# 國立交通大學

## 電機與控制工程學系

## 博士論文

# 兩個關於晶圓製造及測試程序的產能與良率之問題及解決方法

# Two Related Issues on the Throughput and Yield of Wafer Fabrication and Testing Processes

研究生：洪士程

指導教授：林心宇 教授

中華民國九十五年九月

# 兩個關於晶圓製造及測試程序的產能與良率之問題及解決方法

# Two Related Issues on the Throughput and Yield of Wafer Fabrication and Testing Processes

研究生：洪士程　　Student： Shih-Cheng Horng

指導教授：林心宇　　Advisor： Shin-Yeu Lin

國立交通大學

電機與控制工程學系

博士論文

A Dissertation

Submitted to Department of Electrical and Control Engineering

College of Electrical Engineering

National Chiao Tung University

in Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy

in

Electrical and Control Engineering

Septemper 2006

Hsinchu, Taiwan, Republic of China.

中華民國九十五年九月

# 兩個關於晶圓製造及測試程序的產能與良率之問題及解決方法

研究生：洪士程　　　　　指導教授：林心宇　博士

國立交通大學電機與控制工程學系

## 摘要

在本論文中，我們提出兩個關於晶圓製造及測試程序的產能與良率之問題及解決方法。第一個問題為離子植入機的錯誤偵測與隔離，第二個問題為如何在可容忍的重測範圍內降低晶圓誤宰。要偵測一個複雜系統中的錯誤，由於缺少適當的模型，所以是一個困難的任務；正因如此，這也是讓資料採礦技術具有吸引力的地方。對於離子植入機，我們提出了一個以分類為基礎的錯誤偵測與隔離方法。所提出的方法包含了兩部分：分類部分以及錯誤偵測與隔離部分。在分類部分，我們提出具有學習能力的混合型分類樹，針對離子植入機裡正在運作晶圓的配方進行分類，所得到的k-交疊相互驗證錯誤率則用來作為分類結果的準確性。在錯誤偵測與隔離部分，則提出一個基於分類結果準確性，決定是否產生警報信號的標準，而錯誤隔離的機制則是依據混合型分類樹來隔離離子植入機的真正錯誤。我們將所提出的分類器與現有的分類軟體以實例進行比較，並測試所提出的錯誤偵測與分離方法的正確性，皆獲得很成功的結果。

降低晶圓測試程序中的誤宰與重測可以形成一個具有巨大決定變數空間$\Theta$的隨機模擬最佳化問題，針對此問題我們提出一個以序的最佳化方法為基礎的兩層次演算法，來求解一個足夠好的解。在第一層次，對於所考慮的問題透過類神經網路建立一個粗略但有效率的模型。這個粗略的模型被用來在基因演算法中當做適應函數的計算工具，用以有效的從$\Theta$中挑選出$N$個表現較佳的解。在第二層次，從挑選出來的$N$個表現較佳的解，繼續以現有之序的最佳化搜尋方法來找出一個足夠好的解。並且利用模擬的方式來證明所得到解的優質性。我們將所提出的方法應用在降低晶圓測試程序中的誤宰與重測問題，這是一個由測試程序中門限值向量所組成，含有巨大決定變數空間的隨機模擬最佳化問題。經由提出的演法算所得到足夠好的門限值向量，在解的優質性與計算效率上都非常成功。

# Two Related Issues on the Throughput and Yield of Wafer Fabrication and Testing Processes

Student: Shih-Cheng Horng Advisor: Dr. Shin-Yeu Lin

Department of Electrical and Control Engineering
National Chaio Tung University

## Abstract

In this dissertation, we present two related issues on the throughput and yield of wafer fabrication and testing processes. The first issue is a fault detection and isolation problem of the ion implanter, and the second is a reducing overkills under a tolerable retest level problem in wafer testing process. To detect the fault of a complex manufacturing system is a difficult task because of the lack of proper model; indeed, this is the key that makes the data mining technique attractive. We propose a classification based fault detection and isolation scheme for the ion implanter. The proposed scheme consists of two parts: the classification part and 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 and obtain successful results.

Reducing the overkills and retests in a wafer testing process can be formulated as a stochastic simulation optimization problem with huge decision-variable space $\Theta$. For this problem, we have proposed an ordinal optimization theory based two-level algorithm to solve for a good enough solution. In the first-level, we construct a crude but efficient model for the considered problem based on an artificial neural network. This crude model will then be used as a fitness function evaluation tool in a genetic algorithm to efficiently select $N$ roughly good solutions from $\Theta$. In the second-level, starting from the selected $N$ roughly good solutions we proceed with the existing ordinal optimization searching procedures to search a good enough solution of the considered problem. We have justified the quality of the obtained solution using simulations. We applied the proposed algorithm to the reduction of overkills and retests in a wafer testing problem, which is formulated as a stochastic simulation optimization problem that consists of a huge decision-variable space formed by the vector of threshold values in the wafer testing process. The vector of good enough threshold values obtained by the proposed algorithm is very successful in the aspects of solution quality and computational efficiency.

# 誌謝

首先要感謝我的指導教授林心宇博士在研究上所給予的指導及鼓勵，且提供一個良好的研究環境，使我能順利的完成博士學位。林心宇教授的學士淵博，對知識探求的積極態度及待人誠懇且幽默風趣的特質，更是我努力追求的典範。對於本論文的完成，除了林教授的指導之外，也非常感謝諸位口試委員寶貴的意見，使得本論文更加完備。

其次，要感謝我的學長林啟新博士與林謝興博士，實驗室的學弟們黃榮壽、蔡佶興、梁傑愷、林志遠、張文賢、張紹興、林成梓，以及卓建文學弟。多年來，因為有你們的參與，使得我在交通大學的求學過程中更加多采多姿。

同時也要感謝我的服務單位，親民技術學院的長官與同事們多年來的支持及熱心協助，得以使我順利完成本論文。

最後，感謝作者的父母、兄長，在求學期間所給予的支持與鼓勵，讓我得全心全力的專注於研究上。僅以本論文獻給我的家人與關心我的師長與朋友們。

# Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

## 1.1 Motivation

Semiconductor manufacturing is a complex process that involves monitoring a great number of parameters from the early stages of the production to the packaging of an end product. The two most significant factors that determine the manufacturing performance are throughput and yield. The throughput and yield are the most important indexes for measuring the quality of semiconductor manufacturing process. Throughput is defined as the achieved unit output rate of a particular type of equipment asset. Yield is defined as the fraction of total input transformed into shippable output. The process of IC manufacturing often requires hundreds of sequential steps, each one of which could lead to yield loss. Consequently, maintaining product quality in an IC manufacturing facility often requires the strict control of hundreds or even thousands of process variables. Traditional statistical methods are no longer feasible nor efficient, if possible, in analyzing the vast amounts of data in a modern semiconductor manufacturing process. Traditional approaches have limits in extracting the full benefits of the data. Therefore, the manufacturing data is poorly exploited even in the most sophisticated processes. Small improvements on throughput and yield for tenths of a percent can save hundreds of millions of dollars annually in lost products, product rework, energy consumption, and the reduction of waste streams. Considering the big set of parameters and large volume of data in semiconductor manufacturing process, improving throughput and yield is indeed an extremely difficult.

Thereofre, we narrow our focus of throughput and yield improvements on two issues: the fault detection and isolation of an ion implanter and reducing overkills and retests in wafer testing process.

## 1.2 Throughput and Yield Enhancement of Ion Implanter

The semiconductor manufacturing process can be divided into four basic phases: wafer fabrication, wafer probe, assembly or packaging and final testing. Wafer fabrication is the most technologically-complex and capital-intensive phase among the four. In the wafer fabrication phase, ion implanter is a bottleneck machine in the semiconductor manufacturing process because of its expensiveness; thus, ion implantation is a critical operation to the throughput. The damaged wafer due to the malfunction of the ion implanter is not re-workable hence significantly affects the yield. Thus, the real-time fault detection and isolation for minimizing the possible down time of the ion implanter is a crucial issue in semiconductor manufacturing process.

## 1.3 Throughput and Yield Improvement of Wafer Testing

Semiconductor testing of ICs or chips is required at various stages during the fabrication process. Each IC must be individually tested in wafer and in packaged form to ensure that it functions as intended. Demand for testing products is driven by two considerations: new chip designs and higher unit volumes. As chips become increasingly powerful and complex, the need for high-speed and accurate testing becomes more important than ever. The process of testing individual chips in wafer form is referred to as wafer probing. Wafer probing establishes a temporary electrical contact between the chip and the automatic test equipment. This is the critical test for design and performance of the IC, and for sorting ICs before separation and costly packaging. A probing system, which transmits electrical signals to the wafer and analyzes the signals upon their return, has four principal components: the prober, the probe card, the probe station, and automatic test equipment**.** Although there exist techniques such as the Statistical Process Control (SPC) for monitoring the operations of the wafer probes, the probing errors may still occur in many aspects and cause some good dies being over killed, which will degrade the yield. Thus, reducing the number of overkills is always one of the main objectives in wafer testing process. The key tool to identify or save overkills is retest, which is an additional wafer probing. However, retest is a major factor for decreasing the throughput. Thus, the overkill and the retest possess inherent conflicting factors, because reducing the former can gain more profit, however, at the expense of increasing the latter, which will degrade the throughput and increase the cost. Consequently, to save more overkills using less retests is a goal of the wafer testing process.

## 1.4 Dissertation Outline

This dissertation introduces two related issues on the throughput and yield of wafer fabrication and testing processes. The first issue is a fault detection and isolation problem of the ion implanter, and the second one is a reducing overkills under a tolerable retest level problem in wafer testing process.

In Chapter 2, we propose a classification based fault detection and isolation scheme for the ion implanter. The proposed scheme consists of two parts: the classification part and 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 and obtain successful results.

In Chapter 3, we have formulated a stochastic optimization problem to find the optimal threshold values to reduce the overkills of dies under a tolerable retest level in wafer testing process. We have proposed an ordinal optimization (OO) theory based two-level algorithm to solve for a vector of good enough threshold values of the stochastic simulation optimization problem. In the first-level, we construct a crude but efficient model for the considered problem based on an artificial neural network. This crude model will then be used as a fitness function evaluation tool in a genetic algorithm to efficiently select $N$ roughly good solutions from decision-variable space. In the second-level, starting from the selected $N$ roughly good solutions we proceed with the existing ordinal optimization searching procedures to search a good enough solution of the considered problem. We have justified the quality of the obtained solution using simulations. We applied the proposed algorithm to the reduction of overkills and retests in a wafer testing process, which is formulated as a stochastic simulation optimization problem that consists of a huge decision-variable space formed by the vector of threshold values in the wafer testing process. The vector of good enough threshold values obtained by the proposed algorithm is very successful in the aspects of solution quality and computational efficiency.

Finally, some conclusions for the dissertation are drawn in Chapter 4. We also suggest some possible future research issues concerning the methods developed in this dissertation.

# Chapter 2

# Fault Detection and Isolation for the Ion Implanter

## 2.1 Introduction

Ion implanter [1] is a bottleneck machine in the semiconductor manufacturing process because of its expensiveness; thus, ion implantation is a critical operation to the throughput. The damaged wafer due to the malfunction of the ion implanter is not re-workable hence 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]-[4]. Model free methods, which do not use the mathematical model of the plant, range from physical redundancy, limit value checking [5] and spectrum analysis [6]. Among them, limit value checking method is widely used in practice. There are also two types of fault isolation methods [7], 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 models 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 Figure 2.1 [1]. In general, the equipment supplier provides a 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 [8] of the measured parameters for other major subsystems, the ion source (filament), extraction electrode, mass analysis, and acceleration subsystems to monitor their operations.

The measured parameters can be, for examples, 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 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

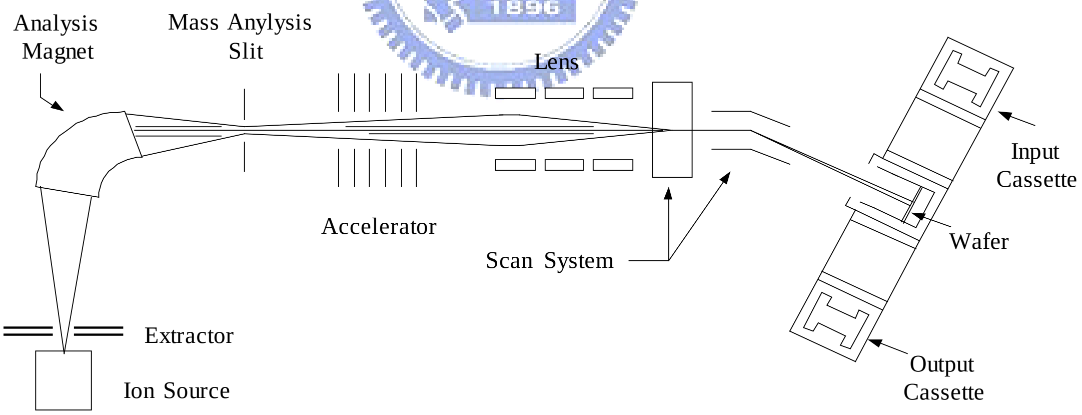
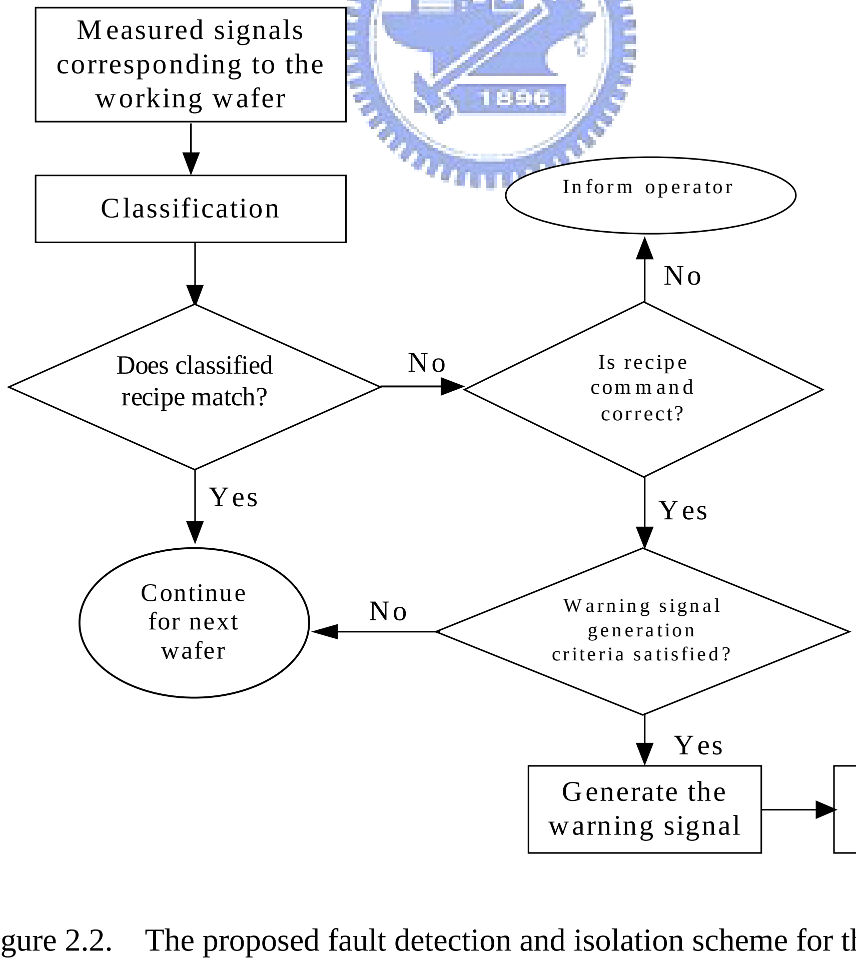


Figure 2.1. The structure of an ion implanter.

[1] A recipe controls how vectors 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 chapter, a recipe is corresponding to a specific product of integrated circuit.

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 not even mention the collection of a large set of abnormal data patterns with known faults due to ion implantation. Thus, the *purpose* of this chapter is to propose an automatic (i.e. no need of 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 [9] 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 above 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 Figure 2.2. Our scheme starts from 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 few damaged wafers in the worst case. A detailed analysis of this claim will be addressed in Section 2.4. 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 chapter, we propose a *Hybrid Classification Tree (HCT)* with good *learning capability* to deal with the classification problem. The HCT combines a proposed *clustering algorithm* with the



Measured signals corresponding to the working wafer
Classification
Does classified recipe match?
No
Is recipe command correct?
No
Inform operator
Yes
Yes
Continue for next wafer
No
Warning signal generation criteria satisfied?
Yes
Generate the warning signal
Isolate the fault

Figure 2.2. The proposed fault detection and isolation scheme for the ion implanter.

*Classification and Regression Tree (CART)* [10] to take the advantages 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 so as not to 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 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 classified recipe matches the destined one; however, we need not worry about the latter one by the no fault assumption mentioned above. 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. Dislike the existing methods, which need to collect a fairly large set of measured data patterns with known faults as indicated earlier, the proposed fault isolation scheme almost spend no extra effort as will be seen in Section 2.3.2. 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 chapter 2 in the following manner. In Section 2.2, we will present the HCT and its learning capability. In Section 2.3, we will analyze the probability of no alarm while 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 2.4, 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. In the meantime, we will also investigate the learning capability of HCT by reporting the computation time needed to update the classification rules of HCT. In Section 2.5, we will make a conclusion.

## 2.2 The Hybrid Classification Tree (HCT)

There exist numerous classification techniques for classification problems of continuous attributes such as the neural network approach [11], maximum-likelihood approach [12], fuzzy set theory based approach [13], decision tree [14], CART [10], kernel based learning algorithms [15], and recent methods like random forests [16], multiple additive regression trees (MART) [17] and the boosting flexible learning ensembles with dynamic feature selection technique [18], etc.. 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 measurement 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 are large. 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 technique, disadvantage of CART is inaccuracy due to its nature of piecewise constant approximation. However, the biggest advantage of CART is its interpretability whereas the above 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 as for applying to the fault detection of the ion implanter, which will be presented in Section 2.3.1.

The tree sizes of CART are closely related to the interpretability and accuracy. Small tree can be easily interpreted, while the interpretability of a large tree is questionable. On the other hand, 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 so as 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 Figure 2.10) attributes' mean of each individual recipe step are still a key to distinguish the recipes. Thus, we can exploit this property to fulfill the above mentioned 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.

### 2.2.1 The Separation Matrices Based Clustering Algorithm

Due to the above 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* are as shown in Figure 2.3(a), then these two recipes are separable based on that attribute; while in the case of Figure 2.3(b), the two recipes are not. Throughout this section, we will use the terminology *class* in classification techniques to represent *recipe*.

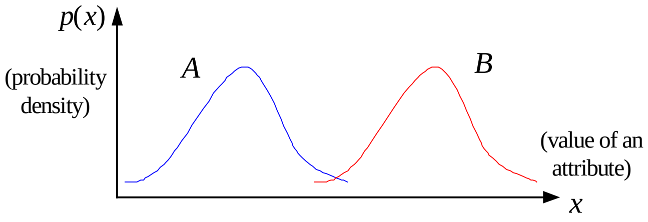


Figure 2.3(a). Separable recipes.

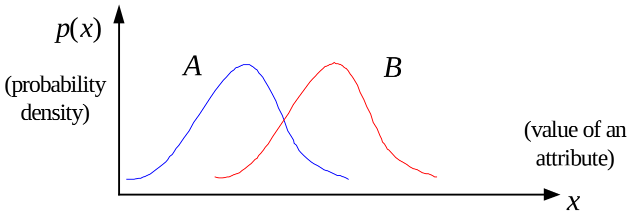


Figure 2.3(b). Non-separable recipes.

#### 2.2.1.1 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

$$D(C_i,C_j)_k = \begin{cases} 0, & \text{if } C_i \text{ and } C_j \text{ are separable based on attribute } k, \\ 1, & \text{otherwise.} \end{cases} \tag{2.1}$$

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 Chebyshev inequality [19] as described below. We let the random variable $X_i^k$ denote the $k$th attribute of class $C_i$, and let $\mu_i^k$ and $\sigma_i^k$ denote the mean and standard deviation of $X_i^k$, respectively. Let $a_i^k$ be a positive real number such that $P[\left|X_i^k - \mu_i^k\right| \geq a_i^k] \leq \alpha$, where $P[(\cdot)]$ denotes the probability of the event $(\cdot)$, and $\alpha$ 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 $\alpha$ can be calculated from setting $\left(\sigma_i^k / a_i^k\right)^2 = \alpha$ using Chebyshev inequality. Without loss of generality, we can assume $\mu_i^k < \mu_j^k$. We let $\bar{p}_j = \min(1, \left[\sigma_j^k / \max(\delta, \mu_j^k - \mu_i^k - a_i^k)\right]^2)$, where $a_i^k$ is defined above and $\delta$ is a very small positive real number to avoid the denominator of the square term being 0 or negative. $\bar{p}_j$ 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 Chebyshev inequality. If $\mu_j^k$ is sufficiently larger than $\mu_i^k + a_i^k$, $\bar{p}_j$ 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 Figure 2.4. 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 the following:

$$D(C_i,C_j)_k = \begin{cases} 0, & \mathit{if} \quad \bar{p}_j < \hat{p}, \\ 1, & \mathit{otherwise.} \end{cases} \tag{2.2}$$

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 $D(C_i, C_j)_k$.

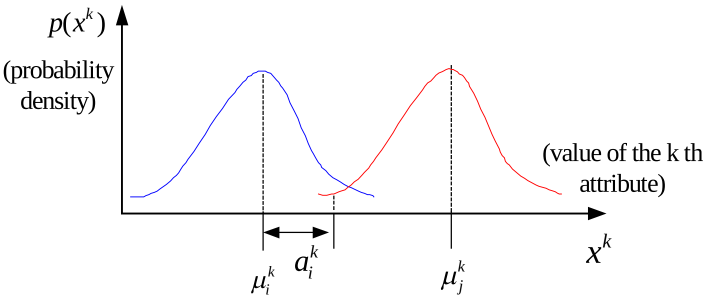


Figure 2.4. Illustration of the separation between $C_i$ and $C_j$ based on $\overline{p}_j$.

### 2.2.1.2 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 sub-graphs. Each connecting sub-graph represents a cluster of non-separable classes based on attribute $k_1$, and the number of disjoint sub-graphs represent 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 Figure 2.5(a) is shown in Figure 2.5(b), which consists of two disjoint clusters, or two separate connecting sub-graphs, $\mathscr{A}$ and $\mathscr{B}$. The resulted clusters can be further split by other attributes. For example, cluster $\mathscr{A}$ in Figure 2.5(b) can be further split by attribute $k_2$, whose $[D(C_i,C_j)_{k_2}]$ is shown in Figure 2.6(a), in the following manner. Collecting the rows and columns of $[D(C_i,C_j)_{k_2}]$ corresponding to the classes in cluster $\mathscr{A}$ to form the submatrix shown in Figure 2.6(b). Repeating the same process of splitting $Cr_0$ using $[D(C_i,C_j)_{k_1}]$, cluster $\mathscr{A}$ can be split into two clusters $\mathscr{C}$ and $\mathscr{D}$ as shown in Figure 2.6(c) by using the submatrix shown in Figure 2.6(b).

$$\begin{array}{c|cccccc|}
 & C_1 & C_2 & C_3 & C_4 & C_5 & C_6 \\
\hline
C_1 & 1 & 1 & 0 & 0 & 0 & 0 \\
C_2 & 1 & 1 & 1 & 0 & 0 & 0 \\
C_3 & 0 & 1 & 1 & 0 & 0 & 0 \\
C_4 & 0 & 0 & 0 & 1 & 1 & 0 \\
C_5 & 0 & 0 & 0 & 1 & 1 & 1 \\
C_6 & 0 & 0 & 0 & 0 & 1 & 1 \\
\hline
\end{array}$$

Figure 2.5(a). A separation matrix example $[D(C_i, C_j)_{k_1}]$.

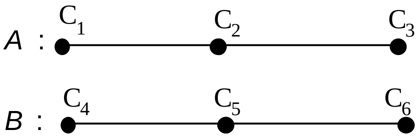


Figure 2.5(b). A separation graph example resulted from the separation matrix in Figure 2.5(a).

$$\begin{array}{c|cccccc|}
 & C_1 & C_2 & C_3 & C_4 & C_5 & C_6 \\
\hline
C_1 & 1 & 0 & 0 & 0 & 0 & 0 \\
C_2 & 0 & 1 & 1 & 0 & 0 & 0 \\
C_3 & 0 & 1 & 1 & 1 & 0 & 0 \\
C_4 & 0 & 0 & 1 & 1 & 1 & 0 \\
C_5 & 0 & 0 & 0 & 1 & 1 & 1 \\
C_6 & 0 & 0 & 0 & 0 & 1 & 1 \\
\hline
\end{array}$$

Figure 2.6(a). The separation matrix $[D(C_i, C_j)_{k_2}]$.

$$\begin{array}{c|ccc|}
 & C_1 & C_2 & C_3 \\
\hline
C_1 & 1 & 0 & 0 \\
C_2 & 0 & 1 & 1 \\
C_3 & 0 & 1 & 1 \\
\hline
\end{array}$$

Figure 2.6(b). Submatrix of $[D(C_i, C_j)_{k_2}]$ corresponding to cluster $\mathcal{A}$ in Figure 2.5(b).

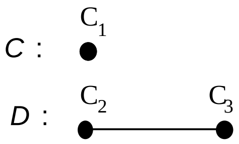


Figure 2.6(c). Clusters split from cluster $\mathcal{A}$ using the submatrix shown in Figure 2.6(b).

### 2.2.1.3 The Choice of Attributes for Cluster Splitting and the Construction of the Clustering Tree

Because the separation matrix has already indicated 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 range of attribute values, more finer fuzzy partition is needed to classify a cluster with 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 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 Figure 2.7(a) and 2.7(b), suppose that we use the attribute $k_1$ to split the cluster first, we obtain three child clusters. One consists of 1 class, and the other two consist of three and four classes. While 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 above, we would choose $k_2$ to split the cluster. To put this criteria into a mathematical form, we let $L_k$ and $n_{k,\,l}(Cr_j)$ denote the number of child clusters and the number of classes in the $l$ th child cluster resulted

$$\begin{array}{c|cccccccc|}
 & C_1 & C_2 & C_3 & C_4 & C_5 & C_6 & C_7 & C_8 \\ \hline
C_1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
C_2 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
C_3 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
C_4 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\
C_5 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\
C_6 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\
C_7 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\
C_8 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ \hline
\end{array}$$

Figure 2.7(a). The separation matrix $[D(C_i, C_j)_{k_1}]$.

$$\begin{array}{c|cccccccc|}
 & C_1 & C_2 & C_3 & C_4 & C_5 & C_6 & C_7 & C_8 \\ \hline
C_1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
C_2 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
C_3 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\
C_4 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\
C_5 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\
C_6 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\
C_7 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\
C_8 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ \hline
\end{array}$$

Figure 2.7(b). The separation matrix $[D(C_i, C_j)_{k_2}]$.

from using attribute $k$ to split the cluster $Cr_j$, respectively. Then, the *criteria* for choosing the attribute for splitting $Cr_j$ is

$$\min_k \left\{ \left( \sum_{l=1}^{L_k} n_{k,\,l}\left(Cr_j\right) \middle/ L_k \right) \cdot \left( \sum_{l=1}^{L_k} \left(n_{k,\,l}(Cr_j) - \bar{n}_{k,\,l}\left(Cr_j\right)\right)^2 \middle/ L_k \right) \right\} \tag{2.3}$$

where the first term inside the big bracket represents the average number of classes in the resulted child clusters and the second term denotes the variance of the number of classes in the resulted child clusters, where $\bar{n}_{k,\,l}\left(Cr_j\right) \equiv \sum_{l=1}^{L_k} n_{k,\,l}\left(Cr_j\right) \Big/ L_k$.

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*)={ $Cr_0$ }.

Step 1: For each cluster in *YSC*, obtain the corresponding splitting attribute $k$ based on the criteria (2.3). Using 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 $YSC = \phi$, stop; otherwise return to Step 1.

Figure 2.8 shows a clustering tree built by using the separation matrices of two attributes shown in Figure 2.5(a) and 2.6(a) to split the root cluster $Cr_0$ ={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 [20] for 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 indicated in Section 2.2.

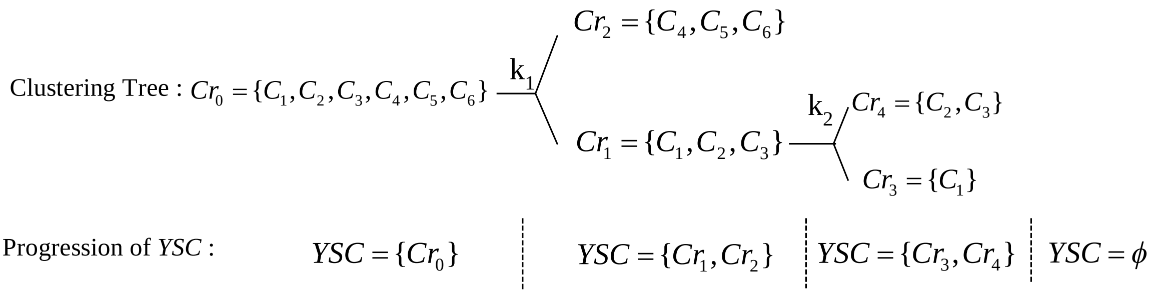


Figure 2.8.   An example of using Algorithm I to build the clustering tree.

### 2.2.1.4 The 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: (i) construction of the separation matrices for all attributes, (ii) determine the cluster splitting attribute and build the clustering tree, and (iii) throughout the clustering tree, generate the fuzzy if-then rules needed to classify a data pattern into proper child cluster based on a given set of training data patterns with known TCs. In the above three steps, (i) and (ii) had been presented in previous subsections. The details of (iii) as well as the classification part are described below.

#### *1. The Fuzzy-Rule Generation Procedures of the 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 $x_k^s$, $s = 1,...,g$, denote the $k$ th attribute of $g$ data patterns, $x^s$, $s = 1,...,g$, from $M_j$ known child clusters,

$CCr_{j1},\ldots,CCr_{jM_j}$. These $g$ data patterns form the training data set for splitting $Cr_j$. The fuzzy rules for splitting 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 $k$ th attribute values,

*Rule* $R_i(Cr_j)$: *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* $i$ *th 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* $R_i(Cr_j)$ *(2.4)*

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

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 chapter, 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

$$\beta_{CCr_{jl}}(R_i(Cr_j)) = \sum_{x^s \in CCr_{jl}} f_i^K(x_k^s) \tag{2.5}$$

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 $CCr_{jl}$, $l=1,\ldots,M_j$ of the to-be-split cluster $Cr_j$ and the corresponding splitting attribute $k$. Set $i=1$.

Step 1: Calculate the sum of grade of compatibility of child cluster $CCr_{jl}$, $l=1,\ldots,M_j$, with respect to $A_i^K$ by (2.5).

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

$$\beta_{CCr_{jx}}(R_i(Cr_j)) = \max\{\beta_{CCr_{j1}}(R_i(Cr_j)), \ldots, \beta_{CCr_{jM_j}}(R_i(Cr_j))\} \tag{2.6}$$

then $CCr_{jx}$ is the consequent $CCr_{ji}$ in rule $R_i(Cr_j)$.

Step 3: Determine $CF_i^K$, the grade of certainty of rule $R_i(Cr_j)$, by

$$CF_i^K = \left(\beta_{CCr_{jx}}(R_i(Cr_j)) - \beta(R_i(Cr_j))\right) \Big/ \sum_{l=1}^{M} \beta_{CCr_{jl}}(R_i(Cr_j)) \tag{2.7}$$

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

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

**2.** *Training Part of the Clustering Algorithm*

Combining the construction of separation matrices, determination of the splitting attributes, building of the clustering tree and the above 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 $\mu_i^k$ and $\sigma_i^k$ of each class $C_i$ and each attribute $k$; compute the separation matrices $[D(C_i, C_j)_k]$ based on (2.2) 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.

**3.** *Classification Part of the 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 $k$ th attribute of $x'$ corresponding to the splitting attribute $k$ at cluster $Cr_j$. We define $\gamma_{CCr_{jl}}$, 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 $R_i(Cr_j)$ over all $K$ trained rules whose consequent are $CCr_{jl}$. We can express $\gamma_{CCr_{jl}}$ mathematically as $\gamma_{CCr_{jl}} = \sum_{R_i(Cr_j), CCr_{ji}=CCr_{jl}} f_i^K(x'_k)\cdot CF_i^K$. Then the classification procedures for the new data can be stated below.

*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, $CCr_{jy} = \arg(\max\{\gamma_{CCr_{j1}},\ldots,\gamma_{CCr_{jM_j}}\})$.

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

**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 (*PCr*)= $Cr_0$.

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.

### 2.2.2 The CART for Terminal Cluster (TC)

The TCs resulted 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 help reduce the computational complexity and the tree size of CART when applies to $Cr_0$ alone.

The CART is a well-developed classification tool. The details of this classification technique can be found everywhere [10]. 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: (a) the splitting rule, (b) the goodness-of-split criteria, and (c) the criteria for choosing an optimal or final tree for analysis. Regarding (a), there are three major splitting rules in CART. The one we employed here is the Gini's criteria [10]. This criteria starts the tree-building process by partitioning the TC into binary nodes based upon a very simple question of the form: is $k \leq b$? where $k$ is an attribute and $b$ is a real number. Regarding (b), the CART uses a computation-intensive algorithm that searches for the best split at all possible split points for each attribute that decrease the Gini's impurity measure most. CART will recursively apply this splitting rule to split non-terminal 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 (c). The computational complexity of the training part of CART mainly lies in the exhaustive search for the best split required in (b). Once the classification tree and the splitting rules are obtained, the classification procedure of CART is simply asking whether $k \leq b$? to determine which of the binary child nodes the new data pattern belongs to throughout the classification tree.

### 2.2.3 Classification of a 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 Figure 2.2.

## 2.2.4 Learning Capability

The learning capability of a classifier is very important in current application, because for every fourteen minutes, 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 slight modification is needed for the case of 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: (i) updating the separation matrices, (ii) updating the attributes used to split clusters as well as the clustering tree, and (iii) updating the fuzzy rules for splitting clusters. Learning of CART is just to update the best split for each node in the classification tree.

### 2.2.4.1 Learning of the 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 $m$, then we will update $\mu_m^k$ and $\sigma_m^k$ for all $k$, which will be used to update the separation indices $D(C_m, C_j)_k$ 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 (ii) can be skipped. In fact, $\mu_m^k$ and $\sigma_m^k$ will only slightly deteriorate when the new data pattern 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 cause the corresponding separation matrices changed in updating step (i), we need to proceed with updating step (ii) 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 (iii), 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, along which the new data pattern belongs to. 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 $R_i(Cr_j)$ in (2.4) is to update the consequent and grade of certainty after the new data pattern join in. To update the consequent, we need to update $\beta_{CCr_{jz}}(R_i(Cr_j))$ 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 (2.5). The updated $\beta_{CCr_{jz}}(R_i(Cr_j))$ 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 $\beta_{CCr_{jz}}(R_i(Cr_j))$ to update the corresponding $CF_i^K$ by (2.7). 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 if the clustering tree or any splitting attributes changes 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 previous case. However, no matter what case, it will not affect HCT to work real-time and on-line as will be demonstrated in Section 2.4.

### 2.2.4.2 Learning of CART

Following from 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 are very large, the best split point of each attribute in each node of the CART will alter at most slightly when new data pattern 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 ±*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 subsection, this will not affect HCT to work real-time and on-line.

## 2.3 Warning Signal Generation and Fault Isolation

### 2.3.1 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 (i) the machine is in normal condition, or (ii) the actual implantation has been wrong due to a machine fault but a misclassification makes the classified recipe match the destined one. Case (ii) indicates a fault situation that cannot be observed from the matched result. We let $\eta_i$ denote the misclassification rate of recipe $i$, which can be calculated by the following

$$\eta_i = \sum_{j \neq i} \pi(j) m(i \mid j) \tag{2.8}$$

where $\pi(j)$ denotes the prior probability of recipe $j$ and $m(i \mid j)$ denotes the misclassification rate of classifying recipe $j$ to be recipe $i$. If case (ii) occurs to recipe $i$, then the probability of a series of $n$ such events occur is $\eta_i^n$. This indicates the probability of an undetected machine fault will be extremely small provided that $\eta_i$ 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 $\eta_i$ for all $i$ based on HCT will be given through the tests presented in Section 2.4. This addresses the comment cited in Section 2.1 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 $\eta_i^n$ extremely small. This also indicates when the classified recipe matches the destined one, we can continue for next wafer as shown in Figure 2.2. 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 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 mismatch the destined one and require the checking of warning signal generation criteria as indicated in Figure 2.2. To minimize the probability of 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 $\varepsilon$, 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 mismatched recipe is not classification errors. Thus we can state our *warning signal generation criteria* as follows.

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

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

$$q_{i_l}(l)\times q_{i_{l+1}}(l+1)\times\cdots\times q_{i_{l+n}}(l+n)\le\varepsilon \tag{2.9}$$

*and*

$$n \ge n_1, \tag{2.10}$$

*where* $i_l$ *denotes the destined recipe of the* $l$ *th wafer,* $q_i(l)$ *denotes the* $q_i$ *of the* $l$ *th wafer,* $\varepsilon$ *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 condition (2.9) holds, we can exclude the possibility of false alarm due to classification errors. If condition (2.10) holds, we can exclude the possibility of false alarm due to the electrical spikes.

### 2.3.2 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. While 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 conditions (2.9) and (2.10) 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 Figure 2.9, 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 using the classification tree of CART.

Step 4: List all the different faulty attributes found from the $m_1 \times m_2$ searches in Step3 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.

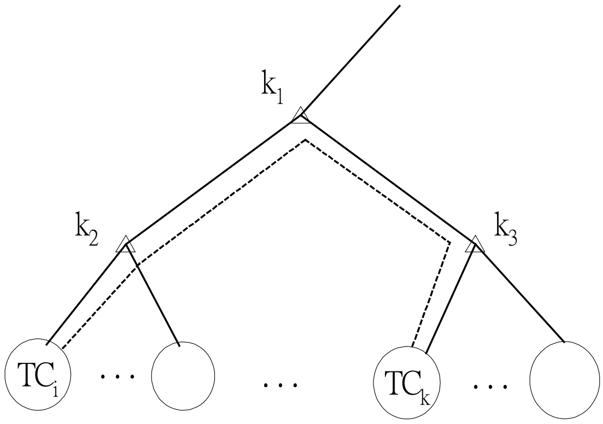


Figure 2.9. Using clustering tree to find the faulty attribute.

## 2.4 Test Results of HCT, Warning Signal Generation and Fault Isolation

### 2.4.1 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, $k_1$,…, $k_{11}$ 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 2.1 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. It should be noted that all the test results shown in this section are simulated in a Pentium IV PC using Matlab.

A data set of 42-recipe case, and each recipe consists of a thousand to 10,000 wafers are 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 one second to measure a 12-attribute data pattern. The ion implantation time for a wafer is around 10 seconds. Thus, ten data patterns are taken while a wafer is under work. The wafer changeover time is 26 seconds on average. Each lot contains 24 wafers, and the setup time for a new lot is 13 minutes. For all the measured data patterns in this case, we randomly divide them in wafer base into ten parts. We take 9 parts as training data set and 1 part as test data set. We set $\hat{p} = 0.075$ in (2.2), 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 Figure 2.10, 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 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 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 10 measured data patterns, and each test data pattern will be classified to a recipe, thus a *majority voting scheme* is used to conclude the classified recipe of the wafer corresponding to the 10 test data patterns. Repeating this process for ten times by circulating the training data set and test data set, Table 2.2 shows the resulting 10-fold cross validation classification error rate of all recipes in this test. We also indicate the 10-fold cross validation classification error rate using the software See5 [21] and CART [10] 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.

Table 2.1. The units and related subsystems of the 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 |

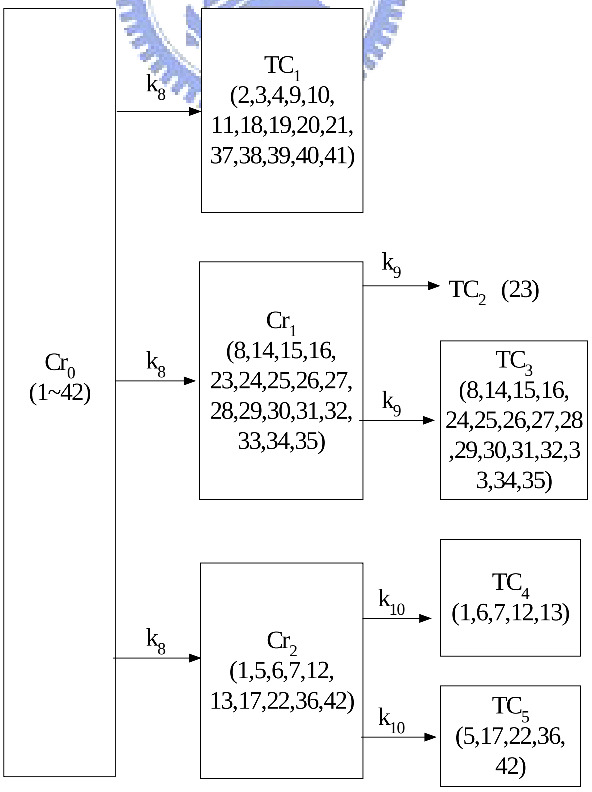

Cr0 (1~42)
k8
TC1 (2,3,4,9,10, 11,18,19,20,21, 37,38,39,40,41)
k8
Cr1 (8,14,15,16, 23,24,25,26,27, 28,29,30,31,32, 33,34,35)
k9
TC2 (23)
k9
TC3 (8,14,15,16, 24,25,26,27,28 ,29,30,31,32,3 3,34,35)
k8
Cr2 (1,5,6,7,12, 13,17,22,36,42)
k10
TC4 (1,6,7,12,13)
k10
TC5 (5,17,22,36, 42)

Figure 2.10. The clustering tree of the 42-recipe case.

Table 2.2. The 10-fold cross validation classification error rate of the 42-recipe case.

| Recipe | Classification error rate(%) | | | Recipe | Classification error rate (%) | | | Recipe | Classification error rate (%) | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | HCT<br>$\hat{p}$ =0.075 | CART | See5 | | HCT<br>$\hat{p}$ =0.075 | CART | See5 | | HCT<br>$\hat{p}$ =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 |

**Remark 1:** From the test results shown in Table 2.2, 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, recipe 14 and 20 have been classified into different TCs as can be observed from Figure 2.10. 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 2.3 for 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 subsection, 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 (2.8) for the three classifiers as shown in Table 2.3. 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 2.3 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 classification error rate shown in Table 2.2. 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 10-fold 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 2.4. From the fourth row of this table, we can observe that when $\hat{p} \leq 0.075$, the 10-fold 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 one second, thus HCT can work real-time. This shows that HCT not only performs better than See5 and CART in the aspect of 10-fold 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 computational time consuming as expected. This also demonstrates why the clustering algorithm helps reduce the computational complexity of CART.

Table 2.3. The misclassification rate $\eta$ of the 42-recipe case.

| Recipe | Misclassification rate (%) | | | Recipe | Misclassification rate (%) | | | Recipe | Misclassification rate (%) | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | HCT $\hat{p}$ =0.075 | CART | See5 | | HCT $\hat{p}$ =0.075 | CART | See5 | | 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 2.4. The training time, classification time and 10-fold cross validation for the sum of classification error rates for different values of $\hat{p}$ .

| Classifier | HCT | | | | CART | See5 |
|---|---|---|---|---|---|---|
| | $\hat{p}$ =0.05 | $\hat{p}$ =0.075 | $\hat{p}$ =0.08 | $\hat{p}$ =0.13 | | |
| 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 |

### 2.4.2 Test Results of the 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 algorithm and the splitting rules for CART take only 0.1637 seconds 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 seconds, which is even shorter than the wafer changeover time, which takes 26 seconds. For different values of $\hat{p}$, the updating times of HCT when separation matrices change are shown in Table 2.5. From the above 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 real-time and on-line. 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.

Table 2.5. The 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 |

### 2.4.3 Test Results of the 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 above 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 10 abnormal wafers are locating 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 10 abnormal wafers are locating from the 31st to the 40th wafers. The first abnormal wafer is caused by attribute $k_9$, and the rest 9 wafers are caused by both $k_9$ and $k_{10}$. The third set consists of 20 wafers, and the 2 abnormal wafers are locating 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, the fifth and the sixth set of wafers; these three sets consist of 30 wafers each, and the two abnormal wafers are locating 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 6 out of 10 existing HCT test data sets and insert the above 6 small sets of data patterns into the 6 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 2.2, we apply the HCT associated with the majority voting scheme to classify the above six test data sets. In the first test data set, the warning signal generation criteria, i.e. conditions (2.9) and (2.10), are satisfied at the 5th abnormal wafer of the first small data set, because $q_{39} = 0.0455\%$ according to Table 2.2, 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 are the 16th to the 20th wafers in the first small set of test data. The 5 misclassified wafers are all classified to recipe 24 while the previous 5 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 $TC_3$-$Cr_1$-$Cr_0$ and $TC_1$-$Cr_0$, respectively, as can be observed from Figure 2.10. 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 5th abnormal wafer, because $q_6 = 0.05\%$, thus $q_6^5 < \varepsilon = 10^{-15}$, and $5 > n_1 = 4$. Thus, we have $m_1 = 5$, and we also set $m_2 = 5$. The 5 misclassified wafers are all classified to recipe 7, while the previous 5 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 Figure 2.10, we need to apply Step 3.2 to perform fault isolation. The CART for this TC is shown in Figure 2.11. The latest 5 correctly classified wafers lie in the same terminal node for recipe 6 as indicated by □ in Figure 2.11. However, there are two different terminal nodes for the 5 misclassified wafers as indicated by ∇ in Figure 2.11. This is because the first abnormal wafer consists of one faulty attribute $k_9$ only, while the rest 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 □ is shown by the solid line and for recipe 7 indicated by ∇ are shown by dashed lines in Figure 2.11. The faulty attributes, which are the splitting attribute 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 with probability 1.0. For the third set to the sixth set of test data, the details of the misclassified abnormal wafers are tabulated in Table 2.6. From this table and Table 2.2, we can easily find that conditions (2.9) and (2.10) can not hold simultaneously. Thus, no warning signal is generated in any of these four cases.

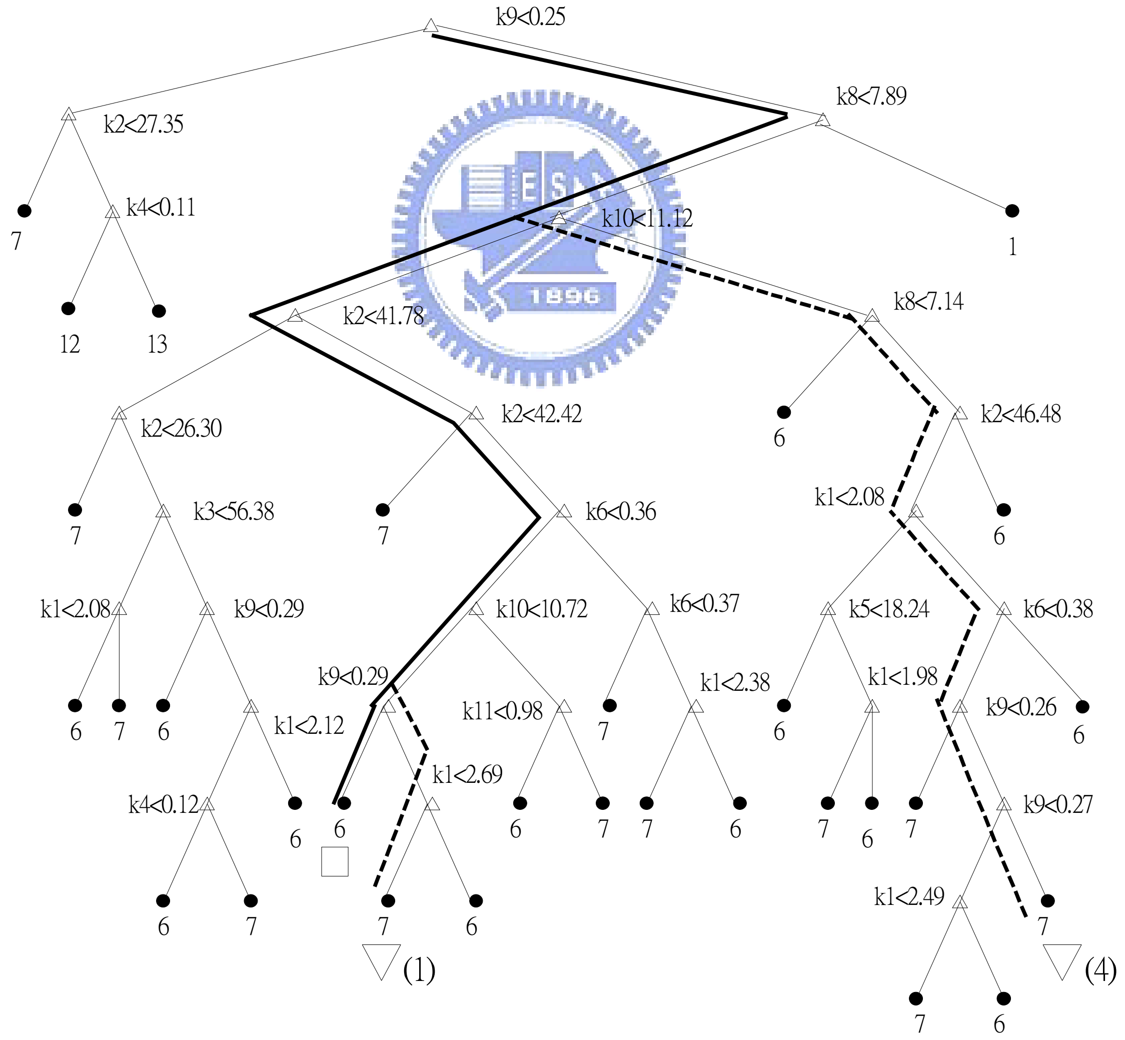


Figure 2.11. Classification tree of CART for $TC_4$.

Table 2.6. The 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 |

## 2.5 Concluding Remarks

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 to the ion implanter. Due to the efficient learning capability of HCT and the 0.05 seconds classification time for classifying the recipe of a working wafer, the proposed fault detection and isolation scheme can work on line and real-time.

# Chapter 3

# Reducing Overkills and Retests in Wafer Testing Process

## 3.1 Introduction

The wafer fabrication process is a sequence of hundreds of different process steps, which results in an unavoidable variability accumulated from the small variations of each process step. Chips are tested multiple times throughout the design and manufacturing process to ensure the integrity of the chip design and the quality of the manufacturing process. Semiconductor testing of chips is required at various stages during the fabrication process.

Wafer probing, or testing chips while they are still in semiconductor wafer form is critical to both engineering and production test. A typical wafer is 8 inches or 200 mm in diameter and usually contains 600 to 15,000 chips. Wafer probing establishes a temporary electrical contact between test equipment, such as an Agilent analyzer, and each individual die (or chip) on a wafer to determine whether each chip meets design and performance specifications. The test transmits electrical signals to the chip and analyzes the signals that return. Wafer probing ensures that the chip manufacturer avoids incurring the significant expense of assembling and packaging chips that do not meet specification by identifying flaws early in the manufacturing process.

Although there exist techniques such as the Statistical Process Control (SPC) [22] for monitoring the operations of the wafer probes, the probing errors may still occur in many aspects and cause some good dies being over killed; consequently, the profit is diminished. Thus, reducing the number of *overkills* is always one of the main objectives in wafer testing process. The key tool to identify or save overkills is *retest*, which is an additional wafer probing. However, retest is a major factor for decreasing the *throughput*. Thus, the overkill and the retest possess inherent conflicting factors, because reducing the former can gain more profit, however, at the expense of increasing the latter, which will degrade the throughput and increase the cost.

There may be different testing procedures in different chip manufacturers. But, no matter what testing procedures are used, the decision for carrying out the retest should be based on whether the number of good dies and the number of *bins*[2] in a wafer exceed the corresponding *threshold values*. Deciding whether to go for a retest is a decision problem. In current wafer testing process, this decision is made based on whether the number of good dies and the number of *bins* in a wafer exceed the corresponding *threshold values*. Manually adaptive adjustments of the threshold values based on engineering judgment, three-sigma limit [23] or a looser six-sigma limit are currently used in some semiconductor manufacturing companies. Consequently, determining these threshold values so as to *minimize the overkills under a tolerable level of retests* is the main theme of the stochastic optimization problem considered here. What implies is that drawing a fine line for deciding whether to go for a retest to save possible overkills is an important research issue in this *stochastic optimization problem* of the wafer testing process.

Various techniques such as the weighting objective method, hierarchical optimization method, trade-off method, global criterion method, and method of distance functions and min-max method described in [24] can be used to solve stochastic optimization problems. Considering the economic situation regarding throughput requirement, it would be most beneficial for us to use the trade-off method [25] to solve the current problem. That is to minimize the overkills subject to a tolerable level of retests provided by the decision maker.

The *purpose* of this chapter is using a systematic approach to determine these threshold values. We first formulate a stochastic optimization problem on the threshold values. Since the formulated stochastic optimization problem consists of a huge decision-variable space, this makes the problem becomes a hard optimization problem. Thus, to cope with the

[2] A *bin* denotes a type of circuitry-defect in a die. There are various types of bins, and a die of any type of bin is considered to be a bad die.

enormous computational complexity, we propose an ordinal optimization theory based two-level algorithm to solve the formulated problem for a good enough solution. This computationally intractable problem is most suitable for the application of our OO theory based two-level algorithm to seek for good enough threshold values.

We organize this chapter in the following manner. In Section 3.2, we formulate a stochastic optimization problem on the simulated wafer testing procedures. In Section 3.3, we will present the proposed OO theory based two-level algorithm and justify its performance using simulations. In Section 3.4, we will present the application of the OO theory based two-level algorithm to reduce overkills and retests in semiconductor wafer testing process. In Section 3.5, we will show the test results of applying the proposed algorithm on two real cases and demonstrate the solution quality by comparing with a vast number of randomly generated solutions and competing methods. Finally, we will make a conclusion in Section 3.6.

## 3.2 Problem Statements and Mathematical Formulation

### 3.2.1 Testing Procedures

Wafer probing establishes a temporary electrical contact between test equipment and each individual die (or chip) on a wafer to determine the goodness of a die. We employ typical testing procedures used in a local world-renowned wafer foundry. Figure 3.1 shows the flow chart of the real and simulated testing procedures. All the solid blocks represent the real testing procedures, while the dashed blocks are added for the purpose of computer simulation. The operation of the real testing procedures is briefly described in the following.

For every wafer, the wafer probing is performed twice as shown in the solid square marked by I in Figure 3.1. The second probing applies only to those dies failed in the first one. A die is considered to be good if it is good in either probing. If a die is detected to have bins in both tests, the bin detected in the second probing is taken as the bin of that die. We let $g_j$ ($\overline{g}_j$) denote the number of good (bad) dies in wafer $j$, and let $b_{jk}$ denote the number of dies of bin $k$ in wafer $j$. Assume there are $K$ types of bins in a wafer, then $\overline{g}_j = \sum_{k=1}^{K} b_{jk}$ and $g_j = TD_j - \overline{g}_j$ as shown in the square marked by II in Figure 3.1, where $TD_j$ denotes the total number of dies in wafer $j$. Following the two times of wafer probing and the calculation of $g_j$ and $\overline{g}_j$, a two-stage checking on the number of good dies is performed to determine the necessity of carrying out a retest, i.e. an additional wafer probing. The mechanism of the two-stage checking described in the part of the testing procedures enclosed in the dotted contour can be summarized below. We let $g_{W\min}$ denote the threshold value for the *lower bound* of the number of good dies in a wafer to determine whether to pass or hold the wafer; we let $n_{k\max}$, $k = 1,\ldots,K$, denote the threshold value for the *upper bound* of the number of dies of bin $k$ in the hold wafer to determine whether to perform a retest. If $g_j \geq g_{W\min}$, we pass wafer $j$ as shown in the diamond-shape block marked by III.a and the square marked by III.c; otherwise, we will hold this wafer and check its bins. For the hold wafer $j$, if $b_{jk} \leq n_{k\max}$ for all $k$, then wafer $j$ will be passed, as shown in the diamond-shape block marked by III.b and the square marked by III.c. However, if the hold wafer $j$ consists of any bin $k$ with $b_{jk} > n_{k\max}$, retests will be performed for all dies of bin $k$ in wafer $j$ to check for possible probing errors as shown in the diamond-shape block and square marked by IV.b and IV.d. Then, the overkills will be saved when there are probing errors as shown in the square marked by V. For bin $k$ in the hold wafer $j$ with $b_{jk} \leq n_{k\max}$, we pass it as shown in the diamond-shape block and square marked by IV.b and IV.c. This threshold value checking process will continue until all bins are checked as indicated in the diamond-shape blocks and squares marked by IV.e, IV.f, IV.g, and IV.h.

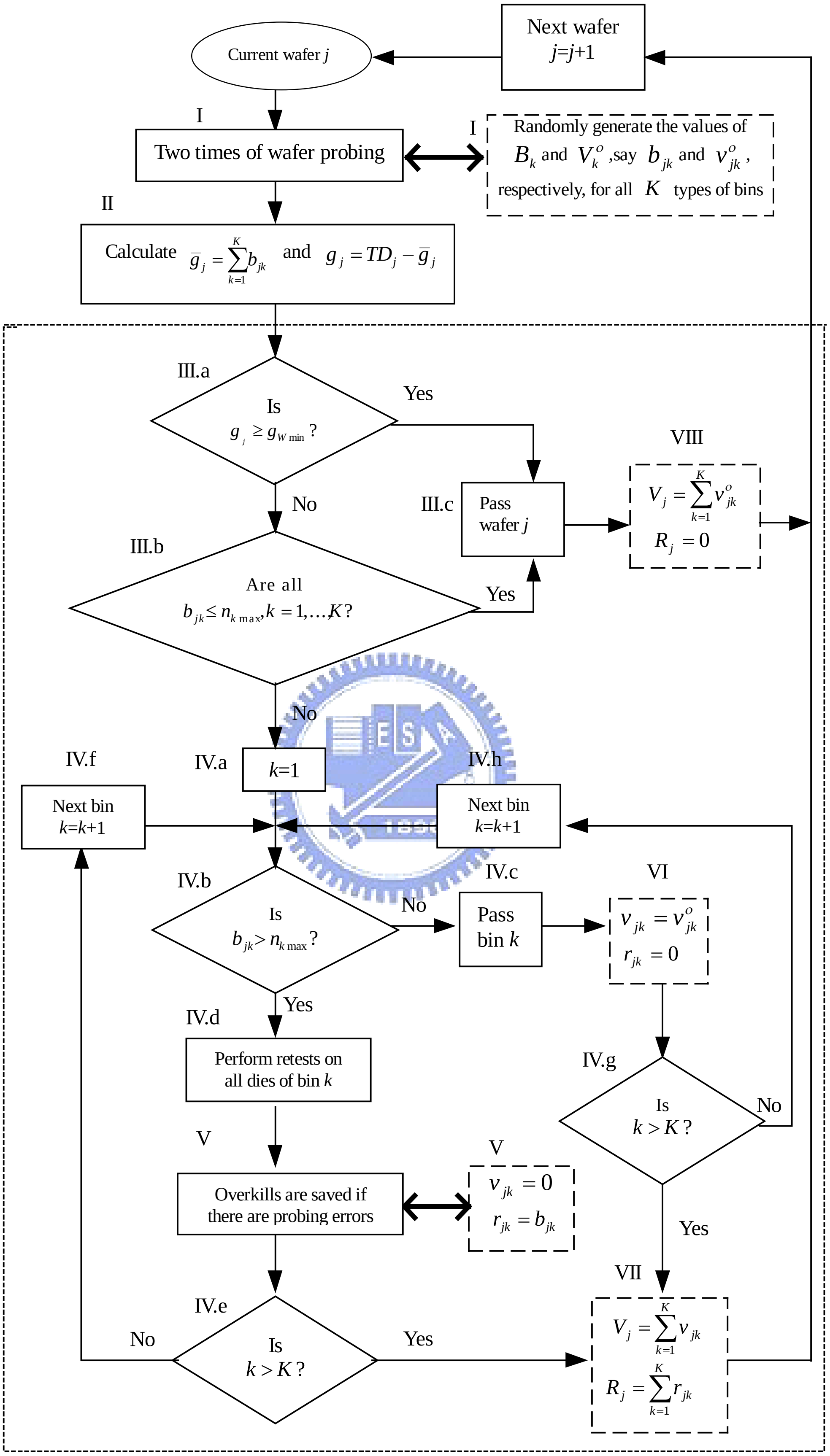


Figure 3.1: Flow chart of the real and simulated wafer testing procedures.

### 3.2.2 Computer Simulation of the Testing Procedures

1. *Simulation Model for the Two-times Wafer Probing*

Since we cannot perform the real wafer probing in computer, for the purpose of simulation, we need to build up a simulation model for the two times wafer probing. We let $B_k$ denote the discrete random variable for the number of dies of bin $k$ in a wafer. Since $P(B_k = n)$ can be provided by the real data, we can randomly generate the value of $B_k$ for a wafer based on the discrete probability mass function $P(B_k = n)$.

Each die of bin $k$ can be either an actual bin caused by manufacturing errors or an overkill caused by testing errors. Thus we can treat the overkills in $B_k$ as a binomial random variable with probability $p_k$, which represents the probability of overkills in dies of bin $k$ and can be provided by real data. We let $V_k^o$ denote the random variable for the number of overkills in $B_k$. Then, once the value of $B_k$ is randomly generated, we can randomly generate the value of $V_k{}^o$ based on a binomial probability distribution with probability $p_k$.

2. *Simulation of the Testing Procedures*

We let $b_{jk}$ and $v_{jk}^o$ denote the values generated from the random variables $B_k$ and $V_k^o$ for wafer $j$, respectively. The two times wafer probing in Figure 3.1 will be replaced by the random generator of $B_k$ and $V_k^o$ shown in the dashed square marked also by I in Figure 3.1. The dashed squares in Figure 3.1 except for the one mentioned above are for calculating the number of overkills and retests resulted from the simulated testing procedures. In contrast to $v_{jk}^o$, we let $v_{jk}$ denote the number of overkills for bin $k$ of wafer $j$ after completing the testing procedures and let $r_{jk}$ denote the corresponding number of retests. In the testing procedures, although we may pass the wafer when the threshold value test is a success, there may be overkills. We let $V_j$ and $R_j$ denote the total number of overkills and retests in wafer $j$, respectively. Thus for the passed wafer $j$,

$V_j = \sum_{k=1}^{K} v_{jk}^{o}$ and $R_j$=0 as shown in the dashed square marked by VIII in Figure 3.1. The same logic applies to the passed bin $k$ of the hold wafer $j$ that $v_{jk} = v_{jk}^{o}$ and $r_{jk}$=0 as shown in the dashed square marked by VI in Figure 3.1. However, for any retested bin, the probability of any unidentified overkill is extremely small, because the dies had been probed three times, which include two times wafer probing before any retest. Thus, for any retested bin $k$, $r_{jk} = b_{jk}$ and we assume $v_{jk}$=0, because the overkills are saved, as shown in the dashed square marked also by V in Figure 3.1; the solid square marked by V will be replaced by this dashed square in the simulated testing procedures. Once all the threshold value tests for all bins of the hold wafer $j$ are completed, we can compute $V_j$ and $R_j$ as shown in the dashed square marked by VII in Figure 3.1. The resulting values of $V_j$ and $R_j$ of wafer $j$ will be used to calculate $E[V] = \frac{1}{L}\sum_{j=1}^{L} V_j$ and $E[R] = \frac{1}{L}\sum_{j=1}^{L} R_j$, which represent the average overkills and retests per wafer, respectively, and $L$ denotes the total number of tested wafers.

### 3.2.3 Problem Formulation

From Figure 3.1, we see that if we increase $g_{W\min}$ while decreasing $n_{k\max}$, that is setting more stringent threshold values, there will be more retests and less overkills. This shows a conflicting nature between the overkills and retests. Thus, to reduce overkills under a tolerable level of retests, we will set minimizing the average number of overkills per wafer, $E[V]$, as our objective function while keeping the average number of retests per wafer, $E[R]$, under a satisfactory level. Thus, our problem for determining the threshold values can be formulated as the following constrained stochastic optimization problem:

$$\min_{x \in X} \; E[V]$$

subject to {simulated wafer testing procedures in Figure 3.1},

$$E[R] \le r_T, \qquad (3.1)$$

where $x \equiv [g_{W\min}, n_{k\max}, k = 1,\ldots,K]$ denotes the vector of threshold values, that is the vector of decision variables; $X$ denotes the decision variable space; $r_T$ denotes the tolerable average-number of retests per wafer.

**Remark 2:** a) The value of $r_T$ can be determined by the decision maker based on the economic situation. When the chip demand is weak, the throughput, in general, is not critical in the manufacturing process; therefore, we can allow a larger $r_T$ so as to save more overkills to gain more profit. On the other hand, if the chip demand is strong, then the throughput is more important, and we should set the value of $r_T$ smaller. Taking the chip demand into account is a *distinguished feature* of the proposed formulation. b) It is possible to pursue the relationships between the number of retests and the throughput. Then if we can derive the profit in terms of the throughput and the overkill, we can formulate an unconstrained optimization problem to maximize the profit. However, the relationships between the profit and throughput are very complicated due to the status of chip demand. For instances, when the chip demand is strong, larger throughput implies higher profit; on the other hand, if the chip demand is weak, larger throughput will cause inventory problem, which will hurt the profit. Therefore, the current formulation is simple and direct for a decision maker.

Since the constraint on $E[R]$ shown in (3.1) is a soft-constraint in a sense, we can use a penalty function to relax that constraint and transform (3.1) into the following unconstrained stochastic optimization problem:

$$\min_{x \in X} \; E[V] + P(E[R] - r_T) \times (E[R] - r_T)^+$$

$$\text{subject to \{simulated wafer testing procedures in Figure 3.1\},} \qquad (3.2)$$

where $P(E[R] - r_T)$ denotes a continuous penalty function for the constraint $E[R] \le r_T$, and $(x)^+ = \max(x, 0)$.

## 3.3 The OO Theory Based Two-Level Algorithm and Performance Evaluation

### 3.3.1 The OO Theory Based Searching Procedures

The considered stochastic simulation optimization problem is stated in the following

$$\min_{\theta\in\Theta} J(\theta) \tag{3.3}$$

where $\Theta$ is a huge decision-variable space, and $J(\cdot)$ is the objective function, which may be an expected output or a function of expected outputs of the simulated system. To cope with the computational complexity of this problem, we will employ the Ordinal Optimization (OO) theory based searching procedure [26]-[27], which efficiently seeks a good enough solution with high probability instead of searching the best for sure based on the observation that the performance order of the decision-variable vectors is likely preserved even evaluated by a crude model. From here on, we will use the word vector to represent the vector of decision variables.

The existing searching procedure of OO can be summarized in the following [27]: (i) Uniformly or randomly select $N$, say 1000, vectors of decision variables from $X$. (ii) Evaluate and order the $N$ vectors using an approximate model, then pick the top $s$, say 35, vectors to form the estimated good enough subset. (iii) Evaluate and order all the $s$ vectors obtained from (ii) using the exact model, then pick the top $k(\geq 1)$ vectors. The basic idea of the OO theory is based on the following observation: the performance order of the decision variables is likely preserved even evaluated using a crude model. Thus, the OO approach can reduce the searching space using cheaper evaluation to save computational time as indicated in (ii), and the best vector of decision variables obtained in (iii) is proved in [27] to be a good enough, top 5%, solution among $N$ (=1000) with probability 0.95.

However, the good enough solution of problem (3.3) that we are searching for should be a good enough vector in $\Theta$ instead of the $N$ vectors unless $\Theta$ is as small as $N$ [28]-[29]. As indicated in a recent paper by Lin and Ho [30], under a moderate modeling noise, the top 3.5% of the uniformly selected $N$ vectors will be among the top 5% vectors of a huge $\Theta$ with a very high probability ($\geq 0.99$), and the best case can be among the top 3.5% vectors of $\Theta$ provided that there is no modeling error. However, for $\Theta$ with size of $10^{30}$, a top 3.5% vector is a vector among the top $3.5 \times 10^{28}$ ones. This certainly not seems to be a good enough solution in the sense of practical optimization; however, it is acceptable only when $\Theta$ consists of lots of good vectors so that even if the performance order of the selected vector is not practically good enough, the corresponding objective value acceptable only when $\Theta$ consists of lots of good vectors so that even if the performance is. As a matter of fact, most of the practical stochastic simulation optimization problems do not have lots of good vectors; otherwise, finding a good enough solution won't be difficult. Therefore to apply the existing ordinal optimization searching procedures, we need to develop a new scheme to select $N$ excellent vectors from $\Theta$ to replace (i) so as to ensure the final selected-vector is a good enough solution of (3.3) from the practical viewpoint.

Heuristic methods for obtaining $N$ excellent vectors may depend on how well one's knowledge about the considered system. For instance in the optimal power flow problems with discrete control variables, Lin et al. proposed an algorithm based on the OO theory and engineering intuition to select $N$ excellent discrete control vectors [31]. However, the engineering intuition may work only for specific systems. Thus, in this section, we will propose an OO theory based systematic approach to select $N$ excellent vectors from $\Theta$ and combine with the existing ordinal optimization searching procedures to find a good enough solution of (3.3). The systematic method we propose here for finding $N$ excellent vectors is a combination of an Artificial Neural Network (ANN) and the Genetic Algorithm (GA). We use the ANN to construct a crude model required to evaluate the objective values of the vectors. Using this efficient evaluation for the fitness value of a vector, GA can efficiently find $N$ excellent vectors from $\Theta$.

## 3.3.2 Finding $N$ Roughly Good Vectors from Decision Variables Space

As indicated in the OO theory [26]-[27], performance “order” of the vectors is likely preserved even evaluated using a crude model. Thus, to select $N$ roughly good vectors from $\Theta$ without consuming much computation time, we need to construct a crude but effective and efficient model to evaluate the objective value of (3.3) for a given vector $\theta$, and use an efficient scheme to select $N$ roughly good vectors. Our crude model is constructed based on an ANN [32], and our selection scheme is GA [33].

### 3.3.2.1 The Artificial Neural Network (ANN) Based Model

Considering the inputs and outputs as the vectors $\theta \in \Theta$ and the corresponding objective values $J(\theta)$, respectively, we can use an ANN to implement the mapping from the inputs to the outputs [32]. First of all, we will select a representative subset of $\Theta$ by uniformly picking $M$, say 1000, vectors from $\Theta$. Then we will evaluate the objective values of these $M$ vectors using an *exact model*, which can be a stochastic simulation with moderate number of test samples as indicated in [28]. These collected $M$ input-output pairs of $(\theta, J(\theta))$ will be used to train the ANN to adjust its arc weights. Once this ANN is trained, we can input any vector $\theta$ to obtain an estimation of the corresponding $J(\theta)$ from the output of the ANN; in this manner, we can avoid an accurate but lengthy stochastic simulation to evaluate $J(\theta)$ for a given $\theta$. This forms our crude but efficient model to roughly estimate the objective value of (3.3) for a given vector $\theta$. Effectiveness of this crude model is justified by the OO theory as mentioned above, because what we care here are the relative order of $\theta$ ’s, not the value of $J(\theta)$ ’s.

### 3.3.2.2 The Genetic Algorithm (GA)

By the aid of the above effective and efficient objective value (or the so-called fitness value in GA terminology) evaluation model, we can efficiently select $N$ roughly good vectors from $\Theta$ using GA, which is briefly described as follows. Assuming an initial random population produced and evaluated, genetic evolution takes place by means of three basic genetic operators: (a) parent selection; (b) crossover; (c) mutation. The population in GA terminology represents a vector $\theta$ in our problem, and each population is encoded by a string of 0s and 1s. The string is called a chromosome. Parent selection is a simple procedure whereby two chromosomes are selected from the parent population based on their fitness values. Solutions with high fitness values have a high probability of contributing new offspring to the next generation. The selection rule we used in our approach is a simple roulette-wheel selection [33]. Crossover is an extremely important operator for the GA. It is responsible for the structure recombination (information exchange between mating chromosomes) and the convergence speed of the GA and is usually applied with relatively high probability, say 0.7. The chromosomes of the two parents selected are combined to form new chromosomes that inherit segments of information stored in parent chromosomes. There are many crossover scheme, we employ the single-point crossover [33] in our approach. While crossover is the main genetic operator exploring the information included in the current generation, it does not produce new information. Mutation is the operator responsible for the injection of new information. With a small probability, random bits of the offspring chromosomes flip from 0 to 1 and vice versa and give new characteristics that do not exist in the parent population. In our approach, the mutation operator is applied with a relatively small probability 0.02 to every bit of the chromosome.

There are two criteria for the convergence of GA. One is when the fitness value of the best population does not improve from the previous generation, and the other is when evolving enough generations.

The initial populations of the GA employed in our first-level approach are $I$, say 5000, randomly selected vectors from $\Theta$. After the applied GA converges, we rank the final generation of these $I$ populations based on their fitness values and pick the top $N$ populations, which form the $N$ roughly good vectors that we look for.

### 3.3.3 Searching the Good Enough Solution Among the $N$

Starting from the selected $N$ roughly good vectors, in the second-level, we will proceed directly with step (ii) of the existing ordinal optimization searching procedures described in Section 3.3.1. In this step, we will evaluate the objective value of each vector using a more refined model[3] than the crude one employed in the first-level. We will order the $N$ vectors based on the estimated objective values and choose the top $s$ vectors to form the Selected Subset (SS). Then, we will evaluate each of the $s$ vectors using the exact model, which is a stochastic simulation with sufficiently large number of test samples that makes the value estimation of $J(\theta)$ for a given $\theta$ sufficiently stable, of the considered problem as indicated in step (iii) of the existing ordinal optimization searching procedures. The vector associated with the smallest objective value of (3.3) among $s$ is the good enough solution that we seek.

### 3.3.4 The OO Theory Based Two-level Algorithm

Now, our OO theory based two-level algorithm can be stated as follows.

**Step 1**: Uniformly select $M$ $\theta$'s from $\Theta$ and use an exact model to compute the corresponding $J(\theta)$'s. Train an ANN (or ANNs) by adjusting its (or their) arc weights using the mapping between the given $M$ input-output pairs, that are the $M$ $(\theta, J(\theta))$'s.

**Step 2**: Randomly select $I$ vectors from $\Theta$ as the initial populations. Apply GA to these populations using the efficient and effective fitness-value evaluation model based on the ANN trained in Step 1. After the algorithm converges, we rank all the final $I$ populations based on their fitness values and select the top $N$ populations.

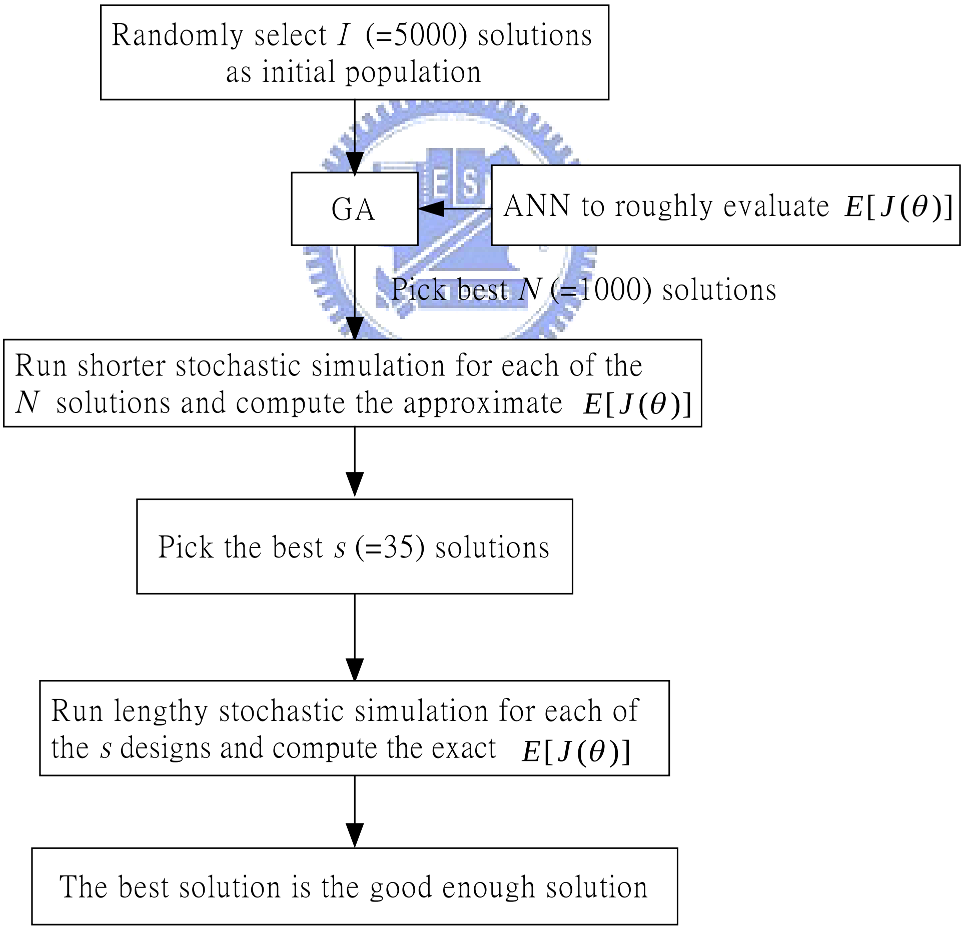
Steps 1 and 2 constitute the first-level approach.

[3] This more refined model can be, for example, a stochastic simulation with small number of test samples [28] to evaluate the objective value of a given vector in the considered problem.

**Step 3**: Use a more refined model than the ANN to estimate the objective values of the $N$ vectors obtained in Step 2. Rank the $N$ vectors based on their estimated objective values and select the top $s$ vectors.

**Step 4**: Use the exact model of the considered problem to compute the objective values of the $s$ vectors. The vector with the smallest objective value of (3.3) is the good enough solution.

Steps 3 and 4 represent the procedures of the second-level approach. Thus, the overall structure of the proposed OO theory based two-level algorithm can be shown in Figure 3.2.



Randomly select $I$ (=5000) solutions as initial population
GA
ANN to roughly evaluate $E[J(\theta)]$
Pick best $N$ (=1000) solutions
Run shorter stochastic simulation for each of the $N$ solutions and compute the approximate $E[J(\theta)]$
Pick the best $s$ (=35) solutions
Run lengthy stochastic simulation for each of the $s$ designs and compute the exact $E[J(\theta)]$
The best solution is the good enough solution

Figure 3.2. The structure of the OO theory based two-level algorithm.

## 3.3.5 Performance Evaluation

### 3.3.5.1 Performance Evaluation of the First-level Approach

Since the performance of the second-level approach had been thoroughly investigated in [27], what we need to address here is how excellent the $N$ selected vectors are among the various types of decision-variable space $\Theta$ so as to demonstrate the validity of our first-level approach. This evaluation is carried out in the following, while the performance of the two-level algorithm will be presented afterwards.

As indicated in [27], the Order Performance Curve (OPC) of all the ordered vectors $\theta_1, \theta_2, \ldots, \theta_{|\Theta|}$ in $\Theta$ is determined by the spread of the order performance $J_{[1]}, J_{[2]}, \ldots, J_{[|\Theta|]}$, where $J_{[i]}$ denotes $J(\theta_i)$. Without loss of generality, $J_{[i]}$'s can be normalized into the range [0,1], i.e., for $i = 1,2,\ldots,|\Theta|$, $y_i = (J_{[i]} - J_{[1]})/(J_{[|\Theta|]} - J_{[1]})$. Meanwhile, the ordered $|\Theta|$ vectors, spaced equally, are also mapped into the range [0,1] such that for $i = 1,2,\ldots,|\Theta|$, $z(\theta_i) \equiv z_{[i]} = (i-1)/(|\Theta|-1)$. There are five broad categories of OPC models: (i) lots of good vectors, (ii) lots of intermediate but few good and bad vectors, (iii) equally distributed good, bad and intermediate vectors, (iv) lots of good and lots of bad but few intermediate vectors, and (v) lots of bad vectors. Figure 3.3 shows a graphical expression of these five types of OPCs. More precisely, a standardized OPC can be determined by a two-parameter smooth curve $F^{-1}(z \mid \alpha,, \beta) = F(z \mid \frac{1}{\alpha}, \frac{1}{\beta})$, where $F(z \mid \cdot,\cdot)$ is the Incomplete Beta function of the two parameters $(\cdot,\cdot)$. In general, $\alpha$ <1, $\beta$ >1

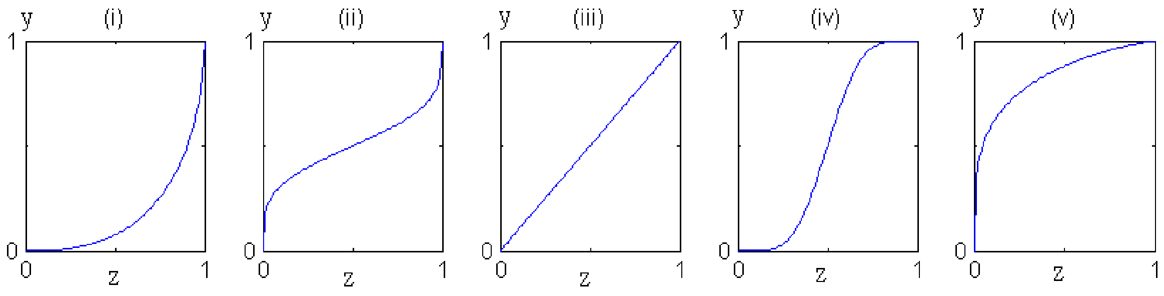


Figure 3.3: Five types of standardized OPCs.

corresponds to the OPC of type (i); $\alpha>1$, $\beta>1$ corresponds to the OPC of type (ii); $\alpha=1$, $\beta=1$ corresponds to the OPC of type (iii); $\alpha<1$, $\beta<1$ corresponds to the OPC of type (iv); $\alpha>1$, $\beta<1$ corresponds to the OPC of type (v). As indicated in Section 3.1, we need not consider the types of $\Theta$ consisting of lots of good vectors in this evaluation, thus we take only the three OPC types (ii), (iii) and (v) into account. For the purpose of evaluation, we assume the size of the decision-variable space $\Theta$ to be $10^{30}$.[4]

The roughness of the ANN model can be described by adding a uniform noise to the normalized performances $y_i$'s [26]-[27]. That means, the model of ANN can be described by the noisy model $y_i+\omega$, where the random noise $\omega$ is generated from the uniform distribution random variable U=[-0.01,0.01][5]; note that this range of noise seem conservative however it can switch the order of $2\times10^{28}$ vectors for a type (iii) OPC.

We studied a total of 28 OPCs distributed uniformly among the three broadly generic types, (ii), (iii) and (v), formed from the following parameters: $\alpha$=1.0, 2.0, 4.0, 5.0 and $\beta$=0.2, 0.4, 0.8, 1.0, 2.0, 4.0, 5.0. In all of our Monte-Carlo calculations, we simulate 10000 realizations of noisy OPCs. We found that the top 5% of the top ranked $N$ (=1000) populations obtained after GA converges are lying in the top $10^{-6}$% of the $|\Theta|$ (=$10^{30}$) populations with probability 0.99. This result is extremely better than the uniformly selected $N$ vectors whose top 3.5% vectors can at best (i.e. no modeling error) be the top 3.5% vectors of $\Theta$ as indicated in [30]. This shows that the $N$ vectors obtained by our first-level approach are really excellent.

[4] Since what we care here is the ranking percentage of the selected $N$ vectors among $\Theta$, we can, without loss of generality, assume $|\Theta|=10^{30}$ for a typical huge decision-variable space.

[5] The magnitude of noise for describing the roughness of a crude model is determined either based on an engineering judgment or an empirical experiment; in our case, it is estimated from an experiment of this crude model for the application problem of this chapter.

**Remark 3:** Though we do not investigate the actual order of the $N$ vectors for the OPC types (i) and (iv), our first-level approach can still be applied for problems with $\Theta$ of these two types of OPCs. This is because even if the order of the obtained $N$ vectors of the two types of OPC may not be as good as those of the other three OPC types due to the sharp sensitivity of the noise to the performance in these two types, however their actual objective values will still be good enough due to the existence of lots of good vectors. That means in both OPC types (i) and (iv), there can be a big difference in the order of good vectors but the difference in objective values are very small. Thus, no matter what types of OPC we are facing, our first-level approach processes the same.

### 3.3.5.2 Performance Evaluation of the Two-level Algorithm

As indicated in Section 3.3.1, for $N$ =1000, $s$ =35, the top vector we obtain in Step 4 of the two-level algorithm must be among the top 5% of the $N$ vectors with probability 0.95. Then, combining the performance evaluation for the first-level approach, we can conclude the following: the good enough solution obtained by the OO theory based two-level algorithm is among the top $10^{-6}$ % of $\Theta$ with probability $0.95\times 0.99$.

## 3.4 Application of the OO Theory Based Two-Level Algorithm

The size of the decision variable space $X$ in (3.2) is huge; for example, for an 8-inch wafer, which consists of, say 2500 dies, the possible ranges of the integer values $g_{W\min}$ and $n_{k\max}$ are [1, 2500] and [1, 2500], respectively. Consequently for the number of bin types $K=12$, the size of $X$ will be more than $10^{30}$. The evaluation of the performance of each vector of decision variables requires a lengthy stochastic simulation of the testing procedures. Therefore, any global searching techniques for solving the simulation optimization type problem (3.2) will be very computationally expensive. To cope with the computational complexity of this problem, we propose an Ordinal Optimization (OO) theory based two-level algorithm to solve for a good enough solution with high probability instead of searching the best for sure.

As indicated in Section 3.3.1, we see that the quality of the good enough solution heavily depends on the quality of the randomly selected $N$ vectors of decision variables. Thus to improve the existing OO searching procedures, we can apply the OO theory to select $N$ roughly good vectors of decision variables from $X$, to ensure the top 5% solutions among $N$ to be the good enough solutions of $X$. This is what we called the first-level OO approach for replacing the existing searching procedure (i). Combining first level approach with the existing searching procedures (ii) and (iii) forms a two-level OO algorithm.

### 3.4.1 Constructing a Metamodel for (3.2)

The very first step for choosing $N$ roughly good vectors from $X$ should be constructing a *metamodel* or *surrogate model* for the considered stochastic simulation optimization type problem. There are various techniques to approximate the relationships between the inputs and outputs of a system such as the linear regression, response transformation regression, projection-pursuit regression and artificial neural network (ANN) [34], etc…. Among them, ANN is considered to be a universal function approximator [35] due to its genetic, accurate and convenient property to model complicated nonlinear input-output relationships. ANN not only approximate the continuous functions well [36]-[37], but also being used to construct metamodels for discrete event simulated systems in [38] and [39]. Since what we care here is the performance *order* of the solution rather than the performance *value* as considered in [38] and [39], we can trade off the accuracy of the ANN based metamodel with the training time by using simple ANN with reasonable size of training data set. Two simple feed forward two-layer ANNs are employed here. One is to approximate the relationships between $x \in X$ and the corresponding $E[V]$, and the other is for $x \in X$ and $E[R]$. In these two ANNs, there are 16 neurons with hyperbolic tangent sigmoid function in the first layer, and 1 neuron with linear function in the second layer. We obtain the set of training data for the two ANNs by the following two steps. (a) Narrow down the decision-variable space $X$ by excluding the irrational threshold values and denote the reduced decision variable space by $\hat{X}$[6]. (b) Uniformly select $M$ vectors from $\hat{X}$ and compute the corresponding outputs $E[V]$ and $E[R]$ using a stochastic simulation of the testing procedures shown in Figure 3.1. As indicated above, $M$ need not be a very large value. The objective value of (3.2) can be computed based on the values of $E[V]$ and $E[R]$. Thus, we can obtain $M$ pairs of decision variables and the corresponding objective values for (3.2). To speed up the convergence of the back propagation training, we employed the Levenberg-Marquardt algorithm [40] and the scaled conjugate gradient algorithm [41] to train the ANNs for $E[V]$ and $E[R]$, respectively. Stopping criteria of the above two training algorithms are when any of the following two conditions occurs: (i) the sum of the mean squared errors is smaller than $10^{-5}$, and (ii) the number of epochs exceeds 500. Once these two ANNs are trained, we can input any vector $x$ to the two ANNs to estimate the corresponding $E[V]$ and $E[R]$, which will be used to compute the objective value of (3.2). This forms our metamodel to estimate the objective value of (3.2) for a given vector of decision variables $x$.

### 3.4.2 Using GA to Select $N$ Roughly Good Vectors of Decision Variables from $\hat{X}$

By the aid of the above ANN model, we can search $N$ roughly good vectors of decision variables from $\hat{X}$ using heuristic global searching techniques.

Since the searching techniques of Genetic Algorithm (GA), Evolution Strategies (ES) and Evolutionary Programming (EP) [42] improve a pool of populations from iteration to iteration, they should best fit our needs. For the sake of explanation and easier implementation, we employ the GA [43, Chapter 14] as our searching tool.

[6] The threshold values, $g_{W\min}$ and $n_{k\max}$, should lie in a reasonable range determined by the corresponding average values of $g_j$ and $b_{jk}$ collected from a wafer foundry.

The coding scheme of the GA we employed to represent all the vectors in $\hat{X}$ is rather straightforward, because each component of the vector $x$ is an integer. We start from $I$, say 5000, randomly selected vectors from $\hat{X}$ as our initial populations. The fitness of each vector is set to be the reciprocal of the corresponding objective value of (3.2) computed based on the outputs of the two ANNs. The members in the mating pool are selected from the pool of populations using roulette wheel selection scheme. 70% of the members in the mating pool are randomly selected to serve as parents for crossover. We use a single point crossover scheme and assume the mutation probability to be 0.02. We stop the GA when the iteration number exceeds 30. After the applied GA converges, we rank the final $I$ populations based on their fitness and pick the top $N$ populations, which are the $N$ roughly good vectors of decision variables.

**Remark 4:** Although there exists in-depth analysis of the approximation errors for ANN to approximate continuous functions [36]-[37], the accuracy of approximating the input and output relationships of a discrete event simulated system is usually addressed using empirical results [38]-[39]. Thus, it is not surprising that we do not get any analytical result for the quality of the $N$ vectors selected above. However, similar to the study in [27], we assume various magnitudes of modeling noise of uniform distribution to represent the approximation errors caused by the proposed ANN based metamodel and make the following simple experiments to compare the quality of the $N$ vectors selected by GA based on the ANN model with those selected in random from the solution space. We let $U$[-0.1,0.1] denote the uniform distribution of a random noise ranging from -0.1 to 0.1 to be added to the normalized performance, i.e. the normalized objective value, of the exact model. The normalized performance for all solutions in a solution space is equally-spaced ranging from 0 to 1 with 0 as the top performance. In [27], a normalized Ordinal Performance Curve (OPC) is used to describe the performance structure of all the solutions in a solution space. Assume $|X| = 10^{30}$, $N = 1000$, we carried out a Monte Carlo study for vast number of OPCs similar to that in [27] for an assumed noise distribution and pick the top $N$ vectors using GA. We found the following results. For the modeling noise distribution, $U$[-0.01,0.01], $U$[-0.1,0.1] and $U$[-0.5,0.5], the top 5% solutions in $N$, which are selected by GA, is at least a top $10^{-6}$%, top $10^{-3}$%, and top $10^{-2}$% solution in $X$ with probability 0.95, respectively. However, the top 5% solutions in $N$, which are selected in random, is at best, i.e. assuming no modeling noise, a top 5% solution in $X$ only. Therefore, we have greatly improved the quality of the $N$ vectors by replacing the existing searching procedure (i).

### 3.4.3 Using an Approximate Model for Selecting the Estimated Good Enough Subset

Starting from the $N$ vectors of decision variables obtained in Section 3.4.2, we will proceed with (ii) of the OO searching procedure to compute the objective value of (3.2) for each vector using an approximate model. As indicated in [28], this approximate model can be a stochastic simulation with moderate number of test wafers, that is to carry out the testing procedures shown in Figure 3.1 for $L_s$, say 300, wafers. We will then order the $N$ vectors of decision variables based on the obtained estimated objective values of (3.2) and choose the top $s$ vectors which form the estimated good enough subset.

### 3.4.4 Using the Exact Model to Determine the Good Enough Solution

We will compute the objective value of (3.2) for each of the $s$ vectors in the estimated good enough subset using the exact model that is a stochastic simulation with sufficiently large number of test wafers that makes the estimated objective value sufficiently stable. This exact model is similar to the approximate model mentioned above however replacing $L_s$ by $L_e (>> L_s)$ wafers. Then the vector associated with the smallest objective value of (3.2) among $s$ is the good enough solution that we seek.

## 3.5 Test Results and Comparisons

Our simulations are based on the following data collected from two products, say product A and product B, of a renowned wafer foundry in Taiwan. Both products are made in 6-inch wafers[7]. Each wafer of product A and product B consists of 203 dies and 206 dies, respectively. In the following, we will focus our description on product A, while the test results of product B will also be presented. It should be noted that all the test results shown in this section are simulated in a Pentium IV PC using Borland C++.

There are 12 bins in the wafers of product A. The probability mass function $P(B_k = n)$, $k$ =1,…,12, and the probability of the number of overkills in bin $k$, $p_k$, $k$ =1,…,12, are given. The yield rate of product A is 68%. The decision-variable space $X = \{x(=[g_{W\min}, n_{k\max}, k = 1,...,K]) \mid g_{W\min} \in [1,203], n_{k\max} \in [1,203], k = 1,...,12\}$. We used the sigmoid-type function as our penalty function $P$ in (3.2), i.e., $P = \eta \frac{1}{1+e^{-(E[R]-r_T)}}$, where $\eta$ ($\cong$ 0.1594) is a normalized coefficient such that $\eta = \frac{\max_{i\in\{1,...,M\}} E[V_i]}{\max_{i\in\{1,...,M\}} E[R_i]}$.

We set $\hat{X} = \{x(=[g_{W\min}, n_{k\max}, k = 1,..., K]) \mid g_{W\min} \in [50,203], n_{k\max} \in [1,6\mu_k], k = 1,...,12\}$, where $\mu_k$ is the mean of the number of dies of bin $k$. The parameters in the proposed two-level algorithm are set as follows: $L_s = 300$, $L_e = 10,000$, $M$ =1000, $I$ =5000, $N$ =1000, and $s$ =35. We have simulated 3 cases of different $r_T$ 's, which are 10, 30 and 50.

The good enough vector of threshold values and the average overkill percentage for the three cases of $r_T$ we obtained from the two-level algorithm are shown in Table 3.1. The CPU time consumed in each case plus the training time is approximately 6.05 minutes. From Table 3.1, we can observe that when $r_T$ increases, the values of $g_{W\min}$ increase as shown in row 2, and the values of leading $n_{k\max}$, $k = 5$ and 6, which account for most of the retests, decrease as shown in rows 7 and 8, respectively. This indicates that if we allow

[7] The reason we use 6-inch wafer products is for easier identification of the bins and overkills in experiments. In fact, our results can apply to any size of wafer.

Table 3.1: The good enough vector of threshold values and the average overkill percentage of product A for three different $r_T$ 's.

| $r_T$<br>Good enough vector of threshold values | 10 | 30 | 50 |
|---|---|---|---|
| $g_{W\min}$ | 146 | 163 | 176 |
| $n_{1\max}$ | 7 | 3 | 8 |
| $n_{2\max}$ | 3 | 8 | 5 |
| $n_{3\max}$ | 6 | 6 | 6 |
| $n_{4\max}$ | 5 | 6 | 5 |
| $n_{5\max}$ | 51 | 43 | 34 |
| $n_{6\max}$ | 32 | 23 | 16 |
| $n_{7\max}$ | 7 | 7 | 3 |
| $n_{8\max}$ | 7 | 3 | 6 |
| $n_{9\max}$ | 4 | 3 | 4 |
| $n_{10\max}$ | 4 | 3 | 2 |
| $n_{11\max}$ | 3 | 3 | 2 |
| $n_{12\max}$ | 2 | 9 | 5 |
| * $\frac{E[V]}{TD_A}\times 100\%$ | 1.86% | 1.07% | 0.27% |

* $TD_A$ : the total number of dies in a wafer of product A.

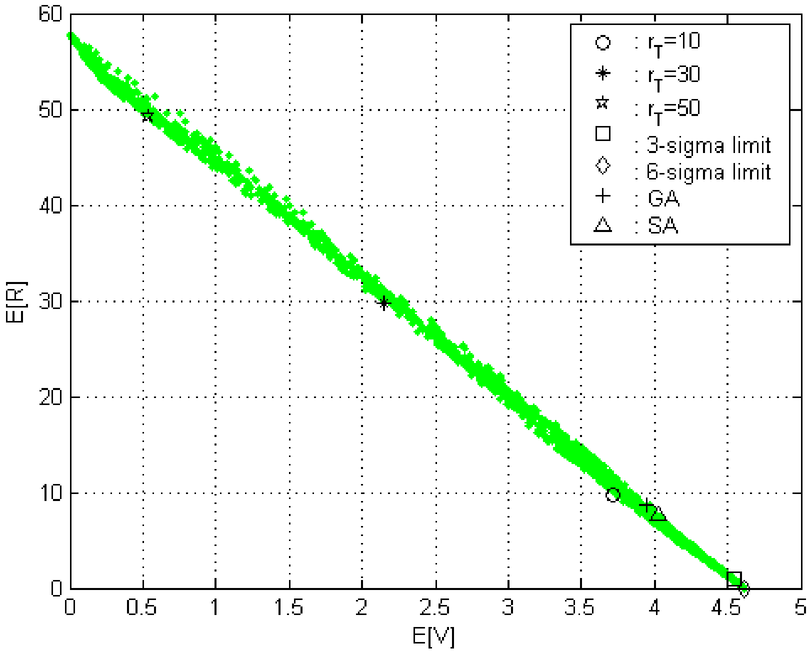


Figure 3.4: The resulted $(E[V], E[R])$ pairs of the 521 test wafers based on the vector of threshold values determined by two-level algorithm, random generator, three-sigma limit, six-sigma limit, GA and SA algorithm.

more retests (that is increasing $r_T$), we can set more stringent threshold values (that are increasing $g_{W\min}$ and decreasing the leading $n_{k\max}$ s), so as to save more overkills (that is the decreased average overkill percentage), as indicated in the last row of Table 3.1.

To demonstrate the real world performance of the vector of threshold values obtained by the two-level algorithm for the three cases shown in Table 3.1, we use 521 real test wafers, whose number of dies of all bins, $b_{jk}$, $j = 1,...,521$, $k = 1,...,12$, and overkills before retest, $v^o_{jk}$, $j = 1,...,521$, $k = 1,...,12$, are known. The corresponding results of the pair of the average overkills per wafer, $E[V](= \frac{1}{521}\sum_{j=1}^{521} V_j)$, and the average retests per wafer, $E[R](= \frac{1}{521}\sum_{j=1}^{521} R_j)$, for these 521 test wafers are shown in Figure 3.4 as the points marked by "☆", "*", "○" with the corresponding $r_T$ shown on the top right corner of the figure. We also use 2000 randomly selected vectors of threshold values to test the same 521 wafers; the resulted pairs of $E[V]$ and $E[R]$ are shown as the points marked by "●" in Figure 3.4. We see that for $E[R] \le 10$, the $E[V]$ resulted by the good enough vector of threshold values obtained by the two-level algorithm is almost the minimum compared with those resulted by the randomly selected vectors of threshold values. Similar conclusions can be drawn for the cases of $r_T$ =30 and 50. Since reducing overkills and retests have conflicting nature, the considered unconstrained stochastic optimization problem (3.2) possesses pareto optimal solutions. From Figure 3.4, we can see that the results we obtained for the cases of $r_T$ =10, 30 and 50 are almost on the boundary of the region resulted from the randomly generated vectors of threshold values; this implicit boundary represents the $(E[V], E[R])$ pairs resulted by the pareto optimal vectors of threshold values.

We also use the three-sigma limit and six-sigma limit to determine the threshold values such that $g^{3\sigma}_{W\min} = \mu_g - 3\sigma_g$, $n^{3\sigma}_{k\max} = \mu_k + 3\sigma_k$, $k = 1,...,12$, and $g^{6\sigma}_{W\min} = \mu_g - 6\sigma_g$, $n^{6\sigma}_{k\max} = \mu_k + 6\sigma_k$, $k = 1,...,12$, where $\mu_g$ and $\sigma_g$, the mean and standard derivation of the number of good dies in a wafer, and $\mu_k$ and $\sigma_k$, the mean and standard derivation of the number of dies of bin $k$, are obtained from the data set of 521 test wafers. Using these threshold values to test the same set of 521 test wafers, the resulted $(E[V], E[R])$ pairs from three-sigma limit and six-sigma limit are also shown in Figure 3.4 marked by “□“ and “◇”, respectively. For $E[R] \le 10$, we can see that our method will save 22% and 24% more overkills than the three-sigma limit and six-sigma limit, respectively. Considering the vast number of dies manufactured per month, the increased profit due to saving overkills will be too large to neglect. Furthermore, both three-sigma limit and six-sigma limit do not generate the pareto optimal solution for (3.2), and they cannot control the level of retests like ours.

We have also used typical GA and Simulated Annealing (SA) [42] algorithm to solve (3.2) for the case of $r_T$=10. As indicated at the beginning of Section 3.4, the global searching techniques are computationally expensive in solving (3.2). We stop the GA and SA when they consumed 50 times of the CPU time consumed by the two-level algorithm, and the objective values of (3.2) they obtained are still 5.4% and 8.1% more than the final objective value obtained by the two-level algorithm, respectively. Using the threshold values they obtained to test the 521 wafers, the resulted $(E[V], E[R])$ pairs from GA and SA are marked by “+“ and “Δ” in Figure 3.4. We found that using two-level algorithm, we can save 6.2% and 8.6% more overkills than using the GA and SA for $E[R] \le 10$, respectively. In addition, both GA and SA do not generate the pareto optimal solution, because the best so far solution they obtained for 5 hours of CPU time are still far away from the optimal solution of (3.2).

There are 10 bins in the wafers of product B. The probability mass function $P(B_k = n)$, $k$=1,…,10, and the probability of the number of overkills in bin $k$, $p_k$, $k$=1,…,10, are given. The yield rate of product B is 46.6%. We employed the same sigmoid-type function as that used in product A, however the normalized coefficient $\eta$ is 0.1207. Specific data in the two-level algorithm applying to product B are similar to the case of product A. We have also simulated 3 cases of different $r_T$’s, which are 20, 40 and 80. The CPU time consumed in each case plus the training time is approximately 5.2 minutes. More retests are requested here due to the lower yield of product B than A. The good enough vectors of threshold values and the average overkill percentage for the three cases of different $r_T$’s obtained by our algorithm are shown in Table 3.2. The values of $r_T$ versus the threshold values and the average overkill percentage have the same trend as in product A. From Table 3.2, we can observe that when $r_T$ increases, the values of $g_{W\min}$ increase as shown in row 2, and the values of leading $n_{k\max}$, $k=8$ and 9, which account for most of the retests, decrease as shown in rows 10 and 11, respectively.

We use 590 real wafers of product B to test the performance of the vector of threshold values shown in Table 3.2. The resulted pairs of $E[V](=\frac{1}{590}\sum_{j=1}^{590}V_j)$ and $E[R](=\frac{1}{590}\sum_{j=1}^{590}R_j)$ are shown in Figure 3.5 as the points marked by "☆", "*", "○" with the corresponding $r_T$ shown on the top right corner of the figure. The "•" points in Figure 3.5 denote the resulted pairs of $E[V]$ and $E[R]$ of the 2000 randomly generated vectors of threshold values applied to the same set of 590 wafers. From this figure, we can see that the results we obtained for the cases of $r_T$ =20, 40 and 80 are almost on the boundary of the region resulted from the randomly generated vectors of threshold values; this implicit boundary represents the $(E[V], E[R])$ pairs resulted by the pareto optimal vectors of threshold values.

We also use the three-sigma limit and six-sigma limit to determine the threshold values, the resulted $(E[V], E[R])$ pairs from three-sigma limit and six-sigma limit are also shown in Figure 3.5 marked by "□" and "◇", respectively. For $E[R] \le 20$, we can see that our method will save 21% and 24% more overkills than the three-sigma limit and six-sigma limit, respectively. We have also used typical GA and SA algorithm to solve (3.2) for the case of $r_T$ =20. We stop the GA and SA when they consumed 50 times of the CPU time consumed by the two-level algorithm, and the objective values of (3.2) they obtained are still 5.3% and 6.9% more than the final objective value obtained by the two-level algorithm, respectively. Using the threshold values they obtained to test the 590 wafers, the resulted $(E[V], E[R])$ pairs from GA and SA are marked by "+" and "Δ" in Figure 3.5. We found that using two-level algorithm, we can save 6.6% and 4.2% more overkills than using the GA and SA for $E[R] \le 20$, respectively.

Table 3.2: The good enough vector of threshold values and the average overkill percentage of product B for three different $r_T$ 's.

| $r_T$<br>Good enough<br>vector of threshold values | 20 | 40 | 80 |
|---|---|---|---|
| $g_{W\min}$ | 118 | 131 | 146 |
| $n_{1\max}$ | 6 | 2 | 3 |
| $n_{2\max}$ | 4 | 4 | 1 |
| $n_{3\max}$ | 3 | 4 | 5 |
| $n_{4\max}$ | 6 | 3 | 3 |
| $n_{5\max}$ | 5 | 1 | 4 |
| $n_{6\max}$ | 10 | 5 | 5 |
| $n_{7\max}$ | 7 | 4 | 2 |
| $n_{8\max}$ | 65 | 54 | 38 |
| $n_{9\max}$ | 76 | 63 | 45 |
| $n_{10\max}$ | 18 | 13 | 9 |
| * $\frac{E[V]}{TD_B}\times 100\%$ | 2.36% | 1.71% | 0.56% |

* $TD_B$ : the total number of dies in a wafer of product B.

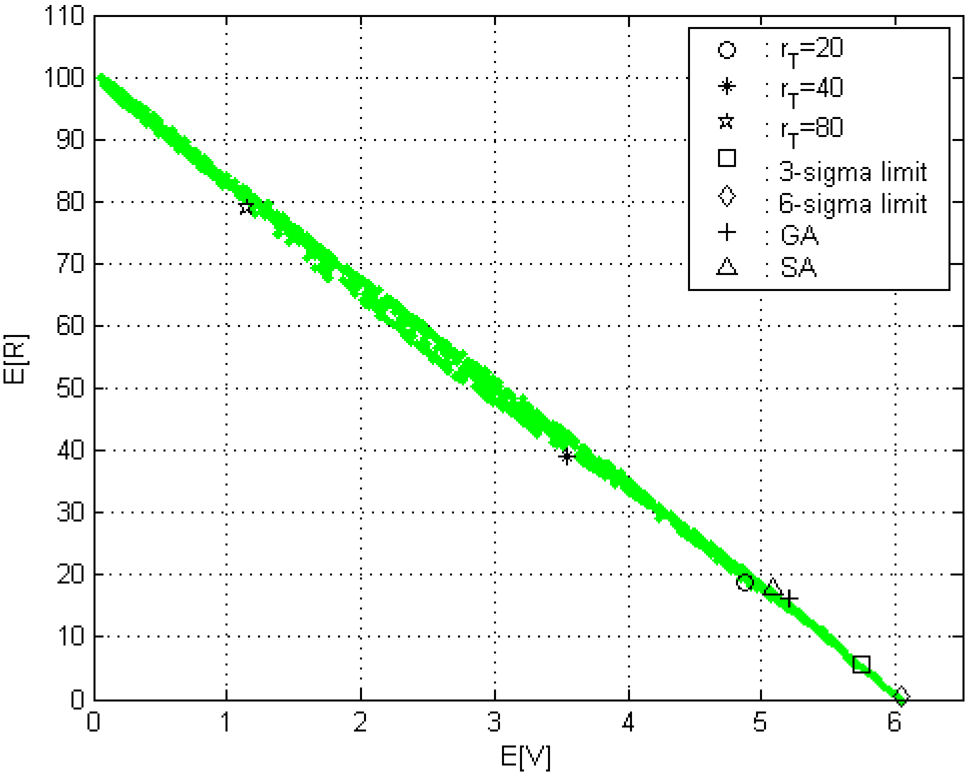


Figure 3.5: The resulted $(E[V], E[R])$ pairs of the 590 test wafers based on the vector of threshold values determined by two-level algorithm, random generator, three-sigma limit, six-sigma limit, GA and SA algorithm.

## 3.6 Concluding Remarks

To cope with the computationally intractable stochastic simulation optimization problems, we have proposed an ordinal optimization theory based two-level algorithm to solve for a good enough solution using reasonable computational time. We have justified the performance of the proposed algorithm based on the simulations.

To demonstrate the applicability of the proposed algorithm, we have used it to solve for a vector of good enough threshold values to reduce overkills and retests in a wafer testing process of a wafer foundry. We have tested the performance of the solution we obtained using the real data and found that the resulting average number of overkills and retests per wafer lie almost on the boundary resulted from the pareto optimal vector of threshold values of the considered stochastic optimization problem. This indicates that the proposed algorithm will not only control the tolerable level of retests by taking the various chip demand into account but also provide a near pareto optimal vector of threshold values. The vector of good enough threshold values obtained by the proposed algorithm is very successful in the aspects of solution quality and computational efficiency.

The proposed formulation for reducing overkills and retests is not limited to the testing process of a foundry, it can easily adapt to any general testing procedures. The proposed ordinal optimization theory based two-level algorithm is not limited to the problem considered in this chapter. In fact, it can be used to solve any hard optimization problem that requires lengthy computational time to evaluate the performance of a decision variable.

# Chapter 4

# Conclusions and Perspectives

Two related issues on the throughput and yield of wafer fabrication and testing processes have been discussed. In the first issue, we have presented a classification based fault detection and isolation scheme for the ion implanter, and in the second issue, we have presented an ordinal optimization approach to find the optimal threshold values to reduce the overkills of dies under a tolerable retest level in wafer testing process.

In the first issue, 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 to the ion implanter. Due to the efficient learning capability of HCT and the 0.05 seconds classification time for classifying the recipe of a working wafer, the proposed fault detection and isolation scheme can work on line and real-time.

In the second issue, the proposed ordinal optimization theory based two-level algorithm is not limited to the problem considered in this dissertation. In fact, it can be used to solve any hard optimization problem that requires lengthy computational time to evaluate the performance of a decision variable. Although the proposed approach presented in this dissertation was illustrated using a wafer testing problem, it is well suited to different application areas.

# Reference

[1] C. M. McKenna, “A personal historical perspective of ion implantation equipment for semiconductor applications,” *2000 International Conference on Ion Implantation Technology*, Alpbach, Austria, pp. 1-19, Sep. 2000.

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

[3] R. Isermann, “Fault diagnosis of machines via parameter estimation and knowledge processing,” *Automatica*, vol. 29, no. 4, pp. 815-835, July 1993.

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

[5] M. Basseville and A. Benveniste, (eds), “Detection of abrupt changes in signals and dynamical systems,” *Lecture Notes in Control and Information Sciences*, vol. 77, Springer-Verlag: Berlin, Dec. 1985.

[6] H. H. Yue, S. J. Qin, R. J. Markle, C. Nauert and M. Gatto, “Fault detection of plasma etchers using optical emission spectra,” *IEEE Transactions on Semiconductor Manufacturing*, vol. 13, no. 3, pp. 374-385, Aug. 2000.

[7] R. Isermann, “Model-based fault detection and diagnosis - status and applications,” *16th Symposium on Automatic Control in Aerospace*, St. Petersburg, Russland, June 2004.

[8] G. M. Smith, *Statistical Process Control and Quality Improvement*, 5th ed., Upper Saddle River, NJ: Prentice Hall, July 2003.

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

[10] L. Breiman, J. H. Friedman, J. A. Olshen and C. J. Stone, *Classification and Regression Trees*, London: Chapman & Hall, Jan. 1984.

[11] G. P. Zhang, “Neural networks for classification: a survey,” *IEEE Transactions on Systems, Man and Cybernetics*, Part C, vol. 30, no. 4, pp. 451-462, Nov. 2000.

[12] L. Bruzzone and D. F. Prieto, “Unsupervised retraining of a maximum likelihood classifier for the analysis of multitemporal remote sensing images,” *IEEE Transactions on Geoscience and Remote Sensing*, vol. 39, no. 2, pp. 456-460, Feb. 2001.

[13] X. Chang and J. H. Lilly, “Evolutionary design of a fuzzy classifier from data,” *IEEE Transactions on Systems, Man and Cybernetics*, Part B, vol. 34, no. 2, pp. 1031-1044, April 2004.

[14] J. R. Quinlan, *C4.5: Programs for Machine Learning*, San Mateo, Calif.: Morgan Kaufmann, Jan. 1993.

[15] K. R. Müller, S. Mika, G. Rätsch, K. Tsuda and B. Schölkopf, “An introduction to kernel-based learning algorithms,” *IEEE Transactions on Neural Networks*, vol. 12, no. 2, pp. 181-202, March 2001.

[16] L. Breiman, “Random forests,” *Machine Learning*, vol. 45, no. 1, pp. 5-32, Oct. 2001.

[17] J. H. Friedman, “Stochastic gradient boosting,” *Computational Statistics & Data Analysis*, vol. 34, no. 4, pp. 367-378, Feb. 2002.

[18] A. Borisov, V. Eruhimov and E. Tuv, “Boosting flexible learning ensembles with dynamic feature selection”, *NIPS 2003 workshop on feature extraction*, British Columbia, CA, Dec. 2003.

[19] G. Grimmett and D. Stirzaker, *Probability and Random Processes*, 3rd ed., Oxford University Press, Oxford, May 2001.

[20] H. Ishibuchi and T. Nakashima, “Effect of rule weights in fuzzy rule-based classification systems,” *IEEE Transactions on Fuzzy Systems*, vol. 9, no. 4, pp. 506-515, Aug. 2001.

[21] J. R. Quinlan, (2003) *Data Mining Tools See5 and C5.0*, version 1.20. [Online]. Available: http://www.rulequest.com/see5-info.html

[22] S. Muriel, P. Garcia, O. Marie-Richard, M. Monleon and M. Recio, “Statistical bin analysis on wafer probe,” *2001 IEEE/SEMI Advanced Semiconductor Manufacturing Conference and Workshop,* Munich, Germany, pp. 187-192, April 2001.

[23] D. C. Montgomery, *Introduction to Statistical Quality Control,* 5th ed., New York: John

Wiley and Sons, July 2004.

[24] J. Andersson, *A Survey of Multiobjective Optimization in Engineering Design*, Technical Report No. LiTH-IKP-R-1097, Department of Mechanical Engineering, Linköpings University, Sweden, 2000.

[25] K. M. Miettinen, *Nonlinear Multiobjective Optimization*, Boston: Kluwer Academic Publishers, Oct. 1999.

[26] Y.C. Ho, “An explanation of ordinal optimization: Soft computing for hard problems,” *Information Sciences*, vol. 113, no. 3-4, pp. 169-192, Feb. 1999.

[27] T.W.E. Lau and Y.C. Ho, “Universal alignment probability and subset selection for ordinal optimization,” *Journal of Optimization Theory and Applications,* vol. 39, no. 3, pp. 455-489, June 1997.

[28] C.-H. Chen, S.D. Wu and L. Dai, “Ordinal comparison of heuristic algorithms using stochastic optimization,” *IEEE Transactions on Robotics and Automation,* vol. 15, no. 1, pp. 44-56, Nov. 1999.

[29] B.-W. Hsieh, C.-H. Chen and S.-C. Chang, “Scheduling semiconductor wafer fabrication by using ordinal optimization-based simulation,” *IEEE Transactions on Robotics and Automation,* vol. 17, no. 5, pp. 599-608, Oct. 2001.

[30] S.-Y. Lin and Y.C. Ho, “Universal alignment probability revisited,” *Journal of Optimization Theory and Applications,* vol. 113, no. 2, pp. 399-407, May 2002.

[31] S.-Y. Lin, Y.C. Ho and C.-H. Lin, “An ordinal optimization theory based algorithm for solving the optimal power flow problem with discrete control variables,” *IEEE Transactions on Power Systems,* vol. 19, no. 1, pp. 276-286, Feb. 2004.

[32] M. Bosque, *Understanding 99% of Artificial Neural Networks: Introduction & Tricks,* San Jose: Writers Club Press, March 2002.

[33] C. R. Reeves and J. E. Rowe, *Genetic Algorithms: Principles and Perspectives: A Guide to GA Theory*, Boston: Kluwer Academic Publishers, Dec. 2002.

[34] George A. F. Seber and C. J. Wild, *Nonlinear Regression*, New York: John Wiley and Sons, Sep. 2003.

[35] K. Hornik, M. Stinchcombe and H. White, “Multilayer feedforward networks are universal approximators,” *Neural Networks*, vol. 2, no. 5, pp. 359-366, 1989.

[36] T. Chen, H. Chen and R. W. Liu, “Approximation capability in C($R^{¯n}$) by multilayer feedforward networks and related problems,” *IEEE Transactions on Neural Networks*, vol. 6, no. 1, pp. 25-30, Jan. 1995.

[37] J. G. Attali and G. Pagès, “Approximation of functions by a multilayer perceptron: a new approach,” *Neural Networks*, vol. 10, no. 6, pp. 1069-1081, Aug. 1997.

[38] C. G. Panayiotou, C. G. Cassandras and W. B. Gong, “Model abstraction for discrete event systems using neural networks and sensitivity information**,**” *Proceedings of the 2000 Winter Simulation Conference,* Orlando, FL, USA, vol. 1, pp. 335-341, Dec. 2000.

[39] R. A. Kilmer, A. E. Smith and L. J. Shuman, “Computing confidence intervals for stochastic simulation using neural network metamodels,” *Computers and Industrial Engineering*, vol. 36, no. 2, pp. 391-407, April 1999.

[40] G. Lera and M. Pinzolas, “Neighborhood based Levenberg-Marquardt algorithm for neural network training,” *IEEE Transactions on Neural Networks*, vol. 13, no. 5, pp. 1200-1203, Sep. 2002.

[41] M. F. Moller, “A scaled conjugate gradient algorithm for fast supervised learning,” *Neural Networks*, vol. 6, no. 4, pp. 525-533, 1993.

[42] S. M. Sait and H. Youssef, *Iterative Computer Algorithms with Applications in Engineering: Solving Combinatorial Optimization Problems*, Los Alamitos: IEEE Computer Society, Aug. 1999.

[43] E. K. P. Chong and S. H. Żak, *An Introduction to Optimization*, 2nd ed., New York: John Wiley and Sons, July 2001.

# List of Publication

## 著作目錄

姓名：洪士程 (Shih-Cheng Horng)

**期刊論文著作：**

1. Shin-Yeu Lin and Shih-Cheng Horng, "A Classification Based Fault Detection and Isolation Scheme for the Ion Implanter", accepted to appear in IEEE Transactions on Semiconductor Manufacturing. (EI, SCI)
2. Shin-Yeu Lin and Shih-Cheng Horng, "Application of an Ordinal Optimization Algorithm to the Wafer Testing Process", accepted to appear in IEEE Transactions on Systems, Man and Cybernetics, Part A. (EI, SCI)

**研討會論文著作：**

1. Shin-Yeu Lin, Shih-Cheng Horng, "Ordinal Optimization Approach to Stochastic Simulation Optimization Problems and Applications", *Proceedings of the 15th IASTED International Conference on Applied Simulation and Modelling*, pp. 274-279, Rhodes, Greece, June 26~28, 2006.
2. Shih-Cheng Horng, Shin-Yeu Lin, "A Hybrid Classification Tree for Products of Complicated Machines in Flexible Manufacturing Systems", *Proceedings of IEEE SMC 2005 - International Conference on Systems, Man and Cybernetics*, pp. 3775-3780, Hawaii, USA, Oct. 10~12, 2005.
3. Shin-Yeu Lin, Shih-Cheng Horng, Chi-Hsing Tsai, "Fault Detection of the Ion Implanter Using Classification Approach", *Proceedings of the 5th Asian Control Conference*, pp. 808-813, Melbourne, Australia, July 20-23, 2004.
4. Chi-Hsing Tsai, Shin-Yeu Lin, Mu-Huo Cheng, Shih-Cheng Horng, Chun-Hung Liu, Wen-Yo Lee, Chia-Hung Tsai, "An Effective and Efficient Hierarchical Fuzzy Rule Based Classifier", *Proceedings of IEEE International Conference on Machine Learning and Cybernetics 2003*, vol.4, pp. 2173-2178, Xi-An, China, Nov. 2-5, 2003.
5. S. C. Horng, S. Y. Lin, M. H. Cheng, F. Y. Yang, C. H. Lin, W. H. Lee, C. H. Tsai, "Reducing the Overkills and Retests in Wafer Testing Process", *Proceedings of the 14th Annual IEEE/SEMI Advanced Semiconductor Manufacturing Conference and Workshop*, pp. 286-291, Munich, Germany, March 31- April 1, 2003.

# 博士候選人學經歷資料

姓名:洪士程

性別:男

生日:中華民國 60 年 4 月 11 日

籍貫:台灣省南投縣

論文題目: 中文:兩個關於晶圓製造及測試程序的產能與良率之問題及解決方法

英文: Two Related Issues on the Throughput and Yield of Wafer Fabrication and Testing Processes

學歷:

1. 國立交通大學電機與控制工程學系學士，民國 78 年 9 月~82 年 6 月
2. 國立交通大學電機與控制工程學系碩士，民國 82 年 9 月~84 年 6 月
3. 國立交通大學電機與控制工程學系博士班，民國 90 年 9 月~ 95 年 8 月

經歷:

1. 私立親民工商專科學校電子工程科專任講師，民國 86 年~93 年
2. 私立親民技術學院電子工程系專任講師，民國 93 年~95 年

# Vita

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, Hsinchu, Taiwan, in 1993 and 1995, respectively. He has been studying for his Ph. D. degree in the same department since September 2001.

Currently, he also works as a lecturer in the Department of Electronic Engineering, Chinmin Institute of Technology, Miaoli, Taiwan. His major research interests are optimization theory with applications to large semiconductor fab related problems, data mining, and modeling of large complex systems.