Regular Papers

R-MINI: An Iterative Approach for Generating Minimal Rules from Examples

Se June Hong, *Fellow, IEEE*

Abstract—Generating classification rules or decision trees from examples has been a subject of intense study in the pattern recognition community, the statistics community, and the machine-learning community of the artificial intelligence area. We pursue a point of view that minimality of rules is important, perhaps above all other considerations (biases) that come into play in generating rules. We present a new minimal rule-generation algorithm called R-MINI (Rule-MINI) that is an adaptation of a well-established heuristic-switching-function-minimization technique, MINI. The main mechanism that reduces the number of rules is repeated application of generalization and specialization operations to the rule set while maintaining completeness and consistency. R-MINI results on some benchmark cases are also presented.

Index Terms—Classification, minimal rules, DNF, Occam's razor, generalization, specialization.

1 Introduction

There are many approaches to generating Disjunctive Normal Form (DNF) rules from examples. The Aq family of rule generation and other approaches [1], [2], [3], [4] incrementally cover the class examples by forming one rule at a time. Most of the other approaches [5], [6] construct a decision tree and derive the rules from it by modifying and pruning the path conditions according to some bias criteria. In all these approaches, the minimality of the rule set has not been a seriously enforced bias. While generating a true minimum rule set is well known to be NP hard, we show that the objective of near minimality can be achieved without a computational explosion.

It is not the purpose of this paper to argue the merits of minimality in detail. The Occam's razor argument [7], [8], [9] strongly favors minimality of the representation for accuracy. Fayyad and Irani [10] provide an analysis for why the minimal number of leaves in a tree (equivalent to the number of rules) and is perhaps the most important bias. Simpler rules are more understandable and more efficient to apply. The minimum-description-length principle has been used by some [11], [25] to prune the trees for reducing error rates. We concentrate only on the generation of complete and consistent rule sets that are near minimum in size. Completeness means the rules cover all of the positive examples in the given class, and consistency means the rules cover none of the negative examples. Others also argue for generating a complete and consistent rule set first and then applying a pruning process for error consideration to both rules and decision trees [12], [13]. We advocate that the ***minimal*** complete and consistent rule set be generated first; pruning to avoid overfitting noisy data can be a separate process.

The technique presented here is confined to generating rules for examples expressed in categorical features. This is not as restrictive as it may seem, because we can assume that all numeric features have been discretized by some preprocess, many of which are described by Dougherty et al. [14]. They establish experimentally that prediscretization is competitive with the dynamic discretization that occurs during the traditional tree induction process. In a separate paper [15], we describe a new context-based feature-analysis technique that is used for both feature ranking and discretization.

We borrow the core heuristics for driving minimality from a successful heuristic minimization technique for large switching functions, MINI [16]. Similar core heuristics were also adopted by ESPRESSO [17], a publicly available package for switching function minimization. They consist of iterating over three key subprocesses:

1) a generalization step, ***EXPAND***, which takes each rule in turn (initially each example is considered a rule) and generalizes it to remove other rules that become covered,
2) a specialization step, ***REDUCE***, which takes each rule in turn and specializes it to the most specific rule still general enough to cover the unique positive examples the rule must cover (if a rule covers no *unique* positive examples, the rule is deleted), and
3) a reformulation step, ***RESHAPE***, which transforms a pair of rules into another pair with the same coverage, for all pairs that can be so transformed.

• *The author is with the IBM Research Division, Thomas J. Watson Research Center, Yorktown Heights, NY 10598. E-mail: hong@watson.ibm.com.*

Manuscript received 16 Sept. 1994; revised 11 Dec. 1995.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number 104413.

MINI and ESPRESSO have been the main methods used for VLSI two-level logic minimization for the past 20 years. The iteration approach has been proven more effective than other heuristic approaches for this difficult problem. The importance of using both generalization and specialization for the induction process was also noted in [18]. Smyth and Goodman's paper [19] describe another rule induction technique using only categorical features, which also uses the generalization and specialization paradigm to derive the n-"best" rules according to their J-measure. The objective there is to induce statistically significant rules about any of the features in terms of other features, as opposed to generating a rule set to model a given class feature, per se.

In our approach, the primary determiner of whether a rule is general or specific is the number of examples in the universe a rule may cover, in contrast to the popular practice where the degree of generality depends only on the number of constraining feature conditions involved in a rule.

The R-MINI technique is programmed in APL and C++. It is now incorporated into our data abstraction system, RAMP (Rule Abstraction for Modeling and Prediction) [20]. It has been applied to many large real data as well as some benchmark data from the UCI repository, summarized in a later section. Since R-MINI rule generation relies on iterative improvement, one can, in principle, use as much computing time as affordable. The results reported here were obtained by setting some reasonable limits on the number of iteration rounds. We remark that the R-MINI algorithm described here is better adapted for rules than an earlier attempt [21] to use MINI in a straightforward manner for generating classification rules.

2 Positional Cube Notation and the Minimality Objective

We consider the examples as vertices in an n-dimensional universe. The axis corresponding to the feature X_i contains m_i discrete value points representing the distinct values of the feature's domain. A rule corresponds to an n-dimensional set of vertices specified by the subsets of values in each feature axis, and is commonly called a "cube" in the switching theory community. Any vertex (example) that is a member of the Cartesian product of the specified subsets is said to be covered by the cube (rule). Conversely, the rule is satisfied by any of the vertices in the cube.

We make use of the positional cube notation as an efficient data structure to denote the vertices and the cubes in the universe. An m-bit vector of 0s and 1s represents the subset of values for an m-valued feature, 1 signifying the presence of the value in the subset. We call such vector a ***part*** in a cube. A cube is represented by simply concatenating parts in the order of the features.

Consider the following case with four features:

X_1	color	{pink, white, blue, black}	$m_1 = 4$
X_2	shape	{round, square, oval}	$m_2 = 3$
X_3	length	{long, short}	$m_3 = 2$
X_4	material	{wood, copper, lead, plastic}	$m_4 = 4$

An example for a class of "OK-for-baby-furniture" might be pink, square, long, and wood, which is represented as:

1000 010 10 1000

(spaces are for readability only).

An example (a vertex) obviously contains exactly one feature value (one "1 bit") per part. A rule for this class might be, "If the color is not black, length is long, and the material is either wood or plastic, then OK," which is represented as the cube:

1110 111 10 1001

The entire universe is just a single cube with all 1s in each of its parts. If a part contains all 1s, that feature is not involved in the rule, because any value may satisfy it. If any part contains all 0s, it denotes a vacuous set of vertices, i.e., a null cube. We say a cube covers another cube or a vertex (an example) when all of the vertices contained in the latter are also contained in the former, i.e., the covering cube has 1s everywhere the covered cube has 1s in their positional cube notation. Logical operations using the positional cube notation are straightforward. (See [16] for more details.)

When there is more than one cube for a given function (for a class), all the conditions of cubes are ORed. The list of cubes for the rule set of a given single class is represented as a matrix whose rows are the cubes. We will call the number of cubes (or rows) in a solution the ***solution size***. Within a cube the conditions of each part are ANDed, therefore representing a DNF expression for the function. ***Cube size*** is defined as the total number of vertices contained in the cube, i.e., the product of the number of 1s in each of its constituent parts. The larger the cube, the more general, in that it contains more vertices in the universe. Larger cubes (rules) tend to be expressed in simpler English. In terms of rules, a more general rule covers more possible examples. The most general cubes (called here "prime cubes") are cubes where no more 1s can be added to any part of a cube without violating consistency. Our minimality objective is first driven by the minimal number of cubes, and then driven by the most general (or prime) cubes. To summarize, once we see examples and rules as vertices and cubes (a vertex is a cube of size 1), the minimality objective translates to finding a minimal number of prime cubes that cover all of the positive example vertices without covering any of the negative example vertices, which is the same objective of many switching function minimization algorithms including MINI.

The R-MINI approach is described here in terms of generating rules for one given class at a time. The positive examples are those examples in the training data that are associated with the given class; the negative examples are those which are associated with all other classes. In practice, the R-MINI algorithm is repeatedly run, once for each class value being modeled.

3 Differences between Switching Function and Classifier Rules

Logically, generating a minimal DNF switching function is identical to generating a minimal set of rules for a single class as long as no erroneous examples are assumed to be present. As mentioned earlier, we assume this for now and concentrate on obtaining a minimal set of complete and consistent rules. However, important differences between these two problem environments necessitate new algorithms for generalizing, specializing, and reformulating the cubes for the rules. We now examine some of the salient differences before describing the R-MINI processes.

For switching functions, most of the vertices in the universe are specified as either belonging to the function F or its complement $\overline{F}$. Since the vertices are too numerous, the specification is usually given in the form of implicants (cubes) of the function F, and a small portion of the universe characterized as the "don't care (DC)" space. The complement $\overline{F}$ is then computed as the negation of (F $\cup$ DC). The situation for switching functions is that the number of specified vertices are generally too large to enumerate while the initial nonminimal DNF specification for F, DC, (and $\overline{F}$) can be of manageable size, for instance, in the thousands. MINI iteratively reduces the number of cubes of F, using the $\overline{F}$ and DC explicitly.

For classification problems, the size of the universe can be much larger than a typical switching function universe. (A 100 Boolean variable universe contains 2^{100} vertices.) The total number of positive examples (initial F) and negative examples ($\overline{F}$), i.e., the specified vertices, can range in the thousands to, say, millions, which may still be a very small (even negligible) fraction of the universe. For an explicit representation of unspecified space (DC), if one were to derive it, one must compute the negation of (F $\cup$ $\overline{F}$) which would take too much time and produce too many resultant cubes, owing to the vastness of the DC space (exponential in terms of the number of features in the worst case). In the absence of domain knowledge about these unspecified vertices, they are opportunistically covered by the rules according to the particular bias of the rule-generation method. In the case of R-MINI, as in MINI, the unspecified vertices are covered by the class rule if doing so minimizes the number of rules.

The specialization process, REDUCE, in MINI takes the cubes in the current solution F and reduces each cube to the smallest subcube that will still cover all the F vertices uniquely covered by the original cube. The REDUCE process takes a cube f of F, in turn, and returns the smallest subcube covering f $\cup$ $\overline{\text{DC} \cup \text{rest of the cubes in F}}$. The negation operation inherent in this process may require a large interim memory space and runtime [16]. We accomplish the same goal in R-MINI by just finding a smallest cube that covers the original positive vertices not covered by the rest of the cubes, without using explicit DC. Computation required for this is modest as we shall show later.

An important consequence of having positive and negative examples that are a small fraction of all possible vertices of F and $\overline{F}$ is that generated cubes may not abut each other, contrary to what often happens in switching functions. MINI uses an effective cube reformulation process, RESHAPE, which reshapes pairs of adjacent cubes into a different pair covering the same space. In R-MINI's situation, such opportunities are rare. So the reformulation is also performed by a generalization and specialization sequence, tailored to effectively change the cube shapes.

All three key subprocesses in MINI make use of heuristics designed to help reduce the solution size. The order the cubes are generalized, the order each part is generalized for a cube, and the order the cubes are reduced and reshaped all follow particular heuristics that have been empirically developed (and then rationalized) and proven to be highly effective. Once an EXPAND, REDUCE, and RESHAPE cycle fails to improve the solution size, it is rarely improved by the subsequent cycles. Thus MINI terminates (default mode) when the solution does not improve after the current cycle. For classification problems, due to the low density of specified vertices, we found that the solution often improves again after some unsuccessful cycles. Multiple nonimproving cycles in essence serve as multiple reformulation attempts. When a deterministic heuristic ordering fails to improve the solution, we found it effective to try again with randomized cube and part ordering (instead of directive heuristics) in all subprocesses, another departure from MINI.

The new algorithms that we describe here are distinguished from those of MINI by naming them using a prefix "R-" (RULE-).

4 R-MINI Overview

We now present an overview of the R-MINI process which will be elaborated in the subsequent sections. We assume that conflicting examples (the same example in both positive and negative example sets) are removed. Initially, the positive examples are encoded in the positional cube notation as a list of cubes F, and likewise the negative examples as $\overline{F}$. For later processes, we keep a copy of the positive examples, the initial F, as FF. The process iterates a generalization, specialization, and reformulation cycle (called a TRY) until there is no reduction in the size of F after a set number (MAXTRY) of **consecutive** TRYs. (Even for large problems, we have seen little improvement beyond a MAXTRY value of 5-7, although a lower MAXTRY value is adequate for many problems.)

Often, it is computationally advantageous to quickly reduce the sizes of F and $\overline{F}$ by a ***single-distance-merge*** before the iteration begins. Two cubes can be single-distance-merged into one that covers exactly the same vertices as before. The bits of a given part of the two cubes are ORed whenever all the other parts of the two cubes are identical. Single-distance-merging a given part takes linear time in the number of cubes. First, cubes are multikey sorted (repeated binary bin sorting) by all the bit positions of the other n – 1 parts. Cubes that are identical in these n – 1 parts must appear adjacent in the sorted cubes. It is only necessary to inspect the adjacent cubes down the list to check if the pair are identical in these n – 1 parts. If so, the given part is ORed, and the new cube replaces the two merged ones in place.

The merging does not change the order of the cubes. Sorting the cubes again by the bits of the part just merged allows the next part merging by adjacent pair inspection. The resultant number of cubes after all the parts are single-distance-merged can be much smaller than the original. Although the merged cubes are more general than the examples, the merged solution does not cover any of the unspecified vertices. The merged solution is, therefore, complete and consistent but does not represent any more generality than the original positive examples taken as a whole. For the UCI mushroom case, this process alone reduces 4,208 edible mushroom examples to 13 cubes and 3,916 poisonous examples to 10 cubes. These 23 cubes cover exactly the given 8,124 example vertices and no more. The README file for this case states that these examples were generated from 23 types of mushrooms. R-MINI further reduces the solution to two edible rules and three poisonous rules (see Appendix A).

The minimizing iteration starts after creating an initial solution by applying one generalization step (R-EXPAND on F). The number of cubes of this initial solution is comparable to the number of complete and consistent rules or tree leaves generated by other methods before pruning. We iterate on this initial solution to minimize the size of the complete and consistent set of rules. One iteration is called a TRY which consists of specialization, reformulation and generalization sequences, performed by R-EXPAND and R-REDUCE algorithms and their variations. A TRY is deemed *successful* if it reduces the number of cubes in F. The iteration terminates after MAXTRY *consecutive* unsuccessful TRY – TRY/R sequences. For each unsuccessful TRY, a run of a randomized version of TRY, TRY/R, is inserted to force the solution out of a local minima. Since the reformulation is again composed of a variation of generalization and specialization sequence, the entire R-MINI process is essentially iterating the generalization and specialization. At the end, the cubes are postprocessed for prime rules, presentation, and error evaluation on test examples, etc. In summary, the R-MINI process consists of encoding F and $\overline{F}$, running single-distance-merge on both F and $\overline{F}$, creating an initial solution by running R-EXPAND on F, and then iterating TRY and TRY/R using the flowchart in Fig. 1.

When rules for multiple classes are to be generated, this process is repeated for each class before postprocessing.

Since R-MINI uses randomization, repeated invocations of R-MINI for the same problem can be used to produce different sets of near minimal solutions. Each such rule set covers the feature space by a differently shaped minimal set of cubes for the given class. (For small noise free problems where the number of given examples is a substantial portion of the feature space, the multiples solutions turn out to be identical). Combining multiple rule sets for classification decision generally improves the accuracy of predictions.

Fig. 2 depicts a typical convergence of the rule size through the TRY iterations for large real problems. The problem was taken from the financial domain [22]. It contains about 1,000 positive examples, 4,000 negative examples, and 40 features. The initial solution size is 320. Fig. 2 shows the progress of the rule sizes for five invocations of R-MINI on the same problem with MAXTRY = 5. The resultant solution sizes range between 65 and 67.

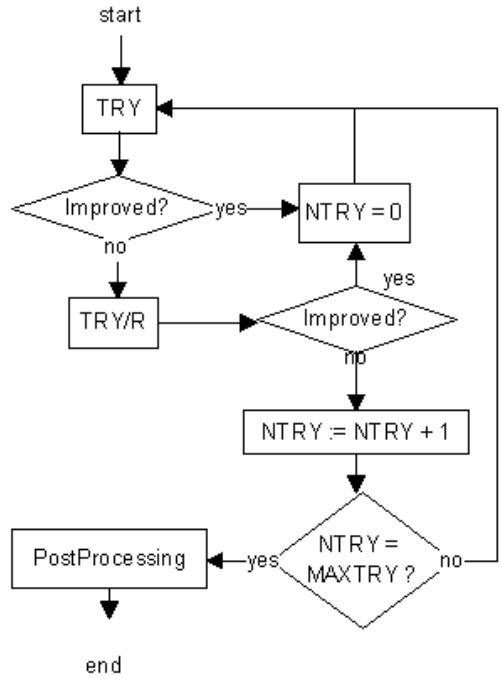

Fig. 1. Flowchart of R-MINI iteration.

We shall now describe the subprocesses.

5 Generalization: R-EXPAND

The main goal of R-EXPAND is to generalize the cubes so that the expanded cube would cover other cubes, which are then removed during the process.

The order in which the cubes are generalized affects the result. We adopt a greedy ordering that selects the current cube to be expanded to be the one that shares the most feature values with other cubes. The rationale is that if there are many cubes that share many coordinate values with the current cube, their covered vertices are more likely to be

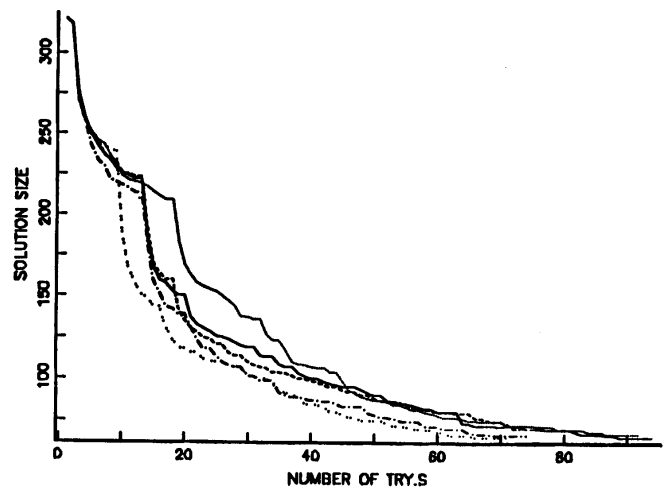

Fig. 2. Convergence of rule size.

covered by the expanded cube. When a cube shares a small number of like values with other cubes, it is difficult to generalize it to cover others. In the MINI situation, where practically all the vertices are specified, we take the opposite cube ordering, expanding the difficult cubes first. This has the effect of "chewing away from the edges," so that there may be nothing left to cover in the "center." This ides does not apply in the sparely specified situation common in classification applications. The greedy ordering reduces the solution size more quickly and, therefore, is computationally advantageous. The shortcoming associated with the "big-chunk-first" approach is compensated by the reformulation step, which is more effective in the R-MINI situation.

The total number of feature values an unexpanded cube shares with other unexpanded cubes is computed as follows: Develop column sums of the number of 1s among the unexpanded cubes (dynamically maintained as cubes are processed). For each unexpanded cube, total the column sums of the columns where the cube has 1. This is then the total number of times the values of the cube are shared with all the unexpanded cubes.

Once a cube is selected, the cube generalization is done by generalizing each part of the cube **recursively**, by ***single-part-expansion*** (SPE), until all parts are maximally general (i.e., prime). To perform SPE on part p of a cube f in F, we make use of the $\overline{F}$ cubes. First, we find the subset of $\overline{F}$ cubes that has a nonnull intersection with f in every part but p. All the values for part p not used by any cube in this subset (i.e., no cube in this subset has bit = 1 in the part p) can be used by f. So the part p of all the subset cubes are ORed (along each bit position) and complemented; this result must cover part p of f and, therefore, replaces it. When SPE is performed recursively on all parts of f in some part ordering, the resultant cube is prime. The order in which parts are expanded affects the size and shape of the resultant cube. We use a part ordering heuristic (same as in MINI) to grow the cube toward a shape that is more likely to cover other cubes of F.

To determine the part ordering for expanding f, first obtain an over expanded cube of f, C(f), by replacing each part of f with its single-part-expanded part from the **original** f. In general, the over expanded cube is not consistent because the parts are expanded maximally from the original f disregarding other expanded parts. However, any cube in F that may be covered by a correctly expanded f, must also be covered by C(f). Let the subset of cubes in F that are covered by C(f) be SF(f). For each part p, count the number of cubes in SF(f) whose part p is not covered by the part p of f. The more this count for the part, there are more cubes that can potentially be covered by expanding f in that part fully. So the part is ordered by this count in descending order. We remark here that the over generalized C(f) is also the smallest cube that covers all possible consistent generalizations of f.

Once the current cube is selected and its part ordering is determined, SPE is performed in the given part order **recursively** to obtain a consistent prime cube E(f) that covers the current cube f.

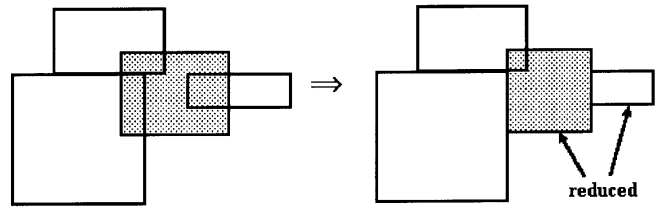

Fig. 3. Local cube reduction.

Before the current cube is replaced with E(f), we perform two local cube reduction steps to reduce the other cubes of F and to reduce E(f) itself, for two reasons:

1) the smaller the cube, the better the chance it may get covered by other cubes being expanded, and
2) since single-part-expansion is greedy (i.e., maximal generalization of the part), it restricts the shape of the prime cube E(f).

Had a part (of f) been expanded only partially, it may have enabled a later expanding part to become more general. This local reduction allows a wider variety of cube shapes to be considered for the next round.

The local cube reduction is depicted in Fig. 3 for a two-feature case. The hatched cube is the just expanded E(f), assuming that positive vertices densely pack these cubes of F.

One cube g can reduce another cube h of F if g covers h in all parts or in all but one part p. The part p of h can be replaced with (part p of h) $\wedge$ $\overline{\text{(part p of g)}}$ without affecting the coverage of vertices. When g covers h, h is removed. This reduction is attempted for all remaining cubes of F with the just expanded E(f). This step removes all cubes from the current F that are fully covered by E(f), including f itself. At this point the current F (which does not yet include E(f)) is temporarily not complete.

Now, E(f) itself is made as small as possible while still covering the unique positive vertices it must cover to remain complete. This is done by first finding the subset SFF of positive vertices FF that are covered by E(f). We remove from SFF all vertices that are covered by all the cubes of the current F. The resultant SFF is column wise ORed to obtain the desired reduced cube of E(f). This cube is marked processed and appended to F, making it complete again.

While the cubes of F are being expanded and locally reduced, the F remains complete and consistent, for SPE ensures consistency and the local cube reductions ensure completeness at each step. Here is a summary of the R-EXPAND process:

E1	Do until all cubes of F are processed	
	E2	Determine the current cube f by finding the most value sharing unprocessed cube
	E3	Determine part ordering for f
	E4	Obtain the prime cube E(f) by applying SPE recursively in the given part order
	E5	Locally reduce all cubes in F
	E6	Locally reduce E(f) against all cubes of F
	E7	Append reduced E(f) to F and mark it processed

The R-EXPAND process is linear in the number of features (parts) and quadratic in the number of cubes in F. The very first application of R-EXPAND to obtain the initial solution is therefore quadratic in the number of examples in the worst case. After the first few iterations, the bulk of the size improvements are achieved; thereafter, the R-EXPAND process becomes roughly quadratic in the final solution size.

A variation where *E2* and *E3* are replaced with a random selection and a random ordering is called R-EXPAND/R. This version is used for the randomized TRY/R which augments an unsuccessful deterministic TRY. Another variation, R-EXPAND/M, omits the local reduction steps *E5* and *E6*, and directly replaces f with E(f) for the step *E7*, leaving the resultant cubes of F all prime. This also has a randomized variation for *E2* and *E3*, called R-EXPAND/MR. The use of these variations will be presented later.

6 Specialization: R-REDUCE

The objective of specializing the cubes is to make them as small as possible while maintaining completeness. The reason this is desired is that a smaller cube has higher chance of being partially or totally covered by other cubes when they are expanded. Also a smaller cube can be generalized later in R-EXPAND with more flexibility. Although cubes are reduced locally during the R-EXPAND process, we seek here a more thorough process. The cubes of F are reduced to the necessary subcubes of the originals, one cube at a time, such that the reduced subcube covers all the positive vertices covered by the original one but not by any other cubes in F. The rules in such a reduced set are therefore necessarily general. (This step is logically equivalent to the REDUSTAR process of the Aq algorithm [23]. The resultant F is called an irredundant DNF in switching function minimization, where no cube may be made smaller without violating the completeness.) Since the number of positive vertices are computationally reasonable in the classification applications, the R-REDUCE process makes direct use of the positive examples FF, avoiding a potentially exponential computation inherent in the REDUCE process used in MINI.

It is readily seen that the order in which cubes are reduced against the other cubes affects the resultant F. Here we use the same heuristic ordering as in the R-EXPAND, for the cube that shares the most like values with other cubes is also the one most likely to be (partially) covered by them.

Once the cubes of F are ordered top-to-bottom by descending number of total shared values (computed in the same manner as in R-EXPAND), we define two subsets, FF1 and FF2, of the positive examples FF. FF1 is the set of positive examples not covered by any cube above the current cube. (Initially, FF1 is set to FF). FF2 is to be the subset of FF1 that is not covered by any cube below the current cube. The minimal subcube of the current cube that is necessary is then the column-wise OR of FF2, which is denoted by R(f)—the reduced f. If FF2 becomes an empty set, the current cube is removed. The R-REDUCE process is:

R1	Initialize FF1 as FF	
R2	Order cubes of F in descending total shared values of the cubes	
R3	Do for each unprocessed cube f, from the top	
	R4	Let FF2 be FF1
	R5	Do for all cubes below f
		R6 Remove covered vertices from FF2
	R7	If FF2 becomes empty, remove f from F, return to R3
	R8	Else, let R(f) be the column-wise OR of FF2
	R9	Replace f of F with R(f)
	R10	Update FF1 by removing all vertices covered by R(f)

Due to *R7*, the number of cubes in F often decreases. The algorithm is efficient because the positive vertices once claimed by a cube are not considered again. The computation is quadratic in the number of cubes in F (due to *R5-R6*). The R-REDUCE process preserves the completeness of F because it maintains the coverage of all positive vertices in FF. Since the process starts with a consistent F and reduces the cubes, the resultant F remains consistent.

For steps *R6* and *R10*, and also for the local reduction step *E6* of R-EXPAND, we must find which of the vertices in G (each vertex being a row of the G array) is covered by a given cube g. This is accomplished by first selecting the columns of G where g has 0s. Perform row-wise OR of these columns. A vertex is covered if and only if the corresponding row-wise ORed result is 0. This also applies when the rows of G are cubes as well. In a reverse situation when there is one vertex h, to find which cubes in H covers it, select the columns of H where h has 1s, and row-wise AND these columns. A cube covers h if and only if the corresponding row result is 1.

For the randomized variation, R-REDUCE/R, the cube ordering step *R2* is replaced with a random cube ordering.

7 Reformulation

Whenever cubes are generalized or specialized the representation of the positive example space is being reformulated. The main objective of the two procedures, R-EXPAND and R-REDUCE, however, is to reduce the size of F, i.e., reduce the number of cubes in F. To change the cube shapes most effectively, we first make all the cubes of F maximally general by R-EXPAND/M and use a random reduction, R-REDUCE/R. The cubes that are as large as they can be enables more flexibility of shapes when they are randomly reduced. The size of F can sometimes decrease during this sequence as well. Practically all cubes change shape here due to the sparsity of specified vertices; in contrast, only a small portion of the cubes change shape during RESHAPE in the MINI situation.

Thus, the main cycle of the iteration on the initial solution, TRY (TRY/R) consists of:

Specialization:	R-REDUCE (R-REDUCE/R)
Reformulation:	R-EXPAND/M (R-EXPAND/MR) and R-REDUCE/R
Generalization:	R-EXPAND (R-EXPAND/R)

Since all the subprocesses maintain completeness and consistency, TRY and TRY/R also do and, therefore, R-MINI produces a complete and consistent rule set.

8 Postprocessing

When the TRY-TRY/R cycle stops, the cubes in F, in general, are neither prime nor reduced (necessarily general). We produce two versions of rules—one reduced and one prime. The prime rules tend to overlap coverages of examples more and also cover more of the unspecified vertices. Depending on the application one may opt for one version or another, or even mix the rules from the two versions for different classes. (The results reported in the next section were obtained using the prime rules only).

Each part of a cube specifies the subset of feature values. A part in a reduced cube defines the necessary subset (to maintain completeness), while the same part in the cube's prime counterpart defines the upper limit of that subset (to maintain consistency). Examining such subset pairs from all the rules, it is sometimes possible to group the values in the feature's domain such that the groups (a higher level concept in the feature's domain) may be used as values in the rules. This application of the two versions of the rule sets will be a subject of further research.

The postprocessing includes R-EXPAND/M on F for the prime rule set PF followed by R-REDUCE on PF for the reduced rule set RF. It is possible that the size of RF may decrease from that of PF when MAXTRY is set too low for the case. This is checked in R-MINI and, when it happens, we go back to the R-EXPAND phase of the TRY iteration, with RF as if the previous TRY were a success. The remainder of the postprocessing operations implemented in the RAMP system [20] to date include processing the rule sets for various statistics on them and for presentation, and doing interactive evaluation.

9 Results and Discussions

There are several reasons why it is difficult to compare the R-MINI results to those of others. There are very few public data that consist of only categorical features. When the data contain numeric features we must combine the result of a an external discretization step with that of R-MINI in order to make a comparison. Another problem is that most of the available rule/tree induction tools generate pruned results, not the complete and consistent rule sets. Since we have not seriously explored rule pruning yet, a fair comparison of the resulting number of rules as well as their accuracy evaluation is difficult. Furthermore, most of the published results report on the accuracy statistics but not the sizes of their models, especially the complete and consistent versions; the size comparison is difficult. (Some rule set sizes reported in [3] are several times their minimal values.) We hope that the minimality bias will be taken more seriously in the future to improve this situation.

R-MINI has been applied to many real cases with hundreds of features and tens of thousands of examples, resulting in several hundreds of rules. We ran eight StatLog [24] cases (all but one is the same used in [25]) through our RAMP system. All cases except DNA contain numerical features which were discretized by the method in [15] before R-MINI was invoked to generate rules. Thus, the DNA case is the only one where the R-MINI result can be compare in isolation. For each subcase (of n-fold cross validation), R-MINI was invoked five times, generating five sets of rules for each class—except for the Letter case for which only two rule sets were generated. The multiple rule sets were combined and no rules were pruned. Instead, we used a simple voting scheme (a default choice in RAMP) to arrive at a classification decision for each test example.

The default rule application strategy (voting scheme) in RAMP is as follows: Each rule is assigned a weight com-

TABLE 1
RAMP Performance on Eight StatLog Cases

Case	*Evaluation Method*	***StatLog Data***					***Ramp Data***		[25] *MDL Tree Size*
		Best	***Error Rates***				*Rank/ 24*	*Avg. best size*	
			Best	*C4.5*	*CART*	*CN2*	*RAMP*			
CrAust	10 fold on 690	Cal5	0.1310	0.155	0.1450	0.2040	0.1390	3	31.2	23.6
Diabetes	12 fold on 768	Logdisc	0.2230	0.270	0.2550	0.2890	0.2720	16	55.1	34.8
Vehicle	9 fold on 846	Quadisc	0.1500	0.266	0.2350	0.3140	0.2890	15	55.6	72.1
Segment	10 fold on 2,310	Alloc80	0.0300	0.040	0.0400	0.0430	0.0329	3	35.6	56.2
DNA	2,000Tr, 1,186Tst	RBF	0.0410	0.076	0.0850	0.0950	0.0489	3	50.0	51.0
SatIm	4,435 2,000Tst	K-NN	0.0094	0.150	0.1380	0.1500	0.1260	5	146.0	167.0
Letter	15,000 5,000Tst	Alloc80	0.0640	0.132	*0.1300	0.1150	0.1080	4	693.0	1,175.0
Shuttle	43,500 14,500Tst	NEWID	0.0001	0.001	0.0008	0.0003	0.0011	8	20.0	---

puted as 1 + N1/N2 where N1 = number of training examples the rule covers, and N2 = number of training examples in the rule's class. For each test example, we first determine the set of rules FM that are "fully matched" and the set AM of rules that are "almost matched," that is, matched in all but one of the constrained features. All the rules in FM contribute their full weight to the rule's class while the rules in AM (by default setting) contribute 0.1 of their weight to the rule's class. The class that accumulates the highest total weight wins. Our evaluation was done exactly the same way (either the n-fold cross validation or separate training and test set) as was done in StatLog.

In Table 1, we show the error rates of the best method and those of C4.5 and CART (in the case of Letter, we show the result of IND-CART as indicated by *) and CN-2 from [24] for reference. The RAMP error rates, its rank among the methods in StatLog (out of 24 including RAMP now), and the number of minimal complete and consistent rules (best among the multiple rule sets generated), averaged over the n-fold trials, are shown in the next three columns. The last column shows the average number of nodes in MDL-pruned decision trees reported in [25] for reference. Although there is a correlation between the minimal rule set size and the MDL-pruned tree size, one needs the number of leaves in the tree for a direct comparison.

The RAMP-generated rules (even without pruning and combined with an extra step of discretization, but with a simple voting on multiple rule sets) came in among the top five in five of the eight cases tried. Except for the shuttle case, the RAMP error rates are smaller than those of CN2, which also generates DNF rules. We feel that with a proper pruning strategy, the modeling accuracy based on minimal rule sets can be made even more robust for a wide variety of data, as long as an adequate set of features are available. The two cases, Diabetes and Vehicle, may contain much noise (therefore pruning must be done), or the decision surfaces in the given feature spaces may not be suited for modeling by DNF rules. We are investigating effective techniques for both removal of outliers (pruning the examples) and pruning the minimal complete and consistent set of rules. We also hope to run the rest of the cases in StatLog through the RAMP system and report on them in the future.

R-MINI runtime is dependent on many factors, including the speed of convergence. The total number of all distinct symbolic values, i.e., the number of bits of the positional cube notation, contributes linearly to the runtime. The number of cubes at each TRY (initially the number of examples) contributes to the runtime no more than quadratically. As the solution size decreases, each TRY time also decreases. On a 63 MHz RS/6000 workstation, the UCI mushroom case takes about 2 seconds to generate the set of complete and consistent rules for both classes (MAXTRY = 5). The best rule set we obtained is shown in Appendix A.

Any complete and consistent rule set can serve as the initial solution F to be further improved by R-MINI's TRYs. MINI has been shown to produce near minimal solutions, and since R-MINI follows the same approach (except in the handling of the unspecified (don't care) space), we claim that R-MINI results are also near minimal. Furthermore, the computation to achieve the near minimality is only quadratic in the number of rules and linear in the number of features. The total memory space needed is also linear in the space needed to hold the initial data.

Finally, we offer an intuitive conjecture for the reason why minimal rule sets may produce a more accurate model when there are some redundant features. Driving for minimality indirectly seeks rules that cover more positive examples. In the presence of redundant features, the minimality approach favors those feature constraints for a rule having more coverage; a combination of features constraints that is more sensitive to noise would tend to cover a smaller number of examples, and would not be preferred. Redundant variables render more flexibility in finding such a context in which the constrained features in their conjunctive contexts are less sensitive to noise.

APPENDIX A

The mushroom example from the UCI depository contains 4,208 examples of edible mushrooms and 3,916 examples of poisonous mushrooms. The unknown value (denoted as ?) in the data was treated as a distinct value of the feature. R-MINI runs on this data consistently produce five rule solutions (with MAXTRY = 5). We ran R-MINI 20 times to obtain the following set of simplest rules (total tests in the rule set is 15). Other solutions had up to three more total tests in the rule set. The RAMP system presents the rules with much more information than shown below.

Class: Edible; Number of examples: 4,208

Rule 1:
Gill-size: (broad)
Ring-type: NOT (none)
Spore-print-color: NOT (chocolate green)
=> **Edible** Coverage: 3,920

Rule 2:
Odor: NOT (creosote pungent)
Gill-size: (narrow)
Gill-color: NOT (buff)
Stalk-surface-to-below-ring: NOT (Scaly)
Population: NOT (clustered)
=> **Edible** Coverage: 288

Class: Poisonous; Number of Examples: 3,916

Rule 3:
Odor: NOT (almond anise none)
=> **Poisonous** Coverage: 3,796

Rule 4:
Ring-type: NOT (evanescent)
Spore-print-color: NOT (buff brown orange yellow)
Population: (abundant clustered several)
Habitat: NOT (woods paths urban)
=> **Poisonous** Coverage: 400

Rule 5:
Stalk-surface-below-ring: (Silky Scaly)
Population: (abundant clustered several)
=> **Poisonous** Coverage: 1,588

ACKNOWLEDGMENTS

Flora Tsai ably programmed the initial C++ version. I thank her for making it possible to run large problems early on. I am very grateful to Chid Apte and my manager Marshall Schor for their encouragement, collaboration on follow-on work, and kind assistance in preparing this paper. Other members of our Data Abstraction Research Group (Barry Rosen, Seema Prasad, and Jorge Lepre) at the Thomas J. Watson Research Laboratory are also gratefully acknowledged for the RAMP system, benchmarking, and discussions. Finally, I thank the reviewers, whose comments prompted me to revise the paper to be more self-contained and relevant to the readership.

REFERENCES

[1] R. Michalski, I. Mozetic, J. Hong, and N. Lavrac, "The Multi-Purpose Incremental Learning System AQ15 and Its Testing Application to Three Medical Domains," *Proc. AAAI*, pp. 1,041-1,045, 1986.

[2] P. Clark and T. Niblett, "The CN2 Induction Algorithm," *Machine Learning*, vol. 3, pp. 261-283, 1989.

[3] S. Clearwater and F. Provost, "RL4: A Tool for Knowledge-Based Induction," *Proc. IEEE Conf. Tools Artificial Intelligence*, pp. 24-30, 1990.

[4] S. Weiss and N. Indurkhya, "Reduced Complexity Rule Induction," *Proc. 12th IJCAI*, pp. 678-684, 1991.

[5] J. Quinlan, "Generating Production Rules From Decision Trees," *Proc. 10th IJCAI*, pp. 304-307, 1987.

[6] G. Pagallo, "Learning DNF by Decision Trees," *Proc. 11th IJCAI*, pp. 639-644, 1989.

[7] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. Warmuth, "Occam's Razor," *Information Processing Letters*, vol. 24, North Holland, pp. 377-380, 1987.

[8] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. Warmuth, "Learnability and the Vapnik-Chervonenkis Dimension," *J. ACM*, vol. 36, pp. 929-965, 1989.

[9] J. Rissanen, *Stochastic Complexity in Statistical Inquiry*. World Scientific Series in Computer Science, vol. 15, 1989.

[10] U. Fayyad and K. Irani, "What Should be Minimized in a Decision Tree," *Proc.* AAAI, pp. 749-754, 1990.

[11] J. Quinlan and R. Rivest, "Inferring Decision Trees Using Minimum Description Length Principle," *Information and Computation*, vol. 80, pp. 227-248, 1989.

[12] L. Breiman, J. Friedman, R. Olshen, and C. Stone, *Classification and Regression Trees*. Wadsworth, 1984.

[13] S. Weiss and C. Kulikowski, *Computer Systems That Learn*. Morgan Kaufmann, 1991.

[14] J. Dougherty, R. Kohavi, and M. Sahami, "Supervised and Unsupervised Discretization of Continuous Features," *Proc. ML*, 1995.

[15] S. Hong, "Use of Contextual Information for Feature Ranking and Discretization," *IEEE Trans. Knowledge and Data Eng.*, vol. 9, no. 5, pp. 718-730, Sept./Oct. 1997.

[16] S. Hong, R. Cain, and D. Ostapko, "MINI: A Heuristic Algorithm for Two-Level Logic Minimization," *IBM J. Research and Development*, vol. 18, no. 5, pp. 443-458, Sept. 1974; also appears in *Selected Papers on Logic Synthesis for Integrated Circuit Design*, R. Newton, ed., IEEE CS/Press, 1987.

[17] R. Brayton, G. Hachtel, C. McMullen, and A. Sangiovanni-Vincentelli, *Logic Minimization Algorithms for VLSI Synthesis*. Kluwer Academic Publishers, 1984.

[18] J. Holland, K.Holyoak, R. Nisbett, and P. Thorgard, *Induction: Process of Inference, Learning, and Discovery*. MIT Press, pp. 84-89, 1986.

[19] P. Smyth and R. Goodman, "An Information Theoretic Approach to Rule Induction from Databases," *IEEE Trans Knowledge and Data Eng.*, vol. 4, no. 4, pp. 301-316, Aug.1992.

[20] e, S. Hong, S. Prasad, and B. Rosen, "RAMP: Rules Abstraction for Modeling and Prediction," IBM Technical Report RC-20271, 1995.

[21] S. Hong and D. Ostapko, "A Practical Approach to Two Level Minimization of Multi-Valued Logic," *Proc. Int'l Symp. Multiple Valued Logic*, May 1974

[22] C. Apte and S. Hong, "Predicting Equity Returns from Securities Data with Minimal Rule Generation," *Knowledge Discovery in Databases*, U. Fayyad, ed. AAAI/MIT Press, 1995.

[23] *Machine Learning: An Artificial Intelligence Approach*, chapters 3-5, R. Michalski, J. Carbonell, and T. Mitchell, eds. Tioga Press, 1983.

[24] *Machine Learning, Neural, and Statistical Classification*, D. Michie, D. Spiegelhalter, and C. Taylor, eds. Ellis Horwood, 1994.

[25] M. Mehta, J. Rissanen, and R. Agrawal, "MDL-based Decision Tree Pruning," *Proc. KDDM*, Aug. 1995.

Se June Hong received his BSc degree from Seoul National University in 1965, and his MS and PhD degrees in electrical engineering from the University of Illinois at Champaign-Urbana in 1967 and 1969, respectively. He joined the IBM Poughkeepsie (New York) Laboratory, working in the areas of fault-tolerant computing and design automation. He joined the IBM Thomas J. Watson Research Center at Yorktown Heights, New York, in 1978. He served as a senior manager in the Mathematical Sciences Department from 1982 to 1991, responsible for projects in natural languages, computer algebra, computer-based education, and knowledge-based systems. He was on an assignment as a senior staff member of the office of vice president of technical plans and controls at Yorktown Heights from June 1991 to October 1992, and then returned to the Mathematical Sciences Department's Computer Science Principles Group as a senior manager until 1994. He is currently a research staff member, working on data mining technology.

During the academic year of 1974-1975, Dr. Hong was a visiting associate professor at the University of Illinois at Urbana-Champaign. He was a visiting professor at the Korea Advanced Institute of Science and Technology (KAIST) in October 1980. He received an honorable mention award as an Outstanding Young Electrical Engineer from Etta Kappa Nu in 1975, and has received three outstanding innovations awards and one outstanding technical achievement award from IBM. He served as an IEEE Computer Society distinguished visitor (1972-1974); guest editor of a special issue of *IEEE Transactions on Computers* on reliable and fault tolerance (July 1982); a member of the ad-hoc visiting team for the IEEE Engineering Accreditation Board (1979-1984); a member of the Edison Medal Awards Committee (1984-1986); and guest editor of a special issue of *Computer* magazine on expert systems for engineering applications (July 1986). He was chair of the Program Committee of the IEEE Conference on Artificial Intelligence Applications (CAIA 90) and general chair in 1991. He chaired the IEEE Emanuel Piore Award Committee in 1991-1992; was a member of the Standing Organizing Committee of the Pacific Rim International Conference on AI (PRICAI) in 1990-1996; and he chaired PRICAI's Standing Scientific Committee in 1992-1994. He is a fellow of the IEEE and a member of the ACM, AAAI, KSEA, and Sigma Xi. He became a foreign member of the Korean National Academy of Science in 1997.