A Classification-Based Fault Detection and Isolation Scheme for the Ion Implanter

Shin-Yeu Lin and Shih-Cheng Horng

Abstract—**We propose a classification-based fault detection and isolation scheme for the ion implanter. The proposed scheme consists of two parts: 1) the classification part and 2) the fault detection and isolation part. In the classification part, we propose a hybrid classification tree (HCT) with learning capability to classify the recipe of a working wafer in the ion implanter, and a k-fold cross-validation error is treated as the accuracy of the classification result. In the fault detection and isolation part, we propose a warning signal generation criteria based on the classification accuracy to detect and fault isolation scheme based on the HCT to isolate the actual fault of an ion implanter. We have compared the proposed classifier with the existing classification software and tested the validity of the proposed fault detection and isolation scheme for real cases to obtain successful results.**

Index Terms—**Classification, classification and regression tree (CART), clustering algorithm, fault detection and isolation, ion implanter.**

I. Introduction

An ION implanter [1] is a bottleneck machine in the semiconductor manufacturing process because of its expensiveness; thus, ion implantation is a critical operation for throughput. A damaged wafer due to the malfunction of the ion implanter is not reworkable, hence it significantly affects the yield. Therefore, a real-time *fault detection* to prevent more wafer damage and a *fault isolation* to reduce the down time of the ion implanter are crucial issues to the yield and throughput of the semiconductor manufacturing process. There are two categories of fault detection methods, the *model-based methods* and *model-free methods*. The model-based methods, which utilize the mathematical model of the plant, originated from chemical process control, aerospace related research, and other areas, have been developed in last three decades [2]–[9]. Model-free methods, which do not use the mathematical model of the plant, range from physical redundancy, limit value checking [10], to spectrum analysis [11], [12]. Among them, the limit value checking method is widely used in practice. There are also two types of fault isolation methods [13], [14], the classification methods and inference methods. If *a priori* knowledge is not available for the relationships between the measured data patterns and faults, classification methods are used. For example, a neural network, trained using a large set of abnormal data pattern and known fault pairs, can be used to classify the corresponding fault of an abnormal data pattern. If there is *a priori* knowledge for the relationships between faults and measured data patterns, a rule-based expert system can be used to inference the corresponding fault of an abnormal data pattern.

Regarding fault detection, since there does not exist any proper model for the ion implanter, the model-based fault detection methods cannot apply. Thus, the limit value checking method is currently employed in some semiconductor manufacturing companies. The structure of an ion implanter is shown in Fig. 1 [1]. In general, the equipment supplier provides digital equipment to monitor the proper operation of the scanning subsystem of the machine. The well-trained engineers employ the limit value checking method to investigate the SPC charts [15] of the measured parameters for other major subsystems, such as the ion source (filament), extraction electrode, mass analysis, and acceleration subsystems, to monitor their operations.

The measured parameters can be, for example, filament voltage, filament current, discharge voltage, etc. However, there are several tens to hundreds of recipes[1] for wafer fabrication in a semiconductor foundry each day. Although the setting of a scanning subsystem is independent of the recipes, the other four subsystems' parameters may vary widely due to various recipes. This induces the first drawback of the limit value checking method, that is the difficulty of defining a *threshold* to distinguish one recipe from the others. Since each recipe involves a combined setting of the four subsystems, this induces the second drawback of the limit value checking method, that it cannot provide combination information of the measured parameters of the four subsystems. In addition, the occurrence of electrical spikes in the ion implanter will make the measured parameters exceed the threshold and indicate a fault situation; however, the electrical spikes are not actual machine faults. This is the third drawback of the limit value checking method. Regarding fault isolation, both classification methods and inference methods require a fairly large set of the abnormal data patterns with known faults to train a classifier and construct a rule-based expert system, respectively. Collecting a large set of abnormal data patterns with known faults in an ion implanter is very difficult, because there are several hundreds of steps in fabricating a chip and the chip failure is most probably known when it is under test. To find out which step in the complete manufacturing process causes the failure is already difficult,

Manuscript received May 31, 2005; revised July 6, 2006. This work was supported in part by the National Science Council, Taiwan, R.O.C., under Grant NSC94-2213-E-009-044.

The authors are with the Department of Electrical and Control Engineering, National Chiao Tung University, Hsinchu 300, Taiwan, R.O.C. (e-mail: sylin@cc.nctu.edu.tw; schong.ece90g@nctu.edu.tw).

Color versions of Figs. 3 and 4 available online at http://ieeexplore.ieee.org.
Digital Object Identifier 10.1109/TSM.2006.883594

[1]A recipe controls how settings are initialized or changed during a process step. Examples include recipe numbers which index tables of set points in furnaces or written instructions to operators. A recipe is usually considered constant during any one process step. In this paper, a recipe corresponds to a specific product of integrated circuit.

0894-6507/$20.00 © 2006 IEEE

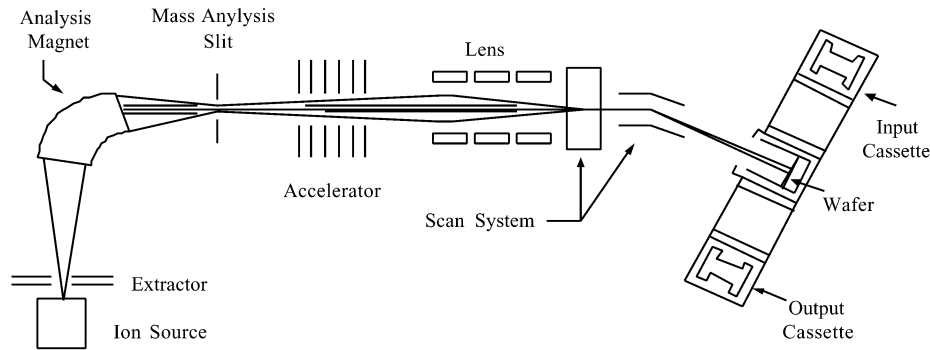


Fig. 1. Structure of ion implanter.

not mention the collection of a large set of abnormal data patterns with known faults due to ion implantation. Thus, the *purpose* of this paper is to propose an automatic (i.e., no need for well-trained engineers) and effective tool to monitor the above-mentioned four subsystems as a whole and generate a warning signal once a machine fault occurs and isolate that fault.

To overcome the first two drawbacks of the limit value checking method, we should be able to identify the recipe of the working wafer from the measured parameters of all the four subsystems. This makes the data mining technique [16] attractive. To overcome the third drawback of the limit value checking method, we need to distinguish electrical spikes from the actual machine faults. Motivated by the these considerations, we propose a *classification-based fault detection and isolation scheme* for the ion implanter. Viewing a recipe as a *class*, we can classify the recipe of the working wafer based on the corresponding measured parameters of the four subsystems. Thus, the overall structure of the proposed fault detection and isolation scheme can be shown in Fig. 2. Our scheme starts by classifying the recipe of the working wafer based on the measured parameters. If the classified recipe of the working wafer matches its destined one, we assume there is no fault and proceed with next wafer. This *no-fault assumption* may cause only a few damaged wafers in the worst case. A detailed analysis of this claim will be addressed in Section IV. On the other hand, if the classified recipe does not match its destined one, a double check of the recipe command should be carried out. If the command is wrong, the operator will be informed; otherwise, the *warning signal generation criteria* will be tested. If the criteria is satisfied, we conclude that there is a machine fault and a warning signal will be generated; otherwise, we will proceed with next wafer. Once a warning signal is generated, we will perform the fault isolation scheme to isolate the fault. In short, the proposed fault detection and isolation scheme consists of three major problems. The first one is a *classification problem*, which is to classify the recipe of a working wafer. The second one is a *fault detection problem*, which is to determine whether there is a machine fault and generate a warning signal if there is one. The third one is a *fault isolation problem* to determine which subsystem has a fault. In this paper, we propose a *hybrid classification tree (HCT)* with good *learning capability* to deal with the classification problem. The HCT combines a proposed *clustering algorithm*

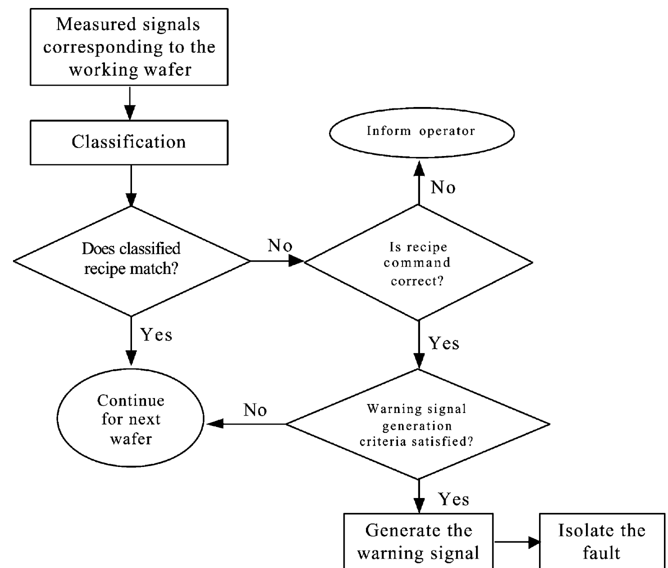


Fig. 2. Proposed fault detection and isolation scheme for ion implanter.

with the *classification and regression tree (CART)* [17], [18] to take advantage of the specific setting of a recipe during a process step. Its good learning capability will enable it to work on line. Since the operator should interrupt wafer processing immediately when a fault is detected, a high standard in the accuracy of fault detection is required to not unnecessarily degrade the throughput. Thus, to account for the possible inaccuracy caused by the HCT, we propose a *warning signal generation criteria* to deal with the fault detection problem. This criteria aims to minimize the probability of false alarm when there is no fault as well as the probability of no alarm while a fault exists; the former tries to eliminate the indicated fault situations due to electrical spikes and classification errors, while the latter tries to find out the hidden machine faults when a classified recipe matches the destined one. However, we need not worry about the latter one by the no-fault assumption mentioned previously. To cope with the fault isolation problem, we propose an HCT-based fault isolation scheme. The basic idea of this scheme is to find the parameter (or parameters) that causes the classification errors. Unlike the existing methods, which need to collect a fairly large set of measured data patterns with known faults, as indicated, the proposed fault isolation scheme almost spends no extra effort as will be seen in Section III-B.

From here on, we will use the terminologies *attribute* and *data pattern* in classification techniques to represent the *parameter* and *data of the measured parameters* of the four subsystems of the ion implanter, respectively.

We organize our paper in the following manner. In Section II, we will present the HCT and its learning capability. In Section III, we will analyze the probability of no alarm, while a machine fault exists to verify the no-fault assumption, and present the criteria for generating the warning signal. We will also present the fault isolation scheme in this section. In Section IV, we will apply the HCT to real data sets to obtain the k*-fold cross-validation classification errors*, based on which we will demonstrate the validity of the proposed warning signal generation criteria and the fault isolation scheme. We will also investigate the learning capability of HCT by reporting the computation time needed to update the classification rules of HCT. In Section V, we conclude.

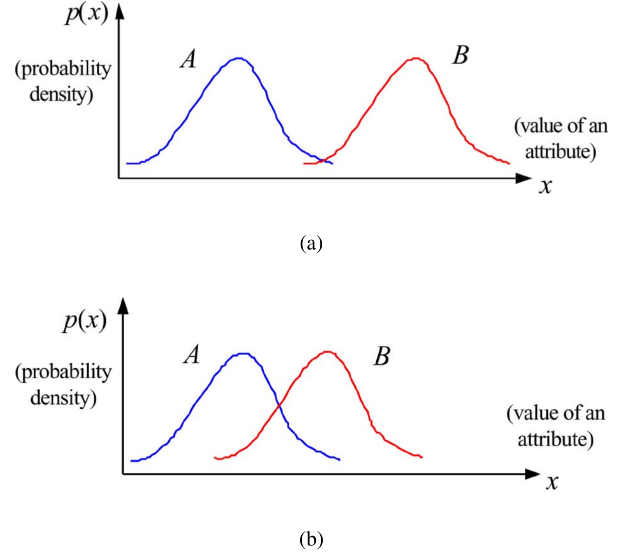


Fig. 3. (a) Separable recipes. (b) Nonseparable recipes.

II. HCT

A. Introduction

There exist numerous classification techniques for classification problems of continuous attributes such as the neural network approach [19], [20], maximum-likelihood approach [21], [22], fuzzy set theory-based approach [23], [24], decision tree [25], [26], CART [17], [18], kernel-based learning algorithms [27], and recent methods like random forests [28], multiple additive regression trees (MART) [29], [30], and the boosting flexible learning ensembles with dynamic feature selection technique [31]. Among them, the neural network approach is superior in the aspects of free data distribution and free data importance; however, they are computationally expensive and produce variable results due to the random initial weights. The maximum-likelihood approach was the most widely used method in classifying remotely measured data; however, its performance was degraded when the target classes could not be adequately described by the statistical model. The fuzzy set theory-based approach had been successfully applied to the pattern classification problem; however, the computational complexity is raised when the number of classes as well as the number of attributes is large. A decision tree is mainly designed for classification of discrete variables. However, CART can handle continuous attributes. Compared with random forest, MART, and boosting flexible learning ensembles with dynamic feature selection techniques, the disadvantage of CART is inaccuracy due to its nature of piecewise constant approximation. However, the biggest advantage of CART is its interpretability; whereas, the previously mentioned three methods and the kernel-based learning algorithms are thought to lack this feature. The interpretability is the key feature of our HCT-based fault isolation scheme, however, at the expense of some classification accuracy. Fortunately, the decrease in accuracy will be remedied by the warning signal generation criteria by applying it to the fault detection of the ion implanter, which will be presented in Section III-A.

The tree sizes of CART are closely related to the interpretability and accuracy. A small tree can be easily interpreted, while the interpretability of a large tree is questionable. On the other hand, a larger tree is more accurate than the smaller one. Thus, to retain the interpretability of a small tree while keeping the accuracy of a large tree, we intend to propose a preprocessing step to reduce the tree size of CART to improve the interpretability while keeping its classification accuracy. In general, a recipe may contain various steps, and a recipe step remains *constant* during the processing of one wafer; however, different attributes (parameters) may be ramped during the entire processing step. Nonetheless, some (*not all*, as can be observed from the experimental results shown in Fig. 10) attributes' mean of each individual recipe step is still a key to distinguishing the recipes. Thus, we can exploit this property to fulfill the objective of preprocessing. To do this, we propose a *separation matrix*-based *clustering algorithm* as a *preprocessing* step for CART. This clustering algorithm will classify the whole data set into a *clustering tree* and the classes in the leaf clusters will be classified by the CART. Because both the size and the number of classes of the leaf cluster are much smaller than the original data set, the computational complexity of CART can be improved.

B. Separation Matrices-Based Clustering Algorithm

Due to the previously mentioned property of a recipe during a processing step, we can investigate the separability between two recipes through the degree of overlapping of the attribute values. For example, suppose the probability density function of an attribute for the two recipes A and B is as shown in Fig. 3(a). Then, these two recipes are separable based on that attribute; while in the case of Fig. 3(b), the two recipes are not. Throughout this section, we will use the terminology *class* in classification techniques to represent *recipe*.

1) Chebyshev Inequality-Based Separation Matrices: We let $D(C_i, C_j)_k$ denote the separation index between classes C_i and C_j based on attribute k and define

$$D(C_i, C_j)_k = \begin{cases} 0, & \text{if } C_i \text{ and } C_j \text{ are separable} \\ & \quad \text{based on attribute } k \\ 1, & \text{otherwise.} \end{cases} \quad (1)$$

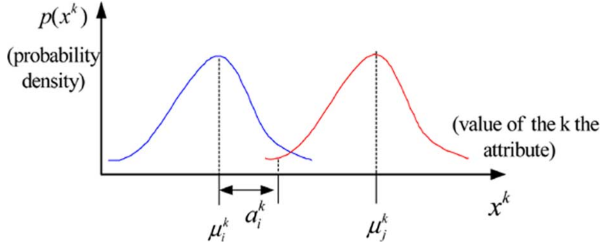


Fig. 4. Illustration of separation between C_i and C_j based on $\overline{p}_j$.

Clearly, $D(C_i, C_i)_k = 1$ and $D(C_i, C_j)_k = D(C_j, C_i)_k$ for any attribute k. The value of $D(C_i, C_j)_k$ is computed using the Chebyshev inequality [32]. We let the random variable X_i^k denote the kth attribute of class C_i and let μ_i^k and σ_i^k denote the mean and standard deviation of X_i^k, respectively. Let a_i^k be a positive real number such that $P[|X_i^k - \mu_i^k| \geq a_i^k] \leq \alpha$, where $P[(\cdot)]$ denotes the probability of the event $(\cdot)$, and α is a small real number representing low probability, which is usually set to be 0.05. The value of a_i^k corresponding to a given α can be calculated by setting $(\sigma_i^k / a_i^k)^2 = \alpha$ using the Chebyshev inequality. Without loss of generality, we can assume $\mu_i^k < \mu_j^k$. We let $\overline{p}_j = \min(1, [\sigma_j^k / \max(\delta, \mu_j^k - \mu_i^k - a_i^k)]^2)$, where a_i^k is as defined and δ is a very small positive real number to avoid the denominator of the square term being zero or negative. $\overline{p}_j$ is an upper bound of $P[|X_j^k - \mu_j^k| \geq \max(\delta, \mu_j^k - \mu_i^k - a_i^k)]$ based on the Chebyshev inequality. If μ_j^k is sufficiently larger than $\mu_i^k + a_i^k$, $\overline{p}_j$ will be very small, which implies the overlapping of the classes C_i and C_j on attribute k will be very small; consequently, the classes C_j and C_i are more likely to be separable, as illustrated in Fig. 4. Therefore, we can define a threshold value $\hat{p}$, such that the separation index for classes C_i and C_j can be calculated by

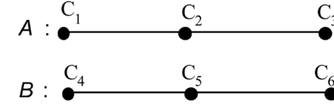
$$D(C_i, C_j)_k = \begin{cases} 0, & \text{if } \overline{p}_j < \hat{p} \\ 1, & \text{otherwise.} \end{cases} \tag{2}$$

Now we can define $[D(C_i, C_j)_k]$ as the separation matrix for all classes based on attribute k, whose (i,j)th entry is $D(C_i, C_j)_k$.

2) Splitting Cluster Using Separation Matrices: We let Cr_0 denote the root cluster, which represents the whole data set. Treating each class in Cr_0 as a node, we can view $[D(C_i, C_j)_{k_1}]$ as an incidence matrix for all nodes in Cr_0 based on attribute k_1. That means nodes C_i and C_j will be connected by an arc if $D(C_i, C_j)_{k_1} = 1$. The graph constructed based on a separation matrix is called a *separation graph*, which may contain separate connecting subgraphs. Each connecting subgraph represents a cluster of nonseparable classes based on attribute k_1, and the number of disjoint subgraphs represents the number of disjoint clusters that can be split from Cr_0 using attribute k_1. For example, the separation graph constructed from the separation matrix $[D(C_i, C_j)_{k_1}]$ given in Fig. 5(a) is shown in Fig. 5(b), which consists of two disjoint clusters or two separate connecting subgraphs $\mathfrak{A}$ and $\mathfrak{B}$. The resulting clusters can be further split by other attributes. For example, cluster $\mathfrak{A}$ in Fig. 5(b) can be further split by attribute k_2, whose $[D(C_i, C_j)_{k_2}]$ is shown in Fig. 6(a), in the following manner. Collecting the rows and columns of $[D(C_i, C_j)_{k_2}]$ corresponding to the classes in cluster $\mathfrak{A}$ forms

	C1	C2	C3	C4	C5	C6
C1	1	1	0	0	0	0
C2	1	1	1	0	0	0
C3	0	1	1	0	0	0
C4	0	0	0	1	1	0
C5	0	0	0	1	1	1
C6	0	0	0	0	1	1

(a)



(b)

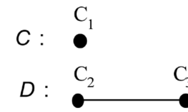
Fig. 5(a). Separation matrix example $[D(C_i, C_j)_{k_1}]$. (b) Separation graph example resulting from separation matrix in (a).

	C1	C2	C3	C4	C5	C6
C1	1	0	0	0	0	0
C2	0	1	1	0	0	0
C3	0	1	1	1	0	0
C4	0	0	1	1	1	0
C5	0	0	0	1	1	1
C6	0	0	0	0	1	1

(a)

	C1	C2	C3
C1	1	0	0
C2	0	1	1
C3	0	1	1

(b)



(c)

Fig. 6(a). Separation matrix $[D(C_i, C_j)_{k_2}]$. (b) Submatrix of $[D(C_i, C_j)_{k_2}]$ corresponding to cluster $\mathfrak{A}$ in Fig. 5(b). (c) Clusters split from cluster $\mathfrak{A}$ using submatrix shown in (b).

the submatrix shown in Fig. 6(b). Repeating the same process of splitting Cr_0 using $[D(C_i, C_j)_{k_1}]$, cluster $\mathfrak{A}$ can be split into two clusters $\mathfrak{C}$ and $\mathfrak{D}$, as shown in Fig. 6(c), by using the submatrix shown in Fig. 6(b).

3) Choice of Attributes for Cluster Splitting and Construction of Clustering Tree: Because the separation matrix has already indicated a certain distribution of the attribute values of all classes, we can employ a coarser partition like fuzzy intervals to classify the disjoint clusters instead of treating each continuous value as a discrete one like CART. In general, for a given

	C1	C2	C3	C4	C5	C6	C7	C8
C1	1	0	0	0	0	0	0	0
C2	0	1	1	0	0	0	0	0
C3	0	1	1	1	0	0	0	0
C4	0	0	1	1	0	0	0	0
C5	0	0	0	0	1	1	0	0
C6	0	0	0	0	1	1	1	0
C7	0	0	0	0	0	1	1	1
C8	0	0	0	0	0	0	1	1

(a)

	C1	C2	C3	C4	C5	C6	C7	C8
C1	1	1	0	0	0	0	0	0
C2	1	1	0	0	0	0	0	0
C3	0	0	1	1	0	0	0	0
C4	0	0	1	1	0	0	0	0
C5	0	0	0	0	1	1	0	0
C6	0	0	0	0	1	1	0	0
C7	0	0	0	0	0	0	1	1
C8	0	0	0	0	0	0	1	1

(b)

Fig. 7(a). Separation matrix $[D(C_i, C_j)_{k_1}]$. (b) Separation matrix $[D(C_i, C_j)_{k_2}]$.

range of attribute values, finer fuzzy partition is needed to classify a cluster with a larger number of classes. In other words, for a given fuzzy partition and the range of attribute values, the classification will be more accurate for a cluster with a smaller number of classes. Considering that any inaccurate cluster splitting will influence the accuracy of the subsequent cluster splitting along the tree path, we set the criteria for choosing the attribute to split a cluster as minimizing the multiplication of the *average number of classes* and the *variation of the number of classes in the resulted child clusters*. This criteria implies that the attribute which results in more child clusters and smaller variation in the number of classes in the child clusters is preferred. For example, for the separation matrices of two attributes shown in Fig. 7(a) and (b), suppose that we use the attribute k_1 to split the cluster first; we obtain three child clusters. One consists of one class, and the other two consist of three and four classes. If we use attribute k_2 first, we will obtain four child clusters, and each child cluster contains two classes. Based on the criteria indicated, we would choose k_2 to split the cluster. To put this criteria into a mathematical form, we let L_k and $n_{k,l}(\mathrm{Cr}_j)$ denote the number of child clusters and the number of classes in the lth child cluster resulted from using attribute k to split the cluster Cr_j, respectively. Then, the criteria for choosing the attribute for splitting Cr_j is

$$\min_k \left\{ \left(\sum_{l=1}^{L_k} n_{k,l}(\mathrm{Cr}_j) \Big/ L_k \right) \cdot \left(\sum_{l=1}^{L_k} \left(n_{k,l}(\mathrm{Cr}_j) - \overline{n}_{k,l}(\mathrm{Cr}_j)\right)^2 \Big/ L_k \right) \right\} \quad (3)$$

where the first term inside the big bracket represents the average number of classes in the resulting child clusters and the second term denotes the variance of the number of classes in the resulting child clusters, where $\overline{n}_{k,l}(\mathrm{Cr}_j) \equiv \sum_{l=1}^{L_k} n_{k,l}(\mathrm{Cr}_j)/L_k$.

Now, our algorithm for choosing the splitting attribute to build the *clustering tree* can be stated as follows.

Algorithm I: Choose the splitting attributes and build the clustering tree

Step 0) Given the original data set Cr_0 and the separation matrices of all attributes. Set Cr_0 as the root cluster and define the set of yet split clusters (YSC) as $(\mathrm{YSC}) = \{\mathrm{Cr}_0\}$.

Step 1) For each cluster in YSC, obtain the corresponding splitting attribute k based on criteria (3). Use the obtained attribute to split the cluster, and put the resulting child clusters into YSC. Discard the clusters that had been split and the clusters that cannot be split using any attribute.

Step 2) If $\mathrm{YSC} = \phi$, stop; otherwise, return to Step 1).

Fig. 8 shows a clustering tree built by using the separation matrices of two attributes shown in Figs. 5(a) and 6(a) to split the root cluster $\mathrm{Cr}_0 = \{C_1, C_2, C_3, C_4, C_5, C_6\}$. Algorithm I uses three iterations to build the tree. The splitting attribute for each cluster and the progression of YSC are also shown in this figure.

We define the leaf cluster in the clustering tree as the *terminal cluster* (TC). Each TC may contain one class only or several classes, which cannot be split further using any attribute. For the purpose of classifying a new data pattern into a TC, we need to use the splitting attributes to construct the cluster splitting rules for each cluster in the clustering tree based on the fuzzy rules [33], [34] for a single attribute as presented in the following section. It should be noted that the fuzzy rules employed here are for single attribute; thus, we can circumvent the computational complexity of the fuzzy set theory-based approach as indicated in Section II-A.

4) Clustering Algorithm: The separation matrix-based clustering algorithm consists of two parts: the training part and the classification part. The training part, which is prepared for the classification part, consists of three steps: 1) construction of the separation matrices for all attributes; 2) determining the cluster-splitting attribute and building the clustering tree; and 3) throughout the clustering tree, generate the fuzzy if–then rules needed to classify a data pattern into a proper child cluster based on a given set of training data patterns with known TCs. Of three steps, 1) and 2) have been presented in previous sections. The details of 3) as well as the classification part are described as follows.

a) Fuzzy-rule generation procedures of clustering algorithm: The fuzzy rules for splitting a non-TC cluster using the corresponding splitting attribute in our clustering algorithm are of the same type. Thus, for the sake of explanation, we will focus on generating the fuzzy rules for one cluster in the clustering tree. We let Cr_j denote a non-TC cluster and k denote the corresponding splitting attribute. We let x_k^s, $s = 1, \ldots, g$, denote the

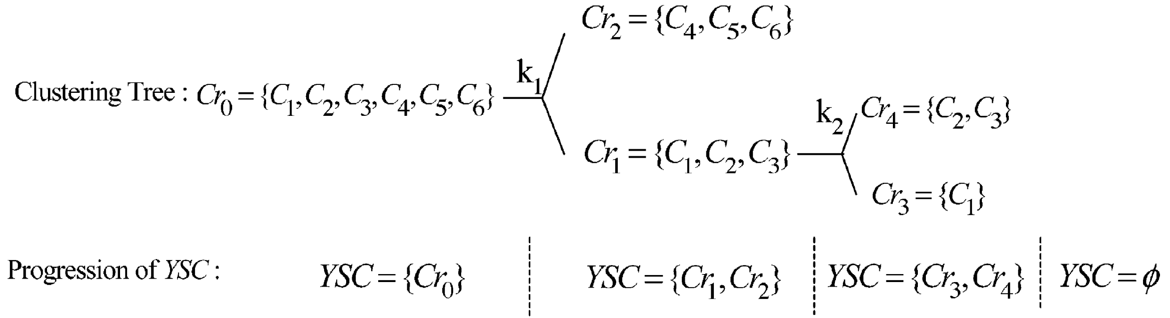


Fig. 8. Example of using Algorithm I to build clustering tree.

kth attribute of g data patterns, and x^s, $s = 1, \ldots, g$, from M_j known child clusters, $\mathrm{CCr}_{j1}, \ldots, \mathrm{CCr}_{jM_j}$. These g data patterns form the training data set for splitting Cr_j. The fuzzy rules for splitting a cluster Cr_j are of the following type.

For $i = 1, \ldots, K$, where K denotes the number of fuzzy partitioned intervals on the range of the kth attribute values

Rule $R_i(\mathrm{Cr}_j)$: If x_k^s is A_i^K, then the x^s belongs to CCr_{ji} with CF_i^K, where A_i^K is the ith partitioned fuzzy interval, CCr_{ji} is the consequent, i.e., one of the M_j child clusters, and CF_i^K is the grade of certainty of rule $R_i(\mathrm{Cr}_j)$. (4)

What need to be determined in the previous rule are CCr_{ji} and CF_i^K, and the procedures for determining them are called fuzzy rules generation procedures for splitting one cluster, described as follows.

Let A_i^K be characterized by the nonnegative fuzzy membership function $f_i^K(\cdot)$. The membership function $f_i^K(\cdot)$ can be triangular, Gaussian, or any other shape. In this paper, we consider the triangular membership function. Then, $f_i^K(x_k^s)$ can be considered as the grade of compatibility of x_k^s with respect to A_i^K. We define

$$\beta_{\mathrm{CCr}_{jl}}(R_i(\mathrm{Cr}_j)) = \sum_{x^s \in \mathrm{CCr}_{jl}} f_i^K(x_k^s) \tag{5}$$

as the sum of grade of compatibility of child cluster CCr_{jl} with respect to A_i^K. Then, the algorithm for generating fuzzy rules for splitting cluster Cr_j can be stated as follows.

Algorithm II: Generation of the K fuzzy rules for splitting cluster Cr_j

Step 0) Given g training data patterns x^s, $s = 1, \ldots, g$, with known child clusters $C\mathrm{Cr}_{jl}, l = 1, \ldots, M_j$ of the to-be-split cluster Cr_j and the corresponding splitting attribute k. Set $i = 1$.

Step 1) Calculate the sum of the rade of compatibility of child cluster CCr_{jl}, $l = 1, \ldots, M_j$, with respect to A_i^K by (5).

Step 2) Find the child cluster CCr_{jx} such that

$$\beta_{\mathrm{CCr}_{jx}}(R_i(\mathrm{Cr}_j)) = \max\left\{\beta_{\mathrm{CCr}_{j1}}(R_i(\mathrm{Cr}_j)), \ldots, \beta_{\mathrm{CCr}_{jM_j}}(R_i(\mathrm{Cr}_j))\right\} \tag{6}$$

then CCr_{jx} is the consequent CCr_{ji} in rule $R_i(\mathrm{Cr}_j)$.

Step 3) Determine CF_i^K, the grade of certainty of rule $R_i(\mathrm{Cr}_j)$, by

$$CF_i^K = \left(\beta_{\mathrm{CCr}_{jx}}(R_i(\mathrm{Cr}_j)) - \beta(R_i(\mathrm{Cr}_j))\right) \Big/ \sum_{l=1}^{M} \beta_{\mathrm{CCr}_{jl}}(R_i(\mathrm{Cr}_j)) \tag{7}$$

where $\beta(R_i(\mathrm{Cr}_j)) = \sum_{\mathrm{CCr}_{jl} \neq \mathrm{CCr}_{jx}} \beta_{\mathrm{CCr}_{jl}}(R_i(\mathrm{Cr}_j))/(M_j - 1)$ denotes the average of the sum of grade of compatibility of the rest of the child clusters with respect to A_i^K.

Step 4) If $i = K$, stop; else, set $i = i + 1$, and return to Step 1).

b) Training part of clustering algorithm: Combining the construction of separation matrices, determination of the splitting attributes, building of the clustering tree, and the fuzzy rule generation procedures, we are ready to summarize the training procedures of the clustering algorithm using the training data set.

Algorithm III: Training procedures of the clustering algorithm

Step 0) Given a set of training data patterns with known classes; compute μ_i^k and σ_i^k of each class C_i and each attribute k; compute the separation matrices $[D(C_i, C_j)_k]$ based on (2) for each attribute k.

Step 1) Apply Algorithm I to obtain the splitting attributes and build the clustering tree.

Step 2) Use Algorithm II to generate the fuzzy rules for each cluster in the clustering tree.

c) Classification part of clustering algorithm: Once the fuzzy rules for splitting the clusters in the clustering tree are generated, we can determine the child cluster to which the new data pattern belongs at each cluster based on a fuzzy reasoning method.

Let the new data pattern be x' and let x'_k be the kth attribute of x' corresponding to the splitting attribute k at cluster Cr_j. We define $\gamma_{\mathrm{CCr}_{jl}}$, the weighting grade of certainty of x'_k with respect to the child cluster CCr_{jl}, as the sum of the multiplication of the grade of compatibility of x'_k with respect to A_i^K and the grade of certainty of rule $R_i(\mathrm{Cr}_j)$ over all K trained rules the consequent of which are CCr_{jl}. We can express $\gamma_{\mathrm{CCr}_{jl}}$ mathematically as $\gamma_{\mathrm{CCr}_{jl}} = \sum_{R_i(\mathrm{Cr}_j),\mathrm{CCr}_{ji}=\mathrm{CCr}_{jl}} f_i^K(x'_k) \cdot CF_i^K$. Then, the classification procedures for the new data can be stated as follows.

Classification Procedures: The child cluster CCr_{jy}, with respect to which the weighting grade of certainty of x'_k is maximum, is the concluded cluster of x', that is, $\mathrm{CCr}_{jy} = \arg(\max\{\gamma_{\mathrm{CCr}_{j1}}, \ldots, \gamma_{\mathrm{CCr}_{jM_j}}\})$.

Now, the classification procedures for classifying a new data pattern x' into a TC can be stated.

Algorithm IV: Classification procedures of the clustering algorithm

Step 0) Given a new data pattern $x' = (x'_1, \ldots, x'_n)$, where n denotes the total number of attributes; set Present Cluster $(\mathrm{PCr}) = \mathrm{Cr}_0$.

Step 1) Use x'_k, where k corresponds to the attribute used for splitting the PCr, and the classification procedures stated above to classify x' into a child cluster of PCr, we denote this child cluster by CPCr. If the CPCr is not a TC, set PCr $=$ CPCr and repeat this step; otherwise, stop.

C. CART for TC

The TCs resulting from the training part of the separation matrix-based clustering algorithm may consist of one or more classes. Since the number of classes and the size of the corresponding data set in each TC should be much smaller than Cr_0, it will be computationally much easier to apply CART to classify the TCs and the resulting tree size of each TC will be much smaller. Therefore, our clustering algorithm helps reduce the computational complexity and the tree size of CART when applied to Cr_0 alone.

The CART is a well-developed classification tool. The details of this classification technique can be found everywhere [17], [18]. Similar to the proposed clustering algorithm, CART also consists of training and classification parts. The training part of CART is to build a classification tree and the splitting rules in each node. In brief, the construction of a CART classification tree and splitting rules centers on three major elements: 1) the splitting rule; 2) the goodness-of-split criteria; and 3) the criteria for choosing an optimal or final tree for analysis. Regarding 1), there are three major splitting rules in CART. The one we employed here is the Gini's criteria [17]. This criteria starts the tree-building process by partitioning the TC into binary nodes based upon a very simple question of the form: "is $k \le b$?" where k is an attribute and b is a real number. Regarding 2), the CART uses a computation-intensive algorithm that searches for the best split at all possible split points for each attribute that decreases the Gini's impurity measure [17] most. CART will recursively apply this splitting rule to split nonterminal child nodes at each successive stage. In order to reduce the complexity of the built tree which is measured by the number of its terminal nodes, CART uses a pruning process to find an optimal tree, as pointed out in 3). The computational complexity of the training part of CART mainly lies in the exhaustive search for the best split required in 2). Once the classification tree and the splitting rules are obtained, the classification procedure of CART is simply asking whether $k \le b$ to determine which of the binary child nodes the new data pattern belongs to throughout the classification tree.

D. Classification of New Data Pattern

Once the training part of the HCT, which combines the training parts of the clustering algorithm and CART, is completed, we are ready to use the classification procedures of both clustering algorithm and CART to classify a new data pattern, as required in the first two blocks in Fig. 2.

E. Learning Capability

The learning capability of a classifier is very important in current application, because for every 14 min, 24 wafers (or a lot) of the same recipe will be ion implanted. Thus, new data patterns arrive with a high frequency. For the sake of explanation, we can assume the recipe of the working wafer is one of the recipes under work, because only a slight modification is needed for the case of a new recipe. The learning of HCT after the new data pattern joins in consists of two parts. The first part is for the clustering algorithm and the second part is for CART. Learning of the clustering algorithm consists of three updating steps: 1) updating the separation matrices; 2) updating the attributes used to split clusters as well as the clustering tree; and 3) updating the fuzzy rules for splitting clusters. Learning of CART is just to update the best split for each node in the classification tree.

1) Learning of Clustering Algorithm: Since the new lot of wafers is of the same recipe, the new data patterns will be used to update the mean and variance of each attribute of the corresponding class. Denoting the class index of the new data pattern by m, we will update μ_m^k and σ_m^k for all k, which will be used to update the separation indexes $D(C_m, C_j)_k$ for all j, all k. Suppose the updated $D(C_m, C_j)_k$ do not change for all j and all k, then the separation matrices remain the same; consequently, the splitting attributes for clusters and the clustering tree also remain the same as can be observed from Algorithm I. This implies that if the separation matrices are unchanged after the new data pattern joins in, the updating step 2) can be skipped. In fact, μ_m^k and σ_m^k will only slightly deteriorate when the new data patterns join in because of the large amount of training data set. This implies that the updated separation matrices may change only when the amount of accumulated new data patterns are large enough. On the other hand, suppose $D(C_m, C_j)_k$ changes for any j and k and causes the corresponding separation matrices to be changed in updating step 1); we need to proceed with updating step 2) by performing Algorithm I (i.e., Step 1 of Algorithm III) to update the splitting attributes and the clustering tree.

To update the fuzzy rules indicated in the updating Step 3), we also consider two cases. In the case of unchanged separation matrices, which implies the clustering tree and splitting attributes remain the same, we only need to update the fuzzy rules for the clusters in the tree path of the clustering tree, in which the new data pattern belongs. To do so, we let Cr_j be a non-TC cluster in this tree path and let CCr_{jz} be the child cluster of Cr_j in this tree path. To update the rules $R_i(\mathrm{Cr}_j)$ in (4) is to update the result and grade of certainty after the new data patterns join in. To update the consequent, we need to update $\beta_{\mathrm{CCr}_{jz}}(R_i(\mathrm{Cr}_j))$ first. To do so, we need to add an extra term of the nonnegative membership function of the new data pattern on the right-hand side of (5). The updated $\beta_{\mathrm{CCr}_{jz}}(R_i(\mathrm{Cr}_j))$ will be larger than the original one. Thus, according to Step 2) of Algorithm II, the consequent will not be changed. Subsequently, we can use the updated $\beta_{\mathrm{CCr}_{jz}}(R_i(\mathrm{Cr}_j))$ to update the corresponding CF_i^K by (7). Thus, in this case, updating fuzzy rules is an easy task because the length of a tree path in the clustering tree is usually short. In the case where the clustering tree or any splitting attributes change due to the changed separation matrices, we need to perform Step 2) of Algorithm III, which is Algorithm II, to update the fuzzy rules. Of course, this is more complicated than in the previous case. However, no matter what case, it will not affect HCT to work in real time and online as will be demonstrated in Section IV.

2) Learning of CART: Following previous discussions, there are also two cases for updating the splitting rules of CART. The first case is a subsequent situation of the unchanged separation matrices such that the TCs of the clustering tree do not change. Since the number of training data patterns is very large, the best split point of each attribute in each node of the CART will alter, at most, slightly when new data patterns join in. Therefore, we need not exhaustively search for the split point of each attribute. Instead, we can search for the split point only within a window of the original best split point of each attribute. The window is set to be $\pm w$ discrete points at the best split point of each attribute. This will, of course, save a lot of computation time. In addition, we need only update the splitting rules for just one TC, to which the new data pattern belongs. The other case is when the separation matrices change and cause the clustering tree changes. In this case, we will rerun the CART for all TCs. As indicated at the end of previous section, this will not affect HCT to work in real time and online.

III. Warning Signal Generation and Fault Isolation

A. Warning Signal Generation

In general, the ion implanter will be stopped whenever there is a warning signal so as not to damage the subsequent wafers. However, this reaction will be justified only when the warning signal is absolutely correct; otherwise, the throughput will be degraded. Thus, to minimize the probability of false alarms should be one of the objectives. On the other hand, thousands of wafers may be damaged if any fault is not detected. Thus, to minimize the probability of overlooking a fault is another objective. In general, a matched classification result implies: 1) the machine is in normal condition or 2) the actual implantation has been wrong due to a machine fault but a misclassification makes the classified recipe match the destined one. Case 2) indicates a fault situation that cannot be observed from the matched result. We let η_i denote the misclassification rate of recipe i, which can be calculated by

$$\eta_i = \sum_{j \neq i} \pi(j) m(i|j) \tag{8}$$

where $\pi(j)$ denotes the prior probability of recipe j and $m(i|j)$ denotes the misclassification rate of classifying recipe j to be recipe i. If Case 2) occurs to recipe i, then the probability of a series of n such events occurring is η_i^n. This indicates the probability of an undetected machine fault will be extremely small, provided that η_i is small and n is large. This also implies that the matched recipe will eventually mismatch, provided that the matched result is due to a misclassification. Real values of η_i for all i based on HCT will be given through the tests presented in Section IV. This addresses the comment cited in Section I that we need not check the existence of a machine fault when the classified recipe matches the destined one, and the cost of such a reaction is at most n damaged wafers, where n is a positive integer that makes η_i^n extremely small. This also indicates when the classified recipe matches the destined one, we can continue for the next wafer as shown in Fig. 2. Thus, using the classification accuracy of the HCT as the basis of generating a warning signal, our objective can be simplified to minimizing the probability of a false alarm.

There are two causes of false alarms. One is the electrical spike and the other is the classification error. Both cases will cause the classified recipe to mismatch the destined one and require the checking of warning signal generation criteria as indicated in Fig. 2. To minimize the probability of a false alarm due to an electrical spike, we should distinguish an electrical spike from a machine fault. The electrical spike is only temporary, which may affect one or two wafers only, while the machine fault will last until it is fixed. Thus, an easier way to distinguish them is checking whether a series of classification errors occur. In other words, if there are more than, say, *four* consecutive classification errors, the causes of the errors should not be the electrical spikes. Similar reasons apply to the classification errors. We let q_i denote the classification error rate, which is defined as (number of misclassified wafers/number of test wafers) $*100\%$ of recipe i obtained using k-fold cross validation. Then, the probability of the occurrence of n consecutive classification errors is $(q_i)^n$, which decreases sharply when n increases. Thus, an easier way to distinguish the classification error from the machine fault is also checking whether a series of classification errors occur. To achieve this, we can predetermine a very small positive real number ε, a probability indication of an event that is almost not possible to occur. Then, if $(q_i)^n < \varepsilon$, we can conclude that the cause of the mismatched recipe is not classification errors. Thus, we can state our *warning signal generation criteria* as follows.

Let the classification error rate of recipe i be obtained using k-fold cross validation denoted by q_i, and let n denote the number of consecutive working wafers; then, the proposed criteria for generating a warning signal is as follows.

Assume the classified recipe of the $(l-1)$th wafer matches the destined one, while the $lth, (l+1)th, \ldots, (l+n)th$ wafers do not; the warning signal will be generated at the $(l+n)th$ wafer provided that the following two conditions hold:

$$q_{i_l}(l) \times q_{i_{l+1}}(l+1) \times \cdots \times q_{i_{l+n}}(l+n) \leq \varepsilon \tag{9}$$

and

$$n \geq n_1 \tag{10}$$

where i_l denotes the destined recipe of the lth wafer, $q_i(l)$ denotes the q_i of the lth wafer, ε is a very small positive real number, and n_1 denotes the maximum number of consecutive wafers that can be affected by the electrical spikes.

If (9) holds, we can exclude the possibility of false alarm due to classification errors. If (10) holds, we can exclude the possibility of false alarm due to the electrical spikes.

B. Fault Isolation

To eliminate the machine fault, we need to isolate the fault first. In general, when there is a fault in a subsystem, the attribute (or attributes) corresponding to that subsystem may become abnormal. Thus, the basic idea of our fault isolation scheme is to find the attribute(s) that causes the classification errors, and this can be easily done in a single-tree classifier like CART and HCT, which is their biggest advantage, the interpretability. In fact, the tree structure of HCT is much simpler than CART, because it largely reduces the tree size of CART by using the clustering tree to separate the whole data set into several TCs. Thus, if the misclassified recipe and the destined recipe belong to different TCs, we can use the clustering tree to find the faulty attribute. If they belong to the same TC, we will use the corresponding CART to find the faulty attribute. Considering that the machine fault may occur abruptly or develop gradually, and there may be single or multiple faulty attributes, we will find the faulty attribute(s) for each misclassified wafer by the aid of its tree path and the tree paths of several latest correctly classified wafers of the same destined recipe. Thus, once a warning signal is generated, our fault isolation scheme will proceed as follows.

Step 1) Collect the m_1 consecutive misclassified wafers that cause the warning signal, i.e., $m_1 = \max(n, n_1)$ such that (9) and (10) hold.

Step 2) Collect the latest m_2 correctly classified wafers, which have the same destined recipes as the m_1 wafers in Step 1).

Step 3) For each of the m_1 wafers in Step 1) and each of the m_2 wafers in Step 2), we will find the faulty attribute(s) that causes the misclassification as follows.

- 3.1) Suppose the two wafers belong to different TCs, say TC_i and TC_k, we will use the clustering tree to find the faulty attribute by tracing the tree paths backward from the corresponding TCs. These two paths will meet at a node whose splitting attribute will be the faulty attribute. As illustrated in Fig. 9, the faulty attribute is k_1.
- 3.2) Suppose the two wafers belong to the same TC and they lie in two different terminal nodes of the corresponding CART, we can find the faulty attribute in a similar manner as in Step 3.1) by using the classification tree of CART.

Step 4) List all the different faulty attributes found from the $m_1 \times m_2$ searches in Step 3) and calculate the corresponding probability, based on the frequency of occurrences. Indicate the corresponding subsystem of the faulty attributes and calculate the corresponding probability by adding the probabilities of the faulty attributes in this subsystem.

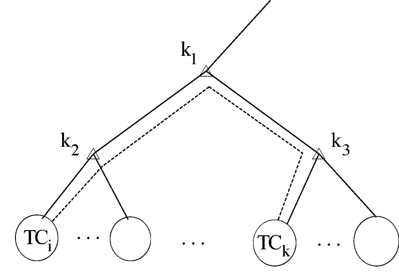


Fig. 9. Using clustering tree to find faulty attribute.

IV. Test Results of HCT, Warning Signal Generation, and Fault Isolation

A. Test Results of HCT

In general, there are quite a few attributes that can be measured from the ion implanter; however, not all attributes are helpful in classification. According to the domain knowledge, the following 12 attributes, $k_1, \ldots, k_{11}$ and k_{12}, are recommended: filament voltage, filament current, discharge voltage, discharge current, extraction electrode voltage, extraction electrode current, acceleration/deceleration voltage, magnetic field strength, high-voltage power supply current, beam current, beam-line pressure, and chamber pressure, respectively. These 12 attributes cover the four subsystems of the ion implanter. Table I shows the units and related subsystems of the above 12 attributes. We have made all the tests on a 26-recipe case and a 42-recipe case. Due to the page limitation, we will present the complicated 42-recipe case only.

A data set of the 42-recipe case and each recipe consists of a thousand to 10 000 wafers supported from a local world-renowned foundry. We use them to test the classification accuracy of the proposed classifier HCT and to demonstrate the validity of the warning signal generation criteria and fault isolation scheme. It takes 1 s to measure a 12-attribute data pattern. The ion implantation time for a wafer is around 10 s. Thus, ten data patterns are taken while a wafer is under work. The wafer changeover time is 26 s, on average. Each lot contains 24 wafers, and the setup time for a new lot is 13 min. For all the measured data patterns in this case, we randomly divide them by wafer base into ten parts. We take nine parts as training data set and

TABLE I
UNITS AND RELATED SUBSYSTEMS OF 12 ATTRIBUTES

Attribute		Unit	Subsystem
k_1	filament voltage	Volts	ion source
k_2	filament current	Amps	ion source
k_3	discharge voltage	Volts	ion source
k_4	discharge current	Amps	ion source
k_5	extraction electrode voltage	KV	extractor
k_6	extraction electrode current	mA	extractor
k_7	acceleration/deceleration voltage	KV	extractor
k_8	magnetic field strength	KGauss	mass analysis
k_9	high voltage power supply current	μA	mass analysis
k_{10}	beam current	mA	mass analysis
k_{11}	beam-line pressure	Torr/e6	accelerator
k_{12}	chamber pressure	Torr/e6	accelerator

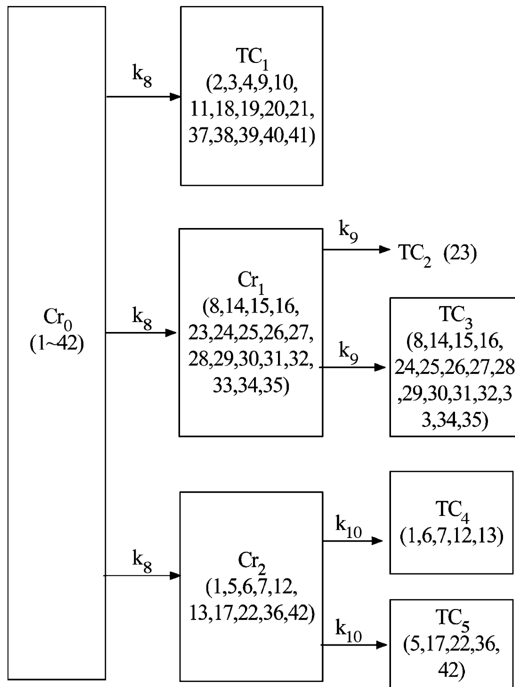


Fig. 10. Cluster splitting tree of 42-recipe case.

one part as test data set. We set $\hat{p} = 0.075$ in (2), the number of fuzzy partitioned intervals $K = 12$, and a triangle nonnegative membership function for $f_i^K(\cdot)$ in Algorithm II. Applying Algorithm III to the training data set, the resulting clustering tree and the splitting attributes are shown in Fig. 10, where each cluster is denoted by a block, and the recipes contained in a cluster are shown inside the parenthesis in each block. The attribute used for a splitting each cluster is indicated at the outgoing branch in the clustering tree. The corresponding fuzzy rules for each splitting attribute are also obtained. There are five TCs, and each TC consists of more than one recipe except for the one consisting of recipe 23 only. Subsequently, we apply CART to the four TCs and build the classification tree and splitting rules for each TC. We then use the part of the test data to test the trained HCT using Algorithm IV of the clustering algorithm and the classification tree and splitting rules of CART. Since each wafer corresponds to ten measured data patterns, and each test data pattern will be classified to a recipe, a *majority voting scheme* is used to conclude the classified recipe of the wafer corresponding to the ten test data patterns. Repeating this process for ten times by circulating the training data set and test data set, Table II shows the resulting tenfold cross-validation classification error rate of all recipes in this test. We also indicate the tenfold cross-validation classification error rate using the software See5 [35] and CART [17] in this table. From this table, we can calculate the sum of classification error rates of the proposed HCT with $\hat{p} = 0.075$ is around 0.2955%; while the sum of classification error rates using See5 and CART are 0.53% and 0.6427%, which are 80% and 117% more than that of HCT, respectively. Thus, HCT obtains a very successful classification result.

Remark: From the test results shown in Table II, we see that the superiority of HCT over CART and See5 is mostly due to the zero classification errors of recipe 14. What causes the classification errors of recipe 14 in CART or See5 is the overlapping of the attribute data between recipes 14 and 20. Thus, some test data patterns of recipe 14 may be classified to be recipe 20 in CART or See5. Fortunately, in HCT, recipes 14 and 20 have been classified into different TCs as can be observed from Fig. 10. This drastically reduces the possibility of classifying recipe 14 to be 20. However, in HCT, recipe 20 may still be classified into recipe 14, which can also be observed from Table III in misclassification rate. Excluding recipe 14 from the data set, we repeat the complete training and test process, and the results show that the sum of classification error rates of HCT, CART, and See5 are 0.173, 0.248, and 0.182, respectively. Indeed, the three sums of classification errors are closer; however, HCT is still the best among them. Furthermore, we also apply the three classifiers to the 26-recipe case that we mentioned at the beginning of this section, and the sum of classification error rates of HCT, CART and See5 are 0.225, 0.577, and 0.405, respectively.

For this 42-recipe case, we also obtain the misclassification rate defined in (8) for the three classifiers as shown in Table III. The largest misclassification rate of HCT, $\eta_{\max} = \max_i \eta_i$, is 0.0043% and the sum of misclassification rates $\sum_{i=1}^{42} \eta_i$ is around 0.00737%. Taking $n = 4$, $\eta_{\max}^n < 10^{-15}$. This demonstrates the analysis stated in Section III for the validity of no-fault assumption, which states that if a machine fault exists, the classified recipe will eventually mismatch the destined one. Compared with CART and See5, the sum of misclassification rates of HCT is better, and this is consistent with the results of the classification error rate shown in Table II. To investigate the training efficiency and the capability of real-time classification of HCT as well as the effects of different values of $\hat{p}$, we have applied HCT to the 42-recipe case with three other value of $\hat{p}$. The resulting tenfold cross validation for the sum of classification error rates, the corresponding average training times, and the classification time for classifying the recipe of a new data pattern are shown in Table IV. From the fourth row of this table, we can observe that when $\hat{p} \leq 0.075$, the tenfold cross validation for the sum of classification error rates of HCT is better than that of See5 and CART. From the second row of the table, we see that when $\hat{p} \geq 0.075$, the training time required by HCT is much shorter than that required by CART and See5. The classification time needed for classifying a new data pattern is much shorter than that of See5 and CART for all the indicated values of $\hat{p}$; in addition, it is also much shorter than the data measurement time, which takes 1 s; thus, HCT can work in real

TABLE II
CLASSIFICATION ERROR RATE OF 42-RECIPE CASE

Recipe	Classification error rate (%) HCT $\hat{p}$=0.075	CART	See5	Recipe	Classification error rate (%) HCT $\hat{p}$=0.075	CART	See5	Recipe	Classification error rate (%) HCT $\hat{p}$=0.075	CART	See5
1	0	0	0	15	0	0	0	29	0	0	0
2	0	0	0	16	0	0	0	30	0	0	0
3	0	0	0	17	0	0	0	31	0	0	0
4	0	0	0	18	0	0	0	32	0	0	0
5	0	0	0	19	0	0	0	33	0	0	0
6	0.05	0.05	0.05	20	0.2	0.125	0.2	34	0	0	0
7	0	0	0	21	0	0	0	35	0	0	0
8	0	0	0	22	0	0	0	36	0	0	0
9	0	0	0	23	0	0	0	37	0	0	0
10	0	0.05	0	24	0	0	0	38	0	0	0
11	0	0	0	25	0	0	0	39	0.0455	0	0
12	0	0	0	26	0	0	0	40	0	0	0
13	0	0	0	27	0	0	0	41	0	0	0
14	0	0.4167	0.28	28	0	0	0	42	0	0	0

TABLE III
MISCLASSIFICATION RATE η OF 42-RECIPE CASE

Recipe	Misclassification rate (%) HCT $\hat{p}$=0.075	CART	See5	Recipe	Misclassification rate (%) HCT $\hat{p}$=0.075	CART	See5	Recipe	Misclassification rate (%) HCT $\hat{p}$=0.075	CART	See5
1	0	0	0	15	0	0	0	29	0	0	0
2	0	0	0	16	0	0	0	30	0	0	0
3	0	0	0	17	0	0	0	31	0	0	0
4	0	0	0	18	0	0	0	32	0	0	0
5	0	0	0	19	0.0006	0.0049	0.0021	33	0.00124	0	0
6	0	0	0	20	0	0.0062	0.0053	34	0	0	0
7	0.00123	0.00123	0.00123	21	0	0	0	35	0	0	0
8	0	0	0	22	0	0	0	36	0	0	0
9	0	0	0	23	0	0	0	37	0	0	0
10	0	0	0	24	0	0	0	38	0	0	0
11	0	0	0	25	0	0	0	39	0	0	0
12	0	0	0	26	0	0	0	40	0	0	0
13	0	0	0	27	0	0	0	41	0	0	0
14	0.0043	0.0025	0.0037	28	0	0	0	42	0	0	0

TABLE IV
TRAINING TIME, CLASSIFICATION TIME, AND TENFOLD CROSS VALIDATION FOR SUM OF CLASSIFICATION ERROR RATES FOR DIFFERENT VALUES OF $\hat{p}$

Classifier	HCT $\hat{p}$=0.05	HCT $\hat{p}$=0.075	HCT $\hat{p}$=0.08	HCT $\hat{p}$=0.13	CART	See5
Training time (sec)	28.105	24.295	22.853	19.246	38.765	26.71
Classification time (sec)	0.052	0.047	0.041	0.037	0.098	0.093
10-fold cross validation for the sum of classification error rates (%)	0.2500	0.2955	0.8455	2.9100	0.6427	0.5300

time. This shows that HCT not only performs better than See5 and CART in the aspect of a tenfold cross validation for the sum of classification error rates but also consumes less training time and classification time when $\hat{p}$ is properly chosen. In the meantime, we found that as $\hat{p}$ increases, the HCT becomes less accurate and less computationally time consuming, as expected. This also demonstrates why the clustering algorithm helps reduce the computational complexity of CART.

TABLE V
UPDATING TIME OF HCT WHEN SEPARATION MATRICES CHANGE FOR DIFFERENT $\hat{p}$

$\hat{p}$	0.05	0.075	0.08	0.13
Updating time of HCT(sec)	24.619	21.741	18.295	16.831

B. *Test Results of Learning Capability of HCT*

We also test the learning capability of the proposed HCT by adding the new data patterns to the training data set. We found that when the accumulated amount of new data patterns is less than 7%, on average, of the amount of training data of the same recipe, the updated separation matrices remain the same. The length of the window in updating the splitting rules of CART, w, is set to be 5. In the case of unchanged separation matrices, the computation time for checking whether there is any change in separation matrices, updating the fuzzy rules of the clustering

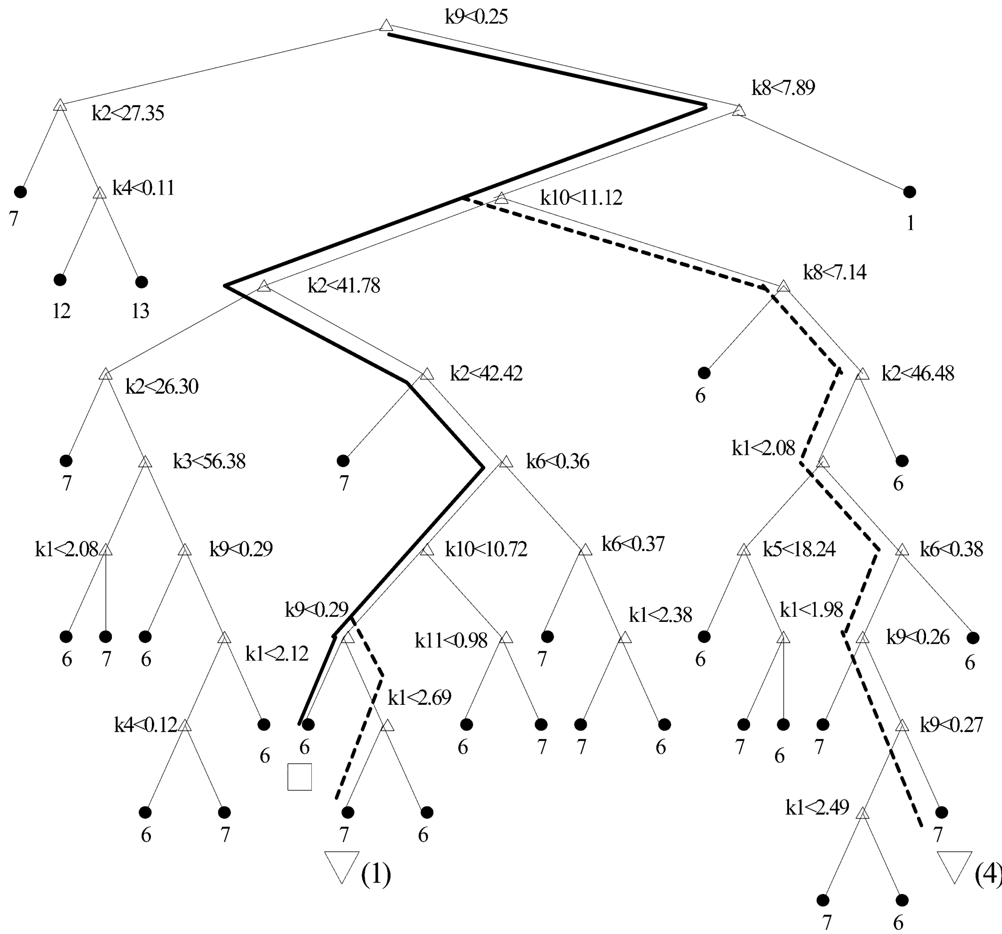


Fig. 11. Classification tree of CART for TC_4.

algorithm, and the splitting rules for CART take only 0.1637 s for each new data pattern when $\hat{p} = 0.075$. This updating time is shorter than measuring a new data pattern; thus, we can perform the online update. In the case when separation matrices change, updating the separation matrices and rerunning Steps 1) and 2) of Algorithm III and the training part of CART for the resulting TCs take only 21.741 s, which is even shorter than the wafer changeover time, which takes 26 s. For different values of $\hat{p}$, the updating times of HCT when separation matrices change are shown in Table V. From the results, we see that we can update the training part of HCT during the wafer changeover period. This indicates that the learning capability of HCT enables it to update in real time and online. It should be noted that the HCT's updating time being shorter than the training time is because updating the separation matrices is much easier than constructing from nothing.

C. Test Results of Warning Signal Generation and Fault Isolation

To test the validity of the proposed warning signal generation criteria and fault isolation scheme, we use six small sets of measured data patterns, which are also collected from the 42-recipe case but not included in the previous data set for constructing the HCT. Among them, the first two sets consist of abnormal wafers caused by machine faults, and the other four sets consist of abnormal wafers caused by electrical spikes. There are 50 wafers with destined recipe 39 in the first set and the ten abnormal wafers located from the 21st to the 30th wafers caused by attribute k_8. The second set consists of 40 wafers with destined recipe 6 and the ten abnormal wafers located from the 31st to the 40th wafers. The first abnormal wafer is caused by attribute k_9, and the remaining nine wafers are caused by both k_9 and k_{10}. The third set consists of 20 wafers, and the two abnormal wafers are located at the 16th and 17th wafers caused by the attribute k_4, whose values are affected by electrical spikes. The abnormal wafers caused by electrical spikes also occur to the fourth, fifth, and the sixth sets of wafers; these three sets consist of 30 wafers each, and the two abnormal wafers are located at the 19th and 20th, 23rd and 24th, and 27th and 28th wafers caused by attributes k_6, k_9, and k_{10}, respectively. We randomly pick six out of ten existing HCT test data sets and insert the previous six small sets of data patterns into the six test data sets, one for each.

Setting $\varepsilon = 10^{-15}$, $n_1 = 4$, and q_i of each recipe i as the result shown in Table II, we apply the HCT associated with the majority voting scheme to classify the six test data sets. In the first test data set, the warning signal generation criteria, i.e., (9) and (10), are satisfied at the fifth abnormal wafer of the first small data set, because $q_{39} = 0.0455\%$ according to Table II.

TABLE VI
MISCLASSIFIED ABNORMAL WAFERS CAUSED BY ELECTRICAL SPIKES

Data set	No. of abnormal wafers	Faulty attribute	Destined recipe	Classified recipe
3rd	2	k_4	6	7
4th	2	k_6	14	24
5th	2	k_9	20	41
6th	2	k_{10}	39	40

Thus $q_{39}^5 < \varepsilon = 10^{-15}$ and $n_1 = 4 < 5$. This demonstrates that our warning signal generation criteria has successfully detected the fault. Now, we have $m_1 = 5$ according to Step 1) of the fault isolation scheme. We also set $m_2 = 5$, which denotes the 16th to the 20th wafers in the first small set of test data. The five misclassified wafers are all classified to recipe 24, while the previous five wafers are correctly classified to recipe 39. Since recipes 24 and 39 belong to different TCs, we apply Step 3.1) of the fault isolation scheme and find that there are only two traced-back tree paths, which are $\mathrm{TC}_3 - \mathrm{Cr}_1 - \mathrm{Cr}_0$ and $\mathrm{TC}_1 - \mathrm{Cr}_0$, respectively, as can be observed from Fig. 10. Thus, the faulty attribute causing the classification errors is k_8 with probability 1.0, and the corresponding subsystem is the mass analysis, which is also with probability 1.0. In the second test data set, the warning signal generation criteria are satisfied at the fifth abnormal wafer, because $q_6 = 0.05\%$; thus, $q_6^5 < \varepsilon = 10^{-15}$ and $5 > n_1 = 4$. Therefore, we have $m_1 = 5$, and we also set $m_2 = 5$. The five misclassified wafers are all classified to recipe 7, while the previous five wafers are all correctly classified to recipe 6. Since recipes 6 and 7 belong to the same TC, TC_4, as can be observed from Fig. 10, we need to apply Step 3.2) to perform fault isolation. The CART for this TC is shown in Fig. 11. The latest five correctly classified wafers lie in the same terminal node for recipe 6 as indicated by $\square$ in Fig. 11. However, there are two different terminal nodes for the five misclassified wafers as indicated by ∇ in Fig. 11. This is because the first abnormal wafer consists of one faulty attribute k_9 only, while the remaining four consist of two faulty attributes, k_9 and k_{10}. Note that the number inside the parenthesis beside ∇ denotes the number of misclassified wafers lying in this node. Applying Step 3.2) of the fault isolation scheme, the traced-back tree paths for recipe 6 indicated by $\square$ are shown by the solid line and for recipe 7 indicated by ∇ are shown by dashed lines in Fig. 11. The faulty attributes, which are the splitting attributes of the nodes where the traced-back paths meet, are k_9 with probability 0.2 and k_{10} with probability 0.8. The corresponding subsystem of both k_9 and k_{10} is the mass analysis, which is thus probability 1.0. For the third to the sixth sets of test data, the details of the misclassified abnormal wafers are tabulated in Table VI. From this table and Table II, we can easily find that conditions (9) and (10) cannot hold simultaneously. Thus, no warning signal is generated in any of these four cases.

V. CONCLUSION

The proposed classification-based fault detection and isolation scheme is a general methodology. Modifying the warning signal generation criteria to meet individual machine's needs, this fault detection scheme is not limited to the ion implanter. The simplicity of the HCT-based fault isolation scheme made HCT worthwhile, especially when its accuracy can be remedied by the warning signal generation criteria when applying it to the ion implanter. Due to the efficient learning capability of HCT and the 0.05-s classification time for classifying the recipe of a working wafer, the proposed fault detection and isolation scheme can work online and in real time.

REFERENCES

[1] C. M. McKenna, "A personal historical perspective of ion implantation equipment for semiconductor applications," in *Proc. Int. Conf. Ion Implantation Technol.*, Sep. 2000, pp. 1–19.

[2] A. S. Willsky, "A survey of design methods for failure detection systems," *Automatica*, vol. 12, no. 6, pp. 601–611, Nov. 1976.

[3] D. M. Himmelblau, *Fault Detection and Diagnosis in Chemical and Petrochemical Processes*. New York: Elsevier, Oct. 1978.

[4] R. Isermann, "Process fault detection based on modeling and estimation methods—A survey," *Automatica*, vol. 20, no. 4, pp. 387–404, Jul. 1984.

[5] D. Barschdorff, Gearbox failure diagnostics VDI-Verlag, Dusseldorf, VDI-Berichte Rep. 644, 1987, pp. 241–248.

[6] S. D. Stearns and D. R. Hush, *Digital Signal Analysis*, 2nd ed. Englewood Cliffs, NJ: Prentice-Hall, 1990.

[7] P. M. Frank, "Fault diagnosis in dynamic systems using analytical and knowledge-based redundancy," *Automatica*, vol. 26, no. 3, pp. 459–474, May 1990.

[8] R. Isermann, "Fault diagnosis of machines via parameter estimation and knowledge processing," *Automatica*, vol. 29, no. 4, pp. 815–835, Jul. 1993.

[9] J. Gertler, *Fault Detection and Diagnosis in Engineering Systems*. New York: Marcel Dekker, 1998.

[10] M. Basseville and A. Benveniste, Eds., *Detection of Abrupt Changes in Signals and Dynamical Systems*. Berlin, Germany: Springer-Verlag, Dec. 1985, vol. 77, Lecture Notes in Control and Information Sciences.

[11] D. Neumann, "Fault diagnosis of machine-tools by estimation of signal spectra," in *Proc. IFAC SAFEPROCESS Symp.*, Sep. 1991, vol. 1, pp. 73–78.

[12] H. H. Yue, S. J. Qin, R. J. Markle, C. Nauert, and M. Gatto, "Fault detection of plasma etchers using optical emission spectra," *IEEE Trans. Semiconduct. Manuf.*, vol. 13, no. 3, pp. 374–385, Aug. 2000.

[13] R. Isermann, "Model-based fault detection and diagnosis—Status and applications," presented at the *16th Symp. Automatic Control Aerospace*, St. Petersburg, Russia, Jun. 2004, unpublished.

[14] ——, "Supervision, fault-detection and fault-diagnosis methods—An introduction," *Contr. Eng. Practice*, vol. 5, no. 5, pp. 639–652, May 1997.

[15] G. M. Smith, *Statistical Process Control and Quality Improvement*, 5th ed. Englewood Cliffs, NJ: Prentice-Hall, 2003.

[16] M. H. Dunham, *Data Mining: Introductory and Advanced Topics*. Englewood Cliffs, NJ: Prentice-Hall, 2002.

[17] L. Breiman, J. H. Friedman, J. A. Olshen, and C. J. Stone, *Classification and Regression Trees*. London, U.K.: Chapman and Hall, 1984.

[18] R. L. Lawrence and A. Wright, "Rule-based classification systems using classification and regression tree (CART) analysis," *Photogrammetric Eng. Remote Sensing*, vol. 67, no. 10, pp. 1137–1142, Oct. 2001.

[19] T. Denoeux, "A neural network classifier based on Dempster-Shafer theory," *IEEE Trans. Syst., Man Cybern.*, vol. 30, no. 2, pt. A, pp. 131–150, Mar. 2000.

[20] G. P. Zhang, "Neural networks for classification: A survey," *IEEE Trans. Syst., Man Cybern.*, vol. 30, no. 4, pt. C, pp. 451–462, Nov. 2000.

[21] J. Ediriwickrema and S. Khorram, "Hierarchical maximum-likelihood classification for improved accuracies," *IEEE Trans. Geosci. Remote Sensing*, vol. 35, no. 4, pp. 810–816, Jul. 1997.

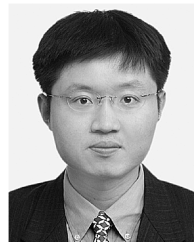
[22] L. Bruzzone and D. F. Prieto, "Unsupervised retraining of a maximum likelihood classifier for the analysis of multitemporal remote sensing images," *IEEE Trans. Geosci. Remote Sensing*, vol. 39, no. 2, pp. 456–460, Feb. 2001.

[23] J. G. Marin-Blazquez and S. Qiang, "From approximative to descriptive fuzzy classifiers," *IEEE Trans. Fuzzy Syst.*, vol. 10, no. 4, pp. 484–497, Aug. 2002.

[24] X. Chang and J. H. Lilly, "Evolutionary design of a fuzzy classifier from data," *IEEE Trans. Syst., Man Cybern.*, vol. 34, no. 2, pp. 1031–1044, Apr. 2004.
[25] D. M. Hawkins and G. V. Kass, "Automatic interaction detection," in *Topics in Applied Multivariate Analysis*, D. M. Hawkins, Ed. Cambridge, U.K.: Cambridge Univ. Press, Apr. 1982, pp. 267–302.
[26] J. R. Quinlan, *C4.5: Programs for Machine Learning*. San Mateo, CA: Morgan Kaufmann, Jan. 1993.
[27] K. R. Müller, S. Mika, G. Rätsch, K. Tsuda, and B. Schölkopf, "An introduction to Kernel-based learning algorithms," *IEEE Trans. Neural Networks*, vol. 12, no. 2, pp. 181–202, Mar. 2001.
[28] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, Oct. 2001.
[29] J. H. Friedman, "Greedy function approximation: A gradient boosting machine," *Ann. Statistics*, vol. 29, no. 5, pp. 1189–1232, Oct. 2001.
[30] ——, "Stochastic gradient boosting," *Computational Statistics Data Analysis*, vol. 34, no. 4, pp. 367–378, Feb. 2002.
[31] A. Borisov, V. Eruhimov, and E. Tuv, "Boosting flexible learning ensembles with dynamic feature selection," presented at the *NIPS 2003 Workshop Feature Extraction*, British Columbia, Canada, Dec. 2003, unpublished.
[32] G. Grimmett and D. Stirzaker, *Probability and Random Processes*, 3rd ed. Oxford, New York: Oxford Univ. Press, May 2001.
[33] O. Cordon, M. Jose del Jesus, and F. Herrera, "A proposal on reasoning methods in fuzzy rule-based classification systems," *Int. J. Approximate Reasoning*, vol. 20, no. 1, pp. 21–45, Jan. 1999.
[34] H. Ishibuchi and T. Nakashima, "Effect of rule weights in fuzzy rule-based classification systems," *IEEE Trans. Fuzzy Syst.*, vol. 9, no. 4, pp. 506–515, Aug. 2001.
[35] J. R. Quinlan, 2003, Data Mining Tools See5 and C5.0 version 1.20 [Online]. Available: http://www.rulequest.com/see5-info.html

Shin-Yeu Lin was born in Taiwan, R.O.C. He received the B.S. degree in electronics engineering from National Chiao Tung University, Taiwan, in 1975, the M.S. degree in electrical engineering from the University of Texas, El Paso, in 1979, and the D.Sc. degree in systems science and mathematics from Washington University, St. Louis, MO, in 1983.

From 1984 to 1985, he was with Washington University working first as a Research Associate and then as a Visiting Assistant Professor. From 1985 to 1986, he was with GTE Laboratory working as a Senior MTS. He joined the Department of Electrical and Control Engineering at National Chiao Tung University in 1987 and has been a Professor since 1992. His research interests include data mining, ordinal optimization theory and applications, and distributed computations.



Shih-Cheng Horng was born in Taiwan, R.O.C. He received the B.S. and M.S. degrees in electrical and control engineering from National Chiao Tung University, Taiwan, in 1993 and 1995, respectively. He is currently pursuing the Ph.D. degree from the same university.

He is a Lecturer in the Department of Electronic Engineering, Chinmin Institute of Technology, Miaoli, Taiwan. His research interests include optimization theory with applications to large semiconductor fab related problems, data mining, and modeling of large complex systems.