A Machine Learning Approach to Diagnosis and Control with Applications in Semiconductor Manufacturing

Keki B. Irani **Jie Cheng** **Usama M. Fayyad** **Zhaogang Qian**[†]

Artificial Intelligence Laboratory
Electrical Engineering and Computer Science Department
The University of Michigan
Ann Arbor, MI 48109-2122

1 Introduction and Motivation

In this chapter, we focus on a machine learning approach to the problem of automating the diagnosis and process control in semiconductor manufacturing. The reactive ion etching (RIE) has been used as a primary vehicle of application. Automation promises cost-effectiveness, reliability, predictability, and accuracy. So far, only fixed simple tasks in manufacturing have been automated. Automation of more complex tasks, currently requiring intelligent decision making or problem solving on the part of humans, is a much more difficult task.

One of the goals of Artificial Intelligence (AI) research is to provide mechanisms for emulating human decision-making and problem solving capabilities, using computer programs. The first AI attempts at such systems appeared as part of the technology known as "expert systems". Expert systems are intended to provide the means of encoding human knowledge about a specific task in terms of situation-action rules. The idea is that if such systems are endowed with sufficient knowledge of the task at hand, they may be able to emulate human expert behavior in most, if not all, situations that arise during task execution.

Serious difficulties arose that hindered the development of successful expert system applications. The first such difficulty is known as the "knowledge acquisition bottleneck" [9]. Human experts find it difficult to express their knowledge, or explain their actions, in terms of concise situation-action rules. If pressed to do so, they typically produce rules that are incorrect, or that have many exceptions. The articulation of specific intuitive knowledge into deterministic rules is a difficult, sometimes unrealistic, problem for human experts. Interviewing domain experts to extract such knowledge is also an expensive process demanding time from experts and knowledge engineers.

A second problem arises in a different situation: What if a task is not well-understood, even by the experts in that area? An example of this situation is man-

[0†] The author is currently with Artificial Intelligence Services, Electronic Data Systems, 5555 New King Street, Troy, MI 48007-1079.

ifested in our experience with the automation of the RIE process in semiconductor manufacturing. In such domains, abundant data are available from the experiments conducted, or items produced. However, models that relate output variables to controlling (input) variables are not available. Experts strongly rely on familiarity with the data and on "intuitive" knowledge of such a domain. How would one go about constructing an expert system for such a domain?

The machine learning approach to circumventing the aforementioned hurdles calls for extracting classification rules from data directly. Rather than require that a domain expert provide domain knowledge, the learning algorithm attempts to discover, or induce, rules that emulate expert decisions in different circumstances by observing examples of expert-executed tasks.

In addition to the motivations listed above, two other reasons exist for the need of a machine learning approach. The first is the growing number of large databases that store instances of diagnostic tasks. Such data is typically accessed by keyword or condition lookup. As the size of the database grows, such an approach becomes less effective. Suppose an expert needs to look up cases similar to a case being diagnosed. A query may easily return hundreds of matches. A method for determining relevant conditions automatically would be needed in this case.

Another motivation is the evolution of complex systems that have an error detection capability. Communication networks are an example. Faults are detectable by the network hardware. Several thousand faults may occur during a day. To debug such a network, a human would need to sift through large amounts of data in search of an explanation. An automated capability of deriving conditions under which certain faults occur may be of great help to the engineer in uncovering underlying problems in the hardware.

There are several approaches to inducing diagnostic rules from data. In this paper we do not cover all the details, nor do we review the relevant machine learning literature. We restrict our discussion to briefly presenting the problem and its complexity, and then we focus our attention on the induction of decision trees as an efficient solution. We illustrate this discussion with simple examples. We then briefly motivate and outline our algorithm (GID3) for inducing decision trees.

The second part of the paper provides some details of several industrial applications in semiconductor manufacturing domains for which GID3 and two of its extensions were used and were found useful by the process and knowledge engineers. The extensions to the basic decision tree algorithm were in response to two problems that we faced in our dealings with industrial data. The typical assumption is that large amounts of data are available when machine learning is to be applied. However, there are cases when experiments may be very expensive. In such cases, training data are limited. We developed a system, KARSM, that uses the Response Surface Methodology (RSM), coupled with GID3, to generate rules under such conditions. Another problem we face with industrial data is that in some processes the data may be noisy. Human recording errors, limited sensor resolution, or sensor or equipment reliability

Table 1: A Simple Training Set of Examples.

example	Selectivity	Δ line width	class
e-1	normal	normal	*power is high*
e-2	normal	high	*power is low*
e-3	high	high	*power is low*
e-4	high	low	*power is high*
e-5	low	normal	*flow rate is low*
e-6	low	high	*flow rate is low*

problems introduce inaccuracies in the values of the attributes. We developed a system, RIST, that utilizes statistically robust techniques along with GID3 to deal with the noise problem.

2 The Machine Learning Approach

The machine learning approach calls for learning the relation between the input variables and the output variable directly from training data. A training example consists of a description of a situation and the action performed by the expert in that situation. The situation is described in terms of a set of *attributes.* An attribute may be *continuous* (numerical) or *discrete* (nominal). For example, a nominal attribute may be *shape* with values { *square, triangle, circle*}. An example of a continuous attribute is pressure or temperature. The action associated with the situation, the *class* to which the example belongs, is a specification of one of a fixed set of allowed actions. The class of each training example is typically determined by a human expert during normal task execution. Example actions may be *raise temperature, decrease pressure, accept batch,...* The goal of the learning program is to derive conditions, expressed in terms of the attributes, that are predictive of the classes. Such rules may then be used by an expert system to classify future examples. Of course, the quality of the rules depends on the validity of the conditions chosen to predict each action.

A training example is therefore a list of the values of all the attributes along with the class to which the example belongs. Assume there are m attributes $A_1, \ldots, A_m$, k classes $C_1, \ldots, C_k$. A training example is an $m+1$-tuple $\langle b_1, b_2, \ldots, b_m; C_j \rangle$, where each b_i is one of the values of the attribute A_i: $\{a_{i1}, \ldots, a_{ir_i}\}$, and C_j is one of the k classes. A rule for predicting some class C_j consists of a specification of the values of one or more attributes on the left hand side and that class on the right hand side.

As an example, consider the simplified small example set shown in Table 1. This set consists of six examples e-1 through e-6. There are two attributes: *selectivity* and Δ *line width.* The attributes can take the values *low, normal,* and *high.* There are three classes: *flow rate is high, power is low,* and *power is high.* A simple rule consistent with these examples may be:

IF (Selectivity = low) THEN *Flow rate is low*

Note that this is only an illustrative simplification. Typically, the number of examples of a meaningful training set is at least in the hundreds, while the number of attributes is usually in the tens.

Note that the rule shown above uses a very simplified set of conditions. Each condition is a simple test of equality on a single attribute. Even with such a simple language, the problem of discovering rules from data is very difficult. Assume that there are m attributes as described above. and that on average an attribute takes on one of r values. There are $k \cdot (r+1)^m$ possible rules for predicting the k classes. It is computationally infeasible for a program to explore the space of all possible classification rules. In general, the problem of determining the minimal set of rules that cover a training set is NP-hard. It is therefore likely that a heuristic solution to the problem is the only computationally feasible one.

2.1 Inducing Decision Trees from Training Examples

A particularly efficient method for extracting rules from data is to generate a decision tree [2, 14]. A decision tree consists of nodes that are tests on the attributes. The outgoing branches of a node correspond to all the possible outcomes of the test at the node. The examples at a node in the tree are thus *partitioned* along the branches and each child node gets its corresponding subset of examples. A popular algorithm for generating decision trees is Quinlan's ID3 [14], now commercially available.

ID3 starts by placing all the training examples at the root node of the tree. An attribute is then chosen to partition the data. For each value of the chosen attribute, a branch is created and the corresponding subset of examples that have the attribute value specified by the branch are moved to the newly created child node. The algorithm is then applied recursively to each child node until either all examples at a node are of one class, or all the examples at that node have the same values for all the attributes. An example decision tree generated by ID3 for the sample data set given in Table 1 is shown in Figure 1.

Every leaf in the decision tree represents a classification rule. The path from the root of the tree to a leaf determines the conditions of the corresponding rule. The class at the leaf represents the rule's action.

Note that the critical decision in such a top-down decision tree generation algorithm is the choice of attribute at a node. The attribute selection is based on minimizing an information entropy measure applied to the examples at a node. The measure favors attributes that result in partitioning the data into subsets that have low class entropy. A subset of data has low class entropy when the majority of examples in it belong to a single class. The algorithm basically chooses the attribute that provides the locally maximum degree of discrimination between classes. Let us examine this attribute selection method more carefully.

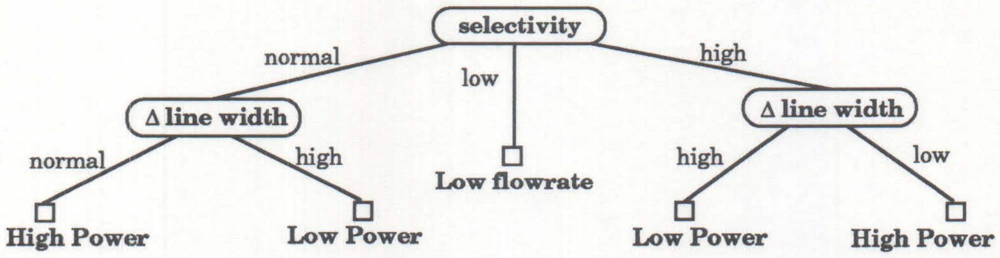

Figure 1: The Decision Tree Generated by ID3.

Let A be a set of m attributes $\{A_1, A_2, \ldots, A_m\}$, let C be a set of k classes $\{C_1, C_2, \ldots, C_k\}$, and let S be a set of examples at some node. The set of possible values for an attribute $A_i \in A$ is referred to as $Range(A_i)$. Each example in S is an $(m+1)$-tuple of the form: $\langle V_1, V_2, \ldots V_m; C_k \rangle$, where $V_i \in Range(A_i), i = 1, \ldots, m$, and $C_k \in C$ is the class of the example. Define P_{S,C_j}, the probability of occurence of class $C_j \in C$ in a set S of examples, to be the proportion of examples in S that are in class C_j: $P_{S,C_j} = \frac{|\{e \in S | Class(e) = C_j\}|}{|S|}$. The first notion that needs to be defined is a measure of class uncertainty in a set S of examples. This is achieved by using an uncertainty or information entropy measure: The *class information entropy* in the set S of examples is defined to be:

$$\mathrm{Ent}(S) = -\sum_{j=1}^{k} P_{S,C_j} \log_2(P_{S,C_j}).$$

Note that $\mathrm{Ent}(S)$ is minimum when all examples in S are in one class. It is maximum when all k classes are equally likely in S and its maximum value then is $\log_2(k)$. Thus, $0 \leq \mathrm{Ent}(S) \leq \log_2(k)$. When the entropy is minimum, there is only one class in the set S, hence one's uncertainty regarding the class of an example from S is minimum. On the other hand, if all classes in S are equally likely, it is most difficult to correctly "guess" the class of an example from S, whence, uncertainty is maximum.

ID3 measures the "goodness" of a partition on S by the average class entropy of its component blocks. It favours a partition of S that results in subsets in which the examples are distributed "less randomly" over the possible classes, i.e., subsets for which the class uncertainty is small. To choose the attribute that would best achieve this, for each attribute A_i that takes more than one value for the examples in S, ID3 partitions S into the sets S_{ij} consisting of all examples in S having value V_j for attribute A_i: $S_{ij} = \{e \in S | A_i = V_j \text{ for } e\}$. The weighted sum of of the individual class entropies of the subsets in the partition on S induced by the attribute A_i is referred to as *the information entropy of attribute A_i with respect to the set S*:

$$E(A_i, S) = \sum_{V_j \in Range(A_i)} \frac{|S_{ij}|}{|S|} \mathrm{Ent}(S_{ij}) \qquad (1)$$

Hence, the *information gain* of attribute A_i is defined to be the decrease in entropy due to the partition induced by A_i:

$$Gain(A_i, S) = \mathrm{Ent}(S) - E(A_i, S). \tag{2}$$

It is easy to show that $0 \leq Gain(A_i, S) \leq \log_2(k)$.

For further partitioning of a node S in the tree, the ID3 algorithm selects the attribute A_i for which the information gain is maximized. Note that for any given S, the value of $\mathrm{Ent}(S)$ is constant. Thus, ID3 selects the attribute that induces the partition having the least average class information entropy.

One problem with the information entropy minimization heuristic introduced above is that the formula of Equation 2 is biased in favour of attributes with a larger number of values[14]. A new measure to compensate for this bias is introduced. For an attribute $A_i \in A$, and a set S of examples partitioned into the subsets S_{ij} consisting of all examples in S that have value V_j for the attribute A_i,

$$IV(A_i, S) = -\sum_{V_j \in Range(A_i)} \frac{|S_{ij}|}{|S|} \log_2\left(\frac{|S_{ij}|}{|S|}\right).$$

$IV(A_i, S)$ measures the degree of randomness of the distribution of the examples in S over the values of A_i. Note that it does not take into account what the classes of these examples are. The gain formula of equation 2 is modified to be the *Gain Ratio*:

$$GainR(A_i, S) = \frac{\mathrm{Ent}(S) - E(A_i, S)}{IV(A_i, S)} = \frac{Gain(A_i, S)}{IV(A_i, S)}.$$

Although this correction has its problems, especially when IV is very small, it seems to work well in ID3 on the average. From our experiments, ID3 generally produced better trees with the IV measure than without. We refer to this version of ID3 as ID3-IV.

2.2 Problems with the ID3 Approach

ID3 is essentially employing a heuristic, hill-climbing, non-backtracking search through the space of possible decision trees. Thus, weaknesses in the ID3 algorithm may cause it to "miss" better decision trees for the same data.

We discuss what we precisely mean by "better" trees in [5]. For the purposes of this paper, we simply say that one decision tree is better than another decision tree for the same training data set, if the former has a smaller number of leaves. The other important performance measure is the error rate of the tree on classifying examples outside the training set. For other performance measures used see [3, 5, 8]. This section addresses some of the problems *inherent* in the ID3 approach that cause it to generate *overspecialized* decision trees.

Perhaps the most pronounced of the problems is the **irrelevant values problem.** When ID3 chooses an attribute for branching out of a node, it creates a branch for each attribute value that appears in the examples. Some of the values of that attribute may be relevant to the classification, yet the rest may not be. The subtrees generated by such irrelevant values will result in overspecialized classification rules—rules that check for unnecessary or irrelevant preconditions. Consider the following example. Assume we are in a world where we are trying to classify objects into the classes *nutritious, useful, poisonous, dreadful,...* Suppose one of the attributes is colour. Assuming that an object whose colour is either *blue* or *red* may, under certain other conditions, be *poisonous.* Suppose that colour is not relevant to any other classification; testing on the individual values of the other colours { *green, yellow,* ...} does not make any sense. The only relevant information that the colour *green* contains is the fact that the colour *is not blue or red.* If ID3 used colour for branching, however, some rules for classifying an object into the class *nutritious* may actually test for the irrelevant fact of whether an object's colour is, say, *yellow.*

The problem of irrelevant values also leads to the **problem of reduced training data**. Since the data are unnecessarily partitioned along the irrelevant values of an attribute, each of the subtrees generated under an irrelevant branch will be based on a subset of the training data that was *unnecessarily* reduced. Consequently, the quality of subsequent choices of attributes made in each subtree are likely to be of a lower quality, leading to an overall worse tree. The problem reduced training data is an important problem in decision tree generation, in general.

Another problem related to the problem of irrelevant values is the **problem of missing branches.** Missing branches essentially represent a reduction in the inductive capacity of the tree. They are due to the fact that some of the reduced subsets at the non-leaf nodes do not necessarily contain examples of every possible value of the branching attribute. The following example illustrates this problem: Consider the ID3 tree of Figure 1 generated for the data set of Table 1. Assume it is the case that values *high* and *normal* for attribute **selectivity** do not have any particular relevance to the classes, however, the value *low* is predictive of the class: *low flowrate.* Consider two possible unclassified examples which are to be classified by the tree of Figure 1.

$$\begin{aligned} e &: (\text{Selectivity} = \mathit{low}) \quad\;\;\; \& \ (\Delta \text{ line width} = \mathit{low}) \\ e' &: (\text{Selectivity} = \mathit{normal}) \ \& \ (\Delta \text{ line width} = \mathit{low}) \end{aligned}$$

Both e and e' have combinations of attribute values that did not appear in the training set. However, the tree readily classifies e as an example of *low flow rate* . e' fails to be classified by the tree because the subtree under the normal selectivity branch has no branch for *low* Δ*line width.*

In summary, the irrelevant attribute values cause the occurence of other serious problems in ID3-generated tree. The final effect is that these problems collectively result in overspecialized trees. If our goal is to find a classifier that is as general as possible, then these problems should be avoided whenever possible.

2.3 The GID3 Algorithm

To avoid some of the problems described above, we developed the Generalized ID3 (GID3) algorithm. We generalized the ID3 algorithm so that it does not necessarily branch on each value of the chosen attribute. GID3 can branch on arbitrary individual values of an attribute and "lump" the rest of the values in a single *default branch.* Unlike the other branches of the tree which represent a single value, the default branch represents a subset of values of an attribute. Unnecessary subdivision of the data may thus be reduced.

The tendency of GID3 to branch on all or some of the values of an attribute is controlled by a user-specified parameter, TL. For a set S of examples, GID3 first evaluates each attribute value pair separately. Suppose an attribute $A_i \in A$ has values in its range: $Range(A_i) = \{V_1, \ldots, V_r\}$. Then an attribute value pair, $\langle A_i, V_j\rangle$ for $V_j \in Range(A_i)$, partitions the set S into $S_{A_i=V_j}$, the subset of examples in S that have $A_i = V_j$, and $S_{A_i \neq V_j}$, the remaining examples in S. We can therefore evaluate the entropy of the partition induced by the pair $\langle A_i, V_j\rangle$ as we did for an attribute:

$$E(\langle A_i, V_j\rangle, S) = \frac{|S_{A_i=V_j}|}{|S|}\mathrm{Ent}(S_{A_i=V_j}) + \frac{|S_{A_i\neq V_j}|}{|S|}\mathrm{Ent}(S_{A_i\neq V_j}) \tag{3}$$

Thus, for each attribute-value pair we can define a "temporary" attribute that takes on the values { True, False } for examples in S. We can thus find the "best" attribute-value pair, the pair having the minimum entropy or maximum gain:

$$Gain(\langle A_i, V_j\rangle, S) = \mathrm{Ent}(S) - E(\langle A_i, V_j\rangle, S) \tag{4}$$

We multiply the best gain by the tolerance level (TL) parameter to get a threshold gain. All pairs whose gain is at least as high as this threshold gain are considered relevant, otherwise they are not. Thus, only a subset of the values of any given attribute may be considered relevant.

For each attribute, this evaluation allows us to define a corresponding new attribute whose values are a subset of the original attribute values. We refer to the new attribute AP_i that has a subset of the values of A_i as a *phantom attribute.* The term phantom refers to the fact that it is defined only at the evaluation stage. Once an attribute is chosen for branching, the attribute A_i is restored to its original state. The algorithm for deciding which of the values of A_i are relevant is given below:

Begin Algorithm: GID3 Phantomization

Inputs: TL: user-determined parameter in range $[0, 1]$.
  S: a set of training examples.
  A: a set of discrete attributes defined over S.

```
1. For each attribute-value pair $\langle A_i, V_j \rangle$ appearing in the examples[1] in $S$, create
   a binary-valued "temporary attribute" $\langle A_i = V_j \rangle$ which takes the value TRUE
   for examples that have value $V_j$ for the attribute $A_i$, and the value FALSE for
   all other examples in $S$.
2. For each binary-valued temporary attribute $\langle A_i = V_j \rangle$ defined in step 1, $E(\langle A_i =
   V_j \rangle, S)$ as defined in equation 4 is computed. Note that $Range(\langle A_i = V_j \rangle) =
   \{TRUE, FALSE\}$.
3. Let MAX-Gain be the maximum of the values computed in step 2 above.
4. Let Threshold-G = TL $\times$ MAX-Gain.
5. Construct a set of *phantom attributes*, $AP$, as follows:
   (a) $AP \leftarrow \emptyset$.
   (b) **For** each attribute $A_i \in A$, for which more than one value appears in the
       examples in $S$, **Do**
       i. Default $\leftarrow \emptyset$; $R \leftarrow \emptyset$.
       ii. **For** each temporary attribute $\langle A_i = V_j \rangle$ defined in step 1 **Do**
                IF $Gain(\langle A_i = V_j \rangle, S) \geq$ Threshold-G
                   THEN $R \leftarrow R \cup \{V_j\}$
                   ELSE Default $\leftarrow$ Default $\cup \{V_j\}$
       iii. IF $R \neq \emptyset$ THEN $AP \leftarrow AP \cup \{AP_i\}$
            where $AP_i$ is the *phantom attribute* corresponding to $A_i$,
            and $Range(AP_i) = R \cup \{$ Default $\}$.
6. Conduct the ID3-IV algorithm to choose an attribute out of the set of phantom
   attributes $AP$ constructed in 5.
```

End Algorithm.

In order to avoid branching on irrelevant values of the attribute, only the values that appear to be relevant, according to the information measure, may potentially be branched on. All other values are lumped together in one default value for the attribute. The tolerance level (TL) is user determined. TL specifies the degree of tolerance for deviation of the information gain of an attribute-value pair from the maximal gain value over all pairs (see algorithm for the definitions of TL and the set of *phantom attributes* AP.) Attribute values that fall outside this tolerance range are treated as a single "default" value. Note that if all values of an attribute fall outside the tolerance range, no corresponding phantom attribute is constructed. Thus $|AP| \leq |A|$. Furthermore, $|Range(AP_i)| \leq |Range(A_i)|$ for all $AP_i \in AP$.

Setting TL= 0 results in behaviour that exactly matches that of ID3 since a pair's gain is allowed to be as low as 0% of the best gain; thus all pairs pass the filtering

[1]Values V_j appearing on the path from the root to the node representing S are excluded from consideration.

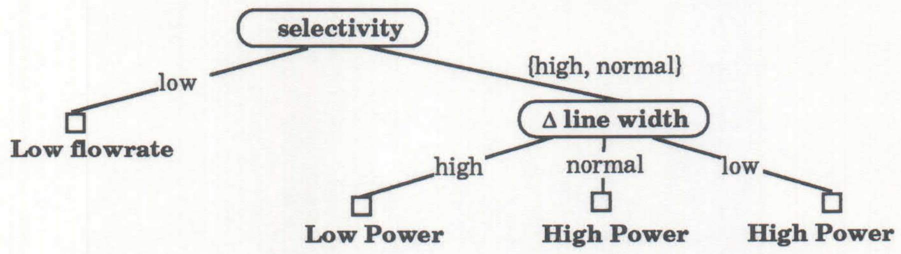

Figure 2: The Decision Tree Generated by GID3.

stage. The other extreme occurs when TL= 1.0, where only the attribute-value pairs whose gain measure *exactly matches* the best gain may potentially be branched on; the rest are grouped together in a "default" branch. In most cases, this setting is expected to result in a binary decision tree—branching on one value with the rest of the values on the default branch. As TL varies from 0.0 to 1.0, the algorithm generates different trees. For a given set of data, we claim the existence of a setting for TL that results in the generation of a "better" decision tree than that of ID3.

The issue of determining the proper setting of TL for a given set of data set is beyond the scope of this presentation. Our initial motivation for developing the GID3 algorithm was to empirically verify our hypothesis that for some TL setting, GID3 will outperform ID3. We have experimentally verified that, indeed, there is always a TL setting which allows GID3 to outperform ID3 [3, 6, 8]. We have since formulated a new algorithm, GID3*, which does not depend on the user-specified parameter TL [6]. Extensive empirical tests, over many synthetic and industrial data sets from several domains, have demonstrated that GID3* outperforms GID3 for any of the TL settings tested. A description of GID3* is beyond the scope of this paper. Furthermore, the applications described in this paper were initially performed using GID3 with TL set by the user.

The algorithm described above is obviously designed to avoid the problem of irrelevant values. As a side effect, the procedure will also generate trees that are less likely to suffer from missing branches as shown in the following example. Figure 2 shows the tree that GID3 would generate for the data set of Table 1. Recall that the ID3 tree for this data set (shown in Figure 1) suffered from the problem of missing branches as illustrated by the example e'. Note that both examples e and e' above are now classifiable by the GID3 tree, indicating a higher inductive power for the tree generated by the new approach.

Finally, we have not discussed how continuous-valued attributes are handled in GID3. A continuous-valued attribute is *discretized* by quantizing its range into intervals. The algorithm for discretizing the range into two intervals is described in detail in [7]. This is done by selecting a value in the range of the attribute and cutting the range in two at that point. We have generalized the algorithm so that multiple intervals can be obtained rather than just two. This enables GID3 and GID3* to discover even better decision trees. For details, the reader is referred to [6, 8]. A new discretization is derived for each node depending on the data set at that node. Once a continuous-valued attribute has been discretized, it is subsequently treated as a discrete attribute by the learning algorithm. Hence, only a subset of the derived intervals may be selected for branching.

We now turn our attention to the application of GID3 to problems in the domain of semiconductor manufacturing. Interested readers are referred to [8, 3] for detailed accounts of the ID3 and GID3 algorithms, the attribute selection criterion, the weaknesses of the ID3 approach, and various performance measures to evaluate the quality of the resulting trees.

2.4 Applications of GID3 in Semiconductor Manufacturing

In this section we discuss several applications of the GID3 algorithm to semiconductor manufacturing domains. Most of these domains involve the reactive ion etching (RIE) process. The RIE process is a wafer etching process that promises increased precision and higher device density. It has been targeted for automation by the Semiconductor Research Corporation (SRC), a consortium of major U.S. companies in semiconductor manufacturing. One of the steps necessary for automation is the development of expert systems that determine process parameter settings based on given output constraints. The problem is that the process is not well-understood and no satisfactory methods for determining proper control settings exist.

To illustrate the types of industrial tasks to which GID3 can be applied, we describe application tasks from three categories: RIE process diagnosis, RIE process optimization, and an emitter piloting application.

2.4.1 Process Diagnosis and Optimization

The goal of process diagnosis is to derive rules for diagnosing faults by deciding which process parameters are not correctly set. We mention two process diagnosis applications. The first project's goal is to acquire a set of RIE process diagnostic rules from a collection of production log data that contain fault inspection results by process engineers where the specified pattern was not etched correctly in the metal on a wafer. We discuss this application in detail when we discuss an extension of GID3 to make it more robust in Section 4.

The second project was aimed at identifying relationships between RIE process problems, such as reduction in yield, and corresponding process parameters including the flow rate of each gas component and the chamber pressure for different etching steps. This data set was derived by regression analysis which statistically identifies the geometric patterns such as length, corners and gaps responsible for the yield loss. Classes consisted of dominating defect patterns. The details of this project with

Westinghouse and the National Bureau od Standards are discussed in [11]. The rules derived by GID3 were used to analyze and understand the process behavior.

In the process optimization category, the project targeted deriving rules for dealing with situations where the operating point drifts away from the optimal operating point in the parameter space. We discuss this application in detail in Section 5 in the context of combining the GID3 approach with the response surface methodology (RSM) for process modelling.

2.4.2 An Emitter Piloting Application

Finally, we briefly discuss our effort on a project aimed at facilitating the knowledge acquisition effort for the development of an emitter piloting advisory expert system at Harris Corporation. GID3 has been applied successfully to acquire knowledge for minimizing steps in emitter piloting dispositions. Emitter piloting is a process of tuning integrated circuits printed in wafer so that device specification can be satisfied. This task is typically carried out by a human operator. The initial cycle time is determined by experience and cycle time adjustment is then guided by the measurement of two device parameters. If the values of the two parameters fall in their respective desired ranges, the cycle time is accepted for batch tuning and is called the "shooting time". Otherwise, it is either increased or decreased to bring parameter values within desirable ranges. The number of steps needed before success in such a process is greatly affected by operator experience. Such experience is very valuable but is very difficult to encode in rules. The purpose of the project was to collect all sequences of trials conducted and to use the machine learning approach to attempt to extract the knowledge underlying the actions of human operators. See [16] for further details of this domain.

The raw data used in knowledge acquisition is composed of numerous experiment data logs, each of which consists of sequences of cycle time adjustments targeting one shooting time. For every trial in each sequence, the cycle time used and two parameter measurement values were recorded. GID3 was used to learn rules for jumping to a shooting time from an arbitrary cycle time by letting each data point be the condition under which certain adjustments can be made to achieve a certain shooting time. The difference between the current cycle time and the actual shooting time for each example was taken to be the predetermined class. The rules induced by GID3 were evaluated by an expert and were deemed satisfactory. In this project GID3 was used as a knowledge acquisition tool to gather rules for incorporation into an expert system developed by Harris Semiconductor.

3 Dealing with Industrial Data Problems

For the rest of this chapter, we focus on two special problems encountered in industrial applications and the solutions we devised to combat them. The problems are:

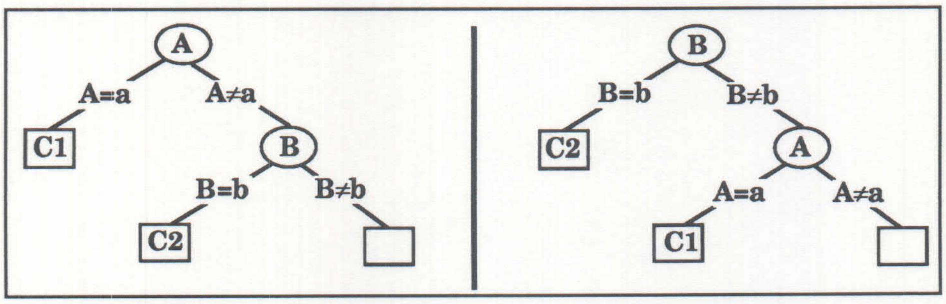

Figure 3: Two Decision Trees Representing the Two Rules.

Noisy Data: attribute values may be erroneous due to human recording errors, imperfect sensor repeatabilty, or defects in process equipment or sensors.

Limited Training Data: the training data may be small in size and conducting more experiments may be too costly.

Along with each presentation of the extension to GID3 developed to deal with the respective problems, we present a detailed account of a related application in RIE.

The extension developed to deal with noisy data is also capable of surmounting some of the limitations imposed by a decision tree approach to the learning problem. To illustrate this, consider the following example: Assume that a set of examples can be classified by the following two rules:

If $(A = a)$ **Then C1** **If** $(B = b)$ **then C2**

where A and B are attributes and **C1** and **C2** are classes. Although both rules have only one condition, any equivalent decision tree for this set of rules will necessarily introduce irrelevant conditions. Figure 3 shows two possible decision trees that can represent these two rules. The rules that we can extract from the first decision tree, labeled Rule1 and Rule2, and the two extracted from the second tree, labeled Rule3 and Rule4, are given below:

Rule1: If $(A = a)$ Then C1
Rule2: If $(A \neq a)$ and $(B = b)$ Then C2

Rule3: If $(B = b)$ Then C2
Rule4: If $(B \neq b)$ and $(A = a)$ Then C1

Obviously, the condition $(A \neq a)$ in Rule2 and the condition $(B \neq b)$ in Rule4 are irrelevant to the classification. However, their presence is mandated by the fact that the set of rules must come from a single decision tree.

By overcoming the single tree representation restriction, one may derive more general and more compact rules. We shall show how this is attained, without sacrificing the efficiency afforded by the decision tree based approach, in the next section.

4 Dealing with Noisy Data

Empirical learning algorithms are typically sensitive to the presence of noise because they rely solely on data to discover rules. Typically, they are not intended to have access to special domain knowledge to guide their decisions. Noise in a training data set may cause irrelevant rule conditions to be selected. The solution we devised combats noise in two ways: (1) statistical evaluation (pruning) to identify and remove irrelevant conditions from rules, and (2) random sampling of multiple training sets and selection of statistically significant rules from the trees generated for these training sets.

As described in previous sections, GID3 and GID3* can efficiently induce classification rules from a set of data. This efficiency is mainly due to the decision tree representation and the powerful hill-climbing heuristic. Because of its heuristic nature and its non-backtracking search method, however, GID3 cannot avoid selecting irrelevant attributes during decision tree generation. This problem becomes more serious when data sets containing noise are used as training data sets for GID3.

The sources contributing to the above problem of the GID3 approach are identified to be (1) its persistence in achieving a complete and perfect classification (if possible) while employing a simple hill-climbing hueristic, and (2) its decision tree representation of classification knowledge. It is not surprising to see that the very features of GID3 that make it efficient also cause problems.

To obtain better decision trees, two approaches are possible. The first approach is to improve the decision tree generation algorithm, while the second is to apply a robust statistical testing method to iteratively evaluate and prune the rules that are derived from decision trees. The first approach was described in earlier sections and consists of developing the GID3 and GID3* algorithms for generating better decision trees. In this section, we focus on the second approach which is attractive because it is independent of the way in which rules are generated. The statistical testing method used in this approach is Fisher's Exact Test [10]. This approach is implemented in a software package, RIST (Rule Induction and Statistical Testing).

RIST extends GID3 by introducing three important methods, (1) a control strategy that calls upon GID3 to iteratively induce decision trees from randomly sampled subsets of a training data set, (2) statistical criteria for testing and pruning conditions of rules extracted from decision trees, and (3) a heuristic method for selecting a minimal set of rules from a pool of statistically-good rules satisfying certain constraints.

By introducing the second method, RIST is able to derive a set of good rules in the sense that all conditions of these rules are known to be statistically relevant to the rules' classifications. While the second method tends to reduce the coverage of a set of rules because of possible elimination of some rules, the first method can enlarge the coverage and make it possible to derive a diverse set of rules that might not be available from any single decision tree. The first method can be perceived as a trade-off of two extreme alternatives: generating a set of rules from a single decision

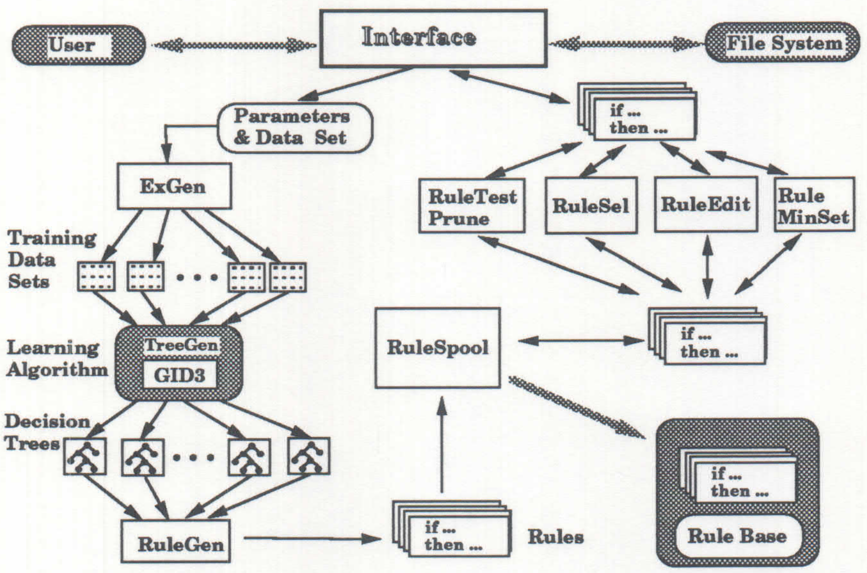

Figure 4: Data Flow Diagram of the RIST System.

tree, or exhaustively searching for all possible rules. By combining rules derived from multiple decision trees, however, it is possible that redundant rules are generated. The third method is therefore introduced to search for a compact set of rules without compromising rule classification. Figure 4 illustrates this description of the RIST program. In the following sections, we cover the algorithm in more detail.

4.1 The RIST program

We briefly describe the nine components of RIST. *Interface* is used for communication between the user and various parts of the program. It provides a set of menus to enable the user to easily inquire, access, and manipulate the information stored and processed by the program, including examples, decision trees and rules. *ExGen* samples random subsets of examples out of a training data set. *TreeGen* is a decision tree induction program (we use GID3.) *RuleGen* extracts rules from a given decision tree. For each rule, it detects and removes redundant conditions. *RuleSpool* merges rules that are derived from different sources such as decision trees or users. *RuleTestPrune* tests and prunes rule conditions or rules based on a statistical measure. Details of this component will be given later in this section. *RuleSel* enables the user to examine or change a set of rules by posing queries on rule conditions, classes or test measures. *RuleEdit* lets the user manipulate the existing rules or create new rules manually. These rules can then be mixed with the rules already in *RuleSpool*. *RuleMinSet* can be called to produce a minimal set of rules which satisfy certain constraints.

While in GID3 the focus of attention is how to derive a (single) decision tree to best encapsulate the classification knowledge, in RIST, a set of rules constitutes the end product. These rules can come from different decision trees and can even come from users. The rationale behind this shift of emphasis is that rules are more modular and are therefore more managable and easy to extend. In addition, a set of rules can potentially provide a greater coverage of any task domain than a single decision tree can.

As shown in Figure 4, RIST starts with a set of training examples drawn from a problem domain. These examples are then randomly sampled by $ExGen$ to provide many training data sets. Decision trees are generated, sequentially or in parallel, by $TreeGen$, and the trees are then converted into a set of rules by $RuleGen$ and merged into a pool of rules (with redundancy removed) by *RuleSpool*. The rules in the pool are tested and pruned by $RuleTestPrune$ according to a statistical measure. Finally, the user can participate by querying rules, manipulating rules, re-testing rules with progressively-tighter test criteria or constraints, and/or selecting a minimal set of rules satisfying certain constraints. These user options are supported by $RuleSel$, $RuleEdit$, $RuleTestPrune$, and $RuleSetMin$, respectively.

4.1.1 Rule Testing and Pruning

As its name suggests, $RuleTestPrune$ performs a test on a set of rules, and based on the test results, it prunes rule conditions or rules themselves according to users' requirements and constraints. The core of this RIST component is the Fisher's exact test. This test is used to determine the statistical confidence with which we can reject the hypothesis that a rule condition is irrelevant to the rule's class for any example that satisfies all other preconditions of the rule. It is important to note that, it is possible for a condition to actually be relevant while a belief of its relevance is not strongly supported by the test. The approach taken by RIST is conservative, namely, it judges a rule precondition to be irrelevant if the test does not strongly support the rejection of the hypothesis that it is irrelevant. The rule conditions that are judged to be irrelevant are pruned. This sometimes leads to the removal of the entire rule.

In addition to testing every individual precondition of a rule, RuleTestPrune also tests the whole conjunctive precondition as one condition. Finally, it tests the rules against a constraint to see whether the rule, with some of its preconditions possibly pruned, can classify the test data accurately. This constraint is implemented as a threshold on the rule hit ratio, a ratio of the number of correctly classified instances to the total number of instances classified by the rule.

RuleTestPrune evaluates the conditions and rules using the procedure $CondTestPrune$.

---

procedure: $CondTestPrune(R, TSet, T_c)$

R: a rule to be evaluated (input, output)
$TSet$: a set of testing data (input)

T_c: a threshold value for determining whether a precondition is acceptable (input)

1. If the precondition set of R is empty, then $R \leftarrow NIL$; RETURN;
2. For each precondition *prec* of R,
   (a) Get a subset of examples, E, of $TSet$ such that E contains only examples satisfying all preconditions of R except for *prec*, the one that is being evaluated.
   (b) Calculate the entries of the following table:

s_1	s_2	A
s_3	s_4	$N-A$
r	$N-r$	N

   where

s_1	$=$	number of examples in E satisfying *prec* and classified correctly
s_2	$=$	number of examples in E not satisfying *prec* but classified correctly
s_3	$=$	number of examples in E satisfying *prec* and misclassified
s_4	$=$	number of examples in E not satisfying *prec* and misclassified
r	$=$	$s_1 + s_3$ = number of examples satisfying *prec*
A	$=$	$s_1 + s_2$ = number of examples classified correctly
N	$=$	$s_1 + s_2 + s_3 + s_4$= number of examples in E

   (c) Calculate the statistical significance level s_{prec}[2]at which we can reject the hypothesis that the precondition *prec* is irrelevant to the rule classification of any example in the test set assuming that all other preconditions are satisfied by that example.

$$s_{prec} \leftarrow \sum_{i=s_1}^{M} \binom{M}{i} \binom{N-M}{r-i} / \binom{N}{r}$$

   where $M = min(A, r)$.

3. $S_{prec} \leftarrow \{prec | s_{prec} > T_c\}$. Prune a precondition of R that is in S_{prec} and whose tree level is the highest among all elements in S_{prec}. If no precondition is pruned, then return. Otherwise, update the set of preconditions of R. Go to step 1.

As illustrated in the above procedure, $CondTestPrune$ evaluates one precondition at a time. For each precondition, CondTestPrune measures its statistical confidence

[2]$(1 - s_{prec}) * 100\%$ is the confidence level at which one can reject the hypothesis that *prec* is irrelevant to the rule's class, assuming all other preconditions are satisfied.

on the relevance of the condition to the rule classification of any example assuming the example already satisfies all other conditions. After all preconditions are evaluated, a precondition is removed from the rule precondition set if the test value of that precondition is the lowest and the precondition lies highest in the tree hierarchy among the ones having the same evaluation. The heuristic decision of removing the one highest in the tree is based on empirical experience. Once a precondition is removed, the whole process of testing and pruning is iterated on the remaining set of preconditions. The procedure terminates when either no precondition is removed or all preconditions have been removed.

The procedure *RuleTestPrune* first invokes *CondTestPrune* to test and prune each of its preconditions. If at least one precondition survives after *CondTestPrune* returns, then it forms a new rule by combining all the preconditions of the current rule into a single conjunctive precondition. It then invokes *CondTestPrune* again to process the rule with the new rule precondition. If the rule survives this test, then RuleTestPrune checks its hit ratio, an estimation of the rule's accuracy in future classification.

4.2 Rule Set Testing and Minimal Rule Set

In the previous section, we described how RIST tests and prunes a rule with a conjunctive precondition. Since the rules are derived from decision trees, it is typical that each rule's precondition is a conjunctive condition, and multiple rules may exist for the same class. One can naturally view a set of rules predicting the same class as a single rule with a disjunctive precondition. A practical question arises: can we get a minimal subset of rules for a given class without loss of the classification coverage of the original set? In this section, we provide procedures that give a heuristic solution to this problem.

Procedure *RuleSetMin* finds the desired small subset of rules. In doing so, it calls on procedure *RuleSetEval*, which evaluates a set of rules that predict the same class. *RuleSetEval* uses two measures, namely, the rule set domain coverage and the rule set hit ratio. The rule set hit ratio is the ratio of the number of correctly classified instances to the total number of instances that are classified by at least one rule in the set. The rule set domain coverage is the ratio of the number of instances that are classified by at least one rule to the total number of instances of that class in the test set. If an example satisfies all the preconditions of at least one rule in the set, it is considered classified by the whole rule set. If an example is misclassified by at least one rule in the set, it is considered misclassified by the whole rule set; although there might exist another rule in the set that correctly classifies this example. Hence, the call: $RuleSetEval(SR, TSet, h, c)$ sets h to the rule set hit ratio and c to the rule set domain coverage, for a set SR of rules that predict the same class and a set of test data $TSet$.

It is now desired to develop a procedure that derives, from a given rule set, a subset of rules that predict the same class so that this subset (1) has the same domain coverage as the whole rule set, (2) has a hit ratio that is at least as good as that of the whole rule set, and (3) has the minimal number of rules. A rule subset which satisfies the first two conditions can be easily found. To satisfy the third condition, however, is an NP-hard problem. With this in mind, we developed a simple procedure which can find a subset of rules that satisfies the first two conditions and whose size is at least a local minimum, meaning that no rule can be removed without sacrificing either the domain coverage or the hit ratio of the rule subset.

```
procedure: $RuleSetMin(SR, TSet, SR_{min})$

$SR$ a set of rules predicting the same class $C$ (input)
$TSet$ a set of testing data (input)
$SR_{min}$ a subset of rules that (1) has the same coverage as that of $SR$, (2) has a hit
        ratio at least as good as that that of $SR$, and (3) has a size that is at least a
        local minimum. (output)

  1. Copy $SR$ to $SR_{min}$.
  2. call $RuleSetEval(SR_{min}, TSet, H_{min}, C_{min})$
  3. $worst_R = NIL$
  4. For each rule, $R$, in $SR_{min}$, do
      (a) call $RuleSetEval(SR_{min} - \{R\}, TSet, h, c)$
      (b) if $c \geq C_{min}$ and $h \geq H_{min}$, then $H_{min} \leftarrow h$, $worst_R \leftarrow R$.
  5. if $worst_R = NIL$ then RETURN; else remove $R$ from $SR_{min}$ and goto step 2.
```

4.3 An application example of RIST

We now give an example of an application of RIST to a real-world problem. The problem domain considered is RIE process diagnostics. The goal is to acquire a set of symbolic diagnostic rules from a collection of production log data that contain process fault inspection results. The data was obtained from Hughes Microelectronics Center. Each data log contains about 60 data entries, including machine type, device specification, material and resist thickness, plasma time, power, DC bias, chamber pressure, gas flow, temperature, valve position and number of wafers. Since this is a multi-stage (three stage) etch, each stage has its own set of measurements. A distinct table slot is used for visual inspection after the etch. For this project, three types of inspection results are commonplace, namely, *normal, PR erosion,* and *sleeves.*

Process fault diagnosis has been a regular demand on process engineers. Whenever a fault is detected, its immediate cause needs to be identified and a decision is then

Table 2: A Partial List of Data Logs for an RIE Process.

power1	power2	power3	time1	time2	time3	wafer#	topology	...	outcome
1117	1134	490	8.98	7.20	5.0	18	v	...	erosion
895	1139	492	10.26	8.2	5.0	24	v	...	erosion
833	854	442	7.4	6.0	5.0	1	v	...	sleeves
835	818	491	9.7	7.76	5.0	5	v	...	none
859	835	490	9.9	7.92	5.0	2	v	...	none
867	886	466	9.9	7.9	5.0	7	v	...	sleeves
847	871	473	9.8	7.8	5.0	8	v	...	sleeves
776	833	500	8.6	6.9	5.0	8	v	...	none
771	813	490	8.7	7.0	5.0	4	v	...	none
847	825	491	9.20	7.36	5.0	13	v	...	erosion
851	843	490	17.2	4.3	5.0	6	v	...	none
806	896	493	10.5	8.40	5.0	2	v	...	none
844	822	493	9.16	7.33	5.0	14	v	...	erosion
860	809	490	9.8	7.84	5.0	2	v	...	none
867	825	452	9.7	7.76	5.0	2	v	...	sleeves
878	819	454	24.1	14.4	5.0	1	t	...	none
848	816	455	321.0	16.8	5.	9	t	...	none
806	778	484	22.5	9.0	5.0	8	t	...	sleeves
881	795	467	22.7	13.6	5.0	1	t	...	none
772	868	490	8.7	7.0	5.0	7	v	...	none
...	...	...	...	...	...	...	...	...	...
842	826	494	17.0	13.6	5.0	1	v	...	none

made to correct the problem by adjusting process parameters directly or indirectly. Determining such adjustments is a nontrivial task. Abstracting these daily routines into rules that cover the task has been found to be difficult. Extracting general rules that can be transferred across different processes and be used to guide further reasoning to find physical laws governing the observed phenomena has been especially difficult. For this project, the difficulty in diagnosing the faults is due to the large number of process parameters, most of which vary greatly. These conditions naturally make a domain appear promising as an application for GID3 or RIST type of machine learning techniques.

Before RIST can be applied, the data slots of the log sheets were first screened to remove attributes that are known to be of not relevance to process diagnosis. We found that data can be grouped into two categories: one for process recipe parameters and one for process parameters that are measured but not controlled. Since the processes we dealt with used the same recipe, the recipe parameters cannot be used to discriminate the process conditions that are responsible for the faults. Therefore, only 9 primitive data slots were chosen as process parameters used to determine causes of various process faults. They were etching power (transmitted power - reflected power), etching time, number of wafers, device topology, and process outcome inspection record. Etching time and etching power are further divided into three etching stages. A subset of data is shown in Table 2.

Once the attributes and classes are determined, RIST can be used to induce rules. For example, RIST may be invoked to generate ten decision trees. For the generation of each decision tree, a randomly sampled training subset consisting of 60% of the training data is selected for each tree. The entire data set will be used as the test data. It is further specified that only rules whose class is "erosion" is of interest for this run. The three thresholding values for rule testing are also given in the specification. Example values for the condition test threshold, the rule test threshold, and the rule hit ratio threshold are 0.05, 0.05, and 0.9, respectively. Notice that the first two are upper bounds on acceptable statistical significance values (lower bounds on acceptable statistical confidence) while the third is a lower bound on the acceptable rule hit ratios. Given the above specification, RIST generated ten decision trees from which 19 distinct rules of class "erosion" are derived. After the statistical and hit ratio testing, however, only nine rules are accepted. The hit ratio and coverage for this set of nine rules are identical and are equal to 0.9231.

Now, RIST finds the smallest subset of these rules that can achieve the same coverage as that of the original rule set with no loss in performance with respect to other criteria. This results in the selection of only four rules. The rule set hit ratio for this set is 1.0 while the coverage is 0.9231.

To let readers get a better feeling of RIST's approach to rule induction, testing, and pruning, we give two examples below to show how RIST evaluates and prunes rules derived from a decision tree for this problem domain. First, we show a rule and its initial evaluation right after it is derived from a decision tree.

```
RULE9: <ETIME2 in [9.07, 9.45)> & <ETIME1 in [8.1, _)>
       & <WAFER in [_, 11.5)> & <PW_2 in [_, 975.5)> ==> [class: Erosion]
<ETIME2 in [9.07, 9.45)> Test=0.001899     <WAFER in [_, 11.5)> Test=1.0
<ETIME1 in [8.1, _)>     Test=1.0          <PW_2 in [_, 975.5)> Test=1.0
              rule's test= 0.017845, certainty= 1.000000
2 exs of class Covered, 11 exs of class Not Covered, 0 Misclassified.
```

Obviously, all but the first condition of this rule are irrelevant to the rule's class. These irrelevant conditions are therefore pruned, resulting in the following new compact rule.

```
RULE9: <ETIME2 in [9.07, 9.45)> ==> [class: Erosion]
<ETIME2 in [9.07, 9.45)> Test=0.017845
             rule's test= 0.017845, certainty= 1.000000
2 exs of class Covered,  11 exs of class Not Covered, 0 Misclassified.
```

In this case, although two conditions are pruned, the rule's hit ratio is not affected because these two conditions are completely redundant. In the second example, we study a different case in which a rule's hit ratio is affected after its condition being pruned. Again, we first give the rule's original form as derived directly from a decision tree.

```
RULE3: <PW_2 in [816.5, 828.5)> <WAFER in [11.5, _)> ==> [class: Erosion]
<PW_2 in [816.5, 828.5)> Test=0.254545    <WAFER in [11.5, _)> Test=0.000327
           rule's test= 0.000234, certainty= 1.000000
4 exs of class Covered, 9 exs of class Not Covered, 0 Misclassified.
```

As can be seen, the attribute PW_2 is not quite as relevant as the attribute WAFER with respect to the rule classification "erosion". After RIST's rule pruning with the given threshold, the first rule condition is then removed, giving the following result:

```
RULE3: <WAFER in [11.5, _)>  ==> [class: Erosion]
<WAFER in [11.5, _)> Test=0.000000
         rule's test= 0.000000, certainty= 0.750000
9 exs of class Covered, 4 exs of class Not Covered, 3 Misclassified.
```

This new rule's precondition now satisfies the test threshold. The rule's hit ratio, however, is now too low to meet the requirement (0.9). This rule is therefore not accepted.

5 Dealing with Limited Training Sets

The learning algorithms we discussed do not use any special domain knowledge about the data during tree or rule generation. They rely on the availability of large training samples to detect the presence of meaningful reliable patterns or correlations. In some cases however, obtaining training examples may be an expensive process. In this case, only a limited training set may be available.

5.1 Process Optimization

To achieve high yield, low defect density, RIE must operate at an optimal point in the input parameter space. The problem, known as process optimization, is the most important problem that a process engineer has to face.

The goal of process optimization is to find a set of input parameters, usually referred to as *recipe*, such that the outputs can be optimized. The input parameters in RIE usually include RF power applied to the electrodes, gas mixture, gas flow rate, chamber pressure. The outputs are usually the measurements made after the wafers are unloaded from the etching chamber. They include: etch rate, selectivity(the ratio of etch rate of two different etch materials), edge profile, loss of critical dimension (Delta CD) and uniformity. In most cases, we want to optimize several of these output variables with the others as constraints. For example, one might want to maximize the etch rate and minimize the side wall loss under the condition that selectivity is greater than a certain number.

The difficulty of the problem lies in getting an accurate physical model of the complex process. Most models of RIE process are too ideal to be used in practice. The process is complex because all the input parameters can interact with each other. In addition, the optimization problem is multivariable, this means that several variables should be optimized according to the process specification.

A popular way to solve the process optimization problem is to use Response Surface Methodology(RSM) [4, 13]. RSM is a statistical method which regards the input parameters of RIE as independent variables and output parameters as response variables. For each response variable, multiple regression analysis is done to generate a response surface. This is usually done by fitting a polynomial equation to a set of independent variables. Optimization techniques are then used to find a point that optimizes the response variables under the given constraints (see [1]). However, RSM has its drawbacks. It is a static method in the following sense. Given a set of experimental data, a set of response surfaces can be generated and a fixed optimal point can be found. According to this optimal point, a recipe can be formulated. One would have to rely on this recipe to have an optimal process. The problem is that under the same recipe, the operating point might not be optimal because certain hidden variables which were not considered in the design process may influence the process later. In other words, the response surface may drift so that the operating point may no longer be optimal. Obviously, what we need is a set of rules which tell us where to move in the parameter space to achieve the optimal output under the given constraints. This makes for a dynamic, rather than static, solution to the optimization problem.

The above analysis led us to consider using machine learning technique for RIE process optimization. Our objective was to generate a set of rules which would determine the input parameters that need to be changed, and the direction in which to change them, if the outputs are not optimal or do not satisfy the constraints. For example, the rule "if Oxide selectivity is lower than normal, and CD (Critical Dimension) uniformity is higher than normal, then pressure is high, and total flow is high" tells us that pressure and total flow should be decreased since selectivity and uniformity are not optimal.

The following are some of the advantages of extracting symbolic rules from experimental data: 1. Knowledge is widely represented by rules in expert systems. Applying knowledge-based approach to semiconductor automation and building expert systems are the primary reasons for acquiring RIE optimization knowledge. 2. As mentioned above, rules give trends instead of an optimal point so that they are more general and more dynamic. 3. Rules are more understandable than equations and numbers. They indicate the qualitative behavior of the process. Thus, rules may help process engineers understand the process and rules may also be used for training new process personnel.

A machine learning program, such as GID3, provides us with a tool to derive qualitative, symbolic rules from data (as discussed in the previous sections and in [12].) However, it was found that combining RSM with machine learning, rather than applying machine learning alone, is more successful in deriving robust, meaningful rules from experimental data. Following are two of the reasons why direct application of GID3 to a set of data is not preferred: 1) To induce a decision tree, GID3 usually requires a large set of data as input. However, we can only have a limited number of experiments because an RIE experiment is expensive. Regression analysis enables us to generate response surfaces. From the response surfaces, we can derive as many samples as necessary for use as input to GID3. 2) GID3, as a classification algorithm, is noise sensitive. RSM can filter out some of the influence of the noisy data.

In the next section, we present a procedure for deriving RIE process optimization rules by using both RSM and GID3. In that section we also provide some results derived from experimental data.

5.2 Procedure and Results

To derive a set of qualitative rules about a process from a set of experimental data, we implemented the KARSM (Knowledge Acquisition from RSM) system. KARSM first constructs response surfaces for the process and then generates random samples from the response surfaces. Those samples are then classified according to the optimum criteria and are fed into machine learning program to induce rules which provide ranges for optimal operation and sub-optimal operations. Finally, after analyzing these ranges and conditions, a set of rules about the trend for optimization is derived. This procedure is schematically illustrated in the diagram of Figure 5. In what follows, we describe the procedure of KARSM through a detailed example of applying KARSM to a set of experimental data obtained from Hughes Microelectronics Center.

Our data is from experiments for reactive ion etching of poly-silicon gates. The three independent variables are the three controllable process parameters. They are: pressure, total flow rate and percentage of Cl_2 flow rate. The experiments are designed using face-centered cube (FCC) design with center point replications. By replicating center point six times, we have 20 trials for three variables and three levels. Measurements are then performed on the etched wafers. Four response variables are used for optimization, they are: Delta CD, CD uniformity, Oxide selectivity (Oxide etch rate divided by poly-silicon etch rate), and Oxide uniformity.

The objective of the optimization is to minimize Delta CD under constraints. The constraints can be formulated as:

$$\text{Oxide selectivity} \geq c_1 \ \& \ \text{CD uniformity} \leq c_2 \ \& \ \text{Oxide uniformity} \leq c_3$$

Here, c_1, c_2, and c_3 are known constants. First, multiple least square regression analysis is applied to generate response surfaces for all the four response variables. For response variable Delta CD, we use quadratic equation to fit the data. The

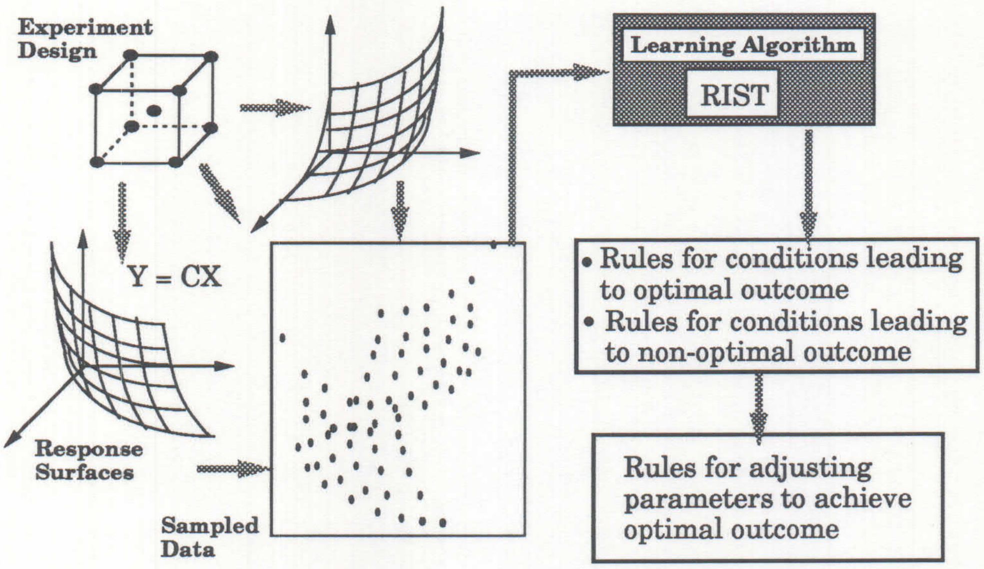
Experiment Design
Y = CX
Response Surfaces
Sampled Data
Learning Algorithm
RIST
• Rules for conditions leading to optimal outcome
• Rules for conditions leading to non-optimal outcome
Rules for adjusting parameters to achieve optimal outcome

Figure 5: A Dataflow Diagram of the KARSM System.

normalized equation with $R^2 = 0.98$ is as follows:

$$\begin{aligned} DeltaCD &= 2045.8 - 7.8987p + 4.6933tf - 84.357pf + 0.0086p^2 \\ &\quad +0.50041pf^2 - 0.012333p \cdot tf + 0.16239p \cdot pf \end{aligned}$$

where p, tf, and pf stand for pressure, total flow, and percentage Cl_2 flow respectively.

For the rest of the response variables, we use polynomials of degree three. We use degree three because the quadratic equations do not give us good fittings, i.e., R^2 for these equations are very low. Since there are only 20 trials, we use forward stepwise regression analysis. The resulting equations are quite satisfactory:

$$\begin{aligned} OxSel &= -250690 - 175.26p + 5490.2tf - 7.331tf^2 + 540.74pf^2 - 3.1968p \cdot tf \\ &\quad -208.01tf \cdot pf + 0.0012673p^3 - 6.5863pf^3 - 0.023512p^2 \cdot pf \\ &\quad +0.21271tf^2 \cdot pf + 0.98478tf \cdot pf^2 + 0.12781p \cdot tf \cdot pf \quad (0.965) \\ CdUnif &= 319750 + 0.089362tf^3 - 0.054928p \cdot tf^2 - 0.79226tf^2 \cdot pf \\ &\quad -0.92434 * p * pf^2 + 1.5004tf \cdot pf^2 + 0.49059p \cdot tf \cdot pf \quad (0.944) \\ OxUnif &= 2500700 - 15069p + 11.982p^2 - 3.0201tf^2 - 3144.9pf^2 \\ &\quad +507.46p \cdot pf + 39.813pf^3 - 0.30148p^2 \cdot pf + 0.0061702p \cdot tf^2 \\ &\quad +0.061437tf^2 \cdot pf - 3.0468p \cdot pf^2 - 0.062042p \cdot tf \cdot pf \quad (0.974) \end{aligned}$$

The numbers in the parentheses following the equations are the values of R^2.

To derive qualitative description of the causal relation between the controllable process variables and the response variables, we use GID3 to learn symbolic rules from random samples on the response surfaces. A random sample on the response surfaces is generated in the following way: For each independent variable, a random

Table 3: Classified Random Samples for Training KARSM.

No.	*pressure*	*total flow*	*percent Cl_2 flow*	*class*
1	377	192	49	ab,l,l,l
2	342	297	36	h,l,l,h
3	491	241	34	ab,h,l,h
4	374	235	30	l,l,l,h
5	419	289	37	ab,h,l,l
6	319	253	38	m,l,l,l
7	305	157	31	vl,h,l,h
8	303	185	42	vl,h,l,l
...	...	...	...	...

value within the range of that variable is assigned. A tuple of all random values is called a random input. Then, for each response surface above, we calculate the response for the random input.

GID3 views a set of random samples on the response surfaces as a set of examples. Naturally, three controllable parameters, pressure, total flow, and percentage of Cl_2 flow are regarded as the attributes of the examples. Class of an example is assigned in the following fashion. For each constraint variable, there is a normal region and an abnormal region. For example, value of CD uniformity is in normal region if it is less than or equal to c_1 and is in abnormal region otherwise. For the response variable to be optimized, say, Delta CD, we discretize its value into different levels such as very high, high, medium, low and very low. Therefore, for each random sample, its responses are coded as a string with four segments, with each segment representing the region of one response. For example, a string "vh,h,l,h" means: very high Delta CD, high Oxide selectivity, low CD uniformity, and high Oxide uniformity. Obviously, overall there are 40 possible classes, although samples from the response surfaces may not exhaust all these 40 classes. Among these classes, only one class is optimal: the class corresponding to the string "vl,h,l,l". All other classes are sub-optimal either because one or more constraints are not satisfied or because Delta CD is not minimized. A subset of classified random samples is shown in Table 3.

By performing machine learning using GID3 on a set of examples in the form of Table 3, we can derive a decision tree which corresponds to a set of rules. To obtain a more robust set of rules, we use the RIST program which invokes GID3 repeatedly to obtain many trees. A set of rules that pass the RIST statistical tests is considered robust. Some of the rules in these set are given below:

a. If $300 \leq$ pressure < 320.5 mTorr, &
 $170.5 \leq$ total flow < 185.5 sccm, &
 $38.5\% \leq Cl_2$ flow percent $< 42.5\%$,
 Then Oxide selectivity is normal,
 CD uniformity is normal,
 Oxide uniformity is normal, &
 Delta CD is very low.

b1. If $319.5 \leq$ pressure 395 mTorr, &
 $296 \leq$ total flow < 300 mTorr,
 Then Oxide selectivity is lower than normal,
 & CD uniformity is higher than normal.

b2. If $326.5 \leq$ pressure < 407 mTorr, &
 total flow in { [283,284.5) , [296.5,300) },
 Then Oxide selectivity is lower than normal,
 & CD uniformity is higher than normal.

Notice that the rule (a) gives the conditions for optimal operation. The other two rules give the conditions for sub-optimal operation. What we need is a set of rules which point to the direction of change for optimal operation when the operation is not optimal. This will actually reflect causes underlying sub-optimal operation.

To derive such rules, the conditions(path) of a rule with sub-optimal class are compared to the conditions of the rule with optimal class. The differences are then the changes one needs to make to bring a sub-optimal operation to the optimal one. For instance, rules b1 and b2 give the conditions for a sub-optimal class "Oxide selectivity is lower than normal and CD uniformity is higher than normal". By comparing the conditions of these rules with those of rule a, we derive the following rule: if Oxide selectivity is lower than normal and CD uniformity is higher than normal, to optimize Delta CD with other responses within the constraints, both pressure and total flow should be decreased. Below, we list four of the rules derived in this way:

1. If Oxide uniformity is higher than normal, then percentage of Cl flow is low.
2. If Oxide selectivity is lower than normal, then total flow is high.
3. If Delta CD is higher than normal, then pressure is high.
4. If Oxide selectivity lower than normal, & CD uniformity higher than normal, then pressure and total flow are high.

Now, the question is how "good" are these rules? Do they really reflect the qualitative behavior of the process? We may not get a good answer to these questions until these rules have been applied at the actual production lines for months and optimal process has been maintained by the application of these rules. However, a partial answer could be given if we compare these rules with the contour plots of the response surfaces. Although, as is mentioned above, response surfaces may drift with time, and the optimum on the response surfaces may not be the real optimum, the relative shape of a response surface most probably will not change. This is because the relative dependency of a response variable to independent variables is governed by the causal relationship between the response variable and the controllable parameters. Therefore, we believe that if correspondence can be found between a rule and the contour plots of the response surfaces, the rule does reflect the qualitative behavior of the process.

We also note that process engineers frequently derive "rules of thumb" of their own by reading contour plots of a process. This is by no means simple and easy. It can be very time consuming. In this respect, the above procedure of deriving rules

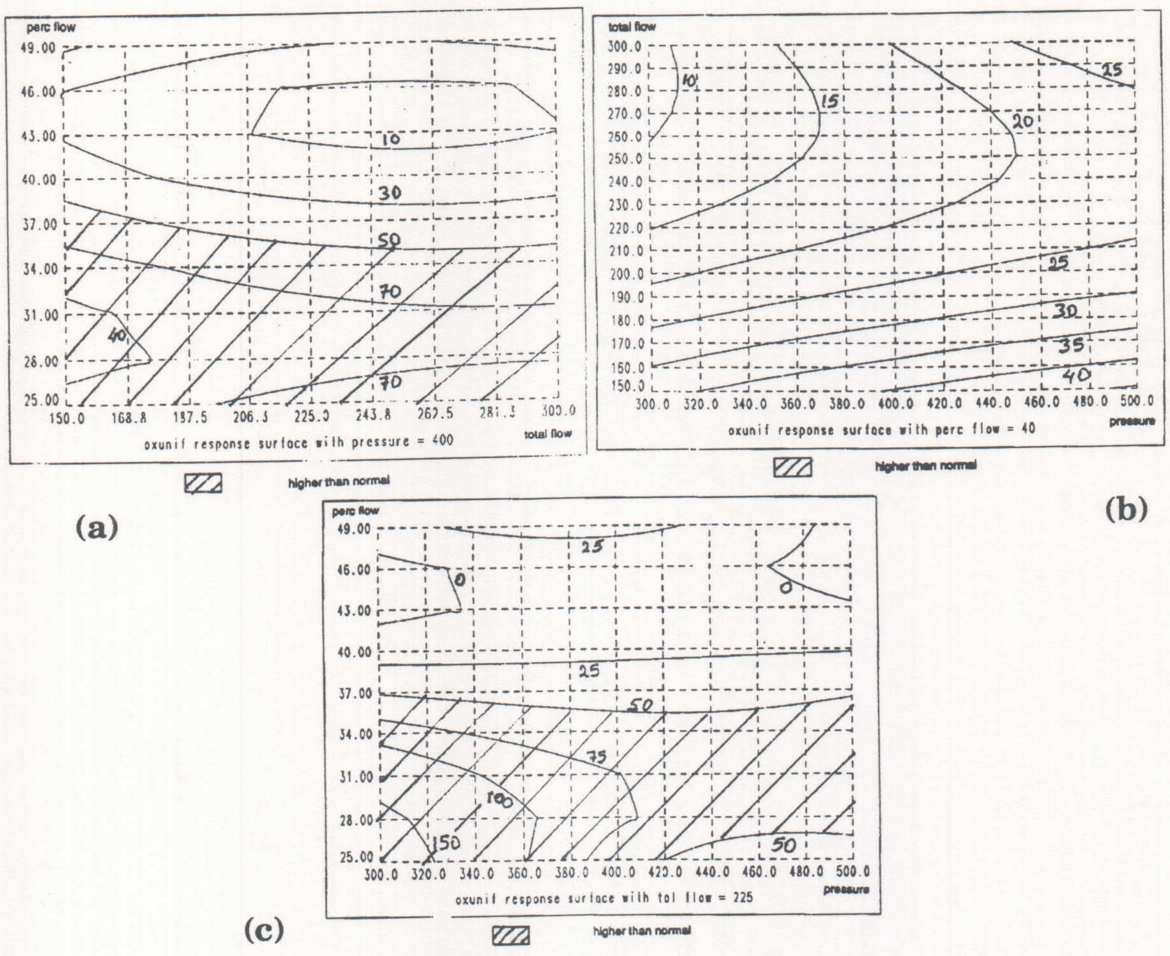

Figure 6: Contour Plots from the Generated Response Surfaces.

by machine learning as well as RSM from experimental data can be thought of as a mechanization of extracting information from response surfaces.

The contour plots in Figure 6 show the correspondence between the response surface for oxide uniformity and rule 1. The constraint on Oxide uniformity is that it must be less than 50. Obviously, the shaded regions in plots (a) and (c) correspond to "Oxide uniformity is higher than normal". From both (a) and (c), it is easy to see that total flow and pressure do not affect Oxide uniformity too much. This is again verified by (b) where Oxide uniformity is normal in all region. From (a) and (c), it is clear that the lower the percentage flow of Cl_2, the higher the Oxide uniformity, which is exactly what rule 1 says.

6 Concluding Remarks

We have introduced the general problem of extracting rules from data for the purpose of automating the knowledge acquisition process. We described the GID3 algorithm for inducing decision trees and its use in automating knowledge acquisition in several application domains. Given a training set of data, the algorithm produces a decision tree for predicting the outcome of future experiments under various, more general conditions. The tree may then be translated into a set of rules for use in expert systems. We also introduced two extensions of GID3 to deal with noise and limited training set availability and detailed their application to RIE related domains.

We have used GID3 and its extensions to extract knowledge from data in several application domains. In each case, the decision trees obtained, or the English version of the generated rules, were sent to the engineer who supervises the experiments and provides the data. The derived rules were judged to be consistent with the data and conformant with the engineers' expectations. A derived model, or a discovered pattern, that is consistent with a process engineer's expectation is of great value in two ways: (1) it provides a previously unavailable mechanical means for classifying events or relating faults to parameters; and (2) it gives the process engineer further insight into the process by making explicit a pattern that was previously implicit as part of the engineer's "intuition" about the process.

We believe that machine learning techniques have an important role to play in the automation and improvement of manufacturing techniques as well as many other diagnostic tasks. A machine learning approach avoids the ubiquitous "knowledge acquisition bottleneck" by minimizing the required interaction with domain experts and focussing on a resource that is much easier to obtain: experimental data. A further advantage of using an induction systems such as GID3, GID3*, RIST and KARSM is that at least some portions of the tedious and difficult knowledge acquisition process can be made efficient, accurate, and most importantly, can be automated.

Acknowledgements

This work was supported by the University of Michigan SRC Research Program under contract #89-MC-085. We would also like to thank Hughes Microelectronics Center for their partial support of this work in the form of an unrestricted grant.

References

[1] Bergendahl, A.S., Bergeron, S.F., and Harmon, D.L. (1982). "Optimization of plasma processing for silicone-gate FET manufacturing applications." *IBM Journal of Res. Dev. vol. 26*, no. 5.

[2] Breiman, L., Friedman, J.H., Olshen, R.A., and Stone, C.J. (1984). *Classification and Regression Trees.* Monterey, CA: Wadsworth & Brooks.

[3] Cheng, J., Fayyad, U.M., Irani, K.B., and Qian, Z. (1988). "Improved decision trees: A generalized version of ID3." *Proceedings of the Fifth International Conference on Machine Learning.* pp. 100-108. Ann Arbor, MI.

[4] Draper, N.R. and Smith, H. (1966) *Applied Regression Analysis.* New York: John Wiley.

[5] Fayyad, U.M. and Irani, K.B. (1990) "What should be minimized in a decision tree?" *Proceedings of the Eighth National Conference on Artificial Intelligence AAAI-90* (pp. 749-754). Cambridge, MA: MIT Press.

[6] Fayyad, U.M. and Irani, K.B. (1991) "A Machine Learning Algorithm (GID3*) for Automated Knowledge Acquisition: Improvements and Extensions." *General Motors Research Report CS-634.* Warren, MI: GM Research Labs.

[7] Fayyad, Usama M. and Irani, K.B. (1991) "On the handling of continuous-valued attributes in decision tree generation." *Machine Learning,* in press.

[8] Fayyad, U.M. (1991). *On the Induction of Decision Trees for Multiple Concept Learning.* Dissertation in preparation, EECS Dept., The University of Michigan, Ann Arbor.

[9] Fiegenbaum, E.A. (1981). "Expert systems in the 1980s." In Bond, A. (Ed.), *State of The Art Report on Machine Intelligence.* Maidenhead: Pergamon-Infotech.

[10] Finney, D.J., Latscha, R., Bennett, B.M., and Hsu, P. (1963). *Tables for Testing Significance in a 2x2 Contingency Table.* Cambridge: Cambridge University Press.

[11] Friedhoff, C.B., Cresswell, M.W., Lowry, C.R., and Irani, K.B. (1989). "Analysis of intra-level isolation test structure data by multiple regression to facilitate rule identification for diagnostic expert systems." *Proceedings of the International Conference on Microelectronic Test Structures.* Edinburgh, Scotland.

[12] Irani, K.B., Cheng, J., Fayyad, U.M., and Qian, Z. (1990) "Applications of Machine Learning Techniques in Semiconductor Manufacturing." *Proceedings of The S.P.I.E. Conference on Applications of Artificial Intelligence VIII* (pp. 956-965). Bellingham, WA: SPIE: The International Society for Optical Engineers.

[13] Jenkins, M.W., Mocella, M.T., Allen, K.D., and Sawin, H.H. (1986). "The modelling of plasma etching processes using response surface methodology." *Solid State Technology (4).*

[14] Quinlan, J.R. (1986). "Induction of decision trees." *Machine Learning 1, No. 1.* pp. 81-106.

[15] Quinlan, J. R. (1987). "Generating Production Rules From Decision Trees." *Proceedings of the Tenth International Joint Conference on Artificial Intelligence.* Milan, Italy.

[16] Yang, Y.K. (1989). "EPAS: An emitter piloting advisory expert system for IC emitter disposition." *Proceedings of Semicon West.*

(a)

(b)

3 contour plots go here

(c)

Figure 6: Contour Plots from the Generated Response Surfaces.

by machine learning as well as RSM from experimental data can be thought of as a mechanization of extracting information from response surfaces.

The contour plots in Figure 6 show the correspondence between the response surface for oxide uniformity and rule 1. The constraint on Oxide uniformity is that it must be less than 50. Obviously, the shaded regions in plots (a) and (c) correspond to "Oxide uniformity is higher than normal". From both (a) and (c), it is easy to see that total flow and pressure do not affect Oxide uniformity too much. This is again verified by (b) where Oxide uniformity is normal in all region. From (a) and (c), it is clear that the lower the percentage flow of Cl_2, the higher the Oxide uniformity, which is exactly what rule 1 says.