarized as follows.

1) Input: training set T and error bound β.
2) Set $n = 2$.
3) Generate rule base with $(n-1)^2$ rules defined by the $n \times n$ grid.
4) Calculate maximum error er over the set of training points.
5) If $er < \beta$ accept the rule base, else set $n = n + 1$ and go to 2).

The applicability of the preceding strategy for generating the original rule base is limited by the number of input dimensions. With k input dimensions, the ith iteration of the preceding loop would be required to produce with $(i-1)^k$ rules. For automatic control, function approximation and many system modeling problems with a small number of variables, the refinement process provides a computationally tractable strategy for producing a rule base that satisfies a predetermined precision bound. For classification problems with high dimensionality, the growth of the exponent k would make this approach to the construction of the rule base computationally unfeasible.

The greedy rule reduction algorithm presented in the next section, however, does not require a rule base produced by partition refinement. The original rule base may be generated by any learning algorithm that produces rules whose supports are hyper-rectangular regions over the grid formed by input domain decompositions.

V. Rule Reduction by Merging

The objective of the merging strategy is to reduce the number of rules while preserving the satisfaction of the precision bound. Ideally, the goal would be to identify the smallest set of rules satisfying the precision bound. However, the algorithm that we present is greedy and does not guarantee the generation of a minimal size rule base. Rule reduction based on merging fuzzy sets to extend the scope of fuzzy rules has been considered in [6], [8], [30], and [31]. The technique presented here differs from previous methods due to use of multi-dimensional fuzzy sets in the antecedents of the rules and the complications associated with ensuring the continuity of the model.

Regions in the input domain in which the system has high variation may require a fine partition to reach the desired precision. Consequently, the uniform partitioning of the input space that was employed in the generation the original model may require many rules in regions of little variation where a small number of rules would suffice. The merging algorithm combines adjacent rules that specify similar responses. Enlarging the region of applicability of a rule may produce two beneficial results. First, increasing the size of the support produces a rule with a greater degree of generalization of the data and guards against the classic overfitting problem. The second benefit is that rules with large granularity are more likely to admit a linguistic interpretation.

Rules from the rule base with uniform partitions of the input domains generated by the refinement process are combined to produce rules with rectangular regions of applicability of the form

$$\begin{aligned} &\text{if } X \times Y \text{ is } D = [(a_i, b_j), (a_r, b_s)] \\ &\quad \text{then } g(x, y) = [c_{i,j}, c_{i,s}, c_{r,j}, c_{r,s}] \end{aligned}$$

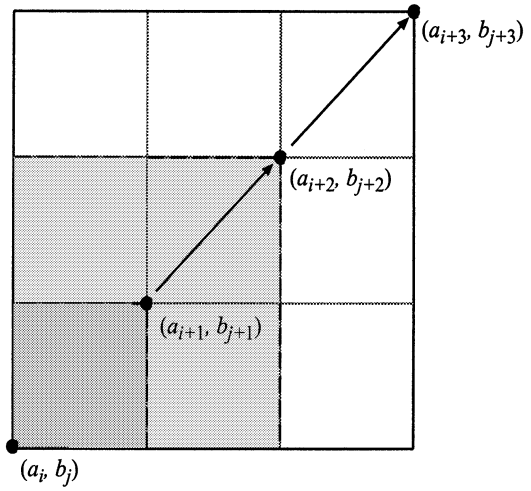


Fig. 5. Merging regions as a square.

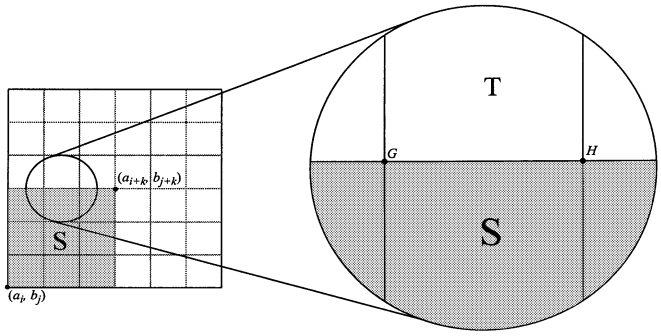


Fig. 6. Continuity considerations.

that preserve the precision bound β. Merging begins with a region $[(a_i, b_j), (a_{i+1}, b_{j+1})]$ (see Fig. 5) and attempts to enlarge the rule. Starting at the lower left corner of the region, the merging strategy combines regions by expanding diagonally to produce rules with support $[(a_i, b_j), (a_{i+2}, b_{j+2})]$, $[(a_i, b_j), (a_{i+3}, b_{j+3})], \ldots$.

The first step in accepting a new rule is to verify that the training data within the support of the expanded rule still satisfy the precision bound. The second requirement for accepting a potential expansion is that the surface generated by the new rule must be continuous with those in adjacent regions. Once a rule with enlarged support is proposed, the surfaces associated with adjacent rules may no longer be continuous. Fig. 6 shows the relationship in the input domain between an expanded rule and the adjacent rules. Before expansion, the surface is continuous along the line segment $\overline{GH}$ since the values at points G and H are the same for the adjacent rules. When $\mathbf{S}$ is expanded, the values specified by the expanded rule at points G and H are no longer the values determined from the local training data but rather are obtained from the values at the corner points (a_i, b_{j+k}) and (a_{i+k}, b_{j+k}). To maintain continuity, the grid point values for the region $\mathbf{T}$ are modified to match the values generated by the rule over $\mathbf{S}$. Since this alters the function associated with the region $\mathbf{T}$, the modified rule over $\mathbf{T}$ must also be checked for compliance with the precision bound. If these conditions are met, the enlarged rule is accepted.

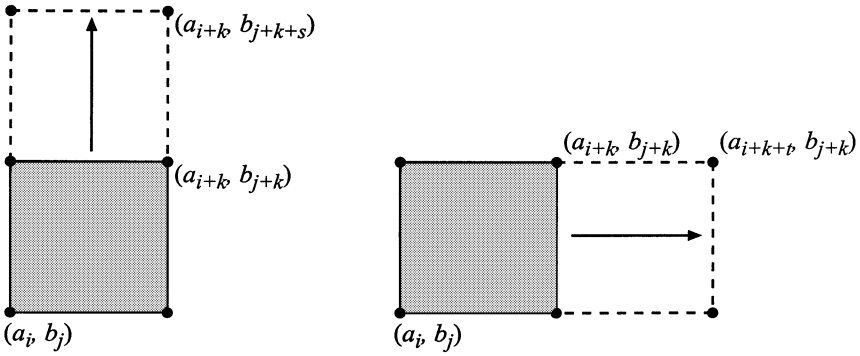


Fig. 7. Merging as rectangles.

After the rule has been expanded as far as possible diagonally, expansions are attempted along the U and V axes. The region $[(a_i, b_j), (a_{i+k}, b_{j+k})]$ resulting from the diagonal expansion is then expanded in the V direction until halted by the precision bound. This expansion produces a rectangle $[(a_i, b_j), (a_{i+k}, b_{j+k+s})]$ (see Fig. 7). In like manner, the region $[(a_i, b_i), (a_{i+k}, b_{j+k})]$ is expanded in the U direction producing $[(a_i, b_i), (a_{i+k+t}, b_{j+k})]$. The larger rectangle that preserves the precision bound is selected as the antecedent of the new rule.

The merging process is greedy; it begins with rectangle $[(a_1, b_1), (a_2, b_2)]$ and merges regions until the precision bound stops the process. Once an expanded rule is accepted, the values at the grid points on the boundary of the rule are fixed and cannot be changed by subsequent expansions. The merging process continues left-to-right, bottom-to-top until the entire input domain has been considered.

A. Features of the Rule Base

The rules constructed by the refine-and-merge algorithm have several desirable features. Along with the reduction in the number of rules required to produce a model with the desired precision, merging produces rule antecedents that are formed by the data rather than obtained from an *a priori* selection of a partition of the input space. The supports of the antecedents, however, are not allowed to be arbitrary regions in the input space but rather are restricted to rectangular regions. This restriction permits an efficient implementation of the model.

Although the merging process constructs rules with multi-dimensional antecedents, the run-time efficiency of a model built from decomposable partitions is maintained using indirect hashing. When the rule base is generated using an $n \times m$ grid, an $(n-1) \times (m-1)$ matrix R is constructed whose entries are pointers to the rule associated with each region. Initially, each entry in the array points to the rule generated by the refinement process for the region. When regions are merged and a new rule is created, the entries for each of the regions that make up the support of the new rule are set to point to the merged rule.

With this formulation, the identification of the appropriate rule during the execution is a direct computation. Consider input (x, y) with n and m evenly spaced points defining the partitions of U and V respectively. As previously noted, the computation of the value $\hat{f}(x, y)$ utilizes only one rule and no aggregation is required. The antecedent of the rule has the form

$$\text{if } X \times Y \text{ is } D = [(a_i, b_j), (a_r, b_s)]$$

where $x \in [a_i, a_r]$ and $y \in [b_j, b_s]$. The indexes i and j in the rule array can be obtained directly from the input: $i = \lfloor (n-1)(x+1)/2 \rfloor + 1$ and $j = \lfloor (m-1)(y+1)/2 \rfloor + 1$. The value in the $R[i, j]$ indicates the rule needed to obtain $\hat{f}(x, y)$.

An objective of the reduction of the number of rules in a model is to increase the interpretability of the rules. Linguistic interpretation has generally considered Mamdani-style rules since fuzzy sets, with their associated linguistic terms, occur in both the antecedent and consequent of such a rule. For example, a common form of a rule for a fuzzy PD controller is

$$\text{if } Error \text{ is } Positive\ small \text{ and } Change\ of\ error \text{ is } Negative\ small \text{ then } Response \text{ is } Zero$$

where $Positive\ small$, $Negative\ small$, and $Zero$ are fuzzy sets over the error, change of error, and response domains, respectively. This rule has a straightforward interpretation: if there is little error and the change is reducing it in the correct direction, do not alter the system.

TSK rules do not admit such a linguistic analysis since the consequent is a function of the input values and not a potentially interpretable fuzzy set. The result of expanding the support of a rule by merging regions for TSK rules does, however, have useful interpretation. A rule

$$\text{if } X \text{ is } D \text{ then } z = g_{i,j}(x, y)$$

indicates that the relationship between the input variables is the same throughout the support of D. The TSK rule simply gives a mathematical formulation of the relationship rather than a linguistic one and the antecedent defines the extent of the relationship.

This observation is particularly pertinent for the models built by the refine-and-merge algorithm. The result of such a model is obtained from a single rule and not from the aggregation of multiple rules. Thus, a rule with a large region of applicability indicates an intuitive stability of the model throughout that region. The fundamental relationships between the variables are unchanged within the region and change only when traversing the borders of the region.

VI. Experimental Results

Experiments have been conducted to determine the ability of the refine-and-merge strategy to produce small rule bases that satisfy a prescribed precision bound β. An experiment begins with selection of a *target function* f, which represents the underlying system. A training set $T = \{(x_t, y_t, f(x_t, y_t)) \mid t = 1, \ldots, N\}$ is generated by randomly selecting N elements from the input space. After the selection of the target and the generation of the training set, the refinement process is used to produce a rule base that satisfies the precision bound on the training set. The merging strategy is then applied to reduce the number of rules.

The function $\hat{f}(x, y)$ defined by rule base is then compared with the target function to determine the precision of the resulting model. The test set consists of 784 points uniformly distributed over the input domain $U \times V$. The experimental results reported in this section consist of the average number of rules generated, the average of the average error and the average maximum error over 20 iterations of the algorithm for each set of parameters.

The algorithm was tested with a number of target functions to evaluate the ability to construct models with systems of different levels of variability. The robustness of the algorithm was examined by varying the precision bounds and the size of the training set.

A. FLM and Refinement

The baseline for comparison is the proximity rule learning algorithm introduced by Wang and Mendel [21], [32], which we will refer to as the FLM algorithm. This technique was originally presented for the construction of Mandami-style rules but can easily be adapted to produce TSK rules. The process begins by decomposing the input domains into triangular fuzzy partitions $\{A_1, \ldots, A_n\}$ and $\{B_1, \ldots, B_n\}$.

FLM: A training example (x_k, y_k, z_k) that has the maximal membership in the fuzzy set $A_i \times B_j$ is selected from the training set T. If more than one example assumes the maximal membership in $A_i \times B_j$, one is selected arbitrarily. The rule "if X is A_i and Y is B_j then $z = z_k$" is added to the rule base.

The consequent of the rule, which we previously have denoted as $c_{i,j}$, is simply the output value of the training set element that has maximal membership in the antecedent. The intuition behind the approach is that the training instance nearest to satisfying the antecedent represents the best prototype of the rule. Variations to the FLM algorithm that use all training examples that fall within the support of the antecedent to determine the consequent have been examined in [23].

The FLM algorithm presented above requires at least one training point to have nonzero membership in the antecedent of the rule to produce a value for the consequent. Techniques for completing a rule base were introduced to extend this algorithm to situations with distributed or sparse training data [23], [33]. Rule base completion uses existing rules and interpolation to produce rules in regions with no training data. When we refer to the FLM algorithm, we will mean the rule selection process described above augmented with region growing completion.

To adapt the FLM algorithm to the production of a rule base that satisfies a precision bound, the iterative refinement of partitions of the input domains will be employed. In the experiments, the partition of each dimension will have the same number of fuzzy sets. Table I shows the results of iterative applications of the FLM algorithm and rule learning strategy from Section IV for target function $f(x, y) = x^2 - y^2$. For simplicity, we will label the latter *refine* as the first step in the refine-and-merge process.

We first observe several general characteristics exhibited by both of the learning strategies. As expected, the average and maximum error over the test set decreases with the number of training points. Note that with a precision bound of .2, the average error is well below the bound regardless of the size training set used. As the precision bound becomes more demanding, additional training data are required to reduce the average error below the bound. Even 10 000 data points, however, are not sufficient for the FLM algorithm to produce a rule base with maximal error below .05. The iterative refinement built FLM rule bases with up to 22 500 rules in the attempt to reach this level of precision, but was unsuccessful.

As observed in previous experiments with the FLM algorithm, the number of rules required initially increases with the acquisition of training data. When sufficient training data is available, the number of rules declines until it reaches an essentially constant number. Initially, the acquisition of training data permits the opportunity of finding errors in more regions in the input and thereby requiring additional rules. Eventually *training data saturation* occurs. That is, there are few or no regions without training data and in each region there is sufficient information to approximate each of the rules to a high degree of precision. The accurate generation of rules reduces the number needed. The point at which saturation occurs is dependent upon the precision bound. This phenomena is exhibited by both of the learning strategies. In the *refine* column, the number of rules decreases between 100 and 500 training points for precision bound .2, between 500 and 1000 training points for precision bound .1, and between 1000 and 10 000 training points for precision bound .05.

Comparing the results of FLM and *refine*, we see that *refine* produces rule bases with between 50% and 95% fewer rules than the FLM algorithm. In each case, the maximum error was also less than that in the corresponding FLM experiment. Similar results were exhibited by experiments with a variety of target functions.

B. Rule Reduction

The rule base constructed by the iterative refinement is subjected to the merging algorithm to reduce the number of rules. Table II shows the results obtained of merging the rule bases constructed in Table I. The lower the precision bound, the more effective the merging is in reducing the number of rules. For precision .2, the number of rules is reduced by 40 to 50 percent for training sets of size 50, 100 and 500. This increases to a reduction of approximately 200 to 300% of the rules for rule bases generated with precision bound .1 and 200 to 600% with .05 precision bound.

TABLE I
FLM VERSUS REFINEMENT: TARGET $f(x, y) = x^2 - y^2$

		FLM			Refine		
Prec Bound	Training Examples	Rules	Ave Error	Max Error	Rules	Ave Error	Max Error
0.2	50	87.8	.118	.704	35.5	.100	.577
	100	109.8	.077	.548	38.2	.076	.440
	500	167.2	.036	.280	23.9	.039	.198
	1000	841.5	.020	.198	13.2	.039	.171
	10000	126.0	.015	.196	9.0	.042	.167
0.1	50	253.9	.110	.607	131.2	.105	.656
	100	344.6	.074	.549	163.5	.069	.520
	500	653.4	.029	.304	193.5	.027	.249
	1000	841.5	.020	.198	140.4	.020	.147
	10000	625.4	.009	.102	25.0	.018	.080
0.05	50	750.0	.124	.762	289.0	.109	.721
	100	1188.3	.082	.568	409.4	.073	.516
	500	1600.0	.032	.290	784.8	.029	.254
	1000	2372.2	.021	.226	873.9	.019	.183
	10000	-	-	-	495.0	.006	.052

TABLE II
RESULT OF MERGING: TARGET $f(x, y) = x^2 - y^2$

Prec Bound	Training Examples	Rules	Ave Error	Max Error	Percent Reduction
0.2	50	14.9	.111	.485	138%
	100	14.8	.084	.385	158%
	500	11.7	.058	.217	104%
	1000	12.4	.046	.188	6%
	10000	9.0	.043	.171	0 %
0.1	50	43.7	.093	.563	200%
	100	53.7	.064	.395	203%
	500	42.0	.034	.147	311%
	1000	33.2	.032	.122	323%
	10000	25.0	.018	.081	0%
0.05	50	89.0	.088	.515	224%
	100	132.8	.062	.391	296%
	500	199.3	.021	.229	293%
	1000	175.5	.027	.149	397%
	10000	65.6	.015	0.053	654%

As the saturation of training data occurs, the impact of reduction lessens. In this case, the original *refine* component is producing high quality models with fewer rules than those produced with fewer training data. Since these rules have larger scopes, it becomes more difficult to successfully merge regions.

Table III gives a comparison of the models generated by refinement alone and by refinement-and-merging over a series of target functions. In each case the training set consisted of 1000 points. The target f_1 is a planar surface and the surface generated by a single rule can approximate f_1 to within the desired precision. The function f_3 is a cylindrification of the sine curve to two dimensions, creating a surface that resembles a wave. The merging algorithm reduces the number rules between 300% and 800% from that produced by iterative refinement alone. The target f_4 defines a more complicated "ripple-like" surface. Minimal reduction occurs with .2 precision, indicating that approximately 25 rules are required to achieve that degree of precision. As the precision bound decreases, the effect of the reduction becomes substantial.

In general, one may expect an increase in the error with a reduction of rules. Enlarging the region of applicability provides a greater generalization from the training data. This has occurred in the experimentation, but the increases in both average and maximum errors have been marginal and frequently the reduced model has less test set error than the original.

C. Observations

The surface given in Fig. 8 illustrates several properties and shortcomings of the refine-and-merge algorithm and indicates directions of future improvements. The surface was produced by a rule base generated using target function $f(x, y) = x^2 - y^2$, 1000 training points and precision bound .2. Refinement required 25 rules to reach the desired precision. The squares along the bottom of the figure show the size of the local regions in the original rule base. The shading indicates the 14 rules resulting from the mergers.

The large region in the upper corner is the antecedent of a single rule. In the terminology used to describe rectangles, the antecedent of this rule is "if $X \times Y$ is $D = [(a_4, b_1), (a_6, b_3)]$." As discussed in Section V-A, merging TSK rules indicates that similar relationships hold between the input variables in the expanded region. A standard fuzzy rule interpretation of the rule with antecedent $A = [(a_1, b_1), (a_3, b_6)]$ might be stated: "if X is *Large* and Y is *Small* then $z = g(x, y)$" where g is the function from the consequent of the rule.

Fig. 8 also exhibits artifacts of the greedy approach to rule merging. The function $f(x, y) = x^2 - y^2$ is symmetric and one might expect, or at least hope for, an equally large rule in the other corners. However, the greedy strategy dictates that once an expanded rule has been accepted, it will not be changed in later analyses. Once the rule with antecedent $D = [(a_2, b_3), (a_3, b_5)]$ is accepted, the values for grid points (a_2, b_5) and (a_2, b_6) cannot be altered. When the square $[(a_1, b_4), (a_2, b_5)]$ is considered for expansion, the region to the right has already been assigned to a rule and will not be reallocated. Even potential expansion upward is limited since the value at point (a_2, b_5) is fixed. This example shows that, as more rules are accepted, the flexibility to merge regions in the remainder of the input space is restricted.

There are two techniques for potentially mitigating this limitation. The first is to allow backtracking in the merging process; accepting an expanded rule does not preclude it from being changed

TABLE III
REFINE VERSUS MERGE: 1000 TRAINING POINTS

		Refine			Merge		
Target	Prec Bound	Rules	Ave Error	Max Error	Rules	Ave Error	Max Error
$f_1(x,y) = (x + y/2)/2$	0.2	1.0	.023	.069	1.0	.022	.062
	0.1	1.0	.023	.065	1.0	.022	.065
	0.05	4.5	.013	.036	1.0	.013	.036
$f_2(x,y) = x^2 - y^2$	0.2	13.2	.039	.171	12.4	.046	.188
	0.1	140.4	.020	.147	33.2	.032	.122
	0.05	873.3	.019	.183	175.5	.019	.130
$f_3(x,y) = .5(\sin(2\pi x))$	0.2	68.3	.069	.225	7.5	.072	.196
	0.1	306.4	.032	.233	52.1	.036	.147
	0.05	1491.8	.025	.217	339.2	.019	.205
$f_4(x,y) = (\sin(\pi x) + \sin(\pi y))/2$	0.2	26.6	.054	.178	24.0	.058	.198
	0.1	98.8	.027	.150	47.6	.037	.153
	0.05	703.7	.018	.188	182.4	.021	.163
$f_5(x,y) = (\sin 5x + \sin 5y)/(25xy)$	0.2	34.7	.068	.338	15.9	.085	.310
	0.1	103.6	.062	.396	34.7	.070	.364
	0.05	225.9	.066	.437	71.9	.067	.431

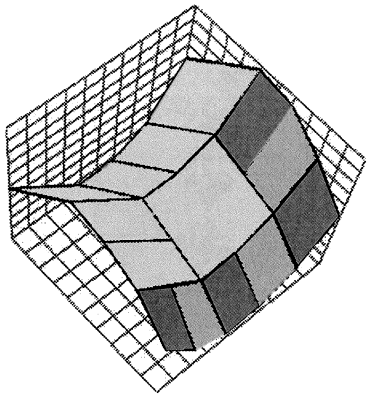


Fig. 8. Merging with target function $f(x,y) = x^2 - y^2$.

when expansions of abutting regions are considered. The drawback with this approach is the increased computational resources required in constructing the reduced rule base. Another approach would be to perform the merging routine a number of times each beginning at a different location (e.g., at each corner) following different patterns. The final model would be obtained from these various rule bases by selecting rules from each that cover the space and whose boundary points can be made consistent to ensure the continuity of the final model.

Building several rule bases from the same set of training data has been employed by Ischibuchi *et al.* [34] for constructing fuzzy classification systems. In their work, the objective was to use multiple rule bases to mitigate the effect of the arbitrary selection of the size of the partitions of the input domains. In the strategy suggested above, the goal would be to retain the efficiency of the greedy strategy but to limit the impact of the arbitrary selection of the initial location and pattern employed in the merging procedure.

VII. IMPRECISE TRAINING INFORMATION

The experiments in the previous section demonstrated the feasibility of making a significant reduction in the number of rules using a greedy merging algorithm. In those experiments, the training set consisted of precise data. However, the training data available for rule generation is frequently imprecise. Imprecise training data requires a modification to the criterion used to halt the refinement in the construction of the original rule base and the merging in the rule reduction process.

For precise data, a partition of the input domains is rejected if a training instance (x,y,z) is encountered for which $|f(x,y) - z|$ exceeds the precision bound β. With imprecise data, this condition may be triggered not by a change in the underlying system but rather by noise in a single training point. To compensate for noisy data, the decision to accept the current rule base or to further refine the partition must not be dependent on the value of a single training point. This can be accomplished by requiring a certain percentage of the training points in the support of a rule to exceed the error bound. An alternative is to require the average error of all training points within the region to exceed β. The latter strategy was incorporated into the iterative refinement rule generation.

In a like manner, the error of a single training point is sufficient to halt the merging process. Adopting the error averaging strategy defined above, merging will be halted when the average error of all training points within the region considered for expansion exceeds β. With these modifications, a series of experiments was conducted with imprecise training data.

In these experiments, the training set consists of points of the form $(x, y, f(x,y) + e(x,y))$, where f is the target function and e is an error function. The error is randomly generated from a

TABLE IV
IMPRECISE TRAINING DATA: TARGET $f(x, y) = x^2 - y^2$

			Refine			Merge		
Error Std. Dev.	Prec Bound	Training Points	Rules	Ave Error	Max Error	Rules	Ave Error	Max Error
0.1	0.1	100	33.0	.127	.609	14.5	.123	.547
		500	17.5	.100	.430	12.5	.102	.427
		1000	12.8	.095	.418	11.0	.098	.403
0.05	0.1	100	11.6	.106	.488	8.4	.111	.492
		500	7.2	.083	.337	7.1	.084	.385
		1000	6.2	.085	.348	5.9	.088	.368
0.05	0.05	100	84.1	.086	.494	42.4	.082	.454
		500	75.4	.056	.276	38.4	.056	.271
		1000	42.3	.052	.257	29.5	.052	.235

TABLE V
IMPRECISE TRAINING DATA: 1000 TRAINING POINTS

			Refine			Merge		
Target	Error Std. Dev.	Prec Bound	Rules	Ave Error	Max Error	Rules	Ave Error	Max Error
f_2	0.1	0.1	12.8	.095	.418	11.0	.098	.403
	0.05	0.1	6.2	.085	.334	5.9	.088	.368
	0.05	0.05	42.3	.052	.257	29.5	.052	.235
f_4	0.1	0.1	33.9	.100	.393	25.5	.104	.433
	0.05	0.1	16.9	.091	.316	9.1	.091	.316
	0.05	0.05	78.3	.052	.236	50.7	.054	.233
f_5	0.1	0.1	30.5	.097	.404	19.0	.102	.426
	0.05	0.1	11.8	.091	.329	5.7	.099	.349
	0.05	0.05	58.5	.051	.226	35.8	.052	.224

Gaussian distribution with mean 0. In the experimental results in Tables IV and V, training data with error standard deviations of .1 and .05 are used for generating rule bases with precision bound .1 and standard deviation .05 for generating rule bases with precision bound .05.

Table IV gives the result of the rule generation with target function $f(x, y) = x^2 - y^2$. The basic relationships between the precision bound, the number of training points, the number of rules, and the error identified in the precise training data case are also exhibited with imprecise data. As might be expected, increasing the error in the training data increases both the number of rules and the error in the resulting model.

It is interesting to compare Table IV with with the results for the algorithm with precise training data given in Tables II and III. Requiring that the average value of the error exceed the precision bound greatly reduces the number of rules. For example, with 1000 training points and precision bound .1, the precise algorithm produced rule bases with an average of 33.2 rules while the imprecise algorithm with error standard deviation .05 produced rule bases with an average of 5.9 rules. The tradeoff was, of course, in accuracy. With the algorithm for precise data, the maximum error was .122 while it was three times that for the imprecise data. The data from experiments with the other target functions in Table V exhibited similar properties.

Table V gives the results of refine-and-merge rule generation with target functions $f_2(x, y) = x^2 - y^2$, $f_3(x, y) = .5(\sin(2\pi x))$, $f_4(x, y) = (\sin(\pi x) + \sin(\pi y))/2$, and $f_5(x, y) = (\sin 5x + \sin 5y)/(25xy)$. When the precision bound β is high and the training data error is low, $\beta = .1$ and error standard deviation .05, the use of averaging in the refinement process produces rule bases with a small number of rules. In this case, merging rules does not make a significant improvement to the rule base.

With a more stringent precision bound, the size of the original rule bases increases and merging plays a more important role. For the cases with $\beta = .5$ and standard deviation .5 in Table III, the merging process reduced the number of rules by 30%, 35% and 38% with little change to the average and maximum error of the resulting rule bases.

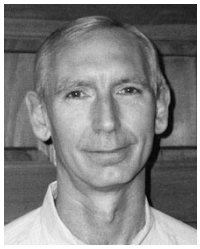
VIII. CONCLUSIONS

A strategy for reducing the number of rules needed to produce a model that satisfies a prescribed precision bound has been presented. The algorithm employs an iterative refinement of the partitions of the input domains to produce a set of decomposable rules that satisfy the bound. When possible, adjacent rules are merged using a greedy strategy to increase the granularity of the rule base. Merging produces rules with multi-dimensional fuzzy sets in the antecedents, which enhances the representational capability of the rules. The merging step has been shown to significantly reduce the number of rules in models built from a variety of target functions and precision bounds. Future research will explore the potential of obtaining further improvements in the ability to reduce the number of rules by modifying the merging algorithm to allow a degree of backtracking (a nongreedy approach) or by combining the results of several iterations of the greedy algorithm.

REFERENCES

[1] J. Yen and L. Wang, "An SVD-based fuzzy model reduction strategy," in *Proc. 5th IEEE Int. Conf. Fuzzy Systems*, New Orleans, LA, Sept. 1996, pp. 835–841.
[2] ——, "Simplifying fuzzy rule-based models using orthogonal transformation methods," *IEEE Trans. Fuzzy Syst.*, vol. 6, no. 4, pp. 13–24, 1999.
[3] Y. Yam, P. Baranyi, and C.-T. Yang, "Reduction of fuzzy rule base via singular value decomposition," *IEEE Trans. Fuzzy Syst.*, vol. 7, no. 2, pp. 120–132, 1999.
[4] D. Nauck and R. Kruse, "A neuro-fuzzy method to learn fuzzy classification rules from data," *Fuzzy Sets Syst.*, vol. 89, pp. 277–288, 1997.
[5] ——, "NEFCLASS-X-a soft somputing tool to build readable fuzzy classifiers," *BT Technol. J.*, vol. 16, no. 3, pp. 180–190, 1998.
[6] M. Setnes, R. Babuska, U. Kaymak, and H. R. van Nauta Lemke, "Similarity measures in fuzzy rule base simplification," *IEEE Trans. Syst., Man, Cybern. B*, vol. 28, no. 3, pp. 376–386, 1998.
[7] M. R. Berthold and K.-P. Huber, "Constructiong fuzzy graphs from examples," *Intell. Data Anal.*, vol. 3, pp. 37–53, 1999.
[8] T. A. Sudkamp, A. Knapp, and J. Knapp, "A greedy approach to rule reduction in fuzzy models," in *Proc. 2000 IEEE Conf. Syst., Man, Cybern.*, Nashville, TN, Oct. 2000, pp. 3716–3622.
[9] T. A. Sudkamp, J. Knapp, and A. Knapp, "Refine and merge: Generating small rule bases from training data," in *Proc. Joint 9th IFSA World Congress and 20th NAFIPS Int. Conf.*, Vancouver, BC, Canada, July 2001, pp. 197–201.
[10] W. Pedrycz, *Granular Computing*, W. Pedrycz, Ed. Heidelberg, Germany: Physica-Verlag, 2001.

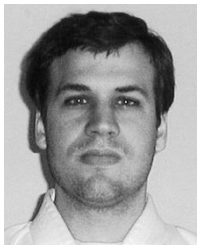
[11] L. A. Zadeh, "Outline of a new approach to the analysis of complex systems and decision processes," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-3, pp. 28–44, 1973.
[12] T. Takagi and M. Sugeno, "Fuzzy identification of systems and its applications to modeling and control," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-15, pp. 329–346, 1985.
[13] R. R. Yager and D. P. Filev, *Essentials of Fuzzy Modeling and Control*. New York: Wiley, 1994.
[14] J. R. Jang, "ANFIS: Adaptive-network-based fuzzy inference system," *IEEE Trans. Syst., Man, Cybern.*, vol. 23, pp. 665–684, 1993.
[15] H. Ishibuchi, "Development of fuzzy neural networks," in *Fuzzy Modeling: Paradigms and Practices*. Norwell, MA: Kluwer, 1996, pp. 185–202.
[16] C. L. Karr and E. J. Gentry, "Fuzzy control of pH using genetic algorithms," *IEEE Trans. Fuzzy Syst.*, vol. 1, pp. 46–53, 1993.
[17] W. Pedrycz, *Fuzzy Evolutionary Computation*, W. Pedrycz, Ed. Norwell, MA: Kluwer, 1997.
[18] O. Cordon, F. Herrera, F. Hoffmann, and L. Magdalena, *Genetic Fuzzy Systems: Evolutionary Tuning and Learning of Fuzzy Rule Bases*, Singapore: World Scientific, 2001.
[19] M. Delgado, A. F. Gomez-Skarmeta, and F. Martin, "Using fuzzy clusters to model fuzzy systems in a descriptive approach," in *Proc. Information Processing Management Uncertainty Knowledge-Based Systems*, Granada, Spain, July 1996, pp. 564–568.
[20] S. Medasani, J. Kim, and R. Krishnapuram, "An overview of membership function generation for pattern recognition," *Int. J. Approx. Reason.*, vol. 19, pp. 391–417, 1998.
[21] L. X. Wang and J. M. Mendel, "Generating fuzzy rules by learning from examples," *IEEE Trans. Syst., Man, Cybern.*, vol. 22, pp. 1414–1427, 1992.
[22] B. Kosko, *Neural Networks and Fuzzy Systems: A Dynamical Systems Approach to Machine Intelligence*. Englewood Cliffs, NJ: Prentice-Hall, 1992.
[23] T. Sudkamp and R. J. Hammell II, "Interpolation, completion and learning fuzzy rules," *IEEE Trans. Syst., Man, Cybern.*, vol. 24, no. 2, pp. 332–342, 1994.
[24] J. A. Dickerson and M. S. Lan, "Fuzzy rule extraction from numerical data for function approximation," *IEEE Trans. Syst., Man, Cybern.*, vol. 26, pp. 119–129, 1995.
[25] T. Thawonmas and S. Abe, "Function approximation based on fuzzy rules extracted from partitioned numerical data," *IEEE Trans. Syst., Man, Cybern. B*, vol. 29, no. 4, pp. 525–534, 1999.
[26] C. C. Lee, "Fuzzy logic in control systems—Part I: Fuzzy logic controller," *IEEE Trans. Syst., Man, Cybern.*, vol. 20, no. 2, pp. 404–418, 1990.
[27] W. Pedrycz, "Why triangular membership functions?," *Fuzzy Sets Syst.*, vol. 64, pp. 21–30, 1994.
[28] E. H. Mamdani and S. Assilian, "An experiment in linguistic synthesis with a fuzzy logic controller," *Int. J. Man-Mach. Stud.*, vol. 7, pp. 1–13, 1975.
[29] E. H. Mamdani, "Advances in the linguistic synthesis of fuzzy controllers," *Int. J. Man-Mach. Stud.*, vol. 8, pp. 669–678, 1976.
[30] T. A. Sudkamp and R. J. Hammell, II, "Granularity and specificity in fuzzy function approximation," in *Proc. NAFIPS*, 1998, pp. 105–109.
[31] J. Espinosa and J. Vandewalle, "Constructing fuzzy models with linguistic integrity from numerical data-AFRELI algorithm," *IEEE Trans. Fuzzy Syst.*, vol. 8, no. 5, pp. 591–600, 2000.
[32] L. Wang and J. M. Mendel, "Generating fuzzy rules from numerical data, with applications," Signal and Image Processing Inst., Univ. Southern California, Los Angeles, Tech. Rep. USC-SIPI-169, 1991.
[33] T. Sudkamp and R. J. Hammell, II, "Rule base completion in fuzzy models," in *Fuzzy Modeling: Paradigms and Practice*, W. Pedrycz, Ed. Norwell, MA: Kluwer, 1996, pp. 313–330.
[34] H. Ishibuchi, K. Nozaki, N. Yamamoto, and H. Tanaka, "Construction of fuzzy classification systems with rectangular fuzzy regions," *Fuzzy Sets Syst.*, vol. 65, no. 2, pp. 237–253, 1994.



Thomas Sudkamp (M'94) received the B.S. degree in mathematics from University of Wisconsin, Madison, in 1974, the M.S. and Ph.D. degrees in mathematics from the University of Notre Dame, Notre Dame, IN, in 1976 and 1978, respectively, and the M.S. degree in computer science from Wright State University, Dayton, OH, in 1982.

Currently, he is a Professor in the Department of Computer Science and Engineering at Wright State University. His research interests are in the application of soft computing techniques to modeling and decision making in complex problem domains.

Dr. Sudkamp is an Associate Editor of IEEE TRANSACTIONS ON SYSTEMS, MAN AND CYBERNETICS and IEEE TRANSACTIONS ON FUZZY SYSTEMS. He is also the President of the North American Fuzzy Information Processing Society.



Aaron Knapp received the B.S. and M.S. degrees in computer science from Wright State University, Dayton, OH, in 1999 and 2001, respectively.

He is currently with Northrop Grumman Information Technologies, Dayton. His research interests include system modeling and the design of fuzzy rule bases.

Jon Knapp received the B.S. and M.S. degrees in computer science from Wright State University, Dayton, OH, in 1999 and 2001, respectively.

He is currently with TriVectus LC, Phoenix, AZ, which specializes in Macintosh-based consulting and development, web-site design, and computer graphics. His research interests in soft computing include system modeling and the design of fuzzy rule bases.