國 立 交 通 大 學

資訊科學研究所

博 士 論 文

知識系統中快速索引機制之研究

A Study of an Efficient Indexing Technology for Knowledge Systems

研 究 生：陳威州

指導教授：曾憲雄 博士

中華民國九十四年四月

知識系統中快速索引機制之研究

A Study of an Efficient Indexing Technology for Knowledge Systems

研　究　生：陳威州　　　　Student：Wei-Chou Chen

指導教授：曾憲雄　　　　Advisor：Dr. Shian-Shyong Tseng

國　立　交　通　大　學

資　訊　科　學　系

博　士　論　文

A Thesis

Submitted to Department of Computer and Information Science

College of Electrical Engineering and Computer Science

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy

in

Computer and Information Science

April 2004

Hsinchu, Taiwan, Republic of China

中華民國九十四年四月

博碩士論文授權書

（國科會科學技術資料中心版本 91.2.17）

本授權書所授權之論文為本人在＿＿＿國立交通大學＿＿＿大學(學院)＿資訊科學＿系所＿＿＿組＿九十三＿學年度第＿二＿學期取得＿博＿士學位之論文。

論文名稱：＿＿＿知識系統中快速索引機制之研究＿＿＿

■同意　　□不同意　（政府機關重製上網）

本人具有著作財產權之論文全文資料，授予行政院國家科學委員會科學技術資料中心、國家圖書館及本人畢業學校圖書館，得不限地域、時間與次數以微縮、光碟或數位化等各種方式重製後散布發行或上載網路。

本論文為本人向經濟部智慧財產局申請專利(未申請者本條款請不予理會)的附件之一，申請文號為：＿92135316＿，註明文號者請將全文資料延後半年再公開。

■同意　　□不同意　（圖書館影印）

本人具有著作財產權之論文全文資料，授予教育部指定送繳之圖書館及本人畢業學校圖書館，為學術研究之目的以各種方法重製，或為上述目的再授權他人以各種方法重製，不限地域與時間，惟每人以一份為限。

上述授權內容均無須訂立讓與及授權契約書。依本授權之發行權為非專屬性發行權利。依本授權所為之收錄、重製、發行及學術研發利用均為無償。上述同意與不同意之欄位若未鉤選，本人同意視同授權。

指導教授姓名:曾憲雄

研究生簽名:　　　　　　　　　　　　學號: 8823805

(親筆正楷)　　　　　　　　　　　　（務必填寫）

日期:民國 94 年 4 月 29 日

知識系統中快速索引機制之研究

學生：陳威州　　　　　　　　　　　　　　　　指導教授：曾憲雄 博士

國立交通大學電機資訊學院

資訊科學系

摘要

近年來，知識發現系統(System for Knowledge Discovery in Database)隨著資訊技術的進步與普及，愈來愈受重視，相關的應用技術及研究也相繼被提出，目的是希望使用資料庫知識發現(Knowledge Discovery from Database)的技術，將企業所累積的交易及製造的資料，透過資料探勘(Data Mining)的方法，找出企業知識(Business Intelligent)與各種行為模式(Behavior Patterns)，進而達到累積企業知識的目的。由於企業在營運的過程中所累積下來的資料量十分可觀，如何即時達成資料挖掘的功能並提出有效的知識則成為一個重要的課題。在此篇論文中，我們將提出一個適用於知識系統及資料庫之資料索引技術-位元組索引技術(Bit-wise Indexing Technology)。在這個技術中，我們總共提出了三個不同的索引方法，包含簡單位元組索引方法(Simple Bit-wise Indexing Method)、概括式位元組索引方法(Encapsulated Bit-wise Indexing Method)及精簡式位元組索引方法(Compacted Bit-wise Indexing Method)，可針對連續性及非連續性型態的資料進行處理，我們

亦也提出了二元化索引編碼及資料搜尋演算法，用以節省搜尋大量資料的處理時間。

為了驗證我們所提出技術的效率、彈性及實際可用性，我們將這個技術分別應用於四個不同的知識系統領域，包含回饋式學習(Reinforcement Learning)，模式學習(Pattern Learning)，監督式學習(Supervised Learning)及非監督式資料(Unsupervised Learning)挖掘知識系統等。而這四個實際系統包含應用在製造過程中由於製程時間的問題所產生的產品缺陷之以遺傳演算法之製造缺陷偵測系統、應用在網路入侵偵測系統中的入侵模式的挖掘與比對以提昇系統彈性及效率、應用在以資料為導向之約略集合論特徵選取技術並使用於知識擷取系統上以節省執行時間及應用在半導體製造過程中用於缺陷偵測的資料挖掘系統以提昇系統效能。其中用於半導體製造過程中用於缺陷偵測的資料挖掘系統已被台灣積體電路公司正式納入該公司之智慧型電子資料分析系統中的良率改善子系統，用以提高良率改善的效率，而以資料為導向之約略集合論特徵選取技術已被實際應用於某國際壽險的客戶關係管理系統專案中之擷取壽險保單回流貸款客戶特徵候選名單用以提昇企業收益。

關鍵詞：知識發現、位元組索引、資料挖掘、模式比對、特徵選取、知識萃取、知識分析

A Study of an Efficient Indexing Technology for Knowledge Systems

Student: Wei-Chou Chen　　　　Advisor: Dr. Shian-Shyong Tseng

Department of Computer and Information Science

National Chiao Tung University

Abstract

Recently, the Knowledge Discovery in Database (KDD) has grown rapidly, as IT and AI technologies have become widely discussed and researched. Relevant research, applications, and tool development in business, science, government, and academia are becoming increasingly popular. Particularly in some worldwide enterprises, KDD systems are applied to discover useful business intelligence and customer behavior patterns using data mining technology. However, since the quantity of data is continuously and rapidly growing in such enterprises, correctly and efficiently discovering useful information is becoming a significant issue. In this thesis, we will propose an efficient indexing technology of knowledge and database systems, called Bit-wise Indexing Technology. There are three indexing models in this technology, including Simple Bit-wise Indexing Method, Encapsulated Bit-wise Indexing Method and Compacted Bit-wise Indexing Method. Also, the corresponding indexing and matching algorithms for such indexing models are also proposed.

In order to demonstrate the suitability, flexibility and efficiency of the proposed indexing methods, we will try to apply the proposed method in four kinds of KDD applications, including reinforcement learning, pattern matching, supervised learning and unsupervised-learning data mining applications, in this thesis. For enhancing the system performance, the simple bit-wise indexing method was applied to the manufacturing defect detection problem, time aspect (*MDDP-t*) for manufacturing domains. For improving the flexibility and accuracy, the encapsulated bit-wise indexing method is applied to the pattern matching module of an Internet intrusion detection system. To reduce the processing time, the compacted bit-wise indexing method is applied to the data-driven rough-set based feature selection. Additionally, the proposed feature selection method was adopted in a KA project to discover the desired feature sets to construct a CBR system for a world-wide financial group customer relationship management system's loan promotion function. In the last application, three proposed methods are hybridly applied to the data mining module of a defect detection mechanism in a semiconductor manufacturing system to improving the accuracy and usability. The proposed method was officially employed in the Yield Explorer Function of Intelligent Engineering Data Analysis system (*iEDA*) in Taiwan Semiconductor Manufacturing Corporation (TSMC) for root cause detection of manufacturing defects and yield enhancement.

Keywords: Knowledge Discovery, Bit-wise Indexing, Data Mining, Pattern Match, Feature Selection, Knowledge Acquisition, Knowledge Analysis

誌　謝

曾幾何時，我從一個北商補校的學生，到今天取得博士學位，一晃眼也已經十幾個年頭了，回首來時路，喜怒哀樂、悲歡離合，雖然辛苦，但也甘之如飴；雖有磨難，卻也逆來順受。隨著年歲的增長，自己也更能體認人生的目的何在，而今天的畢業，不是一個終點，而是另一個蛻變後的起點。

從高雄工學院資管系、義守大學資工所，到交通大學資科博士班，何其有幸讓我能接觸到資訊學門高等教育的三個主要科系，也因為如此，更能體會到資訊教育的理論面、工程面及應用面其實是環環相扣、密不可分的。而啓發我將三個面向的想法，融合一體並帶入產業應用的關鍵人物，就是我的恩師　曾憲雄博士，他不僅在學術領域上給予我無私的指導與啟發，在應用領域上亦給予我很大的空間與舞台，而最最重要的是，在待人處事及應對進退的做人道理上，給了我很多的指導及鼓勵，特別是在博士修業期間，當我面臨人生中最低潮的那二年，恩師對我所展現的寬容與支持，心中的感謝絕非筆墨所能形容的。所以恩師所給予我的指導，絕非僅是這本博士論文而已，而我相信，在未來的道路上，不論是在學界或是業界，定會深深的影響著我的。

而在我的博士論文口試期間，非常感謝　孫春在教授、　袁賢銘教授及　楊維邦教授能不厭其煩在學生博士論文計畫、校內及校外口試中，從不同的角度及觀點，給予指導及提醒；而在校外口試中，亦要感謝針對目前相關研究及產業應用面上，給予許多包含博士論文本身及未來研究發展方向建議的清華大學　陳良弼教授、成功大學　郭耀煌教授、台灣大學　陳銘憲教授及高雄大學　洪宗貝教授。由於他們的指導，才能使這篇論文更臻完整。

從進入交大到現在已經快六個年頭了，知識工程實驗室一直是我與其他研究夥伴們共同生活的地方，從資格考試前挑燈夜戰、論文投稿期限前的秉燈夜書到論文校外口試前的最後準備，一幕幕的記憶晃如昨日般清晰。今天我的畢業，不代表結束，後續的研究仍需仰賴實驗室夥伴們持續不斷的互相扶持及共同努力，加油！

相較於其他博士班學生而言，我的學習歷程有著很大的不同，而我的家人們，所給我的支持與鼓勵，無疑是最大的原動力，而其中最要感謝的就是我的父母親，對於我的母親而言，從來沒有希望自己的兒子能念到多高的學歷，而是一直注重

我的品德教育，在她嚴厲管教的背後，總是不吝給我慈母的關懷，如果沒有她無私無我的關懷與支持，這本博士論文是不可能完稿付梓的，謝謝您，我的母親。

今天我取得博士學位，最高興的應該是我的爸爸了吧，我一生中最大的遺憾，就是未能在父親辭世前，親眼看到我拿到博士學位的樣子，求學的路，我一直走的很辛苦，在高職階段，曾想放棄自己，是父親苦口婆心的勸我回頭，我永遠無法忘記父親在花蓮車站前，等我回家的身影。雖然因為父親的病，而使得博士修業的時程有所延遲，但就我而言，沒有比陪伴父親更重要的事呀。雖然父親無法看到我現在的成就，但是我相信，他仍在我的身旁，無時無刻的陪伴著我、鼓勵著我。

至於我的太太　姿婷，自從嫁給我後，一直克盡妻職，將我的生活起居及一家大小照顧的妥妥當當，讓我能無後顧之憂的努力於學業及事業之上，今天拿到學位的小小榮耀，當然也要跟老婆大人一起分享。

要感謝的人很多，無法一一詳述，對於所有曾經幫助及支持過我的人，在此致上最誠摯的謝意。

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1
Introduction

1.1 Motivation

Recently, the fields of Knowledge System and Data Mining have rapidly grown years, since IT and AI technologies have become widely discussed and researched. Related research, applications, and tool development in business, science, government, and academia are becoming increasingly popular. Especially in some enterprises, KDD systems are applied to discover useful business intelligence and customer behavior patterns via some machine learning and data mining technologies. However, since the amount of data is rapidly increasing in such enterprises, efficiently discovering the useful knowledge becomes a significant issue. In database-related fields, indexing is adopted to provide a global distribution and storage/location information for efficiently retrieving the individual item (record) within a huge dataset (table). This approach can clearly help users to quickly search a dedicated record (set) in a database for the given query conditions, but may not be appropriate for retrieving huge numbers of records, such as OLAP queries in data warehousing and knowledge system analysis requirements, owing to the indexing characteristics. Therefore, the bitmap indexing method [54][74] became popular in data warehousing to obtain the efficient OLAP query requirements. In some previous cases [11][13], the bitmap indexing method has been applied to a case-based knowledge system to accelerate similar-case retrieval and similarity-based computing procedures, but it is not suitable in this domain due to the lack of similarity retrieving ability of such method. The major issue in constructing an effective knowledge system is to propose a flexible and efficient knowledge learning procedure, which can transform the information in the given data set into a well-defined knowledge structure in the knowledge base. Obviously, the data management abilities, including data structure, indexing, processing and manipulation abilities, become very important for the underlying data repositories. Generally, the indexing mechanism for a knowledge system, particularly in the learning procedure, should provide an encoding and representation method to compare and analyze the individuals (records) of data set efficiently. Additionally, the knowledge base indexing method should not only concentrate on the efficient matching query ability, but also provide the similarity analyzing and calculating abilities. Due to immediacy and performance issues, choosing an appropriate data indexing method is an important issue, particularly with large amounts of data. The data representation not only influences the performance of the knowledge system, but also affects the efficiency and accuracy of the underlying knowledge base. In this thesis, we will propose an efficient (using all bit operations), extensive (accepting both symbolic and continuous data formats) and flexible (with similar retrieving ability) indexing technology, called Bit-wise Indexing Technology. Three indexing methods are proposed, the Simple Bit-wise Indexing Method and two advanced indexing methods, including the Encapsulated and Compacted Bit-wise Indexing Methods. Additionally, the corresponding indexing, matching algorithms for such indexing models are also proposed. The proposed bit-wise indexing methods not only accelerate the analyzing performance of knowledge system, but also can be applied in a traditional database system for efficiently similarity-based retrieving. In order to demonstrate the suitability, flexibility and efficiency of the proposed indexing methods, we will try to apply the proposed method in four knowledge system applications, including reinforcement learning, pattern matching, supervised learning and unsupervised-learning data mining applications.

At first, we will propose a novel, efficient and parallelized indexing method, called the *Simple Bit-wise Indexing Method*, to reduce the data processing and query overhead of some knowledge discovery systems. Bit-wise indexing is similar to bitmap indexing method except that the matrix of bit-wise or bitmap indices, which is generated from the related table of the data resource of KDD systems via bit-wise indexes creation algorithm, is partitioned horizontally. Additionally, bit-wise indexing has more powerful similarity retrieving and parallelization capabilities than bitmap indexing. Since the bit length of each attribute in the simple bit-wise indexing or bitmap indexing method depends on the number of distinct attribute values of it, the problem of long bit-wise strings arises when the number of distinct values is huge. For instance), in a huge data warehouse, there may exist millions customer records, meaning that some attributes may have million of distinct values. When these attributes are encoded into the bit strings of the bitmap or Simple Bit-wise Indexing Method, one million bits per record are required, of which only one is set to 1. Although many compression technologies have been proposed for such a problem, they still require additional computational time. Therefore, this study presents two advanced bit-wise indexing methods, including *Encapsulation Bit-wise Indexing* and *Compacted Bit-wise Indexing methods*. *Encapsulation Bit-wise Indexing Method*, is used to solve the problem of long bit-wise lengths. *Encapsulation Bit-wise Indexing Method* partitions the longer bit strings into at least two levels to preserve disk space and memory. Additionally, the computation time of OLAP queries is reduced since the bit length of each record is decreased. For example, the dual-level encapsulation bit-wise indexing method decreases bit length of such attributes from one million to 2,000 bits.

Applying the Encapsulation Bit-wise Indexing Method to the data resource of KDD system can significantly reduce the indexing storage and improve query performance. However, the total number of bit-wise index string that needs to be compared via AND bit-wise operation is still reminded. In order to accelerate the processing time of OLAP queries, another indexing model, called *Compacted Bit-wise Indexing Method*, is proposed. As we know, the attribute is the basic information of all data queries. Additionally, the concept hierarchy of each attribute is an important issue of roll-up and drill-down operations of the data warehouse. In the *Compacted Bit-wise Indexing Method*, the significance of attributes, including attribute weight and concept hierarchy, need to be evaluated via some statistical methods. Compacted Bit-wise Indexing Method compacts the bit strings of higher ranking attributes by at least two levels, including high-level concept hierarchy and the others, the high-level concept attributes is kept in the first level bit-wise indexes matrix, while the others are stored in the second levels. Furthermore, the encapsulated bit-wise indexing method can also be applied to further reduce the bit length. The processing time of queries can be hugely reduced since the total bit length can be largely reduced (via encapsulated bit-wise indexing method) and the irrelevant records can be filtered out (via the higher level concept hierarchy of compacted bit-wise indexing method).

The proposed bit-wise indexing methods are suitable for helping many knowledge discovery systems in order to accelerate the processing performance. In the thesis, the proposed methods is applied in four knowledge system applications, including reinforcement learning, pattern matching, supervised learning and unsupervised-learning data mining applications, to demonstrate the suitability, flexibility and efficiency. The first application consisted of a reinforcement-learning defect detection learning system for the time aspect in manufacturing domains. This implementation employed the Sample Bit-Wise Indexing Method to encode the defect status of manufacturing products and hence accelerate data preprocessing. Additionally, a bit-based Genetic Algorithm is used to learn suitable weights for each computed signature, since the chromosome and the corresponding GA operators are appropriate for the bit operations of BWI indexing method. First, the manufacturing defect detection problem, time aspect, for (*MDDP-t*) is formally modeled and defined. A root-cause evaluation function (*RCEF*), which is a linear combination of three probing functions defined independently according to the experiences of domain experts, is proposed to evaluate whether a specific machine is the root cause of a time problem. The probing function weights are determined separately. Additionally, this study presents a genetic algorithm (GA) with encoding and GA operations appropriate for *MDDP-t* weight-learning problems to obtain suitable weights for the probing functions.

The training examples include *MDDP-t* instances with known root causes provided by the Taiwan Semiconductor Manufacturing Company [TSMC]). Experimental results show that the proposed method can ensure efficiency and accuracy.

The second application introduces a pattern-learning network intrusion detection system. This implementation uses the Encapsulated Bit-wise Indexing Method to encode the networking activity with minimal monitoring time window in order to accelerate the data preparation procedure. Moreover, a bit-based intrusion Pattern Matching mechanism is proposed to efficiently learn, roll-up, drill-down and combine the intrusion pattern with different time-windows/services/ports combinations. In general, the user's pattern can be transformed into a sequence of network activities that are extracted from the related network packets. These kinds of network packets can be collected and then be transformed into some sequence of bit-wise strings showing the intrusion patterns. The Network Activities Analyzing Phase can first filter out the raw network packets and log necessary features (Source IP, Destination IP, Source port, Destination port) in a small time window to perform data sampling and data cleaning and to reduce the amount of data. After that, with combined users and services information, the sufficient service-user activity events are found and used by the second phase. The Features/Pattern Mining Phase transforms the sufficient service-user activity events to some bit-wise strings and next merges the bit-wise strings into some other bit-wise strings with the same source IP. After gathering those bit-wise strings, the Pattern Mining Module and Pattern Merging Module can perform some data mining processes to find possible intrusion patterns that can be the source of the candidates of intrusion patterns for future intrusion detection systems. Finally, the pattern with bit-wise indexing representation can be easily transformed into a corresponding Finite State Machine for efficient real-time tracing and monitoring of networking activities.

The third application is a supervised-learning data-driven feature selection method for CBR systems. As we know, the critical issue in case-based reasoning is to select the correct and enough features to represent a case. However, this task is difficult to carry out since such knowledge is often exhaustively captured and cannot be represented successfully. A new, efficient feature selection method is proposed here. The bit-wise-based feature selection method is proposed for discovering the optimal feature sets for decision–making problems. And the corresponding indexing and selecting algorithms for proposed feature selection method are also proposed. This implementation applies the Feature Selection Method using Rough Set Theory, which is appropriate for finding the optima solution from a given data set, except for the long processing time issue. Therefore, the Compact Bit-wise Indexing Method is used to encode the feature and class relationships to reduce the processing time of feature selection procedure. Finally, some experiments and comparisons are given and the result shows the efficiency and accuracy of our proposed methods.

The last application combines the bit-wise indexing methods (including Sample, Encapsulated and Compact Bit-wise Indexing Methods), Data Mining Technologies, and Statistic Methods to construct an unsupervised-learning data-driven data mining system for an engineering data analysis (EDA) a production-level defect detection system. With large quantities of semiconductor engineering data stored in databases and versatile analytical charting and reporting in production and development, IT systems in most semiconductor manufacturing companies permit users to access and analyze data quickly and conveniently. Making the semiconductor process more sophisticated means that more data must be analyzed and troubleshooting, especially in yield enhancement, becomes more difficult,. Currently, information summarized from these systems is too detailed to be easily assimilated by engineers. Engineers need to daily review thousands of charts and statistical results to undertake trouble shooting jobs. Using simple statistics, these charts and statistics are listed by these IT systems in an order of priority for review. Engineers frequently catch the real root cause of a problem only after reviewing many charts and statistical results. Those simple statistics do not show the complicated intersectional effect resulting from nonlinear interaction among many factors reliably and quickly. This application, describes the experiences that applied such hybrid data mining solutions for low-yield root cause detection situation in the Taiwan Semiconductor Manufacturing Company Ltd. (TSMC). Typically, the data mining solutions have high time and space complexities, but failure to discover the low-yield situation quickly causes significant damage. In this application, the BWI indexing method was applied to the data mining application to accelerate the processing time. As expected, the BWI-indexing-based data mining solution saved over 90% of processing time compared with conventional data mining solutions. The accuracy and performance evaluations for 42 real cases from TSMC are made and reviewed herein. According to the evaluation results, the data mining engine using bit-wise indexing uses only 10% of processing time rather than the in-memory process without the BWI indexing method. Additionally, some critical issues about using a data mining solution to detect semiconductor manufacturing defects are discussed and reviewed herein. Finally, the system framework of the next-generation data mining solution in the future is proposed to provide a knowledgeable, reasonable, reliable and flexible data mining solution in semiconductor manufacturing.

1.2 Contributions

1. Three indexing models, called *Simple bit-wise indexing method*, *Encapsulation bit-wise indexing method* and *Compacted bit-wise indexing method*, are proposed, along with indexing and matching algorithms corresponding to each proposed indexing model.
2. A manufacturing defect detection system for the time aspect problem using the Sample Bit-wise Indexing Method and a Genetic Algorithm are proposed to improve the encoding and computing performance.
3. A network user behavior pattern matching module using Encapsulated Bit-wise Indexing Method of an Internet intrusion detection system is proposed to enhance the usability and flexibility of IDS systems.
4. A data-driven feature selection method using Compact Bit-wise Indexing Method and Rough Set Theory for the CBR system is applied to improve the performance of the feature selection procedure.
5. A data mining module of a defect detection mechanism in a semiconductor manufacturing system using hybrid bit-wise indexing methods is proposed to improve the performance of the data mining and defect detection procedure. Additionally, the system framework of the next-generation data mining solution is also given.

1.3 Reader's Guide

The remaining parts of this thesis are organized as follows. The reviews of the relative works are given in Chapter 2. The *Simple Bit-wise Indexing Method* is proposed in Chapter 3. The advanced indexing methods, including *Encapsulation* and *Compacted Bit-wise Indexing Method* are introduced in Chapter 4. An Intelligent Manufacturing Defect Detection Method for the time issue using Sample BWI indexing method is given in Chapter 5 and a network user pattern matching method of an Internet intrusion detection system using Encapsulated BWI method is discussed in Chapter 6. In Chapter 7, a data-driven feature selection method using Compact BWI indexing method and Rough Set Theory of CBR system is proposed and a data mining module using hybrid BWI indexing methods for low-yield defect detection in a semiconductor manufacturing system is briefly reviewed in Chapter 8. The conclusion and future works are finally given in Chapter 9.

Chapter 2
Related Review

2.1 Data Warehousing

The concept of data warehousing was first proposed by Inmon in 1993. A data warehouse contains information collected from individual data source and integrated into a common repository for efficient querying and analysis. When the data sources are distributed over several locations, a data warehouse is responsible for collecting the necessary data and saving it in appropriate forms. The architecture of a typical data-warehousing system is shown in Figure 2.1.

There are three major components in it: the *data collector*, the *data warehouse*, and the *OLAP and query processor*. The data collector is responsible for collecting necessary information and transaction messages from individual data source through communication networks to meet the requirements of end users and the views defined in the data warehouse. The *data warehouse* receives data from the data collector, filters them, and stores them in its own database. The *OLAP* and *query processor* provide all necessary information for user queries and OLAP requirements. The *data collector* or

OLAP and *query processors* may also be divided into several subparts, each located near a data source.

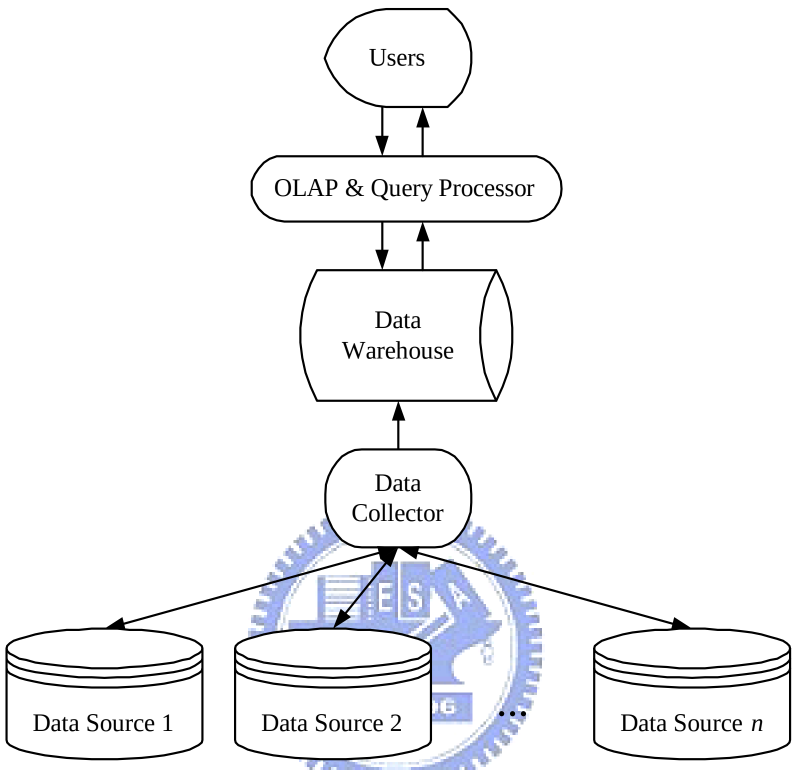


Figure 2.1: Architecture of a typical data warehousing system

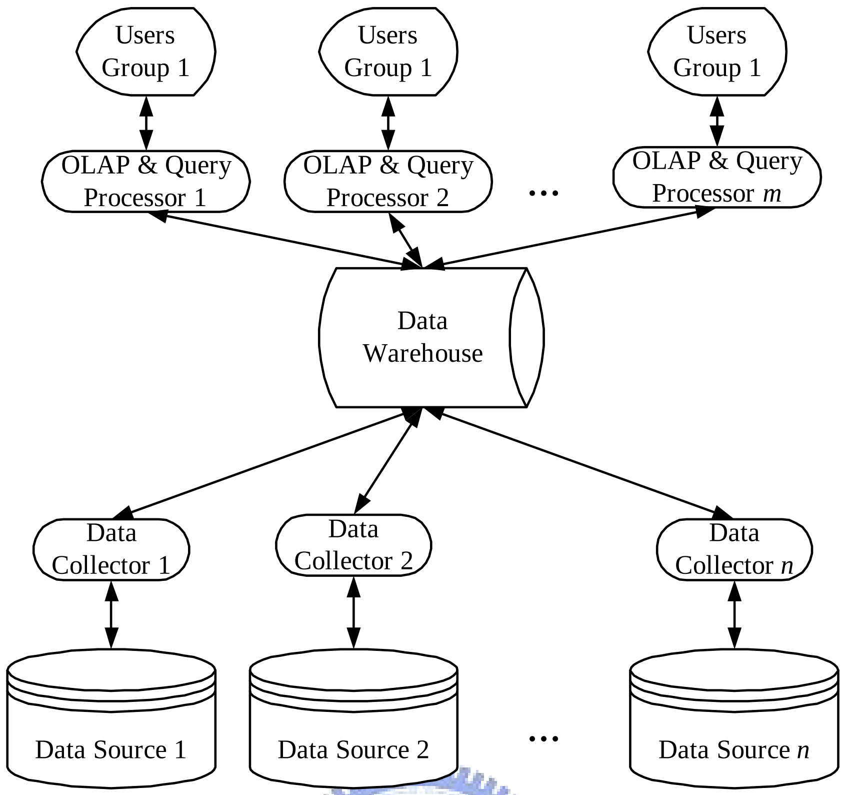


Figure 2.2. Architecture of a data warehousing system with distributed components

A data warehouse usually contains a large number of views in order to speed up query processing and avoid large amounts of network transmission. Views can be defined by query languages and provide particular formats of query results to users. Data warehousing systems use two kinds of views: *materialized views* and *virtual views*. A materialized view retrieves all necessary information from data sources according to the view definition and physically stores the extracted data in a data warehouse. A virtual view retrieves the information from other materialized views using the query language whenever the view contents are required. Each kind of view has its advantages and disadvantages. One of the primary goals of a data warehousing system is to support on-line analytical processing, call OLAP, and help in decision making. For this reason, data warehouses must maintain appropriate views to ensure that OLAPs are efficient.

Data warehouses are often built to support on-line analytical processing. On the other hand, the OLAP is a technology and the DW is an architectural infrastructure. Typical OLAP operation includes rolling-up (increasing the level of aggregation) and drilling-down (decreasing the level of aggregation or increasing detail). The star schema is the most popular data model of data warehousing. The *manufacturing* star schema example of a data warehouse is shown in Figure 2.3. In this figure, there are four dimension tables, including Tool, Product, Recipe and Time tables, and an Ordering fact table. The relationships between fact table and those dimension tables are kept thru relation keys.

Since OLAP queries of data warehouse are usually complex, the performance of OLAP queries is a critical issue in the data warehouse. Therefore, the indexing technology is often embedded in the data warehouse environment [54][74][75].

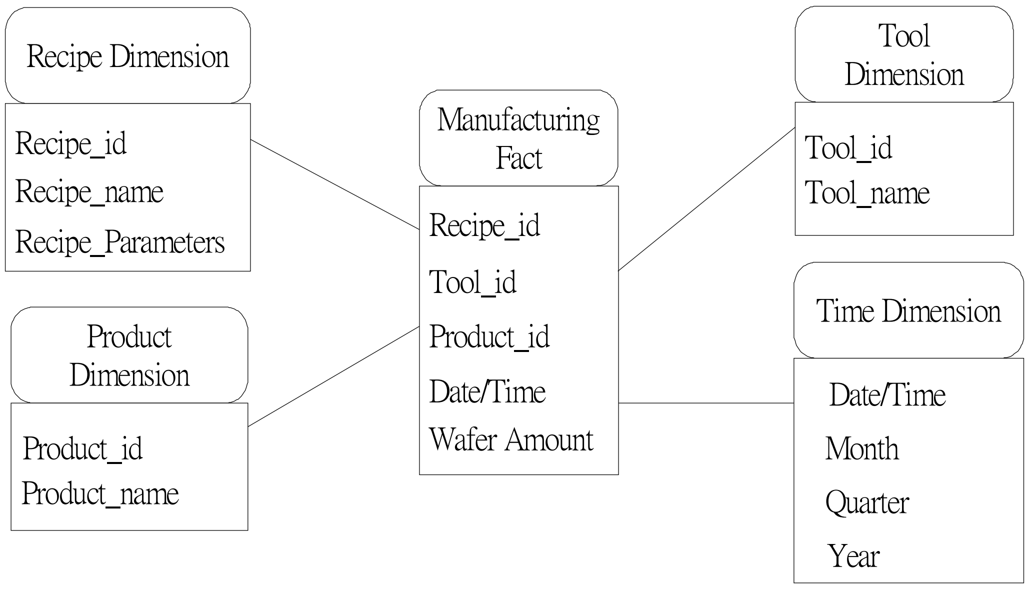


Figure 2.3: An example of a *manufacturing* star schema in a data warehouse.

2.2 Bitmap Indexing methods of Data Warehousing

As mentioned above, the query processing is the critical issue in the data warehouse environment. In recent years, many indexing technologies, such as B-tree, k-d tree, R-tree, Value List and Bitmap indexing methods [54][74][75], have been proposed in data warehouse system. The Bitmap is the most popular indexing method in OLAP system since it was designed to search and analyze the data for the OLAP queries efficiently. The basic idea of Bitmap indexing method is using a string of bits which is called bitmap vector and formed by 1 or 0 to indicate whether the some attributes are equal to a specific value or not [75]. A bit in the bit string maps the position of a record in the table. If the content of the attribute is associated with a specific value, the bit is set as “1”. The Bitmap indexing method is illustrated in Figure 2.4

In Figure 2.4(a), there are three attributes in the table, including *Tool_id*, *Name* and *Location*. The attribute values domain of *Tool_id*, *Name* and *Location* are {3210, 2688, 6150, 6210, 8850}, {AWOX01, AWOX02, AWOX03, AWOX04, AWOX05} and {FAB 1, FAB 2, FAB 3}, respectively. It can be easily seen that the number of distinct values of *Tool_id*, *Name* and *Location* are 5, 5 and 3, respectively. Therefore, thirteen bitmap indexing vectors are generated as shown in Figure 2.4(b). Assume that a query with conditions (*Name* = AWOX02 or *Location* = FAB 3) is required to execute, the bitmap indexing vectors B_{AWOX02} and $B_{FAB\ 3}$ are operated with operation OR and then the result is { 0, 1, 0, 0, 1 }. Therefore, the records 2 and 4 are formed as the result set of the query. In addition to the simple bitmap indexing method described above, there are still some extension can be found in [74][75]. However, it seems that Bitmap indexing method is more efficient than other indexing methods since the method had been widely used in the commercial products of DWs.

	Tool id	Name	Location
1	3210	AWOX01	FAB 1
2	3688	AWOX02	FAB 1
3	6150	AWOX03	FAB 2
4	6210	AWOX04	FAB 2
5	8850	AWOX05	FAB 3

(a) *Tool* dimension table

B_{3210}	B_{3688}	B_{6150}	B_{6210}	B_{8850}	B_{AWOX01}	B_{AWOX02}	B_{AWOX03}	B_{AWOX04}	B_{AWOX05}	$B_{FAB\ 1}$	$B_{FAB\ 2}$	$B_{FAB\ 3}$
1	0	0	0	0	1	0	0	0	0	1	0	0
0	1	0	0	0	0	1	0	0	0	1	0	0
0	0	1	0	0	0	0	1	0	0	0	1	0
0	0	0	1	0	0	0	0	1	0	0	1	0
0	0	0	0	1	0	0	0	0	1	0	0	1

(b) Bitmap indexes for (a)

Figure 2.4: An example of Bitmap indexes

Since the bitmap indexing method seems to be able to be directly applied to indexing and retrieval phrases in the data warehousing. However, there are still some problems should be solved:

1) When the number of records in the data warehouse is large, the bits in the bitmap indexing vectors will be extended hugely. Also, the number of bitmap indexing vectors is dependent on the summary of distinct value for attributes. If the number of distinct values for some attributes is large, the number of bitmap indexing vectors is also large. Although many solutions are proposed to solve these problems, the extra cost of computing also needs to be spent.

2) In the data warehousing, the ability of similarity retrieving may need to be considered. Some extra computation of the similarity between the records is required.

In other words, it is not quite suitable to straightly apply the bitmap indexing method to the data warehousing directly. It needs some adaptation. We will discuss the details of our new indexing technology in following two chapters.

2.3 Feature Selection and Rough Set

Feature selection is about finding useful (relevant) features to describe an application domain [7][11][14][15][19][24][39][42][47][48][79]. The problem of feature selection can formally be defined as selecting minimum features *M'* from original *M* features where $M' \leqq M$ such that the class distribution of *M'* features is as similar as possible to *M* features. Generally speaking, the function of feature selection is divided into three parts: (1) simplifying data description, (2) reducing the task of data collection, and (3) improving the quality of problem solving. The benefits of having a simple representation are abundant such as easier understanding of problems, and better and faster decision making. In the case of data collection, having less features means that less data should be collected. As we know, collecting data is never an easy job in many applications because it could be time-consuming and costly. Regarding the quality of problem solving, the more complex the problem is if it has more features to be processed. It can be improved by filtering out the irrelevant features which may confuse the original problem, and it will win the better performance. There are many discussions about feature selection, and many existing methods to assist it, such as GA technology [60], entropy measure[31], and rough set theory [78].

Next, the rough set theory is briefly reviewed. The rough set theory, proposed by Pawlak in 1982 [55], can serve as a new mathematical tool for dealing with data classification problems [36][56][76][77][78][79]. It adopts the concept of equivalence classes to partition training instances according to some criteria. Two kinds of partitions are formed in the mining process: lower approximations and upper approximations. Rough sets can also be used for feature reduction. The features that do not contribute to the classification of the given training data are removed. The concepts of equivalence classes and approximations are quite suitable to generate the bit-based class vectors and record vectors, which can then be directly and efficiently transformed to the bit-wise indexing matrixes in CBR system. This work thus adopts these concepts to solve the feature selection problem.

Chapter 3
Simple Bit-wise Indexing Method

In this chapter, the Simple bit-wise indexing methods will be introduced. At first, the general assumptions and notations for BWI Technology will be given. After that, the definitions and algorithms of Simple bit-wise indexing method are proposed.

3.1 General Assumptions and Notations for Simple BWI Method

In the section, the basic assumptions and nations are illustrated in detail. As mentioned above, the bit length of each attribute in the bit-wise indexing or bitmap indexing method depends on the number of its distinct values. This implies that the problem of long bit-wise string arises when the number of distinct values is large. Although there are many compression technologies had been proposed to solve such problem, the extra computational time is usually needed. Therefore, the *condensable bit-wise indexing method* is proposed to solve the long bit-wise length problem. Using this method, only the attributes with longer bit lengths are partitioned into two or more levels for saving the storage of disk and memory both. Also, computation time of

OLAP queries is also reduced since the bit length of each record is shortened.

In order to answer the user’s query statements, we search records in the target table of data warehousing. In the beginning, we transform the data schema of data warehousing to a single target table, called flat target table.

Since the data store in data warehouse is updated periodically, maybe a day, a week, or a longer period, the indexing phase will be executed in initialization and maintenance stages of data warehousing. The querying phase is called during the running time of queries for the current users.

Without loss of generality, we assume that the data schema of the warehouse consists *n* fact table and *m* dimension tables. Definition 3.1 defines a flat target table that was transformed from the data schema of data warehouse.

DEFINITION 3.1 : Flat target table

The flat target table *T* is created by joining all non-redundancy fields of the fact tables and all dimension tables via some SQL statements

EXAMPLE 3.1：

The example of data schema is shown in Figure 2.3. There are one fact table, *Manufacturing* fact and four dimension tables, including *Tool*, *Product*, *Recipe*, and *Time* dimension tables of a manufacturing company. The attribute set of *Tool*, *Product*, *Recipe*, and *Time* dimensions are {*Tool_id* ,*Tool_name*}, {*Product_id*, *Product_name*}, {*Recipe_id*, *Recipe_name*, *Recipe_parameters*}, and {*Date/Time*, *Month*, *Quarter*, *Year*}, respectively. The attribute set of fact table *f* is {*Tool_id*, *Product_id*, *Recipe_id*, *Date/Time*, *Wafer amount*}. Moreover, the referential relationships between fact table and the dimensions are {*Tool.Tool_id=Manufacturing.Tool_id*}, {*Product.Product_id=Manufacturing.Product_id*},

R_3={*Recipe.Recipe_id=Manufacturing.Recipe_id*}, and

R_4={*Time.Date/Time=Manufacturing.Date/Time*}. Therefore, the SQL statement of flat target table *T* can be generate as follows:

```
Select  Tool_id, Tool_name, Product_id, Product_name, Recipe_id, Recipe_name,
        Suppiler.category, Year, Quarter, Month, Date/Time, Wafer amount
Into    TargetTable
From    Manufacturing, Tool, Product, Recipe, Time
Where   Tool.Tool_id = Manufacturing.Tool_id
        and Product.Product_id = Manufacturing.Product_id
        and Recipe.Recipe_id = Manufacturing.Recipe_id
```

and Time.Date/Time = Manufacturing.Date/Time.

After above SQL statement is executed, the flat target table *TargetTable* is thus generated and the structure of this table is shown in Table 3.1.

Table 3.1: An example of a flat target table *T* in a data warehouse that transforms from a data schema

Tool_id	Tool_name	Product_id	Product_name	Suppiler_id	Recipe_name
Recipe_parameters	Time/Date	Month	Quarter	Year	Wafer Amount

After the flat target table is generated, our indexing technologies will focus on this target table in the following sections.

3.2 The Indexing Phase of Simple BWI Method

Assume a set of records *R* is stored in a table *T* for a specific domain, denoted *DOM*. The *i*-th record in *R* is represented by R_i. Also assume all the records in *R* can be abstracted by a set of attributes *A*, denoted $A = <A_1, A_2, \ldots, A_r>$, where *r* is the number of attributes. The value of an attribute A_k for a record R_j is denoted $V_k(j)$, which can not be null. The attribute values of a record R_j can then be represented as $V(j) = <V_1(j), V_2(j), \ldots, V_r(j)>$. The set of possible values for attribute A_i, called *attribute value domain*, is denoted $V_i = <V_{i1}, V_{i2}, \ldots, V_{i\alpha(i)}>$, where $\alpha(i)$ is the number of values for A_i, and V_{ij} is the j-th possible attribute value of A_i.

In a data warehousing system, a set of records is stored in the warehouse for serving a new coming query. A matching function is used to evaluate records based on a weighted sum of matched attributes with a new coming query condition. Attribute value can thus be used for indexing a record. An index of a record using *Simple Bit-wise Indexing Method* can be formally defined as follows.

DEFINITION 3.2 - Record Index :

The index IND_k of a record R_k in a table T for domain DOM is defined as:

$IND_k = \{A_1 = V_1(k), A_2 = V_2(k), \ldots, A_r = V_r(k)\}$.

A record in table T can be formally defined as follows.

DEFINITION 3.3 – Record :

A record R_k in a table T for domain DOM is a pair (IND_k, rv_k), where rv_k is the actual contents of record R_k and $R_k \in R$.

In the most indexing methods of data warehousing, the numeric-type data are usually treated as the computational attributes and thus will not be included in the indexes. However, in some real applications, the numeric attributes also need to be indexed for further investigation. For example, in the data warehouse of manufacturing domain, the numeric recipes are the important factors for processing control and defect detection. The same situation will happen in the data/time-type attributes. The basic operations and notations of future definitions are shown as follows:

OPERATION 3. 1 - Type, Year, Month, Day, Hour, Minute and Second Operations :

$$\text{Type}(A_i)=\begin{cases} A_i \text{ is numeric-type,} & N \\ A_i \text{ is date/time-type,} & D \\ \text{Otherwise,} & S \end{cases}$$

Year(V_i)=The number of year in x for Type(A_i)=D; otherwise, return *Null*.

Month(V_i)=The number of month in x for Type(A_i)=D; otherwise, return *Null*.

Day(V_i)=The number of day in x for Type(A_i)=D; otherwise, return *Null*.

Hour(V_i)=The number of hour in x for Type(A_i)=D; otherwise, return *Null*.

Minute(V_i)=The number of minute in x for Type(A_i)=D; otherwise, return *Null*.

Second(V_i)=The number of second in x for Type(A_i)=D; otherwise, return *Null*.

OPERATION 3.2 - Minima Element (MNE) Operation :

$$\text{MNE}(A_i)=\begin{cases}\text{If Type}(A_i)=N, & \text{the smallest number in } A_i\\ \text{If Type}(A_i)=D, & \text{the earliest date/time in } A_i\\ \quad\text{Otherwise,} & \qquad\qquad Null\end{cases}$$

OPERATION 3.3 : Maxima Element (MXE) Operations :

$$\text{MXE}(A_i)=\begin{cases}\text{If Type}(A_i)=N, & \text{the largest number in } A_i\\ \text{If Type}(A_i)=D, & \text{the latest date/time in } A_i\\ \quad\text{Otherwise,} & \qquad\qquad Null\end{cases}$$

A bit-wise indexing vector used in the proposed indexing method is defined as follows.

DEFINITION 3.4 : Bit-wise indexing vector of an attribute where Type(A_i)=S :

The bit-wise indexing vector B_i of the i-th attribute for record R_k is a bit string $B_i=b_{i1}b_{i2}\ldots b_{i\alpha(i)}$, where b_{ij}=1 if $V_i(k)=V_{ij}$ and b_{ij}=0 otherwise.

EXAMPLE 3.2 :

Assume that the domain of attribute *Name* is <AWOX01, AWOX02, AWOX03, AWOX04, AWOX05> and the attribute value of *Name* in the second record is AWOX01. According to the Definition 3.4, bit-wise indexing method uses the 5 bits as the bit vector of the index in which every bit represents a specific value of the index attribute *Name*.

B_2:

AWOX01	AWOX02	AWOX03	AWOX04	AWOX05
b_{11}	b_{12}	b_{13}	b_{14}	b_{15}

Therefore, we get $B_2 = b_{21}b_{22}b_{23}b_{24}b_{25}$="**10000**"

DEFINITION 3.5 - Bit-wise indexing vector of an attribute where Type(A_i)≠S :

The bit-wise indexing vector B_i of the i-th attribute for record R_k is a bit string $B_i=b_{i1}b_{i2}\ldots b_{if_i(MXE(A_i))}$, where b_{ij}=1 if $f_i(V_i(k))=j$ and b_{ij}=0 otherwise, where the function f_i is given via user for clustering the numeric attribute A_i and $f_i(MXE(A_i)) \le \alpha(i)$.

EXAMPLE 3.3 :

Assume that the second attribute *Recipe_degree* is <10, 12, 14, 16, 18, 20, 22> and the attribute value of *Recipe_degree* in the second record is 16. Also, the given function f_2 is given in following.

$$F_2(V_i(k))=\begin{cases} V_i(k) < 12 & =1 \\ 12 \le V_i(k) < 16 & =2 \\ 16 \le V_i(k) < 18 & =3 \\ V_i(k) \le 18 & =4 \end{cases}$$

According to the Definition 3.5, bit-wise indexing method uses the 4 bits as the bit vector of the index in which every bit represents a specific value of the index attribute *Recipe_degree*.

B_2:

$f_2(V_i(k))=1$	$f_2(V_i(k))=2$	$f_2(V_i(k))=3$	$f_2(V_i(k))=4$
0	0	1	0

Therefore, we get $B_2= b_{21}b_{22}b_{23}b_{24}$="**0010**"

For the data-time type data, assume that the domain of attribute *Manufacturing_Date* is <1992/01/02, 1992/10/01, 1993/10/10, 1994/01/22, 1992/06/07> and the attribute value of *Manufacturing_Date* in the second record is 1992/10/01. Also, the given function f_2 is given in following.

$$F_2(V_i(k))=\begin{cases} Year(V_i(k))=1992 & =1 \\ Year(V_i(k))=1993 & =2 \\ Year(V_i(k))=1994 & =3 \end{cases}$$

According to the Definition 3.5, bit-wise indexing method uses the 3 bits as the bit vector of the index in which every bit represents a specific value of the index attribute *Manufacturing_Date*.

B_2:

$f_2(V_i(k))=1$	$f_2(V_i(k))=2$	$f_2(V_i(k))=3$
1	0	0

Therefore, we get

B_2= "**100**"

DEFINITION 3.6 - Bit-wise indexing vector of a record :

The bit-wise indexing vector BWI_k of a record R_k is the concatenation of the bit-wise indexing vectors of all the attributes for record R_k. That is, $BWI_k=B_1B_2\ldots B_r$, where r is the number of attributes.

DEFINITION 3.7 - Matrix of bit-wise indexes for table *T* :

A matrix T_{BWI} of bit-wise indexes for Table T is represented as $\begin{bmatrix} BWI_1 \\ BWI_2 \\ \vdots \\ BWI_{|R|} \end{bmatrix}$, where $|R|$ is the number of records.

The bit-wise indexes for all saved records are generated by the following two algorithms:

Algorithm 3.1 - *Bit-wise index creation algorithm* :

Input: A record R_i.

Output: A bit-wise index BWI_i of R_i.

Step 1: Create a bit-wise vector of length r, where r is the number of attributes.

Step 2: Repeat the following sub-steps for each attribute j until all attributes are processed.

```
    Step 2.1: If Type($A_j$)≠S, go to Step 2.2, else set $b_{jk}$=1 if $V_j(i)=V_{jk}$; set $b_{jk}$=0
              otherwise.
    Step 2.2: Set $b_{jk}$=1 if $f_j(V_j(i))=k$; set $b_{jk}$=0 otherwise.
Step 3: Return the vector $BWI_i$.
```

Algorithm 3.2 - ***Bit-wise index matrix creation algorithm*** **:**

```
Input:  A set of records in $T$.
Output: A bit-wise index matrix $T_{BWI}$ of the records.
Step 1: Create an empty matrix $T_{BWI}$.
Step 2: Repeat the following sub-steps for each record $R_i$ until all records are
        processed.
    Step 2.1: Use the bit-wise index creation algorithm (Algorithm 3.1) to get the
              index $BWI_i$ of $R_i$.
    Step 2.2: Add $BWI_i$ into $T_{BWI}$.
Step 3: Return $T_{BWI}$.
```

After a bit-wise index matrix is built, bit-wise operations can easily be used to retrieve desired record for the new coming queries.

EXAMPLE 3.4:

Assume that a Target Table *T* containing five records is shown in Figure 2.4(a), The bit-wise indexes for the above records are shown in Table 3.2.

Table 3.2: The T_{BWI} of five records in Figure 2.4(a)

BWI_1	10000	10000	100
BWI_2	01000	01000	100
BWI_3	00100	00100	010
BWI_4	00010	00010	010
BWI_5	00001	00001	001

3.3 The Matching Phase of Simple BWI Method

Calculating the similarities between a query and saved records is a time-consuming task. A two-phase matching approach, called the *Similar-records-seeking algorithm*, is thus proposed here to reduce the matching time. It includes the *relevant-records-retrieving* phase and the *similarity-computing* phase. In the first phase, all irrelevant records are filtered out to avoid calculation of their similarities. The time of calculating the similarities of useful saved records can then be decreased. The similarities of the query with remaining saved records are then computed in the *similarity-computing* phase. The algorithm is described as follows.

Algorithm 3.3 - *Similar-records-seeking algorithm* :

```
Input  : A bit-wise index matrix $T_{BWI}$ and a new query $R_N$.
Output : A set of similar record $Rc$ with their similarity degrees with $R_N$.
Step 1:   Use the bit-wise index creation algorithm (Algorithm 3.2) to get the index
          $BWI_N$ of the new query $R_N$ according to the condition part of the query.
Step 2:   Initialize the counter $j$ to 1 and $Rc$ to an empty set.
Step 3:   For each $BWI_j$ in $T_{BWI}$, do the following sub-steps ($1<j\leq|R|$):
    Step 3.1:   Call the search-relevant-records algorithm (Algorithm 3.4) to compute
                the relevance degree $rdi_j$ between $BWI_N$ and $BWI_j$.
    Step 3.2:   If $rdi_j$=0, ignore the record $R_j$ and go to Step 3.5.
    Step 3.3:   Call the similarity-computing algorithm (Algorithm 3.6) to compute
                the similarity $sim_j$ between $R_N$ and $R_j$.
    Step 3.4:   Add record $R_j$ with its similarity $sim_j$ to $Rc$.
    Step 3.5:   Add 1 to $j$.
Step 4:   Sort the results in $Rc$ in descending order of their similarities.
Step 5:   Output $Rc$.
```

A saving record is relevant to a new query that will be transformed to a desired bit-wise index via *bit-wise index creation algorithm*. If they have at least one same attribute value, the saving record is then similar with the new query in a certain degree. The bits in the corresponding positions of the matched attributes should be set as "1" in their bit vectors. This can easily be found by using the 'AND' bit-wise operation to compare the two bit vectors. The following *Search-relevant-records algorithm* is thus proposed to achieve this purpose.

Algorithm 3.4 - *Search-relevant-records algorithm* :

Input:	The bit-wise indexing vector BWI_N of a new query R_N and the index BWI_j of a saved record R_j in R.
Output:	The relevant degree rdi_j between R_N and R_j.
Step 1:	Use the 'AND' bit-wise operation on BWI_N and BWI_j and store the result as rdi_j, which is also a bit string.
Step 2:	Return rdi_j.

Since the 'AND' bit-wise operation is fast, the *Search-relevant-records algorithm* selects relevant saved records quickly. If *rdi* is zero, then the saved record is thought of as irrelevant and will be filtered out.

After all relevant saved records have been retrieved, the similarities between the query condition and them are computed. As mentioned above, a matching function based on a weighted sum of matched attributes is defined to calculate the similarity degrees. Each attribute has its own weight. Since a record has only one value for an attribute, at most one bit in the bit string *rdi* is set for each attribute after the *Search-relevant-records algorithm* is executed. Accordingly, a special bit-wise vector, called the *Mask Vector*, is proposed to help compute similarities. Let <1> be the string of length α with all bits being 1 and <0> be the string of length α with all bits being 0. The definition of the *Mask Vector* is shown below.

DEFINITION 3.8 - Mask Vector:

A bit-wise indexing mask vector *Mask* is a set of $Mask_k$, where $0 < k \le r$ and r is the number of attributes. Each $Mask_k$, denoting the mask vector of attribute A_k, is a concatenation of r bit strings as $Mask_k = S_1S_2\ldots S_{\sum_{j=1}^{r}\alpha(j)}$, where S_i = <1> for $\sum_{j=1}^{k-1}\alpha(j)+1 \le i \le \sum_{j=1}^{k}\alpha(j)$ and S_i = <0> otherwise.

By applying the 'AND' operation on $Mask_k$ and the bit-wise vectors *rdi*'s generated from the *search-relevant-records algorithm*, the similarities between a query and a saved record for attribute A_k can easily be found by the following similarity-measuring function:

$$SIM(R_i) = \frac{\sum_{j=1}^{r}(PC_{ij} \times W_j)}{\sum_{j=1}^{r} W_j},$$

where $SIM(R_i)$ is the similarity between the *i*-th saved record and the new query, W_j is the weight of the *j*-th attribute, PC_{ij} = 0 if the result of performing the AND bit-wise operation on rdi_i and $Mask_j$ is 0, and PC_{ij} = 1 otherwise.

Several saved records may have the same similarity with a new query as long as they have the same attributes matched. This is especially common when the numbers of possible values for attributes are large. For this situation, the cost for calculating similarities of saved relevant records can be reduced if all possible similarities are pre-computed and stored into the *Similarity Mapping List*. Each element in the *Similarity Mapping List* is a similarity value for some attributes matched. Thus, the similarity of a saved record with a new query for known attributes matched can easily be found from the list, instead of from calculation by the above formula. The *Similarity Mapping List* is formally defined as follows.

DEFINITION 3.9 - Similarity Mapping List:

Let *L* be a *Similarity Mapping List* and L_i be an element in *L* with an index value *i*, which is determined from the attributes matched, $1 \le i \le 2^{|r|}-1$. Let *i* be represented as a binary code $b_{i1}b_{i2}\ldots b_{ir}$, with b_{ij}=1 if the *j*-th attribute is matched and b_{ij}=0 otherwise, 1≤j≤r. The value of L_i is thus $\frac{\sum_{j=1}^{r} b_{ij} \times W_j}{\sum_{j=1}^{r} W_j}$.

Algorithm 3.5 - *Similarity-mapping-list creation algorithm* :

```
Input:    Weights of attributes $W_1$, $W_2$, …, $W_r$ of $R$.
Output:   A similarity mapping list $L$.
Step 1:   Initialize the counter $i$ to 1 and the list $L$ to be empty.
Step 2:   For each $i$, $1 \le i \le 2^{|r|}-1$, do the following sub-steps:
    Step 2.1:   Encode $i$ into a binary string $<b_{i1}b_{i2}…b_{ir}>$.
    Step 2.2:   Calculate the similarity degree $L_i$ by the formula in Definition 3.9.
    Step 2.3:   Put $L_i$ into the list $L$ with index $i$.
Step 3:   Return $L$.
```

After the *Similarity Mapping List* has been built, the similarity of each saved record and a new query can be quickly found by the following algorithm.

Algorithm 3.6 - *Similarity-computing algorithm* :

```
Input:    The relevant degree $rdi_j$ of record $R_j$ with a new query, the Mask Vector, and
          the Similarity Mapping List $L$.
```

Output: The similarity of R_j with a new record.

Step 1: Initialize a zero binary string of length r.

Step 2: For each i, $1 \leq i \leq r$, set the i-th position in the string to 1 if the result of using the 'AND' bit-wise operation on $Mask_i$ and rdi_j is not all 0.

Step 3: Transform the binary string into an integer j.

Step 4: Get L_j from the *Similarity Mapping List*.

Step 5: Return L_j.

Since the *Similarity Mapping List* and the *Mask Vector* are constructed in the pre-processing step, and since only the 'AND' bit-wise operations are executed on *Mask Vectors* and bit-wise vectors of relevant records in the *Similarity-computing algorithm*, the computational time for finding the similarities can thus be significantly reduced.

EXAMPLE 3.5:

Continuing from Example 3.4, the BWI_N of a new query R_N, which is {Toolid=6210, Name=AWOX01, Location=FAB1}, is <10000 10000 100>. Also assume that weight W_1, W_2 and W_3 are set to 0.33. Each BWI_j in T_{BWI} in Table 3.2 is processed as follows.

- For BWI_1, The relevant degree rdi_1 between BWI_1 and BWI_N is found as <10000 10000 100> by the *Search-relevant-records algorithm*. Since more than one bit in rdi_1 are "1", Record 4 is a relevant record. Its similarity is found as 1. Record 1 is then a relevant record.
- For BWI_3, BWI_4 and BWI_5, The relevant degree *rdi* between these records and BWI_N is found as <00000 00000 000>. Since all the bits in these *rdi*s are "0", Records 3, 4 and 5 are thus filtered out.

After the relevant records are sorted in decreasing order of similarities, the results are shown is Table 3.3.

Table 3.3: Two relevant records and their similarities

Relevant Record	**Record 1**	**Record 2**
Similarity	**1**	**0.333**

3.4 Analysis and Experiments of Simple BWI Method

As mentioned above, the proposed matching algorithms include two phases to reduce the computational time. At the *retrieving-relevant-records* phase, irrelevant prior records are filtered out. Thus, only the similarities between relevant prior records and the new query are computed at the *similarity-computing* phase. Assume that the number of records in the target table is *N* and the average filtering percentage is *M*. The time needed to retrieve relevant saved records and to calculate their similarities in STEP 3 of Algorithm 3.5 is analyzed as:

$$\begin{aligned} Time_{with\ filtering} &\approx (N \times t_{and} + N \times M \times (r \times t_{and}) + N \times M \times t_c) \\ &= N \times (M \times \frac{1}{M} \times t_{and} + M \times (r \times t_{and}) + M \times t_c) \\ &= N \times M \times ((\frac{1}{M} + r) \times t_{and} + t_c), \end{aligned}$$

where t_{and} is the time needed for an 'AND' bit-wise operation and t_c is the seek time in the *Similarity Mapping List*. If no filtering is performed, the time needed to calculate their similarities in STEP 3 of Algorithm 3.5 is analyzed as:

$$\begin{aligned} Time_{without\ filtering} &\approx (N \times t_{and} + N \times (r \times t_{and}) + N \times t_c) \\ &= N \times ((1 + r) \times t_{and} + t_c). \end{aligned}$$

The performance due to the filtering is then:

$$\frac{Time_{with\ filtering}}{Time_{without\ filtering}} \approx \frac{N \times M \times ((\frac{1}{M} + r) \times t_{and} + t_c)}{N \times ((1 + r) \times t_{and} + t_c)}$$

$\approx M$.

The proposed method can indeed improve the performance of query although some extra storage spaces are required. These storage spaces are used for storing the bit-wise indexes and the *Similarity Mapping List*. The sizes of extra storage spaces required in our method are analyzed as follows.

- The storage space required for the bit-wise indexes $T_{BWI} = |R| \times \sum_{i=1}^{r} \alpha(i)$, where $\alpha(i)$ is the number of bits used for attribute A_i, r is the number of attributes, and $|C|$ is the number of records in warehouse. For example, assume that there are 100000 records in a warehouse and 16 attributes to describe each record. Also assume each attribute has 4 possible values. The storage space required for T_{BWI} = $(100000) \times \sum_{i=1}^{16} 4$ bits = (6400000/8) bytes = 800000 bytes ≅ 0.8 M bytes.
- The storage space of the *Mask Vector* $= r \times \sum_{i=1}^{r} \alpha(i)$. For the above example, the storage space required for the *Mask Vector* = $(16 \times \sum_{i=1}^{16} 4)$ bits = (1024/8) bytes =128 bytes.
- The storage space required for the *Similarity Mapping List* $L = f \times (2^r - 1)$, where f is the storage space required for storing a similarity value. Assume that f is a 4-byte real number. For the above example, the storage space required for the *Similarity Mapping List* $L = 4 \times (2^{16}-1)$ bytes = 262140 bytes ≅ 256 K bytes.

Note that the size of the extra storage space required for the *Similarity Mapping*

List is exponential to *r*. Therefore, the *Similarity Mapping List* is not suitable for domains with large numbers of attributes.

The result of comparing the Simple BWI indexing method with the Bitmap indexing method is shown in Figure 3.1.

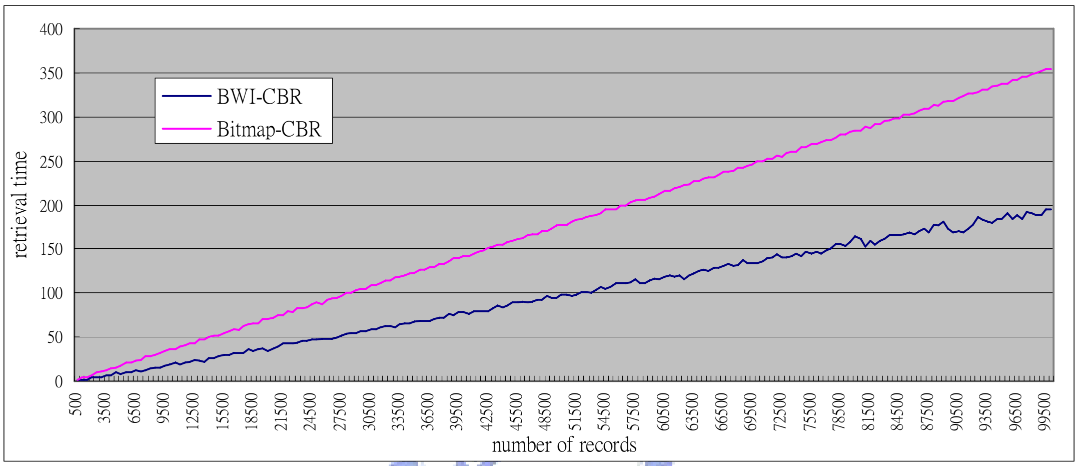


Figure 3.1: Simple BWI indexing method v. s. Bitmap indexing method

We can see that Simple BWI method is faster than Bitmap indexing method, the reasons are:

- In retrieving relevant cases phase, the Bitmap indexing technology is not suitable for retrieving similar cases. For example, when a new case comes, the Bitmap indexing method needs to check all possible attribute combination vectors in order to retrieve relevant prior cases. The more attributes check, the more time it needs.
- In similarity measurement phase, the Bitmap indexing method needs to check the

all corresponding position in all possible attribute combination vectors, especially when the number of attribute of query needs or the number of records in the table T are large. The waste time is lengthy and unbearable. Therefore, the BWI indexing method is faster than that in Bitmap indexing method when the similarity computing is needed.

Also, we compare the Simple BWI indexing method with single processor and the parallel Simple BWI indexing with multiple processors for showing the improvement of the performance. In Figures 3.2 and 3.3, the dual CPUs parallel Simple BWI indexing method can increase the performance about 1.6 times and the quad CPUs parallel Simple BWI indexing method can increase the performance about 3.2 times. It is obvious that Simple BWI indexing method is quite suitable for parallelization since the bit-wise indexing matrix of the proposed method can be separated into several independent sub-matrixes and these sub-matrixes is almost balanced. Therefore, when the Simple BWI indexing method is built in a multiple CPU machine, the workload can be easily shared into each processor and assure that the workload of each processor is almost balanced.

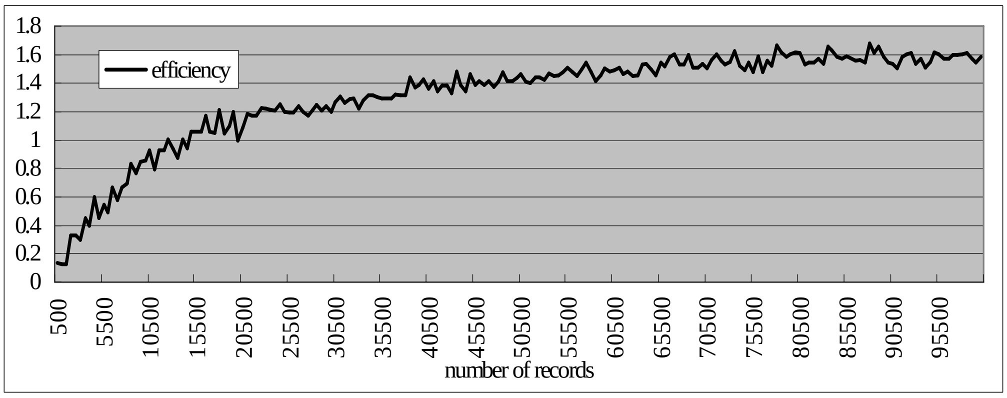


Figure 3.2: Speed-up of parallel BWI indexing on two processors machine.

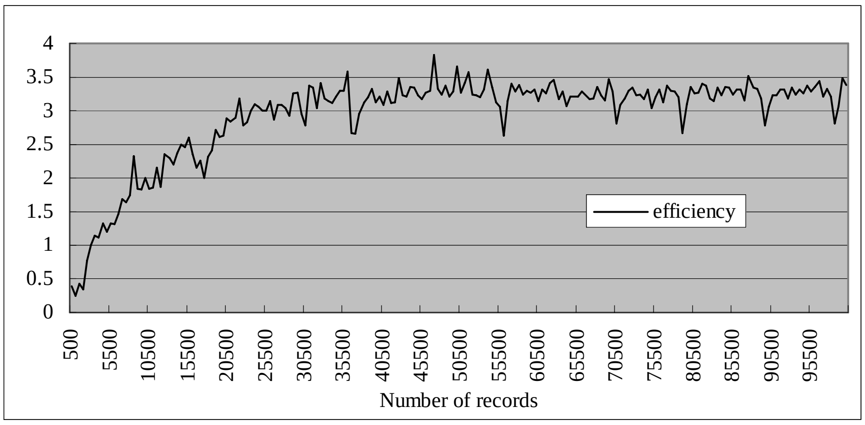


Figure 3.3: Speed-up of parallel BWI indexing on four processor machine.

Chapter 4
Advanced Bit-wise Indexing Method

In the chapter, the advanced bit-wise indexing method, including *Encapsulated bit-wise indexing method* and *Compacted bit-wise indexing method*, are introduced, including the definitions and algorithms of indexing and matching phases in these two bit-wise indexing methods are proposed in the following sections.

4.1 Encapsulated Bit-wise Indexing Method

4.1.1 General Assumptions and Notations for Encapsulated BWI Technology

As we can see, the bit length of bit-wise indexing vector for some attribute depends on the number of its distinct values. When the attribute contains a large amount of distinct values, the size of its corresponding bit-wise indexing vector becomes hugely large, when the required bit-length is too large to handle, partitioning the bit-length to several levels seems helpful for this issue. There is a threshold (*Th*) which can be used to determine whether the encapsulated bit-wise indexing technology is applied or not. That is, when the total length of bit vectors is larger than this threshold (*Th*), the algorithm is applied on. The following notations and definitions are given to describe the encapsulated bit-wise indexing method.

NOTATION 4.1 :

el_i = the maxima encapsulated level of attribute A_i

ei_i^j = the bit length of j-th encapsulated level of attribute A_i

ei_i = the total bit length of BWI index for the given attribute A_i, $ei_i = \sum_{j=1}^{el_i} ei_i^j$

ei = the total bit length of BWI index for the given record R_i, $ei = \sum_{j=1}^{r} ei_j$

Th= the threshold for separating bit-length to levels boundary

We propose an *Encapsulated bit-wise indexing method* on data warehouse to achieve the goal of saving storage and accelerating user query procedure. This method includes two phases. One is creating indexes phase, and the other is querying phase. The indexing phase transforms the contents of table into a bit vector matrix (in here, called a matrix of bit-wise indexes), and the query phase is retrieving records to answer the query statements as soon as possible.

4.1.2 The Indexing Phase of Encapsulated BWI Method

The indexing phase includes *Encapsulated level calculating Algorithm*, *Encapsulated BWI attributes index creating Algorithm* and *Encapsulated BWI matrix of bit-wise indexes creating Algorithm*. The *Encapsulated level calculating Algorithm* calculates an encapsulated level of each attribute for creating the corresponding bit-wise indexes, the *Encapsulated BWI Bit-wise indexes creating Algorithm* creates corresponding BWI index of matrix of multi-level bit-wise indexes. The *Encapsulated BWI Matrix of bit-wise indexes creating Algorithm* creates bit vectors matrix of data warehouse. These algorithms and examples are shown as follows.

In *Encapsulated bit-wise indexing method*, there are several methods to decide the partition size of indexing vector. Here, we use *square root* to calculate the compact size of indexing vector. For instance, when *n* bits are required to represent a specify attributes in simple bit-wise indexing method, $2\lceil \sqrt{n} \rceil$ bits are required by two levels indexing vectors respectively in two-level encapsulated bit-wise indexing method. For example, assume that attribute *A* uses 10,000 bits to be the indexing vector when simple bit-wise indexing method is applied. There are 200 bits are required in two-level condensable bit-wise indexing method. As we can see, the used bits can be largely reduced to 1/50. When the condensable bit-wise indexing method is used in the data warehousing, the used bits in much more compact then using bitmap and simple bit-wise indexing methods. Therefore, we propose the following definitions and algorithms.

Algorithm 4.1 - ***Encapsulated level calculating Algorithm – Square Root*** **:**

Input: Table *T* of data warehouse and threshold *Th*.

Output: The corresponding el_i and ei_i^j for all attribute in *A*.

Step 1 : Let $el_i = 1, ei_i^1 = \alpha(i)$ and $ei = \sum_{j=1}^{r}\sum_{k=1}^{el_i} ei_j^k$,for $1 \le i \le r$.

Step 2: If $ei > Th$, do the following sub-steps; otherwise go to Step 3.

Step 2.1: If not exist a ei_i^j where and $ei_i^j > 2\times\lceil\sqrt{ei_i^j}\rceil$ with minima el_i and j, Return *false* for *Th* limitation

Step 2.2: Let $el_i=el_i+1$, $ei=ei-ei_i^j+2\times\lceil\sqrt{ei_i^j}\rceil$, $ei_i^j=\lceil\sqrt{ei_i^j}\rceil$, $ei_i^{el_i}=\lceil\sqrt{ei_i^j}\rceil$ and go to Step 2.

Step 3: Return the corresponding el_i and ei_i^j for all attribute in *A*.

EXAMPLE 4.1:

Figure 4.1 shows a flat target table T including attribute set A = < LotID, StepID, ToolID, Yield >, four attributes and 23 records. The attribute values domains of Cid, Name, Gender, and City are V1=< 0001, 0002, 0003, ….., 00022, 00023 >, V2=< PS_1, PS_2, PS_3, PS_4, PS_5 >, V3=< AWOX11, AWOX12, AWOX13, AWOX14, AWOX21, AWOX31, AWOX32, AWOX33, AWOX34, AWOX35, AWOX36, AWOX41, AWOX42, AWOX43, AWOX51>, and V4=< 92.1, 92.2, 92.3, 93.1, 93.2, 94.3, 94.4, 94.5, 94.6, 95.5, 95.6, 95.7, 95.7, 95.8, 96.1, 96.5, 99.1, 99.3>, respectively. It can be easily seen that the number of distinct values of *LotID*, *StepID*, *ToolID* and Yield are ei1= 23, ei2=5, ei3=15 and ei4=18, respectively.

	LotID	StepID	ToolID	Yield
1	0001	PS_1	AWOX11	92.1
2	0002	PS_1	AWOX11	92.3
3	0003	PS_1	AWOX12	92.2
4	0004	PS_1	AWOX13	99.1
5	0005	PS_1	AWOX14	99.3
6	0006	PS_2	AWOX21	93.1
7	0007	PS_2	AWOX21	94.5
8	0008	PS_2	AWOX21	95.6
9	0009	PS_2	AWOX21	95.7
10	0010	PS_3	AWOX31	96.1
11	0011	PS_3	AWOX32	92.2
12	0012	PS_3	AWOX33	92.3
13	0013	PS_3	AWOX34	93.1
14	0014	PS_3	AWOX35	94.4
15	0015	PS_3	AWOX36	95.8
16	0016	PS_4	AWOX41	93.2
17	0017	PS_4	AWOX41	94.5
18	0018	PS_4	AWOX41	95.6
19	0019	PS_4	AWOX42	94.6
20	0020	PS_4	AWOX42	94.3
21	0021	PS_4	AWOX43	95.7
22	0022	PS_5	AWOX51	95.5
23	0023	PS_5	AWOX51	96.5

Figure 4.1: An example of Flat target table *T*

$ei = \sum_{j=1}^{4}\sum_{k=1}^{1} ei_j^k = 23 + 5 + 15 + 18 = 61$ bits be the length of an encoded record.

The threshold *Th* is set to 35 initially. Also, all levels of attributes have initial value 1, e.g., $el_1 = 1$, $el_2 = 1$, $el_3 = 1$, $el_1 = 1$ and the vector length of all attribute are thus $ei_1^1=23$, $ei_2^1=5$, $ei_3^1=15$ and $ei_4^1=18$.

Since $ei > Th$, the attribute *LotID* with the max length of indexing string $ei_1^1=23$ is chosen for length reducing. Therefore, the encapsulation level of attribute *LotID* $el_1=1+1=2$, $ei_1^1 = ei_1^2 = \lceil \sqrt{23} \rceil = 5$, $ei_1=10$ and the total length of vectors *ei* is reduced to 48 (61-23+10). However, the length is still larger than the threshold *Th*. The attribute *Yield* is then chosen. Therefore, the encapsulation level of attribute *Yield* $el_4=1+1=2$, $ei_4^1 = ei_4^2 = \lceil \sqrt{18} \rceil = 5$, $ei_4=10$ and the total length of vectors *ei* is reduced to 40 (48-18+10). Since the length is still larger than the threshold *Th*. The attribute *ToolID* is then chosen. Therefore, the encapsulation level of attribute *ToolID* $el_3=1+1=2$, $ei_3^1 = ei_3^2 = \lceil \sqrt{15} \rceil = 4$, $ei_4=8$ and the total length of vectors *ei* is reduced to 33 (40-15+8). Finally, the total length of vector is reduced to 33 and the algorithm stops.

As mentioned in Definition 3.5, the user can provide a suitable transforming function for the continuously type attributes, including numeric and data-time type. In the encapsulated BWI method, user can provide an el_i-level transforming functions f_i for the attribute A_i in order to close for the physical meaning than encapsulated BWI itself only. $f_i = < f_i^1, f_i^2, \ldots, f_i^{el_i} >$ where the number of value domain of f_i^k should equal to ei_i^k. The definition is shown below:

DEFINITION 4.1 – Encapsulated BWI bit-wise indexing vector of an attribute where Type(A_i)≠S :

The bit-wise indexing vector B_i of the *i*-th attribute for the record R_j in T is set of bit strings. B_i=< B_i^1, B_i^2, …, $B_i^{el_i}$ >, where f_i=< f_i^1, f_i^2,…, $f_i^{el_i}$ > is the el_i-level of function that given by user $B_i^k = b_{j1}b_{j2}\dots b_{je_i^k}$, where b_{jl}=1 if $f_i^k(V_i(k))=l$ and b_{jl}=0 otherwise.

EXAMPLE 4.2 :

Assume that the second attribute *Recipe_degree* is <10, 12, 14, 16, 18, 20, 22, 24>. Also, after the ***Encapsulated level calculating Algorithm – Square Root*** executed, and the ei_2^1 , ei_2^1 and el_i are all set to 2. The attribute value of *Recipe_degree* in the second record is 16. Also, user gives the following two-level (fl=2) function f_2^1 and f_2^2 .

$f_2^1(V_i(k))=\lfloor V_i(k)/10\rfloor$

$f_2^2(V_i(k))=\lfloor(V_i(k)-(f_2^1(V_i(k))\times 10))/5\rfloor+1$

According to the Definition 4.1, bit-wise indexing method uses the 4 bits as the bit vector of the index in which every bit represents a specific value of the index attribute *Recipe_degree*.

B_2^1:

$f_2^1(V_i(k))=1$	$f_2^1(V_i(k))=2$
1	0

B_2^2:

$f_2^2(V_i(k))=1$	$f_2^2(V_i(k))=2$
0	1

Therefore, we get B_2=B_2^1 B_2^2="**1001**"

Algorithm 4.2 - *Encapsulated BWI bit-wise indexes creating Algorithm* :

```
Input:    A record $R_i$.

Output:   A bit-wise index $BWI_i$ of $R_i$.

Step 1:   Create a bit-wise vector $BWI_i$ of length 0.

Step 2:   Repeat the following sub-steps for each attribute $A_j$ until all attributes are
          processed.

    Step 2.1:   If Type($A_j$) $\neq$ S and $f_i \neq \varnothing$, go to Step 2.2, else let $m=n$ if $V_j(i)=V_{jn}$, create
                a bit-wise vector $B_i$ with 0 and repeat the following sub-steps for each
                encapsulated level $el_k$ until all encapsulated levels are processed

        Step 2.1.1:   Let $B'=b_1b_2\ldots b_{ei_j^k}$ to a all-zero string with length $ei_j^k$

        Step 2.1.2:   If $k \neq el_k$, go to Step 2.1.3, else if the $m$=0, set $b_{ei_j^k}$=1 and set $b_m$=1
```

```
                otherwise, go to Step 2.1.5.
        Step 2.1.3:  Let $o = \lfloor m/\prod_{p=k+1}^{el_j} ei_i^p \rfloor$, if o=$ei_j^k$, set $b_o$=1 and set $b_{o+1}$=1 otherwise.
        Step 2.1.4:  Set $m$=$m$-($o\times\prod_{p=k+1}^{el_j} ei_i^p$ )
        Step 2.1.5:  Concatenate the bit strings $B_j$ and $B'$ into $B_j$.
    Step 2.2:  If Type($A_j$) $\neq$ S and $f_i\neq\varnothing$, for each $B_j^k$, do the following sub-steps
        Step 2.2.2:  Set $b_{jl}$=1 if $f_i^k(V_i(k))=l$ and $b_{jl}$=0 otherwise.
        Step 2.2.3:  Concatenate the bit strings $B_j$ and $B_j^k$ into $B_j$.
Step 3:  Concatenate the bit strings $B_1$, $B_2$,…, and $B_r$ into $BWI_i$.
Step 4:  Return the vector $BWI_i$.
```

Algorithm 4.3 - *Encapsulated BWI Matrix of bit-wise indexes creating Algorithm* :

```
Input:   Table $T$ of the data warehouse.
Output:  The $T_{BWI}$ of the data warehouse.
Step 1:  Create an empty bit-wise indexes matrix $T_{BWI}$ for table $T$.
Step 2:  Call Encapsulated level calculating Algorithm – Square Root (Algorithm 4.1)
         to get the corresponding $el$s and $ei$s.
Step 3:  Repeat the following sub-steps for each record $R_i$ until all records are
         processed.
```

Step 3.1: Use the *Encapsulated BWI bit-wise index creation algorithm* (**Algorithm 4.2**) to get the index BWI_i of R_i.

Step 3.2: Add BWI_i into T_{BWI}.

Step 4: Return T_{BWI}.

After a bit-wise index matrix is built, bit-wise operations can easily be used to retrieve desired record for the new coming queries.

EXAMPLE 4.3:

Assume that a Target Table *T* containing 23 records is shown in Figure 4.2 and the user gives the following two-level (*fl*=2) function f_2^1 and f_2^2 of attribute *Yield* .

$$f_2^1(V_i(k))=\lceil (V_i(k)-90)/2 \rceil$$

$$f_2^2(V_i(k))=\lceil ((V_i(k) - (90+(f_2^1(V_i(k))-1) \times 2)) /0.4) \rceil$$

The bit-wise indexes for the above records are shown in Table 4.1.

Table 4.1: The T_{BWI} of 23 records in Figure 4.2

BWI	LotID		StepID	ToolID		Yield	
eis	ei_1^1	ei_1^2	ei_2^1	ei_3^1	ei_3^2	ei_4^1	ei_4^2
BWI_1	10000	10000	10000	1000	1000	01000	10000
BWI_2	10000	01000	10000	1000	1000	01000	10000
BWI_3	10000	00100	10000	1000	0100	01000	10000
BWI_4	10000	00010	10000	1000	0010	00001	00100
BWI_5	10000	00001	10000	1000	0001	00001	00010
BWI_6	01000	10000	01000	0100	1000	01000	00100
BWI_7	01000	01000	01000	0100	1000	00100	01000
BWI_8	01000	00100	01000	0100	1000	00100	00010
BWI_9	01000	00010	01000	0100	1000	00100	00001
BWI_{10}	01000	00001	00100	0100	0100	00010	10000
BWI_{11}	00100	10000	00100	0100	0010	01000	10000
BWI_{12}	00100	01000	00100	0100	0001	01000	10000
BWI_{13}	00100	00100	00100	0010	1000	01000	00100
BWI_{14}	00100	00010	00100	0010	0100	00100	01000
BWI_{15}	00100	00001	00100	0010	0010	00100	00001
BWI_{16}	00010	10000	00010	0010	0001	01000	00010
BWI_{17}	00010	01000	00010	0010	0001	00100	01000
BWI_{18}	00010	00100	00010	0010	0001	00100	00010
BWI_{19}	00010	00010	00010	0001	1000	00100	01000
BWI_{20}	00010	00001	00010	0001	1000	00100	10000
BWI_{21}	00001	10000	00010	0001	0100	00100	00001
BWI_{22}	00001	01000	00001	0001	0010	00100	00010
BWI_{23}	00001	00100	00001	0001	0010	00010	01000

4.1.3 The Matching Phase of Encapsulated BWI Method

Calculating the similarities between a query and saved records is a time-consuming task. A two-phase matching approach, called the *Encapsulated BWI Similar-records-seeking algorithm*, is thus proposed here to reduce the matching time.

It includes the *Encapsulated BWI relevant-records-retrieving* phase and the *Encapsulated BWI similarity-computing* phase. In the first phase, all irrelevant records are filtered out to avoid calculation of their similarities. The time of calculating the similarities of useful saved records can then be decreased. The similarities of the query with remaining saved records are then computed efficiently in the *similarity-computing* phase. The algorithm is described as follows.

Algorithm 4.4 - ***Encapsulated BWI Similar-records-seeking algorithm*** **:**

Input : A bit-wise index matrix T_{BWI} and a new query R_N.

Output : A set of similar record Rc with their similarity degrees with R_N.

Step 1: Use the *Encapsulated BWI bit-wise index creation algorithm* (**Algorithm 4.2**) to get the index BWI_N of the new query R_N according to the condition part of the query.

Step 2: Initialize the counter j to 1 and Rc to an empty set.

Step 3: For each BWI_j in T_{BWI}, do the following sub-steps ($1<j\leq|R|$):

Step 3.1: Call the *Encapsulated BWI search-relevant-records algorithm* (**Algorithm 4.5**) to compute the relevance degree rdi_j between BWI_N and BWI_j.

Step 3.2: If rdi_j=0, ignore the record R_j and go to Step 3.5.

```
    Step 3.3:  Call the *Encapsulated BWI similarity-computing algorithm* (**Algorithm
               4.7**) to compute the similarity $sim_j$ between $R_N$ and $R_j$.
    Step 3.4:  Add record $R_j$ with its similarity $sim_j$ to $Rc$.
    Step 3.5:  Add 1 to $j$.
Step 4:  Sort the results in $Rc$ in descending order of their similarities.
Step 5:  Output $Rc$.
```

Even the encoding procedure of BWI index in Encapsulated BWI method is different than the Simple one, it still can easily be found by using the 'AND' bit-wise operation to compare the two bit vectors. The following *Encapsulated BWI Search-relevant-records algorithm* is thus proposed to achieve this purpose.

Algorithm 4.5 - *Encapsulated BWI Search-relevant-records algorithm* :

```
Input:   The bit-wise indexing vector $BWI_N$ of a new query $R_N$ and the index $BWI_j$
         of a saved record $R_j$ in $R$.
Output:  The relevant degree $rdi_j$ between $R_N$ and $R_j$.
Step 1:  Use the 'AND' bit-wise operation on $BWI_N$ and $BWI_j$ and store the result as
         $rdi_j$, which is also a bit string.
Step 2:  Return $rdi_j$.
```

Since the ‘AND’ bit-wise operation is fast, the *Search-relevant-records algorithm* selects relevant saved records quickly. If *rdi* is zero, then the saved record is thought of as irrelevant and will be filtered out. Since the properties of Encapsulated BWI mode, if *rdi* has some ‘1’ bits, it does not mean that the saved record is relevant. As mentioned above, a matching function based on a weighted sum of matched attributes is defined to calculate the similarity degrees. As the same with *Simple BWI* method. the *Mask Vector* and the *Similarity Mapping List* are used in *Encapsulated BWI* method and then be defined at Definition 4.2 and 4.3.

DEFINITION 4.2 - Encapsulated BWI Mask Vector :

A Encapsulated BWI bit-wise indexing mask vector *eMask* is a set of $eMask_k$, where $0 < k \le \sum_{i=1}^{r} el_i$. Each $eMask_k$, denoting the mask vector of attribute A_k, is a concatenation of r bit strings as $eMask_k = S_1 S_2 \dots S_{\sum_{i=1}^{r} el_i}$, where S_i = <1> for $\sum_{i=1}^{k-1} el_i \le i \le \sum_{i=1}^{k} el_i$ and S_i = <0> otherwise.

DEFINITION 4.3 - Encapsulated BWI Similarity Mapping List :

Let L be an *Encapsulated BWI Similarity Mapping List* and L_i be an element in L with an index value i, which is determined from the attributes matched, $1 \le i \le 2^{\sum_{i=1}^{r} el_i}$ -1. Let i be represented as a binary code $b_{i1} b_{i2} \dots b_{i\sum_{i=1}^{r} el_i}$. The value of L_i is thus

$$\frac{\sum_{j=1}^{r} \prod_{k=\sum_{l=1}^{j-1} el_l+1}^{\sum_{l=1}^{j+1} el_l} b_{ik} \times W_j}{\sum_{j=1}^{r} W_j}.$$

Algorithm 4.6 - ***Encapsulated BWI Similarity-mapping-list creation algorithm*** **:**

```
Input:    Weights of attributes $W_1, W_2, \ldots, W_r$ of $R$.

Output:   A similarity mapping list $L$.

Step 1:   Initialize the counter $i$ to 1 and the list $L$ to be empty.

Step 2:   For each $i$, $1 \le i \le 2^{\sum_{i=1}^{r} el_i}-1$, do the following sub-steps:
    Step 2.1:   Encode $i$ into a binary string $<b_{i1}b_{i2}\ldots b_{i\sum_{i=1}^{r} el_i}>$.
    Step 2.2:   Calculate the similarity degree $L_i$ by the formula in Definition 4.3.
    Step 2.3:   Put $L_i$ into the list $L$ with index $i$.

Step 3:   Return $L$.
```

After the *Similarity Mapping List* has been built, the similarity of each saved record and a new query can be quickly found by the following algorithm.

Algorithm 4.7 - ***Encapsulated BWI Similarity-computing algorithm*** **:**

Input: The relevant degree rdi_j of record R_j with a new query, the *Mask Vector*, and the *Similarity Mapping List L*.

Output: The similarity of R_j with a new record.

Step 1: Initialize a zero binary string of length r.

Step 2: For each i, $1 \leq i \leq \sum_{i=1}^{r} el_i$, set the i-th position in the string to 1 if AND($eMask_i$, rdi_j) = AND($eMask_i$, BWI_N).

Step 3: Transform the binary string into an integer j.

Step 4: Get L_j from the *Similarity Mapping List*.

Step 5: Return L_j.

EXAMPLE 4.4:

Continuing from Example 4.3, the BWI_N of a new query R_N, which is {StepID=PS_1, ToolID=AWOX13, Yield=99.1}, is < ei_1^1 =00000 ei_1^2 =00000 ei_2^1 =10000 ei_3^1 =1000 ei_3^2 =0010 ei_4^1 =00001 ei_4^2 =00100>. Also assume that weight W_2, W_3 and W_4 are set to 0.4, 0.4 and 0.2, respectively. Each BWI_j in T_{BWI} in Table 4.1 is processed as follows.

- For BWI_1, BWI_2 and BWI_3, all the relevant degrees rdi_1, rdi_2 and rdi_3 between BWI_1, BWI_1, BWI_1 and BWI_N are found as <00000 00000 10000 1000 0000 00000 00000> by the *Encapsulated BWI Search-relevant-records algorithm*. Since more than one bit in rdi_1 is "1", Records 1, 2 and 3 are possible relevant records. According to the Definition 4.2, the $eMask_2$ = <00000 00000 11111 0000 0000 00000 00000> and $eMask_3$=<00000 00000 00000 1111 1111 00000 00000>. Since the result of AND($eMask_2$, rdi_1) = <00000 00000 10000 0000 0000 00000 00000> is equal to the result of AND($eMask_2$, BWI_N) = <00000 00000 10000 0000 0000 00000 00000> and the result of AND($eMask_3$, rdi_1) = <00000 00000 00000 1000 0000 00000 00000> is not equal to the result of AND($eMask_3$, BWI_N) = <00000 00000 00000 1000 1000 00000 00000>, the similarities of Records 1, 2 and 3 are found as 0.4 via ALGORITHM 4.7. Record 1, 2, 3 are then the relevant records.

- For BWI_4: The relevant degree rdi_4 between BWI_4 and BWI_N is found as <00000 00000 10000 1000 0010 00001 00100> by the *Encapsulated BWI Search-relevant-records algorithm*. Since more than one bit in rdi_1 is "1", Record 4 is a possible relevant record. According to the Definition 4.2, the $eMask_2$ = <00000 00000 11111 0000 0000 00000 00000>, $eMask_3$=<00000 00000 00000 1111 1111 00000 00000> and $eMask_3$=<00000 00000 00000 1111 1111 11111 11111>. Since the results of AND($eMask_2$, rdi_4) is equal to AND($eMask_2$, BWI_N), AND($eMask_3$, rdi_4) is equal to AND($eMask_3$, BWI_N) and AND($eMask_4$, rdi_4) is equal to AND($eMask_4$, BWI_N), the similarity of Record 4 is found as 1 via ALGORITHM 4.7. Record 4 is then a relevant record.

After the relevant records are sorted in decreasing order of similarities, the results are shown is Table 4.2.

Table 4.2: Five relevant records and their similarities

Relevant Record	**Record 4**	**Record 1**	**Record 2**	**Record 3**	**Record 5**
Similarity	**1**	**0.4**	**0.4**	**0.4**	**0.4**

4.1.4 Analysis and Experiments of Encapsulated BWI Method

As we can see, the major different between *Similarity-computing algorithm* (**Algorithm 3.6**) of *Simple BWI method* and *Encapsulated BWI similarity-computing algorithm* (**Algorithm 4.7**) of *Encapsulated BWI method* is in Step 2. In *Simple BWI method*, one 'AND' bit-wise operation and a bit-to-integer operation are used. However, two 'AND' bit-wise operations and a bit-comparing operation are used. In general, the bit-wise and bit-to-integer operations are quite the efficiency operation, however, bit-comparing operations are not and thus highly depends on the length of bit string. The storage saving and computation time of *Encapsulated BWI method* are tradeoff. The more encapsulated level used, the more storage saving, however, the more computing time needing.

4.2 Compacted Bit-wise Indexing Method

In the Section, the *Encapsulated bit-wise indexing method* is introduced, including the definitions and algorithms of indexing and matching phases in Encapsulated Bit-wise Indexing Method are proposed.

4.2.1 General Assumptions and Notations for Compacted BWI Method

In Section 4.1, we propose the *Encapsulated bit-wise indexing method* to a data warehouse and it can largely reduce the width of the matrix of bit-wise indexes. However, the total number of bit-wise index string that needs to be compared via "AND" bit-wise operation are still required. In order to accelerate the processing time of OLAP queries, we propose a more sophisticated indexing model, called *Compacted bit-wise indexing method*.

As we know, the attributes are the base information of all OLAP queries and the concept hierarchy of each attribute is beneficial for roll-up and drill-down operations of the data warehouse. In the *Compacted bit-wise indexing method*, the importance of attributes, including attribute and concept hierarchy, is evaluated via encapsulated level of *Encapsulated BWI method*. Using this method, the bit strings of attributes are partitioned into two levels. For all attribute, and the encapsulated level is smaller than compact level cl_i are kept in the first level bit-wise indexes matrix, called *Main Matrix* T_{BWI}^{M}, and each BWI_i in T_{BWI}^{M} are linked to a bit-wise indexes matrices, called *Drill-Packet Matrix* $T_{BWI}^{DP^i}$, to keep the remain encapsulated levels. According to the BWI indexing structure, the processing time of OLAP queries can be hugely reduced since most irrelevant record will be filtered out in the matching procedure using T_{BWI}^{M}.

NOTATION 4.2:

cl_i = the compacted level of of attribute a_j.

BWI_i^M = the bit-wise indexing for Main Matrix T_{BWI}^{M}.

BWI_i^D = the bit-wise indexing for Drill-Packet Matrix $T_{BWI}^{DP^k}$.

T_{BWI}^{M} = the Main Matrix of bit-wise indexes matrix of T

$T_{BWI}^{DP^i}$ = the Drill-Packet Matrix $T_{BWI}^{DP^i}$ bit-wise indexes matrix of BWI_i of T_{BWI}^{M}

rdm_i = the result vector of matching BWI_N^M and BWI_i^M, where $1 \le i \le |T_{BWI}^{M}|$.

rdd_i = the result vector of matching BWI_N^D and BWI_i^D. where $1 \le i \le r$

We propose a *Compacted bit-wise indexing method* on data warehouse to achieve the goals of saving storage and accelerating user query procedure. This method includes two phases. One is creating indexes phase, and the other is querying phase. The indexing phase transforms the contents of table into a *Main Matrix* T_{BWI}^{M} and $|T_{BWI}^{M}|$ *Drill-down Matrix* $T_{BWI}^{DP^i}$, and the query phase is retrieving records to answer the query statements as soon as possible.

4.2.2 The Indexing Phase of Compacted BWI Method

The indexing phase includes *Compacted BWI attributes index creating Algorithm* and *Compacted BWI matrix of bit-wise indexes creating Algorithm*. In this method, the *Encapsulated level calculating Algorithm* (Algorithm 4.2.1) is still used to calculate an encapsulated level of each attribute for creating the corresponding bit-wise indexes, the *Compacted BWI Bit-wise indexes creating Algorithm* creates corresponding BWI index of matrix of two-level bit-wise indexes. The *Compacted BWI Matrix of bit-wise indexes creating Algorithm* creates *Main Matrix* T_{BWI}^{M} and $|T_{BWI}^{M}|$ *Drill-down Matrix* $T_{BWI}^{DP^i}$ of Table T in the data warehouse. These algorithms and examples are shown as follows.

Algorithm 4.8 - *Compacted BWI bit-wise indexes creating Algorithm* :

Input: A record R_i.

Output: Two bit-wise vectors BWI_i^M and BWI_i^D of R_i.

Step 1: Create two bit-wise vectors, including BWI_i^M and BWI_i^D, of length 0.

Step 2: Repeat the following sub-steps for each attribute A_j until all attributes are processed.

Step 2.1: If Type(A_j) ≠ S and f_i≠∅, go to Step 2.2, else let m=n if $V_j(i)$=V_{jn}. Repeat the following sub-steps for each encapsulated level el_k until all encapsulated levels are processed

Step 2.1.1: Let $B' = b_1 b_2 \ldots b_{ei_j^k}$ to a all-zero string with length ei_j^k

Step 2.1.2: If $k \neq el_k$, go to Step 2.1.3, else if the m=0, set $b_{ei_j^k}$=1 and set b_m=1 otherwise, go to Step 2.1.5.

Step 2.1.3: Let $o = \lfloor m / \prod_{p=k+1}^{el_j} ei_i^p \rfloor$, if o=$ei_j^k$, set b_o=1 and set b_{o+1}=1 otherwise.

Step 2.1.4: Set $m=m-(o \times \prod_{p=k+1}^{el_j} ei_i^p)$

Step 2.1.5: If $k \leq cl_i$, concatenate the bit strings BWI_i^M and B' to BWI_i^M and concatenate the bit strings BWI_i^D and B' to BWI_i^D otherwise.

Step 2.2: If Type(A_j) ≠ S and $f_i \neq \varnothing$, for each B_j^k, do the following sub-steps

Step 2.2.2: Set b_{jl}=1 if $f_i^k(V_i(k))=l$ and b_{jl}=0 otherwise.

Step 2.2.3: If $k \leq cl_i$, concatenate the bit strings BWI_i^M and B_j^k to BWI_i^M and concatenate the bit strings BWI_i^D and B_j^k to BWI_i^D otherwise.

Step 3: Return the vectors BWI_i^M and BWI_i^D.

Algorithm 4.9 - *Compacted BWI Matrix of bit-wise indexes creating Algorithm* :

Input: Table T of the data warehouse.

Output: The T_{BWI}^M and $|T_{BWI}^M|$ $T_{BWI}^{DP^i}$ of the Table T in data warehouse.

Step 1: Create an empty bit-wise indexes matrix T_{BWI}^{M} for table *T*.

Step 2: Call *Encapsulated level calculating Algorithm – Square Root* (**Algorithm 4.1**) to get the corresponding *el*s and *ei*s.

Step 3: Repeat the following sub-steps for each record R_i until all records are processed.

Step 3.1: Use the *Compacted BWI bit-wise index creation algorithm* (**Algorithm 4.8**) to get the indexes BWI_i^M and BWI_i^D of R_i.

Step 3.2: Set *k*=1 and do the following sub-steps.

Step 3.2.1: If $BWI_k^M = \varnothing$, Create an empty Drill-Packet Matrix $T_{BWI}^{DP^k}$, set $BWI_k^M = BWI_i^M$, add BWI_i^D into $T_{BWI}^{DP^k}$, and go to Step 3.

Step 3.2.1: If $BWI_k^M = BWI_i^M$, add BWI_i^D into $T_{BWI}^{DP^k}$ and go to Step 3.

Step 4: T_{BWI}^{M} and $|T_{BWI}^{M}|$ $T_{BWI}^{DP^i}$.

After bit-wise index matrixes, including T_{BWI}^{M} and all related $T_{BWI}^{DP^i}$, are built, bit-wise operations can easily be used to retrieve desired record for the new coming queries.

EXAMPLE 4.5:

Continuing Example 4.3, assume that the compacted level cl_i of all attributes are set to 1. According to the Algorithm 4.9, the Main Matrix T_{BWI}^{M} for the records in

Figure 4.2 is shown in Table 4.3 and the corresponding Drill-Packet Matrixes are shown in Table 4.4 to 4.17.

Table 4.3: The T_{BWI}^{M} of records in Figure 4.2

BWI	LotID	StepID	ToolID	Yield
BWI_1^M	10000	10000	1000	01000
BWI_2^M	10000	10000	1000	00001
BWI_3^M	01000	01000	0100	01000
BWI_4^M	01000	01000	0100	00100
BWI_5^M	01000	00100	0100	00010
BWI_6^M	00100	00100	0100	01000
BWI_7^M	00100	00100	0010	01000
BWI_8^M	00100	00100	0010	00100
BWI_9^M	00010	00010	0010	01000
BWI_{10}^M	00010	00010	0010	00100
BWI_{11}^M	00010	00010	0001	00100
BWI_{12}^M	00001	00010	0001	00100
BWI_{13}^M	00001	00001	0001	00100
BWI_{14}^M	00001	00001	0001	00010

Table 4.4: The $T_{BWI}^{DP^1}$ of records with cl_i=1 in of T_{BWI}^{M}

BWI	LotID	ToolID	Yield
BWI_1^D	10000	1000	10000
BWI_2^D	01000	1000	10000
BWI_3^D	00100	0100	10000

Table 4.5: The $T_{BWI}^{DP^2}$ of records with cl_i=1 in of T_{BWI}^{M}

BWI	LotID	ToolID	Yield

BWI_4^D	00010	0010	00100
BWI_5^D	00010	0001	00010

Table 4.6: The $T_{BWI}^{DP^3}$ of records with cl_i=1 in of T_{BWI}^{M}

BWI	LotID	ToolID	Yield
BWI_6^D	10000	1000	00100

Table 4.7: The $T_{BWI}^{DP^4}$ of records with cl_i=1 in of T_{BWI}^{M}

BWI	LotID	ToolID	Yield
BWI_7^D	01000	1000	01000
BWI_8^D	00100	1000	00010
BWI_9^D	00010	1000	00001

Table 4.8: The $T_{BWI}^{DP^5}$ of records with cl_i=1 in of T_{BWI}^{M}

BWI	LotID	ToolID	Yield
BWI_{10}^D	00001	0100	10000

Table 4.9: The $T_{BWI}^{DP^6}$ of records with cl_i=1 in of T_{BWI}^{M}

BWI	LotID	ToolID	Yield
BWI_{11}^D	10000	0010	10000
BWI_{12}^D	01000	0001	10000

Table 4.10: The $T_{BWI}^{DP^7}$ of records with cl_i=1 in of T_{BWI}^{M}

BWI	LotID	ToolID	Yield
BWI_{13}^{D}	00010	1000	00100

Table 4.11: The $T_{BWI}^{DP^8}$ of records with cl_i=1 in of T_{BWI}^{M}

BWI	LotID	ToolID	Yield
BWI_{14}^{D}	00010	0010	01000
BWI_{15}^{D}	00001	0010	00001

Table 4.12: The $T_{BWI}^{DP^9}$ of records with cl_i=1 in of T_{BWI}^{M}

BWI	LotID	ToolID	Yield
BWI_{16}^{D}	10000	0001	00010

Table 4.13: The $T_{BWI}^{DP^{10}}$ of records with cl_i=1 in of T_{BWI}^{M}

BWI	LotID	ToolID	Yield
BWI_{17}^{D}	01000	0001	01000
BWI_{18}^{D}	00100	0001	00010

Table 4.14: The $T_{BWI}^{DP^{11}}$ of records with cl_i=1 in of T_{BWI}^{M}

BWI	LotID	ToolID	Yield
BWI_{19}^{D}	00010	1000	01000

BWI_{20}^{D}	00001	1000	10000

Table 4.15: The $T_{BWI}^{DP^{12}}$ of records with cl_i=1 in of T_{BWI}^{M}

BWI	LotID	ToolID	Yield
BWI_{21}^{D}	10000	0100	00001

Table 4.16: The $T_{BWI}^{DP^{13}}$ of records with cl_i=1 in of T_{BWI}^{M}

BWI	LotID	ToolID	Yield
BWI_{22}^{D}	01000	0010	00010

Table 4.17: The $T_{BWI}^{DP^{14}}$ of records with cl_i=1 in of T_{BWI}^{M}

BWI	LotID	ToolID	Yield
BWI_{23}^{D}	00100	0010	01000

4.2.3 The Matching Phase of Compacted BWI Method

Calculating the similarities between a query and saved records is a time-consuming task. A two-phase matching approach, called the *Compacted BWI Similar-records-seeking algorithm*, is thus proposed here to reduce the matching time. It includes the *Compacted BWI relevant-records-retrieving* phase and the *Compacted BWI similarity-computing* phase. In the first phase, all irrelevant records are filtered out to avoid calculation of their similarities. The time of calculating the similarities of useful saved records can then be decreased. The similarities of the query with remaining saved records are then computed efficiently in the *similarity-computing* phase. The algorithm is described as follows.

Algorithm 4.10 - ***Compacted BWI Similar-records-seeking algorithm*** **:**

```
Input:    A bit-wise index matrixes $T^{M}_{BWI}$ , $|T^{M}_{BWI}|$ $T^{DP^{i}}_{BWI}$ , Table $T$ and a new query $R_N$.
Output:   A set of similar record $Rc$ with their similarity degrees with $R_N$.
Step 1:   Use the Compacted BWI bit-wise index creation algorithm (Algorithm 4.8)
          to get the indexes $BWI^{M}_{N}$ and $BWI^{D}_{N}$ and of the new query $R_N$ according to
          the condition part of the query.
Step 2:   Initialize the counter $j$ to 1 and $Rc$ to an empty set.
Step 3:   For each $BWI^{M}_{j}$ in $T^{M}_{BWI}$ , do the following sub-steps ($1<j\leq|T^{M}_{BWI}|$):
    Step 3.1:   Call the Compacted BWI Search-relevant-records-main-matrix
                algorithm (Algorithm 4.11) to compute the relevance degree $rdm_j$
                between $BWI^{M}_{N}$ and $BWI^{M}_{j}$ .
    Step 3.2:   If all bits in $rdm_j$ are 0, go to Step 3.5.
    Step 3.3:   For each $BWI^{D}_{k}$ in $T^{DP^{j}}_{BWI}$ , do the following sub-steps:
        Step 3.3.1:   Call the Compacted BWI search-relevant-records-drill-packet
```

```
            algorithm (Algorithm 4.12) to compute the matrix of relevance
            degree $rdd_k$ between $BWI_N^D$ and $BWI_k^D$.
    Step 3.3.2: Call the Compacted BWI Concatenate-rdi-result algorithm
            (Algorithm 4.13) to concatenate the bit strings $rdm_j$ and $rdd_k$ to
            $rdi_k$.
    Step 3.3.3: Call the Compacted BWI similarity-computing algorithm
            (Algorithm 4.15) to compute the similarity $sim_j$ between $R_N$ and
            $R_k$ using $rdi_k$.
    Step 3.3.4: If $sim_k \neq 0$, add record $R_k$ with its similarity $sim_k$ to $Rc$.
  Step 3.5: Add 1 to $j$.
Step 4: Sort the results in $Rc$ in descending order of their similarities.
Step 5: Output $Rc$.
```

Even the encoding procedure of BWI index in Encapsulated BWI method is different than the Simple one, it still can easily be found by using the ‘AND’ bit-wise operation to compare the two bit vectors. The following *Compacted BWI Search-relevant-records-main-matrix algorithm*, is thus proposed to achieve this purpose.

Algorithm 4.11 - *Compacted BWI Search-relevant-records-main-matrix algorithm* :

Input: The bit-wise indexing vector BWI_N^M of a new query R_N and the index BWI_j^M in T_{BWI}^M.

Output: The relevant degree *rdm*$_j$ between BWI_N^M and BWI_j^M.

Step 1: Use the 'AND' bit-wise operation on BWI_N^M and BWI_j^M and store the result as *rdm*$_j$, which is also a bit string.

Step 2: Return *rdm*$_j$.

Since the 'AND' bit-wise operation is fast, the *Compacted BWI Search-relevant-records-main-matrix algorithm* selects relevant saved records quickly. If *rdm*$_i$ is zero, then the saved records in the $T_{BWI}^{DP^i}$ are thought of as irrelevant and will be filtered out. Since the properties of Compacted BWI mode, if *rdm* has some '1' bits, it means that some saved records indexed in the $T_{BWI}^{DP^i}$ are relevant. However, the similarities between query and these records should be calculated based on the matching result of BWI_N^D and all contains indexes in $T_{BWI}^{DP^i}$. The following *Compacted BWI Search-relevant-records-main-matrix-drill-packet algorithm* and *Compacted BWI Concatenate-rdi-result algorithm* are thus proposed to achieve this purpose.

Algorithm 4.12 - *Compacted BWI search-relevant-records-drill-packet algorithm* :

Input: The bit-wise indexing vector BWI_N^D of a new query R_N and the index BWI_k^D in $T_{BWI}^{DP^j}$.

Output: The relevant degree *rdd*$_k$ between BWI_N^D and BWI_k^D.

Step 1: Use the 'AND' bit-wise operation on BWI_N^D and BWI_k^D and store the result as *rdd*$_k$, which is also a bit string.

Step 2: Return *rdd*$_k$.

Algorithm 4.13 - *Compacted BWI Concatenate-rdi-result algorithm* :

Input: The relevant degree bit strings *rdm*$_j$ and *rdd*$_k$.

Output: The relevant degree *rdi*$_k$.

Step 1: Initialize the counter *m*,*n* to 1 and *rdi*$_k$ to an empty bit string.

Step 2: For *m*≤*r*, do the following sub-steps:

Step 2.1: Add the bits between position $\sum_{n=1}^{m-1}\sum_{l=1}^{cl_i} ei_n^l + 1$ to $\sum_{n=1}^{m}\sum_{l=1}^{cl_i} ei_n^l$ of *rdm*$_j$ to *rdi*$_k$.

Step 2.2: Add the bits between position $\sum_{n=1}^{m-1}\sum_{l=(cl_i+1)}^{el_i} ei_n^l + 1$ to $\sum_{n=1}^{m}\sum_{l=(cl_i+1)}^{el_i} ei_n^l$ of *rdd*$_j$ to *rdi*$_k$.

Step 2.3: Add 1 to *m*.

Step 3: Return rdi_k.

EXAMPLE 4.6:

Continuing from Example 4.5, assume that rdm_j = <00000 10000 1000 00000> and rdm_j = <00000 1000 00000>. For the first attribute, the sub-string of rdm_j at position 1 ($\sum_{n=1}^{1-1}\sum_{l=1}^{1} ei_n^l + 1 = 0 + 1 = 1$) to 5 ($\sum_{n=1}^{1}\sum_{l=1}^{1} ei_n^l = 5$) will be appended to rdi_k first. rdi_k is set to <00000>. The sub-string of rdd_j at position 1 ($\sum_{n=1}^{1-1}\sum_{l=(1+1)}^{2} ei_n^l + 1 = 0 + 1 = 1$) to 5 ($\sum_{n=1}^{1}\sum_{l=(1+1)}^{2} ei_n^l = 5$) will then be appended to rdi_k. rdi_k is then set to <00000 00000>. Finally, the rdi_k is set to <00000 00000 10000 1000 1000 00000 00000> after the *Compacted BWI Concatenate-rdi-result algorithm* is executed.

As mentioned above, a matching function based on a weighted sum of matched attributes is defined to calculate the similarity degrees. As the same with *Simple* and *Encapsulated BWI* methods, the *Compacted BWI Mask Vector* and the *Compacted Similarity Mapping List* are used in *Compacted BWI method* then be defined at Definition 4.4 and 4.5.

DEFINITION 4.4 - Compacted BWI Mask Vector :

A Encapsulated BWI bit-wise indexing mask vector *cMask* is a set of $cMask_k$, where $0 < k \le \sum_{i=1}^{r} el_i$. Each *cMask*$_k$, denoting the mask vector of attribute A_k, is a concatenation of r bit strings as $cMask_k = S_1 S_2 \ldots S_{\sum_{i=1}^{r} el_i}$, where S_i = <1> for $\sum_{i=1}^{k-1} el_i \le i \le \sum_{i=1}^{k} el_i$ and S_i = <0> otherwise.

DEFINITION 4.5 - Compacted BWI Similarity Mapping List :

Let L be an *Compacted BWI Similarity Mapping List* and L_i be an element in L with an index value i, which is determined from the attributes matched, $1 \le i \le (2^{\sum_{i=1}^{r} el_i} - 1)$. Let i be represented as a binary code $b_{i1} b_{i2} \ldots b_{i\sum_{i=1}^{r} el_i}$. The value of L_i is thus

$$\frac{\sum_{j=1}^{r} \left(\prod_{k=(\sum_{l=1}^{j-1} el_l)+1}^{(\sum_{l=1}^{j+1} el_l)} b_{ik} \right) \times W_j}{\sum_{j=1}^{r} W_j}.$$

Algorithm 4.14 - *Compacted BWI Similarity-mapping-list creation algorithm*:

Input: Weights of attributes $W_1, W_2, \ldots, W_r$ of R.

Output: A similarity mapping list L.

Step 1: Initialize the counter i to 1 and the list L to be empty.

Step 2: For each i, $1 \le i \le 2^{\sum_{i=1}^{r} el_i} - 1$, do the following sub-steps:

Step 2.1: Encode i into a binary string $<b_{i1} b_{i2} \ldots b_{i\sum_{i=1}^{r} el_i}>$.

Step 2.2: Calculate the similarity degree L_i by the formula in Definition 4.5.

```
    Step 2.3:  Put $L_i$ into the list $L$ with index $i$.
Step 3:  Return $L$.
```

After the *Similarity Mapping List* has been built, the similarity of each saved record and a new query can be quickly found by the following algorithm.

Algorithm 4.15 - ***Compacted BWI Similarity-computing algorithm*****:**

```
Input:   The relevant degree $rdi_j$ of record $R_j$ with a new query, the Compacted Mask
         Vector, and the Similarity Mapping List $L$.
Output:  The similarity of $R_j$ with a new record.
Step 1:  Initialize a zero binary string of length $r$.
Step 2:  For each $i$, $1 \le i \le \sum_{i=1}^{r} el_i$, set the $i$-th position in the string to 1 if
         AND($cMask_i$, $rdi_j$) = AND($cMask_i$, $BWI_N$).
Step 3:  Transform the binary string into an integer $j$.
Step 4:  Get $L_j$ from the Similarity Mapping List.
Step 5:  Return $L_j$.
```

EXAMPLE 4.7:

Continuing from Example 4.6, the BWI_N^M and BWI_N^D of a new query R_N, which is <StepID=PS_1, ToolID=AWOX13, Yield=99.1>, is $<ei_1^1$=00000 ei_2^1=10000 ei_3^1=1000 ei_4^1=00001 > and $<ei_1^2$=00000 ei_3^2=0010 ei_4^2=00100>. Also assume that weight W_2, W_3 and W_4 are set to 0.4, 0.4 and 0.2 correspondingly. Each BWI_j^M in T_{BWI} in Table 4.3 is processed as follows.

- For BWI_1^M, the relevant degree rdm_1 = <00000 10000 1000 00000> since:

	<00000 10000 1000 00001>	(BWI_N^M)
AND	<00000 10000 1000 00000>	(BWI_1^M)
	<00000 10000 1000 00000>	(rdm_1)

Since more than one bit in rdm_1 is "1", all BWI_i^D in Drill-Packet Matrix $T_{BWI}^{DP^1}$ are retrieved to further investigation. There are three BWI_i^D in $T_{BWI}^{DP^1}$, including BWI_1^D, BWI_2^D and BWI_3^D, the relevant degree rdd_1 = <00000 0000 00000> since:

	<00000 0010 00100>	(BWI_N^D)
AND	<00000 1000 10000>	(BWI_1^D)
	<00000 0000 00000>	(rdd_1)

The relevant degree rdd_2 = <00000 0000 00000> since:

	<00000 0010 00100>	(BWI_N^D)
AND	<00000 1000 10000>	(BWI_2^D)
	<00000 0000 10000>	(rdd_2)

The relevant degree rdd_3 = <00000 0000 00000> since:

	<00000 0010 00100>	(BWI_N^D)
AND	<00000 0100 10000>	(BWI_3^D)
	<00000 0000 00000>	(rdd_3)

After the *Compacted BWI Concatenate-rdi-result algorithm* executed, the rdi_1, rdi_2 and rdi_3 are thus generated as following:

rdi_1= <00000 00000 10000 1000 0000 00000 00000>

rdi_2= <00000 00000 10000 1000 0000 00000 00000>

rdi_3= <00000 00000 10000 1000 0000 00000 00000>

According to the Definition 5.1, the $cMask_2$ = <00000 00000 11111 0000 0000 00000 00000> and $cMask_3$=<00000 00000 00000 1111 1111 00000 00000>. Since the result of AND($cMask_2$, rdi_1) = <00000 00000 10000 0000 0000 00000 00000> is equal to the result of AND($cMask_2$, BWI_N) = <00000 00000 10000 0000 0000 00000 00000> and the result of AND($cMask_3$, rdi_1) = <00000 00000 00000 1000 0000 00000 00000> is not equal to the result of AND($cMask_3$, BWI_N) = <00000 00000 00000 1000 1000 00000 00000>, the similarities of record 1, 2 and 3 are found as 0.4. Record 1, 2, 3 are then the relevant records.

- For BWI_2^M, the relevant degree rdm_2 = <00000 10000 1000 00001> since:

	<00000 10000 1000 00001>	(BWI_N^M)
AND	<00000 10000 1000 00001>	(BWI_2^M)

	<00000 10000 1000 00001>	(rdm_2)

Since more than one bit in rdm_2 is "1", all BWI_i^D in Drill-Packet Matrix $T_{BWI}^{DP^2}$ are retrieved to further investigation. There is only one BWI_4^D in $T_{BWI}^{DP^1}$, ,the relevant degree rdd_1 = <00000 0000 00000> since:

	<00000 0010 00100>	(BWI_N^D)
AND	<00000 0010 00100>	(BWI_4^D)
	<00000 0010 00100>	(rdd_4)

After the *Compacted BWI Concatenate-rdi-result algorithm* executed, the rdi_4, is thus generated as following:

rdi_4= <00000 00000 10000 1000 0010 00001 00100>

According to the Definition 5.1, the $cMask_2$ = <00000 00000 11111 0000 0000 00000 00000>, $cMask_3$=<00000 00000 00000 1111 1111 00000 00000> and $cMask_4$=<00000 00000 00000 0000 0000 11111 11111>. Since the results of AND($cMask_2$, rdi_4) is equal to AND($cMask_2$, BWI_N), AND($cMask_3$, rdi_4) is equal to AND($cMask_3$, BWI_N) and AND($cMask_4$, rdi_4) is equal to AND($cMask_4$, BWI_N), the similarity of Record 4 is found as 1. Record 4 is then a relevant record.

- For BWI_1^M , the relevant degree rdm_3 = <00000 10000 1000 00001> since:

	<00000 10000 1000 00001>	(BWI_N^M)
AND	<00000 10000 1000 00001>	(BWI_3^M)
	<00000 10000 1000 00001>	(rdm_3)

Since more than one bit in *rdm*$_3$ is "1", all BWI_i^D in Drill-Packet Matrix $T_{BWI}^{DP^3}$ are retrieved to further investigation. There only one BWI_5^D in $T_{BWI}^{DP^1}$, ,the relevant degree *rdd*$_5$ = <00000 0000 00000> since:

	<00000 0010 00100>	(BWI_N^D)
AND	<00000 1000 00010>	(BWI_5^D)
	<00000 0000 00000>	(*rdd*$_5$)

After the *Compacted BWI Concatenate-rdi-result algorithm* executed, the *rdi*$_5$ is thus generated as following:

rdi$_5$= <00000 00000 10000 1000 0000 00001 00000>

According to the Definition 5.1, the *cMask*$_2$ = <00000 00000 11111 0000 0000 00000 00000>, *cMask*$_3$=<00000 00000 00000 1111 1111 00000 00000> and *cMask*$_4$=<00000 00000 00000 0000 0000 11111 11111>. Since the result of AND(*cMask*$_2$, *rdi*$_1$) is equal to the result of AND(*cMask*$_2$, *BWI*$_N$), however, result of AND(*cMask*$_3$, *rdi*$_1$) is not equal to the result of AND(*cMask*$_3$, *BWI*$_N$), the similarities of record 5 are found as 0.4. Record 5 is then a relevant record.

- For the other BWI_i^M , the relevant degree *rdm*$_i$ are all equal to <00000 00000 0000 00000> , where 6≤ *i* ≤14, since no “1” bit in *rdm*, all other records are filtered out using the Main Matrix only.

After the relevant records are sorted in decreasing order of similarities, the results are shown is Table 4.18.

Table 4.18: Two relevant records and their similarities

Relevant Record	**Record 1**	**Record 2**	**Record 3**	**Record 4**	**Record 5**
Similarity	**0.4**	**0.4**	**0.4**	**1**	**0.4**

4.2.4 Analysis and Experiments of Compacted BWI Method

As we can see, the major different between *Encapsulated BWI Similar-records-seeking algorithm* (**Algorithm 4.4**) of *Encapsulated BWI method* and *Compacted BWI Similar-records-seeking algorithm* (**Algorithm 4.10**) of *Compacted BWI method* is in Step 3, the computation time analysis (worse case analysis) is shown below

In *Encapsulated BWI method*, the “AND” operations should be taken

$$\prod_{i=1}^{r}\prod_{j=1}^{el_i} ei_i^j \quad \text{times.}$$

In *Compacted BWI method*, the “AND” operations should be taken

$$\prod_{i=1}^{r}\prod_{j=1}^{cl_i} ei_i^j + \left(\prod_{i=1}^{r}\prod_{j=1}^{cl_i} ei_i^j \times \prod_{i=1}^{r}\prod_{j=cl_i+1}^{el_i} ei_i^j\right) = \prod_{i=1}^{r}\prod_{j=1}^{cl_i} ei_i^j \times \left(1+\prod_{i=1}^{r}\prod_{j=cl_i+1}^{el_i} ei_i^j\right) \quad \text{times.}$$

The number of extra “AND” operations is:

$$\prod_{i=1}^{r}\prod_{j=1}^{cl_i} ei_i^j \times (1+\prod_{i=1}^{r}\prod_{j=cl_i+1}^{el_i} ei_i^j) - \prod_{i=1}^{r}\prod_{j=1}^{el_i} ei_i^j = \prod_{i=1}^{r}\prod_{j=1}^{cl_i} ei_i^j \quad \text{times}$$

In the worst case analysis, the *Compacted BWI method* uses extra $\prod_{i=1}^{r}\prod_{j=1}^{cl_i} ei_i^j$ time "AND" operations than the *Encapsulated BWI method*. However, the *Encapsulated BWI method* should process extra $\prod_{i=1}^{r}\prod_{j=1}^{cl_i} ei_i^j \times \prod_{i=1}^{r}\prod_{j=cl_i+1}^{el_i} ei_i^j \times \sum_{i=1}^{r}\sum_{j=1}^{cl_i} ei_i^j$ bits than the *Compacted BWI method*.

In *Encapsulated BWI method*, the total bit should be processed

$$\prod_{i=1}^{r}\prod_{j=1}^{el_i} ei_i^j \times \sum_{i=1}^{r}\sum_{j=1}^{el_i} ei_i^j \quad \text{bits.}$$

In *Compacted BWI method*, the total bit should be processed

$$\prod_{i=1}^{r}\prod_{j=1}^{cl_i} ei_i^j \times (\sum_{i=1}^{r}\sum_{j=1}^{cl_i} ei_i^j) + \prod_{i=1}^{r}\prod_{j=1}^{cl_i} ei_i^j \times \prod_{i=1}^{r}\prod_{j=cl_i+1}^{el_i} ei_i^j \times (\sum_{i=1}^{r}\sum_{j=cl_i+1}^{el_i} ei_i^j) \quad \text{bits.}$$

The saving bits are:

$$\prod_{i=1}^{r}\prod_{j=1}^{el_i} ei_i^j \times (\sum_{i=1}^{r}\sum_{j=1}^{el_i} ei_i^j) - [\prod_{i=1}^{r}\prod_{j=1}^{cl_i} ei_i^j \times (\sum_{i=1}^{r}\sum_{j=1}^{cl_i} ei_i^j)] + (\prod_{i=1}^{r}\prod_{j=1}^{cl_i} ei_i^j) \times [\prod_{i=1}^{r}\prod_{j=cl_i+1}^{el_i} ei_i^j \times (\sum_{i=1}^{r}\sum_{j=cl_i+1}^{el_i} ei_i^j)]$$

$$= (|R| \times \sum_{i=1}^{r}\sum_{j=1}^{el_i} ei_i^j) - [\prod_{i=1}^{r}\prod_{j=1}^{cl_i} ei_i^j \times (\sum_{i=1}^{r}\sum_{j=1}^{cl_i} ei_i^j) + (|R| \times \sum_{i=1}^{r}\sum_{j=cl_i+1}^{el_i} ei_i^j)]$$

$$= (|R| - \prod_{i=1}^{r}\prod_{j=1}^{cl_i} ei_i^j) \times (\sum_{i=1}^{r}\sum_{j=1}^{cl_i} ei_i^j)$$

$$= (\prod_{i=1}^{r}\prod_{j=1}^{el_i} ei_i^j - \prod_{i=1}^{r}\prod_{j=1}^{cl_i} ei_i^j) \times (\sum_{i=1}^{r}\sum_{j=1}^{cl_i} ei_i^j)$$

$$= (\prod_{i=1}^{r}\prod_{j=1}^{cl_i} ei_i^j \times \prod_{i=1}^{r}\prod_{j=cl_i+1}^{el_i} ei_i^j - \prod_{i=1}^{r}\prod_{j=1}^{cl_i} ei_i^j) \times (\sum_{i=1}^{r}\sum_{j=1}^{cl_i} ei_i^j)$$

$$= (\prod_{i=1}^{r}\prod_{j=1}^{cl_i} ei_i^j \times (\prod_{i=1}^{r}\prod_{j=cl_i+1}^{el_i} ei_i^j - 1)) \times (\sum_{i=1}^{r}\sum_{j=1}^{cl_i} ei_i^j)$$

When the record size (|R|) of T is quite large, the *Compacted BWI method* can be applied since the disk storage will be largely reduced. Once the record size is smaller then $(\prod_{i=1}^{r}\prod_{j=cl_i+1}^{el_i} ei_i^{j} - 1)\times(\sum_{i=1}^{r}\sum_{j=1}^{cl_i} ei_i^{j})$, the *Encapsulated BWI method* should be used since the extra processing time will be used by *Compacted BWI method*

Chapter 5
Using BWI indexing in an Intelligent Manufacturing Defect Detection Method for the Time Issue

In this chapter, an implementation that consisted of a reinforcement-learning defect detection root-cause learning system for the time aspect in manufacturing domains is introduced. This implementation employed the Sample Bit-Wise Indexing Method to encode the defect status of manufacturing products and hence accelerate data preprocessing. Additionally, a bit-based Genetic Algorithm is used to learn suitable weights for each computed signature, since the chromosome and the corresponding GA operators are appropriate for the bit operations of BWI indexing method.

5.1 Problem Description

In recent years, the problem of detecting defects in the workshop has become increasingly important for manufacturers. In order to raise the quality of products, the root causes of low-quality situations must be found as soon as possible. Thus, process control, statistical analysis, and cause-methodology-analysis techniques have all been widely applied in addressing the problem [10][18][22][27][53][62][70]. However, it is very difficult to identify the root causes of defects due to a wide variety in the types of causes of defects. For example, in the semiconductor manufacturing industry there are many causes of low yields, among them: machine failures, improper operation, improper parameters, manufacturing time problems, and scheduling and material problems. Many studies have been devoted to investigating these issues. The advent of advanced manufacturing technologies has led to overlong queues and increased manufacturing times in workshops that may cause oxidation problems, which are becoming more critical, but the diagnosis of such problems is usually very difficult and time-consuming. In this chapter, we will proposed a manufacturing defect detection problem, time aspect, for manufacturing domains (*MDDP-t*) is formally modeled and defined. In this section, the manufacturing defect detection problem, time aspect, for manufacturing domains (*MDDP-t*) is formally modeled and defined. A root-cause evaluation function (*RCEF*), which is a linear combination of three probing functions defined independently according to the experiences of domain experts, is proposed to evaluate whether a specific machine is the root cause of a time problem. Determining the weights for these probing functions is considered a separate issue here, and a genetic algorithm (GA) with encoding and GA operations suitable for *MDDP-t* weight-learning problems is given to find appropriate weights for the probing functions. Several instances of *MDDP-t* with known root causes, some provided by the Taiwan Semiconductor Manufacturing Company [TSMC]), are given as training examples. Experimental results show the proposed approaches can ensure efficiency and accuracy.

Many technologies or methods are employed to identify the causative factors of manufacturing defects, including Statistical Process Control (SPC), Advanced Process Control (APC) [18][53], and Machine Learning (ML) approaches. However, the real problems are sometimes chaotic, little-understood, and may be caused by complex interactions among multiple factors. Therefore, root-cause sorting becomes a critical issue for all manufacturing enterprises, especially some high technology ones like semiconductor manufacturing corporations.

SPC and APC [10][53] are widely used in the semiconductor industry to monitor manufacturing behavior in workshops via motion and condition sensors. SPC monitors manufacturing by analyzing the statistical results of procedures, generating lists of meaningful results, and warning if the results are outside predefined control boundaries based on machine behaviors and expert experience. However, they sometimes issue warnings for good products (type-two error) and may not always warn of defective products (type-one error). APC, an advanced revision of SPC, not only monitors the statistical results of machines behaviors [18][53], but also takes predefined actions to adjust machine behaviors when machines become unstable. Although APC seems more advanced than SPC, the resulting action-selection problem raises a separate issue that must be resolved.

Certain intelligent methods with self-learning abilities are employed to provide fault analysis and suggest solutions. In [53], a combination of self-organizing neural networks and rule induction was used to identify critical poor-yield factors from normally collected wafer manufacturing data, and the corresponding behavior model thus learned to predict possible behaviors. A decision-tree approach used to locate the root cause of yield loss in integrated circuits was reported in [59]. The utility of decision trees for yield analysis lies in pointing to process steps that may not be captured by analyses of parametric data.

5.2 Problem Definition of *MDDP-t*

As mentioned above, we are concerned with the time aspects of detecting which machines make product defects. In this section, we first define various parameters used in this chapter, and then propose a formal definition of "Manufacturing Defect Detection Problem, time aspects" (*MDDP-t*). Generally, quality baselines must exist for all products in order to ensure good manufacturing procedures. Taking an example from semiconductor manufacturing, the quality baseline for 150-nanometer yields is usually set to 90% or above in a well-tuned manufacturing fab. When yields become unstable and drop below the quality baseline, product engineers (“lot owners” in semiconductor manufacturing fabs) investigate to find the major reason (called the “root cause”) for the low-yield situation. For example, a product engineer may collect data on all low and normal product yields and identify suspect factors, e.g., abnormal machine behaviors, in-line metrologies, processing and queuing times, which are the most likely root causes according to statistical- or data-analysis results. In this chapter, *MDDP-t* is considered a quadruple, including product manufacturing machine information (*PM*), product manufacturing time information (*PT*), product manufacturing yield information (*PY*), and quality baseline(y_θ). The Notation 5.1 is defined as following:

NOTATION 5.1:

M	the set of machines;
c_p	number of products;
c_m	number of machines;
c_s	number of machine clusters;
s^i	i-th machine cluster such that $s^i = \{m_{i,1}, m_{i,2}, \ldots, m_{i,\alpha(i)}\}$, where $1 \le i \le c_s$, and $\alpha(i)$ is the number of machines in s^i and $m_{i,j}$ is the j-th machine in s^i, $1 \le$

	$j \le \alpha(i)$;
p_i	product p_i, $1 \le i \le c_p$;
y_i	product quality p_i, $1 \le i \le c_p$;
y_θ	acceptable product quality baseline;
pm_i	product p_i manufacturing information vector $pm_i = < pm_i^1, pm_i^2, \ldots, pm_i^{c_s} >$, where p_i is processed by the pm_i^j-th machine in s^j and $1 \le i \le c_p$;
pt_i	target manufacturing time vector for product p_i $pt_i = < pt_i^1, pt_i^2, \ldots, pt_i^{c_s} >$, where pt_i^j is the processing time for machine pm_i^j and $1 \le i \le c_p$;
py_i	p_i product yield;
PM	manufacturing procedure for products in P, where PM is a $c_p \times c_s$ matrix and $PM_{i,j} = pm_i^j$;
PT	product manufacturing time, where PT is a $c_p \times c_s$ matrix and $PT_{i,j} = pt_i^j$;
PY	product quality yield, where PY is a column matrix and $PY_i = py_i$;
$MDDP\text{-}t$	a given quadruple manufacturing defect detection problem involving time, where $MDDP\text{-}t=(PM, PT, PY, y_\theta)$.

Table 5.1: An example of products passing through two machine clusters

	s^1	pt^1	s^2	pt^2	Y
p_1	$m_{1,1}$	10	$m_{2,1}$	23	0.85
p_2	$m_{1,1}$	10	$m_{2,1}$	23	0.86
p_3	$m_{1,1}$	11	$m_{2,2}$	23	0.80
p_4	$m_{1,2}$	13	$m_{2,3}$	60	0.60
p_5	$m_{1,2}$	12	$m_{2,3}$	25	0.90
p_6	$m_{1,2}$	12	$m_{2,3}$	66	0.60
p_7	$m_{1,3}$	10	$m_{2,3}$	27	0.83
p_8	$m_{1,3}$	11	$m_{2,3}$	25	0.65
p_9	$m_{1,3}$	11	$m_{2,2}$	25	0.88
p_{10}	$m_{1,3}$	10	$m_{2,2}$	23	0.85

EXAMPLE 5.1.

As shown in Table 5.1, there are 10 products in this example (c_p=10) and each product is processed by two machine clusters (c_s=2), where s^1={$m_{1,1}$, $m_{1,2}$, $m_{1,3}$}(α(1)=3) and s^2={$m_{2,1}$, $m_{2,2}$, $m_{2,3}$} (α(2)=3 and c_m=6). Each product p_i is processed by machine pm_i in target time pt_i. Assume that the given yield threshold y_θ is 0.7. According to the definitions given above, the manufacturing information vector pm_1 and corresponding manufacturing target time vector pt_1 are, respectively, <1,1,1,2,2,2,3,3,3,3> and <10,10,11,13,12,12,10,11,11,10>. Therefore, the manufacturing procedure, target time, and product yield matrixes are

$$PM = \begin{bmatrix} pm_1 \\ pm_2 \\ pm_3 \\ pm_4 \\ pm_5 \\ pm_6 \\ pm_7 \\ pm_8 \\ pm_9 \\ pm_{10} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \\ 1 & 2 \\ 2 & 3 \\ 2 & 3 \\ 2 & 3 \\ 3 & 3 \\ 3 & 3 \\ 3 & 2 \\ 3 & 2 \end{bmatrix}, \quad PT = \begin{bmatrix} pt_1 \\ pt_2 \\ pt_3 \\ pt_4 \\ pt_5 \\ pt_6 \\ pt_7 \\ pt_8 \\ pt_9 \\ pt_{10} \end{bmatrix} = \begin{bmatrix} 10 & 23 \\ 10 & 23 \\ 11 & 23 \\ 13 & 35 \\ 12 & 25 \\ 12 & 66 \\ 10 & 27 \\ 11 & 25 \\ 11 & 25 \\ 10 & 23 \end{bmatrix}, \text{ and } PY = \begin{bmatrix} py_1 \\ py_2 \\ py_3 \\ py_4 \\ py_5 \\ py_6 \\ py_7 \\ py_8 \\ py_9 \\ py_{10} \end{bmatrix} = \begin{bmatrix} 0.85 \\ 0.86 \\ 0.80 \\ 0.60 \\ 0.90 \\ 0.60 \\ 0.83 \\ 0.65 \\ 0.88 \\ 0.85 \end{bmatrix}.$$

Finally, the production for the *MDDP-t* instance in Table 7.1 is set to (*PM*, *PT*, *PY*, 0.7).

Three probing functions, including Individual Machine, Intra-cluster, and Machine Behavior, are proposed to find possible root causes for given *MDDP-t* instances. The three probing functions are described in detail below:

1. Individual-Machine probing function (f_1): This criterion considers individual machine behaviors in given datasets. If the low-product-yield percentage of one machine, especially one with an abnormal target time, is higher than that of other machines, it may be considered a root-cause candidate. For example, Figure 5.1, shows that machine $m_{1,2}$ produces low yields of products p_4 and p_6, 66%, obviously higher than that of machine $m_{1,1}$ with a low-yield percentage of 0%.

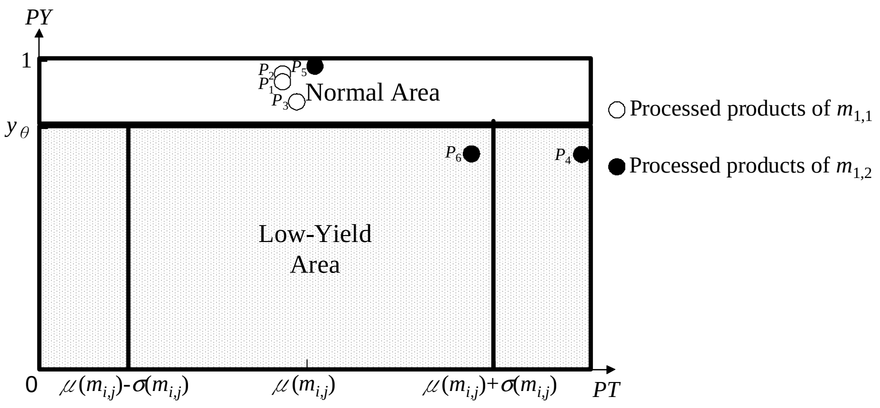

PY
1
y_θ
0
$\mu(m_{i,j})-\sigma(m_{i,j})$
$\mu(m_{i,j})$
$\mu(m_{i,j})+\sigma(m_{i,j})$
PT
P_1
P_2
P_3
P_4
P_5
P_6
Normal Area
Low-Yield Area
○ Processed products of $m_{1,1}$
● Processed products of $m_{1,2}$

Figure 5.1: Products processed by machines $m_{1,1}$ and $m_{1,2}$

Certain notation must be defined in order to calculate the parameters of this function:

NOTATION 5.2:

$mv_{i,j}$	the set of products processed by machine $m_{i,j}$;
$my_{i,j}$	the set of low-yield products processed by machine $m_{i,j}$;
$mty_{i,j}$	the set of low-yield products with abnormal target time processed by machine $m_{i,j}$.

The Individual-machine probing function for machine $m_{i,j}$ is the multiplication of the ratio of processed product ($\frac{|mv_{i,j}|}{n}$) by the ratio of low-yield product processed with abnormal target time ($\frac{|mty_{i,j}|}{|my_{i,j}|}$)$_j$. As mentioned above, a higher result from this function means a higher possibility of being a root cause.

Since applying conventional comparison and computation operators to generate $mv_{i,j}$, $my_{i,j}$, and, $mty_{i,j}$ may be time-consuming, we use the BWI indexing method to reduce the time required to compute this decision variable. The detailed notation and functions resulting from use of the BWI indexing method are defined as follows:

NOTATION 5.3:

$mv_{i,j}$	the machine-bit vector of machine $m_{i,j}$, where $mv_{i,j}$=< $b_1b_2b_3\ldots b_{c_p}$ >, $mv_{i,j}(k)$ is the k-bit (b_k) of $mv_{i,j}$, and $b_k = 1$ if $pm_k^i = j$ for $1 \le i \le c_s$, $1 \le j \le \alpha(i)$, and $1 \le k \le c_p$; otherwise, $b_k = 0$;
mv^{LY}	the machine-bit vector of low-yield products for the given *MDDP-t* instance, where mv^{LY}=< $b_1b_2b_3\ldots b_{c_p}$ > and $b_k = 1$ if $py_k<y_\theta$ for $1 \le k \le c_p$; otherwise, $b_k = 0$;
$mv_{i,j}^{OC}$	the abnormal target time machine-bit vector of machine $m_{i,j}$, where $mv_{i,j}$=< $b_1b_2b_3\ldots b_{c_p}$ > and $b_k = 1$ if $pt_k^i > \mu(m_{i,j})+\sigma(m_{i,j})$ or $pt_k^i < \mu(m_{i,j})-\sigma(m_{i,j})$; otherwise, $b_k = 0$;
$my_{i,j}$	the machine vector for low-yield products from machine $mv_{i,j}$, where $my_{i,j}$ =AND($mv_{i,j}, mv^{LY}$);

$mty_{i,j}$ the machine vector for outlier products of machine $mv_{i,j}$, where $mty_{i,j}$ $=\text{AND}(my_{i,j},\ mv_{i,j}^{OC})$;

$count\_one(x)$ 1-bit count in bit-vector x;

$count\_zero(x)$ 0-bit count in bit-vector x;

$\mu(m_{i,j})$ the average manufacturing time for machine $m_{i,j}$, $\dfrac{\sum_{k=1}^{c_p}\left(pt_k^i \times mv_{i,j}(k)\right)}{count\_one(mv_{i,j})}$;

$\sigma(m_{i,j})$ the standard deviation for machine $m_{i,j}$, manufacturing time

$$\frac{\left(\sum_{k=1}^{c_p}\sqrt{\left(pt_k^i-\mu(m_{i,j})\right)^2}\times mv_{i,j}(k)\right)}{count\_one(mv_{i,j})}.$$

Obviously, the time required to compute $mv_{i,j}$, $my_{i,j}$ and $mty_{i,j}$ is thus largely reduced since all comparison and computation operations use the bit-wise indexing method. The formulation of the Individual Machine probing function (f_1) is thus:

Individual Machine probing function $f_1(m_{i,j})$ for an *MDDP-t*

$$f_1(m_{i,j})=\frac{count\_one(mv_{i,j})}{n}\times\frac{count\_one(mty_{i,j})}{count\_one(my_{i,j})}\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots(5.1).$$

2. Intra-cluster probing function (f_2): The second criterion considers the slopes of machine behavior regression lines within machine clusters. Intra-cluster machine

behavior is represented as a regression line of data points on a two-dimensional plane where the *x* and *y* axes are, respectively, the target time and yield of each product processed by the machine. A higher absolute slope value for the regression line means higher time-issue sensitivity for the corresponding machine. In other words, it may be a root-cause candidate in the time-issue problem. As shown in Figures. 5.2(a) and 5.2(b), the absolute value of the machine-curve slope of $m_{i,j}$ is higher than that of $m_{i,k}$. Therefore, machine $m_{i,j}$ has a higher possibility of being a root-cause candidate. The following definitions and functions are needed to calculate the parameters of this function:

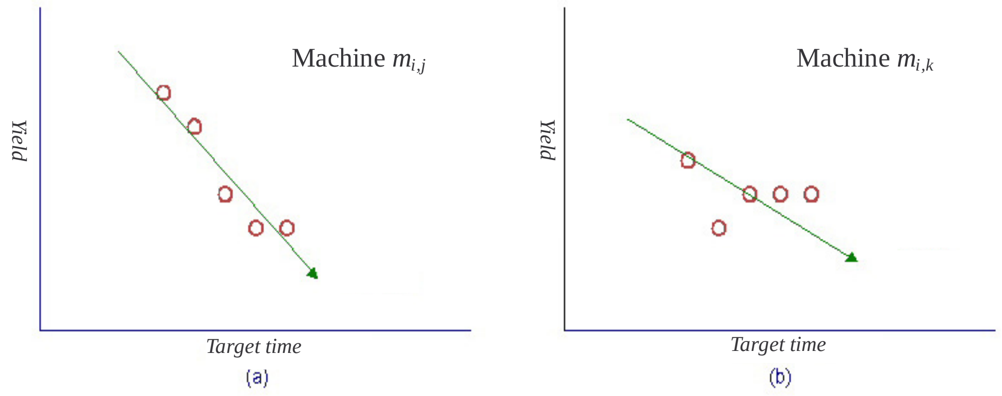


Figure 5.2: The regression lines for (a) $m_{i,j}$ and (b) $m_{i,k}$

Certain notation must be defined in order to calculate the parameters of this function:

$offset(x,i)$	i-th 1-bit offset (l. to r.) in bit-vector x;
$evs_{i,j}$	the set of data points for products processed by machine pm_i^j:

	$evs_{i,j}$={(x_1, y_1), (x_2, y_2), …, ($x_{count\_one(mv_{i,j})}, y_{count\_one(mv_{i,j})}$), } where
	x_k= $py_{offset(mv_{i,j},k)}$, y_k= $pt^{j}_{offset(mv_{i,j},k)}$ for $1 \le k \le count\_one(mv_{i,j})$;
regress($evs_{i,j}$)	the $evs_{i,j}$ regression line;
slope(*regress*($evs_{i,j}$))	the slope of *regress*($evs_{i,j}$).

For the example shown in Table 5.1, the bit operation is $mv_{1,1}(3)$=1, $count\_one(mv_{1,1})$ =3, $count\_zero(mv_{1,1})$ =7 and $offset(m_{1.1},3)$ =3, and we have the evaluation vector set for machine pm_1^1 , $evs_{1,1}$={<0.85, 10>, <0.86, 10>, <0.80, 11>}.

Intra-machine-center probing function $f_2(m_{i,j})$ for the *MDDP-t* problem:

$$f_2(m_{i,j}) = \left| slope(regress(evs_{i,j})) \right| \quad \text{……………………………………………………(5.2)}$$

3. Machine Behavior probing function (f_3): The third criterion considers similarities among machine behaviors in given datasets with respect to the time issue. The behavior of an arbitrary machine can be represented as a machine-behavior vector with $count\_one(mty_{i,j})$ and $count\_one(mv^{OC}_{i,j})$ the respective *x* and *y* axes. The sum of the degrees of included angle between the machine-behavior vector of a machine in a machine cluster and all the other machine-behavior vectors is calculated. The machine with the highest sum has the highest possibility of being

the root cause in that machine cluster. As shown in Figure 5.3, of the four machines in machine cluster s^i, the computed sum for machine $m_{i,4}$ is obviously much higher than the others. Thus, machine $m_{i,4}$ has higher possibility of being the root cause in this example. The following definitions and functions must be defined in order to calculate the parameters for this function.

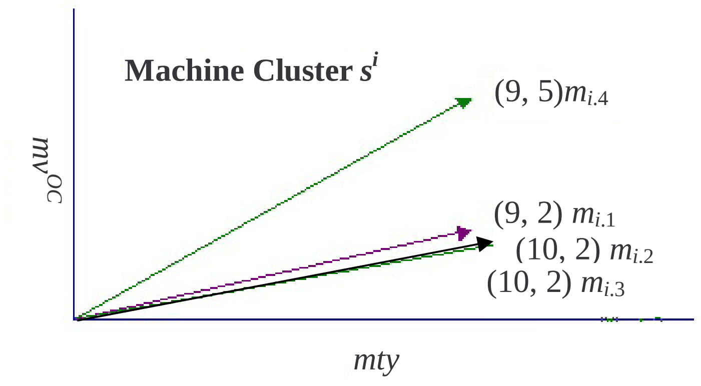


Figure 5.3: The machine-behavior vectors of machine cluster s^i

inner_product(x, y)	the inner product of machine-behavior vector (x, y);
$\theta(x,y)$	the included angle of machine-behavior vector (x, y), where $\theta(x,y)=\cos^{-1}\dfrac{inner\_product(x,y)}{\lvert x\rvert\cdot\lvert y\rvert}$;
mx_inc($m_{i,j}$, $m_{i,k}$)	the included angle between the machine-behavior vectors of machines $m_{i,j}$ and $m_{i,k}$, where *mx_inc*($m_{i,j}$, $m_{i,k}$)=

$$\theta((count\_one(mv_{i,j}^{OC}),count\_one(mty_{i,j})),(count\_one(mv_{i,k}^{OC}),count\_one(mty_{i,k}))).$$

Therefore, formulation of Machine Behavior probing function $f_3(m_{i,j})$ is as follows:

Machine-behavior probing function $f_3(m_{i,j})$ of *MDDP-t* is:

$$f_3(m_{i,j}) = \frac{\sum_{k=1}^{\alpha(i)} mx\_inc(m_{i,j}, m_{i,k})}{\alpha(i)-1} \quad \text{……………………………………………..……(5.3)}$$

EXAMPLE 5.2

Continuing from Example 5.1, the following machine bit-vectors were obtained: $mv_{1,1}$=<1110000000>, $mv_{1,2}$=<0001110000>, $mv_{1,3}$=<0000001111>, $mv_{2,1}$=<1100000000>, $mv_{2,2}$=<0010000011>, and $mv_{2,3}$=<0001111100>; the low-yield machine bit-vector of product *P* is <0001010100> and the out-of-control machine-bit vectors of machines $m_{1,1}$ and $m_{1,2}$ are, respectively, $mbv_{1,1}^{OC}$ =<0010000000> and $mbv_{1,2}^{OC}$ =<0001000000>. And $my_{1,1}$=<1110000000> AND <0001010100> = <0000000000>, $my_{1,2}$=<0001110000> AND <0001010100> = <0001010000> and the corresponding $mty_{1,1}$ and $mty_{1,2}$ are thus ANDed to <0000000000> and <0001000000>.

As mentioned above, we use these probing functions as major criteria in evaluating *MDDP-t* according to experts' experience in the semiconductor manufacturing domain. We then define a Root Cause Evaluation Function $RCEF(m_{i,j})$, which is a linear combination of the three probing functions along with their corresponding weights w_i used to identify the importance of each probing function in the *RCEF*, to compute the root-cause possibility of machine $m_{i,j}$.

Root Cause Evaluation Function *RCEF*($m_{i,j}$) of *MDDP-t*

$$RCEF(m_{i,j}) = \sum_{k=1}^{3} w_k \times f_k(m_{i,j})$$

However, the corresponding weights $W=\{w_1, w_2, w_3\}$ of these three *RCEF* probing functions require further investigation. A genetic algorithm is thus used to solve the weight-learning problem of the three given probing functions in order to determine suitable weights for the *MDDP-t*.

5.3 Genetic Algorithm for *MDDP-t*

The search space in a GA (Genetic Algorithm) consists of possible solutions to a problem [15]. A solution in the search space is called an *individual* and its genotype consists of a set of *chromosomes* represented by sequences of 0s and 1s. These chromosomes can dominate individual phenotypes. Each individual has an associated objective function called its *fitness*. A good individual is one that has a high/low fitness value depending on whether the problem involves maximization or minimization. The strength of a chromosome in an individual is represented by its *fitness value* and the chromosomes of individuals are carried to the next generation. The set of individuals with associated fitness values is called the *population*. The population at a given stage in the GA is referred to as a *generation*. The best individual in each generation is the individual with the best discovered fitness value.

There are three main components in the GA while loop:

(1) *selection/reproduction*, the process of selecting good individuals from the current generation to be carried to the next generation;

(2) *crossover*, the process of shuffling two randomly selected strings (chromosomes) in two parent individuals to generate new offspring;

(3) *replacement*, the replacing of the worst-performing individuals in a generation based on fitness value.

Sometimes one or more bits of a chromosome are complemented to generate a new offspring. This process is called *mutation*. The population size is finite in each GA generation, which implies that only relatively fit individuals in generation *j* will be carried to the next generation *j+1*. The power of GA is that the algorithm terminates rapidly to an optimal or near optimal solution. The iterative process is terminated when the solution reaches the optimum value [16].

Details of the GA developed to solve *MDDP-t* are described in this section. As mentioned above, the weight set *W* is quite important in solving *MDDP-t*. Since the weights are domain-dependent, we propose a GA-based weight-learning function for *MDDP-t* to find weights *w* for each probing function according to *MDDP-t* instances with known root causes. The weight-learning function is described in detail below.

NOTATION 5.4

M_i	machine set for the *i*-th *MDDP-t* instance;
rm_i	root-cause machine already known to cause the *i*-th *MDDP-t* instance defect.
$rank(M_i, rm_i)$	*k*, where rm_i is the *k*-th largest *RCEF* value in set M_i.

Weight-learning Problem: Given *k* *MDDP-t* instances, find weights w_1, w_2 and w_3 to minimize :

$$\sum_{i=1}^{k} rank(M_i, rm_i) \quad \text{(5.4)}$$

EXAMPLE 5.3:

Assume three *MDDP-t* instances with three weight sets. According to the rankings of actual root causes in the three datasets evaluated using these three functions shown in Table 5.2, w_1 is the best choice.

Table 5.2: Weight-learning function example for three *MDDP-t* instances

	$rank(M_1, rm_1)$	$rank(M_2, rm_2)$	$rank(M_3, rm_3)$	$\sum_{i=1}^{3} rank(M_i, rm_i)$
w_1	1	1	2	4
w_2	2	3	4	9
w_3	1	2	4	7

There are five parts to our GA approach: encoding, crossover/mutation, selection/terminal conditions, and fitness determination. In general, the chromosomes in the first generation are created randomly and succeeding generations are generated by crossover and mutation. Details of these four parts are given below.

Encoding

The proposed probing functions are based on expert experiences, and each chromosome is the concatenation of the bit-strings represented by w_1, w_2 and w_3. Since not all probing functions are used in every domain, the n-bit flags e_1, e_2 and e_3 are used to help the GA efficiently determine which probing functions to use in the *RCEF* function. When the one-bit e_i is set to zero, the weight w_i of that probing function is set to zero in the chromosome. Obviously, the n-bit e_i is used to set the probing function probability determination to 1/n. Assume the probing function determination probability is 25% and the number of bits for e_i is set to 4. The corresponding essential flag e_i also uses n bits in the tail of its weight string, the initial values of which are randomly set. According to the above definitions, assume that w_1=00011=3, s_1=01, w_2=00101=5, s_2=10, w_3=00100=4, and s_3=10. The chromosome thus generated is 0001101001011000100l0.

Crossover/Mutation Procedures

Many methods can be employed in the crossover process, thus, suitable operation should be selected according to the application domain. For example, the strings 001111001011001001 and 010011011001001011 could be crossed over after the second locus in each to produce 000011011001001011 and 011111001011001001. Our experience indicates the random one-point crossover method is suitable for solving *MDDP-t* learning problems.

The conventional bit-inversion method can be used in the mutation process. For example, the second position in the string 001111001011001001 might be mutated to yield 011111001011001001 by changing the 0 to 1 in bit 2. Our experience indicates the inversion probability should be set to 0.05.

Selection/Terminal Conditions

The population size in each generation and terminal conditions can be determined according to the application domain. Our experience indicates the initial chromosome number in the population should be set to 300 and the terminal conditions set to 500 generations.

Fitness Function

Many chromosomes are produced in each generation and weights *W* must be evaluated. In order to identify suitable weight sets, all machine information is input to the *RCEF*, which then computes the actual root-cause rankings. An *MDDP-t* GA fitness function and *MDDP-t* GA algorithm are shown below.

***MDDP-t* GA fitness function**

For *n* given *MDDP-t* instances *MDDP-t*$_1$, *MDDP-t*$_2$, …, *MDDP-t*$_n$, let rm_j be the actual root-cause of the *MDDP-t*$_j$ instance. Weight set W_i is better than weight set W_j if $\sum_{k=1}^{n} rank(M_k, rm_k)$ using weight set W_i is smaller than the same function using W_j.

Algorithm 5.1 - *MDDP-t GA algorithm*

```
Input:    Training datasets
Output:   The weight set $W$ for the $RCEF$
Step1:    Initialize population (bit-strings combining $w_1$, $e_1$, $w_2$, $e_2$, $w_3$, $e_3$)
Step2:    Choose parents
Step3:    Construct offspring using one-point crossover
Step4:    Call mutation procedure
Step5:    For all flags $e_i$, if $e_i$ is all 0, set $w_i$=0; otherwise $w_i$=$w_i$
Step6:    Evaluate offspring and replace least-fit individual with better offspring
Step7:    Go to Step2 until a terminal condition is reached
```

Training will generate several weight sets, which can then be applied to detecting root causes in future datasets. When a new dataset with an unknown root cause is input into the manufacturing defect detection system for root cause discovery, it must first be translated into *MDDP-t* terms. After that, the top combination is used to generate a possible root-cause ranking list. Engineers can use these ranking lists to check machines one by one and filter out possible killer machines. Finally, engineers can then record the real root cause and may re-compute the *MDDP-t* learning procedure if the weights resulting from training fail to identify the correct root cause.

5.4 Experiments for *MDDP-t*

There were 21 data datasets in our experiments, some of them are provided by the *Taiwan Semiconductor Manufacturing Company* (TSMC). We divided these into 12 training datasets, shown in Table 5.3, and 9 test datasets, shown in Table 5.4, each with a real root cause. We used the training datasets to find the top 5 weight combinations for the *RCEF*, and used the test datasets to evaluate the accuracy of the weight sets.

Table 5.3: Training Datasets for the GA approach

DataSet	Size of dataset (Lots*Attributes)	Number of machine clusters	Number of machines
Dataset1	300*4211	2314	4456
Dataset 2	302*3345	1842	4235
Dataset 3	255*6625	2356	6822
Dataset 4	187*2568	1108	3684
Dataset 5	427*1548	1001	2265
Dataset 6	392*3954	2304	5262
Dataset 7	265*2879	1105	4552
Dataset 8	267*2265	1096	3665
Dataset 9	321*2451	1664	4556
Dataset 10	367*4325	2025	4456
Dataset 11	357*2848	1456	3698
Dataset 12	285*2525	1875	3308

Table 5.4: Test Datasets

Dataset	Size of dataset (rows*columns)	Number of machine clusters	Number of machines
D_1	365*2234	986	3625
D_2	752*3365	1245	3688
D_3	654*3364	2856	4652
D_4	586*3324	1846	4875
D_5	564*1239	823	2234
D_6	452*2235	1134	3048
D_7	165*3321	1652	3698
D_8	215*1254	656	2043
D_9	346*2236	1134	2365

IThe initial mutation probability was set to 0.05, and the maximum number of generations to 3000. The CPU times and population sizes for the *RCEF* weight combinations are shown in Figure 5.4. As shown, the CPU usage is marginally near the polynomial time cost.

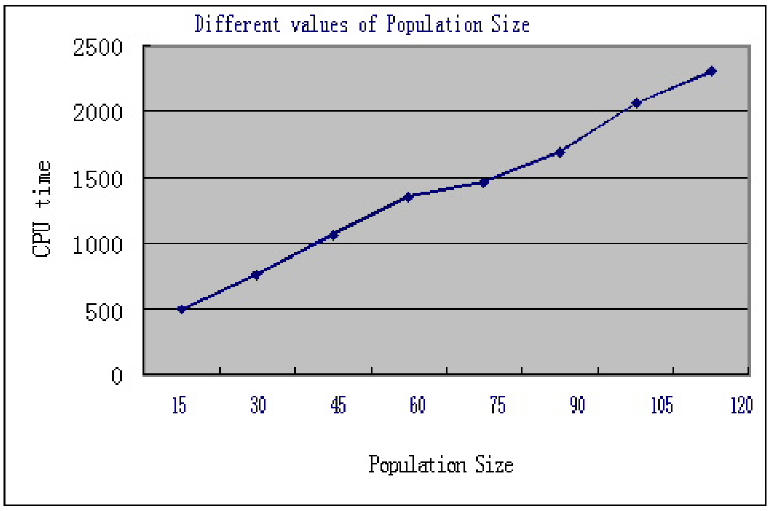


Figure 5.4: Experimental Results for Various Population Sizes

The average root-cause rankings for the training datasets from the top 5 weight sets for various standard deviation values of probing functions are shown in Figure 5.5. When the standard deviation was between 1 and 1.5, the genetic algorithm found the best solutions, but hit errors increased when the time standard deviation was greater than 2 and less than 0.75 since too much information was pruned and computational noise was included.

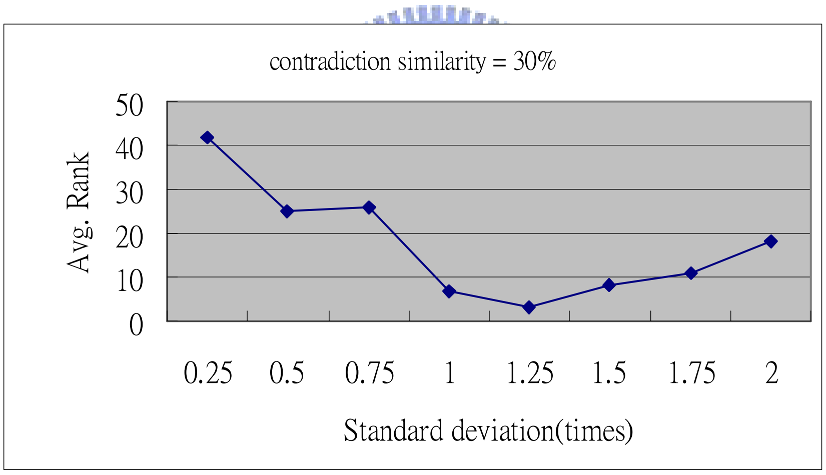


Figure 5.5: Average root-cause rankings from the training datasets by the top 5 functions for various standard deviation values

We chose top 5 chromosomes when the GA training process finished. The results for the α, β, γ combinations are shown in Table 5.5. Clearly, the actual root-cause rankings for the test datasets are quite high and the hit-error averages are all in a tolerable range.

Table 5.5: Actual root-cause rankings for the test datasets

w & *e*	D_1	D_2	D_3	D_4	D_5	D_6	D_7	D_8	D_9	Summary
w={18,5,6} *e*={1,1,1}	2	8	1	2	1	9	7	2	2	34
w={10,12,2} *e*={1,1,1}	3	5	3	2	2	13	9	6	8	51
w={12,8,6} *e*={1,1,1}	1	32	6	8	16	25	13	6	3	110
w={8,15,10} *e*={1,1,1}	25	12	7	12	18	6	3	13	16	112
w={6,7,13} *e*={1,1,1}	46	18	10	9	11	2	4	2	19	120
Average	15.4	8.3	5.4	6.6	9.6	11	7.2	5.7	9.6	9.5

As mentioned above, the proposed method is quite useful for finding actual root causes in actual manufacturing datasets using the weights discovered by the proposed GA learning approach.

Quickly solving product-yield and quality problems in complex manufacturing processes is becoming increasingly difficult. Although the "manufacturing time problem" may be avoided via process control, statistical analyses, and experimental design, it is still very difficult to resolve once it happens. In this section, the manufacturing time problem for the manufacturing domain (*MDDP-t*) has been formally modeled and defined. Accordingly, a root-cause evaluation function (*RCEF*) has been proposed to evaluate whether a specific machine is the root cause of a time problem. The *RCEF* uses three probing functions independently defined according to the experiences of domain experts. Moreover, a genetic algorithm (GA) has been designed to find suitable weights for the proposed probing functions. Experiments were also performed and the results show the proposed approaches can ensure efficiency and accuracy.

In the future, we will continue focusing our research on this topic. We will keep challenging the correctness of the proposed *MDDP-t* by seeking useful probing functions from different perspectives and different application domains. We will also try to apply similar learning models to other *MDDP* problems in the semiconductor manufacturing domain, such as the wafer-in-process (*MDDP-wip*), wafer-acceptance-test (*MDDP-wat*), and in-line metrologies (*MDDP-im*) issues, in order to discover the root causes of *MDDP* problems as correctly and efficiently as possible.

Chapter 6
Using BWI indexing in Intrusion Detection System

In this chapter, a pattern-learning network intrusion detection system is described. This implementation uses the Encapsulated Bit-wise Indexing Method to encode the networking activity with minimal monitoring time window in order to accelerate the data preparation procedure. Moreover, a bit-based intrusion Pattern Matching mechanism is proposed to efficiently learn, roll-up, drill-down and combine the intrusion pattern with different time-windows/services/ports combinations.

6.1 Problem Description

Due to the rapid growth of networked computer resources and the increasing importance of the related applications, intrusions that threaten the infrastructure of these network applications become critical problems today. [35][37][38][40][46][57][66][72] Network intrusion detection (NID) is the process of identifying possible intrusion behaviors from the network that provides information to the security administrators. Although many intrusion detection systems had been proposed and some possible intrusion behaviors had been identified and detected [1][20][21][26][49][57][69], no optimal solution had been found due to the variances of the intrusion patterns. In this work, we are concerned about how to identify possible intrusion behaviors that can help users to build an intrusion detection system through data mining processes to secure the infrastructure of the network. In the intrusion detection domain, five issues need to be considered, including Pattern representation, Computability, Performance, Maintenance and Extendibility. In this chapter, we propose a new, efficient and service-oriented intrusion pattern mining and representation method, called *Bit-wise-based Intrusion Pattern Mining Method* (**BIPAM**), which can provide higher performance, better maintenance and expressive abilities. In our model, BIPAM consists of two phases, Network Activities Analyzing Phase and Features/Pattern Mining Phase, and a database that contains the information about the users and the mined intrusion patterns is used in these two phases.

In general, almost all intrusion patterns can be transformed into a sequence of network activities that are extracted from the related network packets. These kinds of network packets can be collected and then be transformed into some sequence of bit-wise strings showing the intrusion patterns. The Network Activities Analyzing Phase of BIPAM can first filter the raw network packets and log necessary features (Source IP, Destination IP, Source port, Destination port) in a small time window to perform data sampling and data cleaning and to reduce the amount of data. After that, with combined users and services information, the sufficient service-user activity events are found and used by the second phase. The Features/Pattern Mining Phase transforms the sufficient service-user activity events to some bit-wise strings and next merges the bit-wise strings into some other bit-wise strings with the same source IP. After gathering those bit-wise strings, the Pattern Mining Module and Pattern Merging Module can perform some data mining processes to find possible intrusion patterns that can be the source of the candidates of intrusion patterns for future intrusion detection systems.

Since the expression of intrusion pattern is one of the most important things in an intrusion detection system, the expressions of intrusion pattern in current intrusion detection systems will be firstly introduced and the representation of the bit-wise indexing method will be next introduced in this section.

6.2 The Representation of Intrusion Behavior

According to the results of previous researches, the representation of intrusion behavior can be categorized as follows:

Implicit representation of intrusions: Some intrusion detection systems use their own models for detecting some specific intrusion behaviors. For example, the detection system for DDoS (Distributed DoS), which intrudes the system by coordinating hosts, analyzes the network information with the known properties of DDoS intrusion. These kinds of intrusion detection systems may not provide an understandable representation for intrusion behavior, since the knowledge for intrusion detection is imbedded in the system.

Rule oriented intrusion representation: This is the most common representation for intrusion detection knowledge. In an if…then formatted rule, the condition of the rule records the matching criteria for the intrusion, and the action of rule records the reaction for the intrusion. For example, a rule for BO (Back Orifice) intrusion, which is a back door intrusion by using specific program, may check every packet information whether the connection is through port 31337 or not. Once the rule is triggered, the action defined in the action part of the rule, e.g., alert the administrator, is then performed.

Pattern oriented intrusion representation: Many intrusions may not be accomplished by a single step, so does the intrusion detection. Using a single rule can only represent intrusions with single step or intrusions with a significant feature, e.g., BO intrusion, some application vulnerabilities. However, for intrusions with several steps to execute, a pattern oriented intrusion representation for intrusion behavior will be needed. A pattern oriented intrusion representation will represent intrusion in a sequence of states; for example, a sequence of states in a state machine or a state diagram.

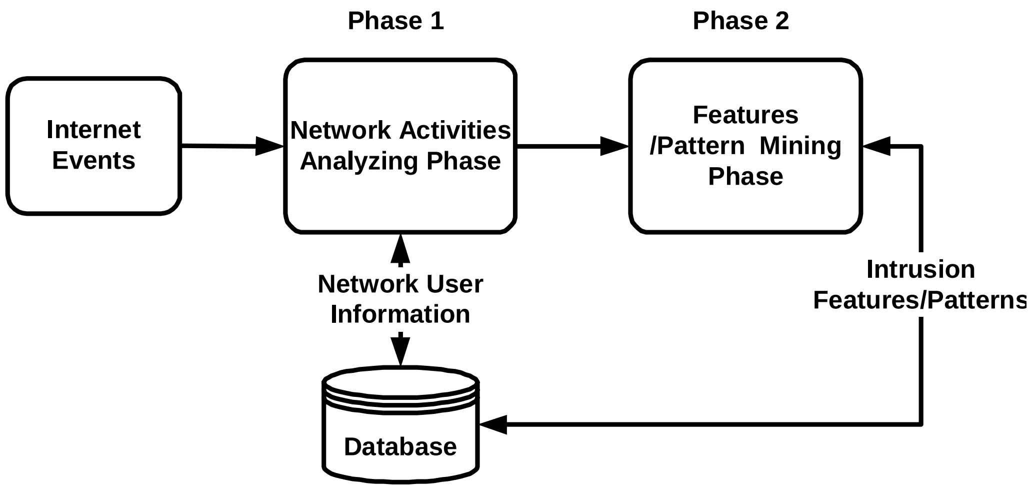
Specific intrusion representation: Many researches are trying to define specific model together with corresponding specific intrusion representation to represent intrusion. For example, goal tree, which has good performance on some specific target intrusions, had been used to represent intrusion pattern in some previous researches. However, the specific representations will sometimes lack the extendibility since the specific representation may be not suitable for all kinds of intrusions.

Each kind of intrusion behavior expression has advantages and disadvantages, but different intrusion detection systems usually require different intrusion behavior expressions. Thus, it is difficult to integrate the intrusion behavior knowledge by these intrusion behavior expressions. In this chapter, we will propose an efficient mining method BIPAM to explore the possible intrusion patterns via monitoring and analyzing the users' behaviors.

As mentioned above, the bit-wise indexing method can be easily indexed and parallelized, the bit-wise indexing method is quite suitable to solve the performance and scalability issues of a real-time IDS.

6.3 Architecture of BIPAM

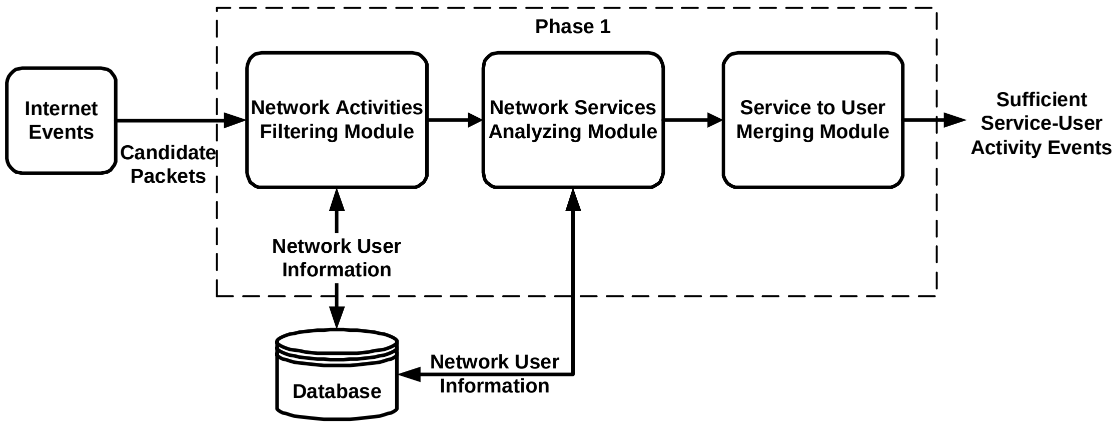
As we know, building an intrusion detection system becomes one of the most popular solutions to secure the network infrastructure in recent years. Since the expression of intrusion patterns is very important in building an intrusion detection system. The architecture of BIPAM consisting of three main components for quickly mining possible intrusion patterns is proposed as shown in Figure 6.1. The database in Figure 6.1 stores the users' information, some users' historical mined data and the possible intrusion patterns gathered in the past.



Phase 1
Phase 2
Internet Events
Network Activities Analyzing Phase
Features /Pattern Mining Phase
Network User Information
Intrusion Features/Patterns
Database

Figure 6.1: The architecture of the BIPAM

Figure 6.2 shows the detailed architecture of the Network Activities Analyzing Phase. In this phase, the Network Activities Filtering Module, Network Services Analyzing Module and Service to User Merging Module are proposed to provide sufficient service-user activity events to the next phase.



Phase 1
Internet Events
Candidate Packets
Network Activities Filtering Module
Network Services Analyzing Module
Service to User Merging Module
Sufficient Service-User Activity Events
Network User Information
Database
Network User Information

Figure 6.2: The detailed process of Phase 1

Before the whole mining procedure is proceed, the IP address of target machines that may be intruded, called *victim IP*, should be defined. These target machines usually provide some important services and thus easy to be treated as the victims by the intruders. The victim IPs are the primary parameters of BIPAM. The network packets, including TCP, UDP and ICMP packets, are checked by Network Activities Filtering Module and all unrelated packets of the victim IPs are filtered out via checking the dumped packet logs. Also, the corresponding IP address for each related packets, called *possible inflictor IP*, is to compare the IP information in the database in order to check the historical status of such IP. If the IP is dangerous, the system alert will be trigged. Also, all the packets from such IP will be restricted. For example, assume that there are twenty packets pass through the Network Activities Filtering Module. The detailed log about these packets is shown in Table 6.1. Also, the victim IP is 140.113.167.100. In Table 6.1, the packet 2 and 4 are filtered out since they are not related packet of victim IP. Also, four possible inflictor IPs, including 140.113.167.122, 140.127.12.113 and 115, denote them as pii_1, pii_2 and pii_3 respectively. Assume that 140.127.12.115 is the known dangerous IP. The system will notify the administrator via sending some warning messages and then all requirements from this IP are denied. After executing the Network Activities Filtering Module, the amount of network packets needed to be logged will be reduced and all connected between the source IPs will be collected. Also, all possible inflictor IPs can be found for further investigation.

Table 6.1: The packet log of Network Activities Filtering Module

#	Source IP	Destination IP	Pt	Packet Type	Serv Type	Prot.	Time	Etc
1	140.113.167.122	140.113.167.100	80	TCP	web	http	12:01:11	……
2	140.113.167.122	140.113.167.121	21	TCP	web	ftp	12:01:11	……
3	140.113.167.122	140.113.167.100	80	TCP	web	http	12:01:12	……
4	140.113.167.122	140.113.167.121	21	TCP	web	ftp	12:01:13	……
5	140.113.167.122	140.113.167.100	80	TCP	web	http	12:01:14	……
6	140.113.167.122	140.113.167.100	80	TCP	web	http	12:01:15	……
7	140.127.12.113	140.113.167.100	1	TCP	u/k	u/k	12:01:16	……
8	140.127.12.113	140.113.167.100	1	TCP	u/k	u/k	12:01:16	……
9	140.113.167.122	140.113.167.100	80	TCP	web	http	12:01:17	……
10	140.127.12.113	140.113.167.100	2	TCP	u/k	u/k	12:01:18	……
11	140.127.12.115	140.113.167.101	80	TCP	web	http	12:01:19	
12	140.113.167.122	140.113.167.100	80	TCP	web	http	12:01:19	……
13	140.127.12.113	140.113.167.100	3	TCP	u/k	u/k	12:01:20	……
14	140.113.167.122	140.113.167.100	80	TCP	web	http	12:01:20	……
15	140.127.12.113	140.113.167.100	4	TCP	u/k	u/k	12:01:21	……
16	140.113.167.122	140.113.167.100	80	TCP	web	http	12:01:22	……
17	140.127.12.113	140.113.167.100	5	TCP	u/k	u/k	12:01:23	……
18	140.127.12.113	140.113.167.100	6	TCP	u/k	u/k	12:01:24	……
19	140.113.167.122	140.113.167.100	80	TCP	web	http	12:01:25	……
20	140.127.12.113	140.113.167.100	7	TCP	u/k	u/k	12:01:27	……

After filtering the network packets, the Network Service Analyzing Module transfers the packet information into packet log table shown in Table 6.2, which contains the attributes about the network activities including Source IP, Destination IP, Destination port, trigger time, and service type. Those packet log table can then be classified according to the source IP, destination IP and service type in order to show the relationships between services and servers.

Table 6.2: The packet log table

#	Source IP	Destination IP	Dest. Port	Service Type	Time
1	140.113.167.122	140.113.167.100	80	Web	12:01:11
3	140.113.167.122	140.113.167.100	80	Web	12:01:12
5	140.113.167.122	140.113.167.100	80	Web	12:01:14
6	140.113.167.122	140.113.167.100	80	Web	12:01:15
7	140.127.12.113	140.113.167.100	1	u/k	12:01:16
8	140.127.12.113	140.113.167.100	1	u/k	12:01:16
9	140.113.167.122	140.113.167.100	80	Web	12:01:17
10	140.127.12.113	140.113.167.100	2	u/k	12:01:18
12	140.113.167.122	140.113.167.100	80	Web	12:01:19
13	140.127.12.113	140.113.167.100	3	u/k	12:01:20
14	140.113.167.122	140.113.167.100	80	Web	12:01:20
15	140.127.12.113	140.113.167.100	4	u/k	12:01:21
16	140.113.167.122	140.113.167.100	80	web	12:01:22
17	140.127.12.113	140.113.167.100	5	u/k	12:01:23
18	140.127.12.113	140.113.167.100	6	u/k	12:01:24
19	140.113.167.122	140.113.167.100	80	web	12:01:25
20	140.127.12.113	140.113.167.100	7	u/k	12:01:27

The third module in this phase is the Service to User Merging Module. In this module, the packet log table has been sorted in ascending order according to the attribute *source IP* and *destination IP*, *service type* and *trigger time* sequentially. After sorting the packet log table sorted, for each segment with the same Soruce IP, Destination IP and service type will be partitioned into several small tables, called service-user activity events tables. These tables can be easily extracted and analyzed. Continuing the example in the above, two segments in Table 6.2 with the same Soruce IP, Destination IP and service type are found. The server-user activity events tables of

Table 6.3 of web and unknown service are shown in Table 6.3(a) and Table 6.3(b), respectively. After the service-user activity events had been generated, this information will be delivered to Phase Two for further processing.

Table 6.3: The service-user activity event tables

(a) Service-user activity event of *pii*$_1$ for web service

#	Source IP	Destination IP	Dest. Port	Service Type	Time
1	140.113.167.122	140.113.167.100	80	web	12:01:11
3	140.113.167.122	140.113.167.100	80	web	12:01:12
5	140.113.167.122	140.113.167.100	80	web	12:01:14
6	140.113.167.122	140.113.167.100	80	web	12:01:15
9	140.113.167.122	140.113.167.100	80	web	12:01:17
12	140.113.167.122	140.113.167.100	80	web	12:01:19
14	140.113.167.122	140.113.167.100	80	web	12:01:20
16	140.113.167.122	140.113.167.100	80	web	12:01:22
19	140.113.167.122	140.113.167.100	80	web	12:01:25

(b) Service-user activity event of *pii*$_2$ for unknow service

#	Source IP	Destination IP	Dest. Port	Service Type	Time
7	140.127.12.113	140.113.167.100	1	u/k	12:01:16
8	140.127.12.113	140.113.167.100	1	u/k	12:01:16
10	140.127.12.113	140.113.167.100	2	u/k	12:01:18
13	140.127.12.113	140.113.167.100	3	u/k	12:01:20
15	140.127.12.113	140.113.167.100	4	u/k	12:01:21
17	140.127.12.113	140.113.167.100	5	u/k	12:01:23
18	140.127.12.113	140.113.167.100	6	u/k	12:01:24
20	140.127.12.113	140.113.167.100	7	u/k	12:01:27

In the Feature/Pattern Mining Phase, there are three modules, including Bit-wise Transforming Module, Pattern Mining Module and Pattern Merging Module. The goals in this phase are transforming the network events to some corresponding bit-wise strings and performing data mining processes in order to find possible intrusion patterns. The detailed architecture of this phase is shown in Figure 6.3.

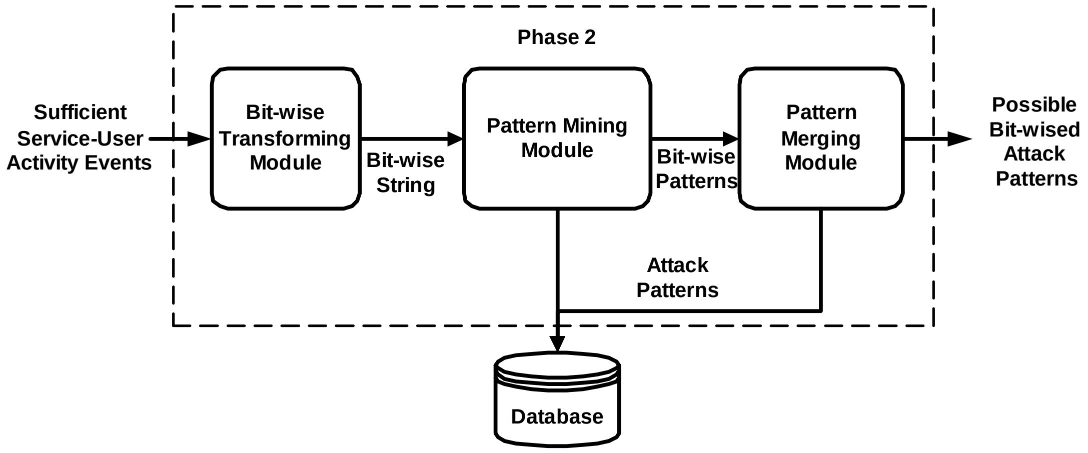


Figure 6.3: The detailed process of phase 2

When the sufficient service-user activity events are collected, the activity events are transformed into some single-services bit-wise strings according to a small time window, which is defined to be the basic time slice of IDS, for every service-user activity event via Bit-wise Transforming Module. These bit-wise strings can then be stored in the database. For further data mining and storage saving, each single-services bit-wise string can be transformed into a new single-services bit-wise string with a larger time slice. Moreover, some single-service bit-wise strings with the same destination and source IPs may be merged into a new multi-services bit-wise string since the most networking intrusion behaviors from several different services in order to setup an proper attack environment. So the final products of this module are these bit-wise strings that not merely contain single-service user behaviors but also contain single-service user behaviors. For example, assume the basic time slice of BIPAM is one second. The bit-wise string of pii_n for service type m using time slice k is denoted as $ppi_n.bs_k^m$, shown as following.

Service-user activity event of pii_1 for web service using time slice one and five seconds

$ppi_1.bs_1^{web}$= 0000000000110110101101001000

$ppi_1.bs_5^{web}$= 001110

Service-user activity event of pii_2 for unknown service using time slice one and five seconds

$ppi_2.bs_1^{unknown}$= 00000000000000001010110101001

$ppi_2.bs_5^{unknown}$= 000111

In the Pattern Mining Module, with the help of the pattern database, these bit-wise strings of each user are first compared with the existing intrusion patterns stored in the database using bit-wise indexing method for similarity search. If there is an existing intrusion pattern is compared with one of these bit-wise strings and the similarity degree is higher than the given threshold (e.g., 0.9), the IDS will announce a warning message and take some appropriate actions. For instance, if there is an existing port scan intrusion pattern, $bp_*^{unknown}$=1111111111111111111, the similarity between $bp_*^{unknown}$ and $ppi_2.bs_5^{unknown}$ is 1 (if leading 0 is avoided), Although there is no such kind of intrusion patterns in the database, the security administrators or expertise may still consider these packet logs as some kinds of intrusions and possible intrusion patterns might be found and then be stored in the database for further evaluations. After finishing the works in this module, the bit-wise strings with possible intrusion patterns of one user will be sent to the next module to find more complex intrusion patterns. These bit-wise strings can be merged and then compared with existing intrusion patterns to find the intrusion patterns of multiple services using Pattern Merging Module. For example, the $ppi_1.bs_1^{web}$ and $ppi_1.bs_1^{web}$ can be merged and thus the bit string $ppi_{1,2}.bs_1^{web,unknown}$ = 000000000011011111111111101001 is formed. The bit string $ppi_{1,2}.bs_1^{web,unknow}$ can then be compared with the existing DDOS patterns for finding some possible intrusion behavior. At last, the possible bit-wise intrusion patterns are mined for further works in building an intrusion detection system.

In this chapter, we have proposed a new, efficient and service-oriented intrusion pattern mining and representation method that provides more expressivities, higher performance. The intrusion patterns are extracted from the some sample packets that can be expressed in sequence of packets and thus are represented by some bit-wise strings for each network service. These bit-wise intrusion patterns can be easily rolled up and drilled down into the intrusion pattern of variant time window efficiently. Also, the bit-wise intrusion patterns of each service can be easily merged with the others. Using this method, the Internet intrusion patterns can be automatically mined from the basic Internet activity logs efficiently and some interesting and unknown patterns may be discovered. Now, we are trying to build an online intrusion detection system using BIPAM for building a high confidence network system.

Chapter 7
Using BWI indexing in Feature Selection Method for Knowledge Acquisition

In this chapter, an application that is a supervised-learning data-driven feature selection method for CBR systems [5][7][23][25][28][31][67][73]is introduced. This implementation applies the Feature Selection Method using Rough Set Theory, which is appropriate for finding the optima solution from a given data set, except for the long processing time issue. Therefore, the Compact Bit-wise Indexing Method is used to encode the feature and class relationships to reduce the processing time of feature selection procedure. Finally, some experiments and comparisons are given and the result shows the efficiency and accuracy of our proposed methods.

7.1 Problem Description

Feature selection is about finding useful (relevant) features to describe an application domain. Selecting relevant and enough features to effectively represent and index the given dataset is an important task to solve the classification and clustering problems intelligently. This task is, however, quite difficult to carry out since it usually needs an exhaustive search to get the features desired. In the past, some approaches have been proposed to solve the feature selection problem [11][19][24][30][43][47][48][60][78]. These approaches can roughly be classified into the following two strategies:

1. Optimal strategy: This kind of approaches considers all the subsets of a given feature set [2][63][76]. Some searching techniques, such as branch and bound, may be adopted to reduce the search space. For example, Liu *et al.* proposed a special feature selector [47], which randomly produced feature subsets according to the Las Vegas algorithm [7]. It thus searched the entire solution spaces and guaranteed to get an optimal feature set.

Heuristic strategy: This kind of approaches prunes search spaces according to some heuristics. The results obtained by these approaches are usually not optimal, but within a short time [79]. There are three typical heuristic approaches for feature selection, including *forward selection, backward selection* and *bi-directional selection.* The forward-selection approach initializes the desired feature set as null and then adds features into it until the results are satisfactory [50][64][78]. The backward-selection approach initializes the desired feature set as all the given features and then removes unnecessary features from it [19][78]. The bi-directional selection approach initializes the desired feature set as a partial feature set, and then either puts good features into it or eliminates bad features from it [24]. In the past, we proposed a bit-wise indexing method based on a given feature set to accelerate case matching in CBR [11][13]. In this section, we further investigate the determination of the appropriate feature set. We propose a two-phase feature selection approach to discover significant feature sets from a given database table, and use them to further investigation. The proposed feature selection approach originates from the bitmap indexing and rough set techniques. Naturally, it is designed to discover optimal feature sets for the given dataset since the proposed method is originated from the rough set theory. The Experimental results also show the efficiency and accuracy of the proposed approach.

7.2 The proposed bitmap-based feature selection method

As we mentioned above, we proposed a heuristic feature-selection approach, called the *bitmap-based feature selection method with discernibility matrix* [14], to find a nearly optimal feature set. However, finding the optimal solutions of feature selection is still needed in some applications. Although some exhaustive search methods can guarantee the optimality of selected feature sets, the computation cost may be very high.

In this section, we thus consider finding an optimal solution via the rough set techniques and the bit-based indexing method for the feature selection. The proposed approach encodes a given data set into a bit vector matrix and uses bit-processing operations on them to reduce the computation time. The proposed approach consists of several main steps, as shown in Figure 7.1.

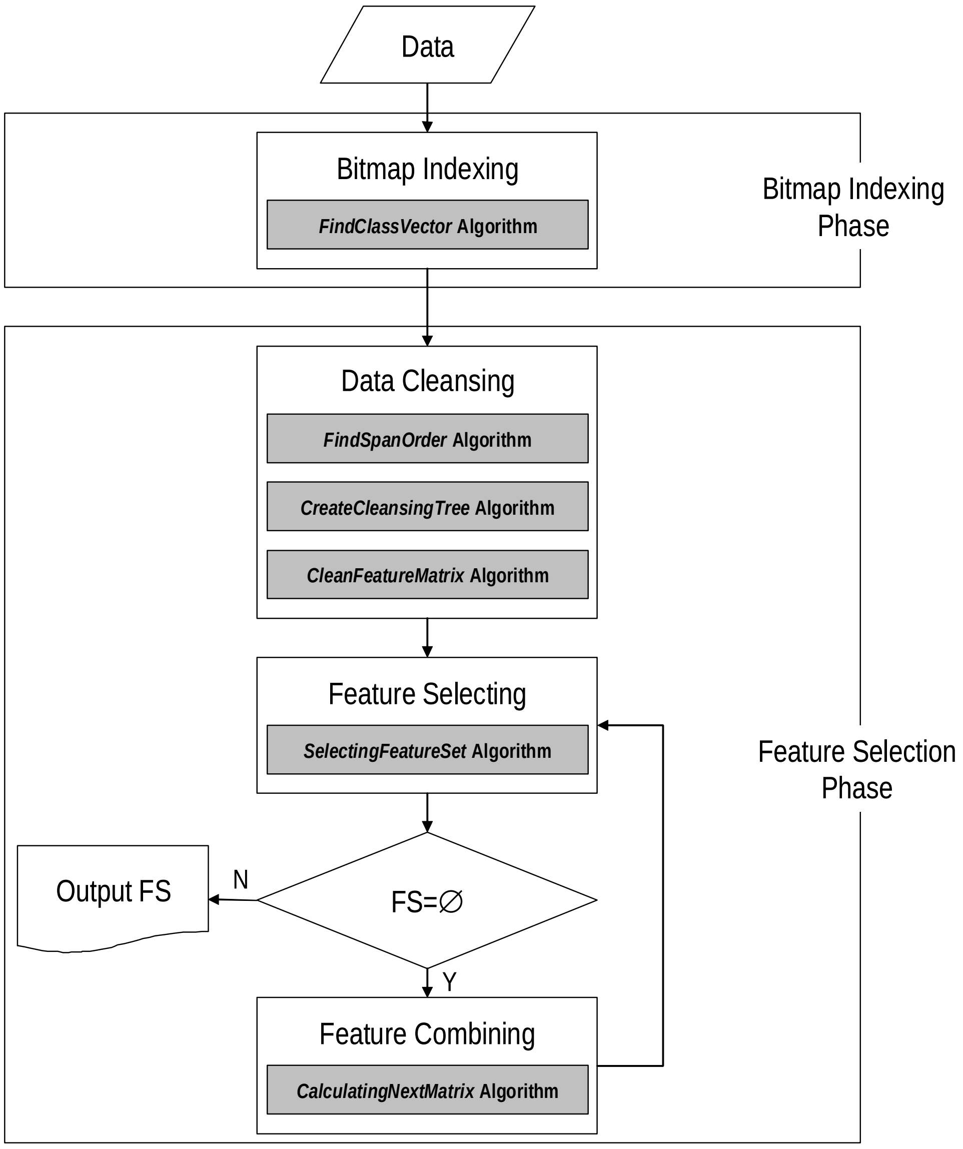

Data
Bitmap Indexing
FindClassVector Algorithm
Bitmap Indexing Phase
Data Cleansing
FindSpanOrder Algorithm
CreateCleansingTree Algorithm
CleanFeatureMatrix Algorithm
Feature Selecting
SelectingFeatureSet Algorithm
Feature Selection Phase
Output FS
N
FS=∅
Y
Feature Combining
CalculatingNextMatrix Algorithm

Figure 7.1: The flowchart of the proposed feature selection approach

There are two phases in the proposed algorithm - bitmap-indexing phase and feature selection phase. In the bitmap-indexing phase, the given dataset is transformed into a bitmap indexing matrix with some additional data information. In the feature selection phase, a set of relevant and enough features are selected and used to represent the dataset. The details of the two phases are described in following sub-sections.

7.2.1 Problem Definitions

Let T denote a target table in a database, R denote the set of n records in T, and C denote the set of m features in T. R can then be represented as $\{R_1, R_2, \ldots, R_n\}$, where R_i is the i-th record. C can be represented as $\{C_1, C_2, \ldots, C_m\}$, where C_j is the j-th feature. The first m-1 elements in C are condition features and the last one, C_m, is a decision feature. Let V_j denote the domain of C_j. V_j can then be represented as $\{V_{j1}, V_{j2}, \ldots, V_{j\sigma_j}\}$, where each element is a possible value of C_j and σ_j is the number of possible values of C_j. Let $V_j(i)$ denote the value of C_j in record R_i, $V_j(i) \neq null$. Table 7.1 shows an example of a target table T with ten records $R = \{R_1, R_2, \ldots, R_{10}\}$ and five features $C = \{C_1, C_2, C_3, C_4, C_5\}$. C_5 is a decision feature and the others are condition features.

Table 7.1: An example of a target table

	C_1	C_2	C_3	C_4	C_5
R_1	M	L	3	M	1
R_2	M	L	1	H	1
R_3	L	L	1	M	1
R_4	L	R	3	M	2
R_5	M	R	2	M	2
R_6	L	R	3	L	3
R_7	H	R	3	L	3
R_8	H	N	3	L	3
R_9	H	N	2	H	2
R_{10}	H	N	2	H	1

The purpose of this method is to find the one of the smallest feature set to effectively index the given table. The definitions and algorithms used in the bitmap indexing phase and in the feature selection phase are described below.

7.2.2 Indexing Phase

In this phase, the target table is first transformed into a bitmap indexing matrix with some additional classification information. Let b_i is a bit of the bit vector. Let ONE_k denote the bit string of length k, with all the bits set to 1, $ZERO_k$ denote the one with all the bits set to 0, and $UNIQUE_k$ denote the one, with only one bit set to 1 and the others set to 0. A record vector, which is used to keep the information of the records with a specific value of a feature, is defined below.

DEFINITION 7.1- record vector :

A record vector RV_{jk} is a bit string $b_1b_2\ldots b_n$, with b_i set to 1 for $V_j(i) = V_{jk}$ and set to 0 otherwise, where $1\leq j\leq m$, $1\leq k\leq \sigma_j$, and $1\leq i\leq n$

RV_{jk} thus keeps the information of the records with the k-th possible value of the feature C_j. For example in Table 7.1, C_1 has three possible values {M, L, H}. The record vector for C_1 = M is 1100100000 since the first, second and fifth records have this feature value. Similarly, the record vector for C_1 = L is 0011010000 and for C_1 = H is 0000001111. All the record vectors are shown in the third column of Table 7.2.

Table 7.2: The record vectors and class vectors from Table 7.1

Feature	Feature-value	Record Vector	Class Vector
C_1	V_{11}	1100100000	110
	V_{12}	0011010000	111
	V_{13}	0000001111	111
C_2	V_{21}	1110000000	100
	V_{22}	0001111000	011
	V_{23}	0000000111	111
C_3	V_{31}	1001011100	111
	V_{32}	0110000000	100
	V_{33}	0000100011	110
C_4	V_{41}	1011100000	110
	V_{42}	0100000011	110
	V_{43}	0000011100	001
C_5	V_{51}	1110000001	100
	V_{52}	0001100010	010
	V_{53}	0000011100	001

A class vector, which is used to keep the information of the classes (values of the decision feature) with a specific value of a feature, is defined below.

DEFINITION 7.2 - class vector:

A class vector CV_{jk} is a bit string $b_1b_2\ldots b_{\sigma_m}$, with b_i set to 1 if $RV_{jk} \cap RV_{mi} \neq ZERO_n$, and set to 0 otherwise, where σ_m is the number of possible values of C_m and n

is the number of records in R.

Here, the "AND" bit-wise operator is used for the intersection in definition 2. CV_{jk} thus keeps the information of the classes related to the k-th possible value of the feature C_j. For example in Table 7.2, the record vector (RV_{11}) for C_1 = M is 1100100000 and the one (RV_{51}) for C_5 = 1 is 1110000001. Since the bit-wise intersection of 1100100000 and 1110000001 is 1100000000, not equal to $ZERO_{10}$, the first bit in RV_{11} is thus 1. Similarly, the second and third bits in RV_{11} are 1 and 0 from the intersection results of RV_{11} with RV_{52}, and with RV_{53}. the class vector CV_{11} is thus 110. All the class vectors are shown in the fourth column of Table 7.2. Formally, a class vector CV_{jk} can be obtained by the following *Find class vector algorithm*.

Algorithm 7.1 – *Find class vector algorithm* :

Input: Record vector RV_{jk}.

Output: Class vector CV_{jk}.

Step 1: Set CV_{jk} to $ZERO\sigma_m$.

Step 2: For each i, $1 \leq i \leq \sigma_m$, set the i-th bit of CV_{jk} to 1 if $RV_{jk} \cap RV_{mi} \neq ZERO_n$; otherwise, set it to 0.

Step 3: Return CV_{jk}.

DEFINITION 7.3 - Feature-value vector :

A feature-value vector F_{jk} is concatenated of RV_{jk} and CV_{jk}.

For example, the feature-value vector F_{11} in Table 7.2 is 1100100000110, which is RV_{11} concatenated with CV_{11}. All the feature-value vectors for a feature are then collected together as a feature matrix. This is defined below.

DEFINITION 7.4 - A feature matrix for a feature :

A feature matrix M_j for the feature C_j is denoted $\begin{bmatrix} F_{j1} \\ F_{j2} \\ \vdots \\ F_{j\sigma_j} \end{bmatrix}$, where σ_j is the number of possible values in C_j.

For example, the feature matrix M_1 in Table 7.2 is show as follows:

$$M_1 = \begin{bmatrix} 1100100000\underline{110} \\ 0011010000\underline{111} \\ 0000001111\underline{111} \end{bmatrix}$$

The bits with underlines are class vectors. From the definition of the feature matrix, it is easily derived that applying the bit-wise operator "OR" on all the record vectors in a feature matrix will get the ONE_n vector, and applying the bit-wise operator "AND" on any two record vectors in a feature matrix will get the $ZERO_n$ vector. Note that, the “OR” and “AND” operators are defined to result for executing “OR” and “AND” operation on all respective bits for the given two bit vectors. Thus, if we apply the bit-wise operator "XOR" on all the record vectors in a feature matrix, we will also get the $ZERO_n$ vector. Take M_1 as an example. The result for 1100100000 OR 0011010000 OR 0000001111 is 1111111111. The result for 1100100000 AND 0011010000 is 0000000000. The result for 1100100000 XOR 0011010000 XOR 0000001111 is 0000000000.

DEFINITION 7.5 - A feature matrix for a table *T* :

A feature matrix M for a table T is denoted $\begin{bmatrix} M_1 \\ M_2 \\ \vdots \\ M_m \end{bmatrix}$, where m is the number of features in T.

For example, the matrix composed of the bit strings from columns 3 and 4 of Table 7.2 is the feature matrix for the data given in Table 7.1. The feature matrix for a table is then input to the feature selection phase to find relevant and enough features.

7.2.3 Feature Selection Phase

In this phase, we want to find a set of relevant and enough features to represent the given dataset. It is further divided into several stages. First, a feature-based spanning tree is built for cleansing the bitmap indexing matrix. The dataset with noisy information is thus judged and filtered out according to the spanning tree. The cleansed, noisy-free bitmap indexing matrix is then used to determine the optimal feature set for some classification and clustering problems.

Before the feature selection phase is executed, the correctness of the target table needs to be verified. If there are some records in the target table with the same values of all condition features, but with different ones of the decision feature, they are treated as noise records and are filtered out from the target table. Intuitively, every two records can be compared to find out the inconsistent records in the target table. Its time complexity is O(n^2m), where n is the number of records and m is the number of features. Below, we propose the concept of a cleansing tree to decrease the time complexity to O(nmj), where j is the maximum number of possible feature values of a feature and *n is usually much larger than j* in the general classification and clustering problems. The formation of a cleaning tree depends on the given feature order. We thus have the following definition.

DEFINITION 7.6 - spanned feature order :

A spanned feature order *O* is a permutation consisting of all the condition features in a target table *T*.

For example in Table 7.1, <C_1, C_2, C_3, C_4> can be a spanned feature order. When a spanned feature order is given, a cleansing tree can then be built according to it. The definition of a cleansing tree is first given below.

DEFINITION 5-7 - cleansing tree :

A cleansing tree *Ctree* is a tree with a root denoted *root*[*Ctree*]. Every node *x* in the tree corresponds to a feature value. A node *y* is the parent of a node *x* if the feature of *y* precedes the feature of *x* in the given spanned feature order. A node *z* is the sibling of a node *x* if they have the same feature, but different values.

A structure of a cleansing tree is shown in Figure 7.2. Its maximum height is *m*-1, where *m* is the number of features in a decision table *T*. Each node *x* has three pointers, which are *p*[*x*], *left-child*[*x*] and *right-sibling*[*x*], respectively pointing to its parent node, its leftmost child node and its first right sibling node. It also contains two additional information, *record*[*x*] and *class*[*x*], which indicate the associated record and class vectors of *x*. If node *x* has no child, then *left-child*[*x*] = NIL; if node *x* is the rightmost child of its parent, then *right-sibling*[*x*] = NIL.

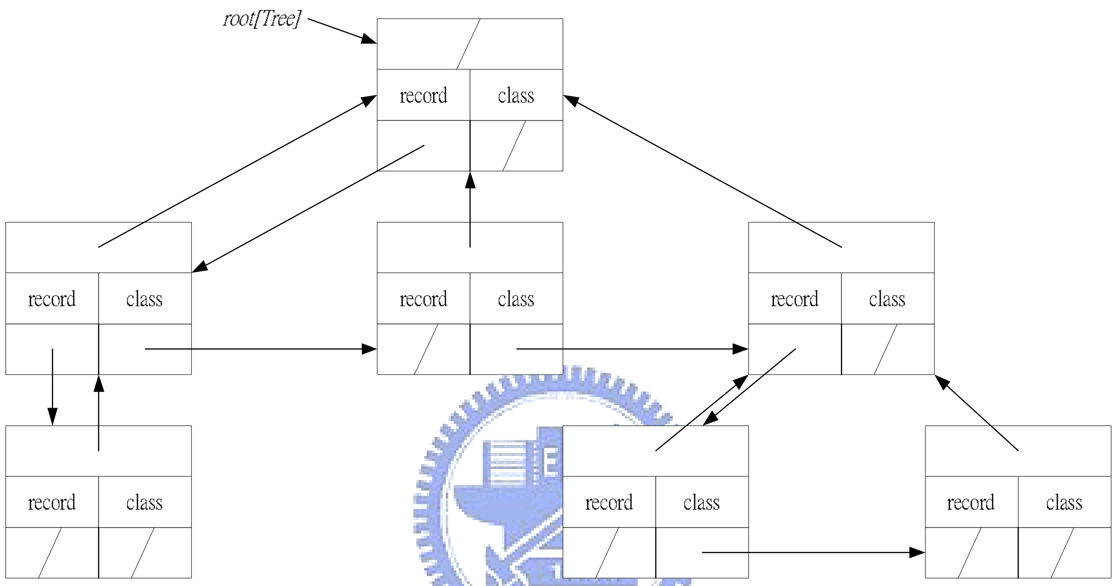


Figure 7.2: The structure of a cleansing tree

As mentioned above, records may have the same values of all condition features, but different value of the decision feature. These records are called inconsistent. Inconsistent records can also be found out when the cleansing tree is built. The building algorithm uses the valid mask vector to find the consistent records. The valid mask vector is defined as follows.

DEFINITION 7.8 - valid mask vector :

A valid mask vector *ValidMask* for a target table *T* a bit string $b_1b_2\ldots b_n$, with b_i set to 1 if the *i*-th record R_i is not inconsistent with other records, and set to 0 otherwise.

The cleansing tree for a given spanned feature order can be built by the following *Create cleansing tree algorithm*. The *ValidMask* is initially set to ONE_n., and will be modified along with the execution of the *Create cleansing tree algorithm*.

Algorithm 7.2 – *Create cleansing tree algorithm* :

Input : A feature matrix *M*, the valid mask *ValidMask* and a spanned feature order *O*.

Output : The valid mask *ValidMask*.

Step 1: Create an empty node *x* and set it as the root node.

Step 2: Initialize $record[x] = ONE_n$, $class[x] = ONE_{\sigma_m}$ and $depth = 0$, where the variable *depth* is used to represent the depth of the node *x* in the cleansing tree.

Step 3: Set $px = x$, where *px* is used to keep the current parent node.

Step 4: If $class[x]$ is not equal to $UNIQUE_{\sigma_m}$ and *depth* is not equal to m-1, do Step 5 to build the child nodes of node *x;* otherwise, go to Step 7.

Step 5: Let C_j be the current feature in the spanned feature order to be considered.

For each feature-value vector F_{jk} in a feature matrix M_j for C_j, if (*record*[*px*] AND RV_{jk}) $\neq$ $ZERO_n$, do the following sub-steps:

Step 5.1: Create an empty node *y*.

Step 5.2: If *left_child*[*x*] = NIL, consider *y* as a child node of *x* and set *p*[*y*] = *x* and *left_child*[*x*] = *y*; otherwise, consider *y* as a sibling node of *x* and set *p*[*y*] = *p*[*x*] and *right_sibiling*[*x*] = *y*.

Step 5.3: Set *record*[*y*] = (*record*[*p*[*y*]] AND RV_{jk}) and *class*[*y*] = (*class*[*p*[*y*]] AND CV_{jk}).

Step 5.4: If *depth* = *m*-1 and *class*[*y*] $\neq$ $UNIQUE_{\sigma_m}$, set *ValidMask* = (*record*[*y*] XOR *ValidMask*).

Step 5.5: set *x* = *y*.

Step 6: If *left_child*[*px*] $\neq$ NIL, set *x* = *left_child*[*px*], *depth* = *depth* + 1 and go to Step 3. Otherwise, do the next step.

Step 7: If *right_sibiling*[*x*] $\neq$ NIL, *x* = *right_sibiling*[*x*] and go to Step 3; otherwise, set *x* = *p*[*x*] and do the next step.

Step 8: If *x* $\neq$ *Tree*[*root*], go to Step 7; otherwise, return *ValidMask* and stop the algorithm.

For example, the cleansing tree for the data in Table 7.1 with the spanned feature order <C_1, C_2, C_3, C_4> will be built as shown in Figure 7.3. At first, the root node is generated and all the bits in *record*[root] and *class*[root] are set to 1. Since *class*[root] is not equal to $UNIQUE_3$, and the current depth is 0, not equal to m-1, the next step is executed to build the child nodes of the root. The first feature C_1 in the spanned feature order is considered. Since it has three possible values and (*record*[root] AND RV_{1k}), k = 1 to 3, is not equal to $ZERO_{10}$, three nodes, represented as nodes 1, 2 and 3, are created as the children of the root. Since node 1, the left child node of the root, is not NIL, it is then processed to generate its child nodes in the same way. Nodes 4 and 5 are then created for the second feature C_2 in the spanned feature order. Since *class*[node 4] has been equal to ONE_{10}, the sibling of node 4, which is node 5, is then considered. Since *class*[node 5] has also been equal to ONE_{10}, the sibling of node 5, is then considered. But since node 5 has no sibling, its parent node, node 1 is considered. The sibling of node 1, which is node 2 is then processed. The same procedure is then executed until the whole cleansing tree is generated.

The numbers at the left of the nodes in Figure 7.3 indicate the order built. In node 15, the second and third bits of the class vector are both "1". It means that the corresponding record vectors will have more than one "1". The corresponding records with bit "1" are then inconsistent since their values of all condition features are the same, but their values of the decision feature are different. In this example, the ninth and tenth records are inconsistent. The *ValidMask* are thus modified from "1111111111" to "1111111100".

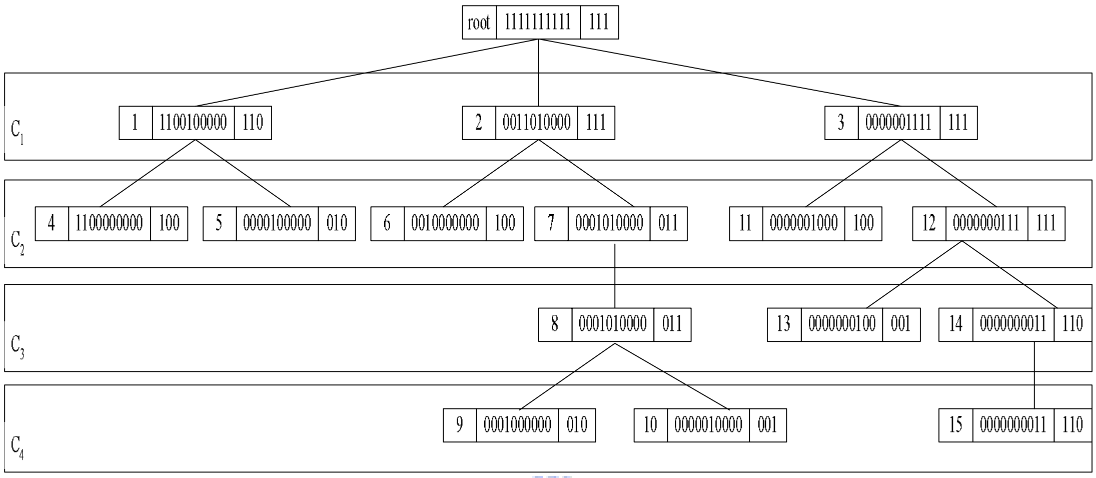


Figure 7.3: Cleansing tree with feature spanned order <C_1, C_2, C_3, C_4>

In the above example, the spanned feature order O is set as <C_1, C_2, C_3, C_4>. Different orders will apparently affect the performance of the cleansing spanning trees built. A cleansing spanning tree with a better spanned feature order can reduce the space and time complexities. In the past, there were some famous tree structures for classification, such as the decision-tree approach[58], which was based on the entropy theory to select the next best feature. In order to reduce the computational complexity for evaluating the spanning order of features, the following heuristics are thus proposed.

H1 : The more '1' bits a record vector for a feature value has, the more weight the feature value has.

H2 : The more '1' bit the class vector for a feature value has, the less weight the feature value has.

These two heuristics show the relationship between feature values and classes. If a feature value appears in most records with a single class, the weight of this feature value is relatively high. These heuristics can be used to save the computation time when compared to using the entropy theory. The following *Find span order algorithm* is thus proposed to determine the spanned feature sequence O of all condition features by evaluating the feature weights according to the above heuristics.

Algorithm 7.3 – *Find span order algorithm* :

Input: A feature matrix M for a table T

Output: A spanned feature order O.

Step 1: Initialize $weight_j = 0$, where $1 \leq j \leq m$-1.

Step 2: For each M_j in M, set:

$$weight_j \leftarrow \sum_{k=1}^{\sigma_j} \frac{Count(RV_{jk})}{[Count(CV_{jk})]^2},$$

where the function $Count(x)$ is used to count the number of '1' bits in x.

Step 3: Order the features in O in the descendent order of the *weight* values.

Step 4: Return O.

For example, according to the feature matrix in Table 7.2, the weight of each feature is calculated as shown in Table 7.3.

Table 7.3: Calculating the weight of each feature

Feature	Weight	Old Order	New Order
C_1	3/4+3/9+4/9=1.53	1	**4**
C_2	3/1+4/4+3/9=4.33	2	**2**
C_3	5/9+2/1+3/4=3.31	3	**3**
C_4	4/4+3/4+3/1=4.75	4	**1**

The new spanned feature sequence O determined by the above algorithm is thus $<C_4, C_2, C_3, C_1>$, instead of the original order $<C_1, C_2, C_3, C_4>$. The cleansing tree generated on the new order is shown in Figure 7.4.

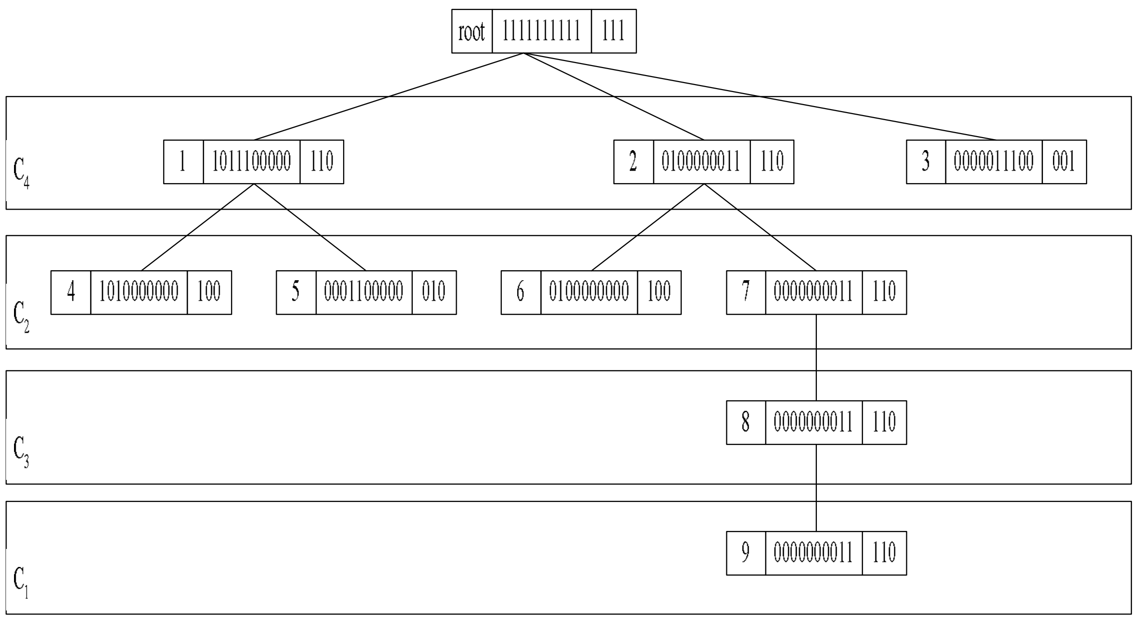


Figure 7.4: The cleansing tree generated on the new order <C_4, C_2, C_3, C_1>

As we can see, the cleansing tree with new feature order O=<C_4, C_2, C_3, C_1> in Figure 7.4 is much smaller than that in Figure 7.3. The number of nodes has decreased from 15 to 9. Therefore, the computational time of generating and traversing the spanning tree can be greatly reduced.

After the cleansing tree is built, the *ValidMask* may not be ONE_n since inconsistent records may exist. The *ValidMask* is then used by the following *Cleansing feature matrix algorithm* to remove the inconsistent records from the feature matrix.

Algorithm 7.4 – *Cleansing feature matrix algorithm* :

Input: A feature matrix M for a table T and a valid mask vector *ValidMask*.

```
Output:   A cleansed feature matrix $M$.

Step 1:   For each feature-value vector $F_{ij}$ in $M$, do following sub-steps:

    Step 1.1:   $RV_{ij} = RV_{ij}$ AND ValidMask.

    Step 1.2:   $CV_{ij}$ = Find class vector algorithm($RV_{ij}$).

Step 2:   Return $M$.
```

For example, the *ValidMask* is set to "1111111100" after the cleansing tree for Table 7.1 is built. Since the ninth and tenth bits of the *ValidMask* are 0, the *Cleansing feature matrix algorithm* will set these two bits of all the record vectors in Table 7.2 to 0. The class vector of each feature value is then recalculated by the *Find class vector algorithm* according to its new record vector. The revised feature matrix is shown in Table 7.4.

Table 7.4: The cleansed feature matrix of Table 7.2

Feature	**Feature-value**	**Record Vector**	**Class Vector**
C_1	V_{11}	1100100000	110
	V_{12}	0011010000	111
	V_{13}	0000001100	001

C_2	V_{21}	1110000000	100
	V_{22}	0001111000	011
	V_{23}	0000000100	001
C_3	V_{31}	1001011100	111
	V_{32}	0110000000	100
	V_{33}	0000100000	010
C_4	V_{41}	1011100000	110
	V_{42}	0100000000	100
	V_{43}	0000011100	001
C_5	V_{51}	1110000000	100
	V_{52}	0001100000	010
	V_{53}	0000011100	001

For effectively distinguishing the classes from the feature values, we must extend the concepts related to a single feature to a feature sets. The following definitions are thus needed.

DEFINITION 7.9 - power of a feature set :

C^s is called the *s*-power of a feature set *C*, if each element in C^s is composed of *s* distinct condition features from *C*, $1 \leq s \leq m$-1.

Thus, we have $C^1 = C$. For example, the power set C^1 for the data in Table 7.1 is

$\{\{C_1\}, \{C_2\}, \{C_3\}, \{C_4\}\}$. The power set C^2 is $\{\{C_1,C_2\}, \{C_1,C_3\}, \{C_1,C_4\}, \{C_2,C_3\}, \{C_2,C_4\}, \{C_3,C_4\}\}$. Let $|C^s|$ denote the cardinality of C^s. Then:

$$|C^s| = \binom{m-1}{s}.$$

Let C^s_j denote the j-th element in C^s, $1 \le j \le |C^s|$. C^s_j is then a feature set. Also let V^s_j denote the domain of C^s_j, σ^s_j denote the number of possible values in V^s_j, and V^s_{jk} denote the k-th feature value of C^s_j. Each feature set can be represented by a name vector, defined below.

DEFINITION 7.10 - name vector of a feature set :

The name vector NV^s_j of a feature set C^s_j is a bit string $b_1b_2\ldots b_{m-1}$, with b_i set to 1 if feature C_i is included in C^s_j and set to 0 otherwise.

For the above example, C^1_1 denotes the first element in C^1, which is $\{C_1\}$. The name vector NV^1_1 is then 1000 since only C_1 is included in C^1_1. For another example, $C^2 = \{\{C_1,C_2\}, \{C_1,C_3\}, \{C_1,C_4\}, \{C_2,C_3\}, \{C_2,C_4\}, \{C_3,C_4\}\}$. C^2_1 denotes the first element in C^2, which is $\{C_1, C_2\}$. The name vector NV^2_1 is then 1100 since C_1 and C_2 are included in C^2_1. Similar to a single feature, some terms related to a feature set is defined below.

DEFINITION 7.11 - record vector of a feature set :

A record vector RV^s_{jk} of a feature set value C^s_{jk} is a bit string $b_1b_2\ldots b_n$, with b_i set to 1 for $V^s_j(i) = V^s_{jk}$ and set to 0 otherwise, where $1 \leq j \leq |C^s|$ and $1 \leq k \leq \sigma^s_j$.

RV^s_{jk} thus keeps the information of the records with the k-th possible value of the feature set C^s_j. A class vector, which is used to keep the information of the classes (values of the decision feature) with a specific value of a feature set, is defined below.

DEFINITION 7.12 - class vector of a feature set :

A class vector of CV^s_{jk} of a feature set value C^s_{jk} is a bit string $b_1b_2\ldots b_{\sigma_m}$, with b_i set to 1 if $RV^s_{jk} \cap RV_{mi} \neq ZERO_n$, and set to 0 otherwise, where σ_m is the number of possible values of C_m and n is the number of records in R.

CV^s_{jk} thus keeps the information of the classes related to the k-th possible value of the feature set C^s_j. A feature-value vector of a feature set is defined below.

DEFINITION 7.13 - Feature-value vector of a feature set :

A feature-value vector F^s_{jk} is composed of RV^s_{jk} and CV^s_{jk}.

DEFINITION 7.14 - A feature matrix for a feature set :

A feature matrix M^s_j for the feature set C^s_j is denoted $\begin{bmatrix} F^s_{j1} \\ F^s_{j2} \\ : \\ F^s_{j\sigma_j} \end{bmatrix}$, where $1 \leq j \leq |C^s|$ and σ^s_j is the number of possible values in C^s_j.

DEFINITION 7.15 - *s*-feature matrix for a table *T* :

An *s*-feature matrix M^s for a table T is denoted $\begin{bmatrix} M^s_1 \\ M^s_2 \\ : \\ M^s_{|C^s|} \end{bmatrix}$, where $1 \leq s \leq m\text{-}1$.

Hereafter, two algorithms are proposed to find the desired feature set. The first algorithm, named the *Selecting feature set algorithm*, is used to find a feature set from a given *s*-feature matrix. If there exists a feature set which is sufficient to decide all the records in the given dataset, the feature set will be returned and the feature selection procedure stops. Otherwise, *s* is incremented and the *Selecting feature set algorithm* is executed again. The second algorithm, named the *Calculating next matrix algorithm*, derives the new feature matrix from the previous feature matrix. The *Selecting feature set algorithm* is described as follows.

Algorithm 7.5 – *Selecting feature set algorithm* :

Input: An s-feature matrix M^s for a table T.

Output: A selected feature set FS.

Step 1: Initialize $FS = \varnothing$, $j = 1$.

Step 2: If $j \leq |C^s|$, do the next step; otherwise go to Step 7.

Step 3: Set $k = 1$, where k is used to keep the number of the value currently processed in a feature set C^s_j.

Step 4: If $k \leq \sigma^s_j$, do the next step; otherwise go to Step 6.

Step 5: If $CV^s_{jk} \neq UNIQUE_{\sigma_m}$, set $j = j + 1$ and go to Step 2; otherwise set $k = k + 1$ and go to Step 4.

Step 6: Set $FS = C^s_j$; That is, for each i from 1 to m-1, set $FS = FS \cup \{C_i\}$ if the i-th bit of the name vector NV^s_j for feature set C^s_j is equal to 1.

Step 7: Return FS.

Take the data in Table 7.1 as an example to illustrate the above algorithm. s is set at 1 at the beginning. The *1*-feature matrix M^1 for the data is the same as the feature matrix M found before. The *Selecting feature set algorithm* will examine the 1-feature sets one by one. The first element M^1_1, which is $\{C_1\}$, is then processed. The class vector CV^1_{11} for the first feature value C^1_{11} is 110, which is not equal to $UNIQUE_3$.

Using the feature set $\{C_1\}$ can thus not completely distinguish the classes. The other elements in the *1*-feature matrix M^1 are then processed in a similar way. In this example, no element is chosen. Thus $\varnothing$ is returned. It means no single feature can completely distinguish the classes. s is then incremented, and the *Selecting feature set algorithm* is then executed from the new s-feature matrix. The new feature matrix can be easily derived from the previous feature matrix by the following *Calculating next matrix algorithm*.

Algorithm 7.6 - *Calculating next matrix algorithm* :

```
Input:    An $s$-feature matrix $M^s$ for a table $T$.
Output:   An $(s+1)$-feature matrix $M^{s+1}$ for a table $T$.
Step 1:   For each $j$, $j$ = 1 to $|C^s|$ - 1, do the following steps.
Step 2:   For each $l$, $l = (j \bmod m) + 1$ to $m$, do the following sub-steps.
    Step 2.1:   Set $NV^{s+1}_j = NV^s_j$ OR $NV^1_l$.
    Step 2.2:   Set the temporary counter $k$ to 1.
    Step 2.3:   For each feature-value vector $F^s_{jx}$ in $M^s_j$, $1 \le x \le |C^s_j|$, do the following
                sub-steps:
        Step 2.3.1:   For each feature-value vector $F^1_{ly}$ in $M^1_l$, $1 \le y \le |C^1_l|$, do the
                      following sub-steps:
```

```
        Step 2.3.1.1: Set $RV^{s+1}_{jk} = RV_{jx}$ AND $RV^{1}_{ly}$.
        Step 2.3.1.2: Set $CV^{s+1}_{jk} = CV^{s}_{jx}$ AND $CV^{1}_{ly}$.
        Step 2.3.1.3: IF $CV^{s+1}_{jk} \neq UNIQUE_{\sigma_m}$, set $CV^{s+1}_{jk}$ = Find class
                      vector algorithm($RV^{s+1}_{jk}$).
        Step 2.3.1.4: Set $k = k+1$.
Step 3: Return the (s+1)-feature matrix $M^{s+1}$.
```

For example, the *2*-feature matrix M^2 for the data in Table 7.1 is generated from the *1*-feature matrix M^1 as follows. The name vector for feature C^2_1 is first calculated. Thus:

$NV^2_1 = NV^1_1$ OR NV^1_1

= 1000 OR 0100

= 1100.

The feature-value vector F^2_{11} in M^2_1 is then calculated. The record vector is found as follows:

$RV^2_{11} = RV^1_{11}$ AND RV^1_{21}

= 1100100000 AND 1110000000

= 1100000000.

The class vector is found as follows:

$CV^2_{11} = CV^1_{11}$ AND CV^1_{21}

= 110 AND 100

= 100.

In a similar way, all the feature-value vectors in the *2*-feature matrix M^2 can be found. The results are shown in Table 7.5:

Table 7.5: The *2*-feature matrix M^2 found by the *Calculating next matrix algorithm*

Feature Set	**Feature Set Value**	**Name Vector**	**Record Vector**	**Class Vector**
C^2_1	V^2_{11}	1100	1100000000	100
	V^2_{12}	1100	0000100000	010
	V^2_{13}	1100	0010000000	100
	V^2_{14}	1100	0001010000	011
	V^2_{15}	1100	0000001000	001
	V^2_{16}	1100	0000000100	001
C^2_2	V^2_{21}	1010	1000000000	100
	V^2_{22}	1010	0100000000	100
	V^2_{23}	1010	0000100000	010
	V^2_{24}	1010	0001010000	011
	V^2_{25}	1010	0010000000	100
	V^2_{26}	1010	0000001100	001

$C^2{}_3$	$V^2{}_{31}$	1001	1000100000	110
	$V^2{}_{32}$	1001	0100000000	100
	$V^2{}_{33}$	1001	0011000000	110
	$V^2{}_{34}$	1001	0000010000	001
	$V^2{}_{35}$	1001	0000001100	001
$C^2{}_4$	$V^2{}_{41}$	0110	1000000000	100
	$V^2{}_{42}$	0110	0110000000	100
	$V^2{}_{43}$	0110	0001011000	011
	$V^2{}_{44}$	0110	0000100000	010
	$V^2{}_{45}$	0110	0000000100	001
$C^2{}_5$	$V^2{}_{51}$	0101	1010000000	100
	$V^2{}_{52}$	0101	0100000000	100
	$V^2{}_{53}$	0101	0001100000	010
	$V^2{}_{54}$	0101	0000011000	001
	$V^2{}_{55}$	0101	0000000100	001
$C^2{}_6$	$V^2{}_{61}$	0011	1001000000	110
	$V^2{}_{62}$	0011	0000011100	001
	$V^2{}_{63}$	0011	0010000000	100
	$V^2{}_{64}$	0011	0100000000	100
	$V^2{}_{65}$	0011	0000100000	010

Note that in Step 2.3.1.2, the *class* vector derived by the bit-wise "AND" operator denotes only the "possible" class distribution. For example, the feature-value vector $F^2{}_{21}$ consists of $RV^2{}_{21}$ = "1000000000" and $CV^2{}_{21}$ = "110" after Step 2.3.1.2. Since each record belongs to only one class, the above results are not correct. In fact, the class vector $CV^2{}_{21}$ = "100". Step 2.3.1.2 is used as a quick check. If $CV^{s+1}{}_{jk} \neq UNIQUE_{\sigma_m}$, then the *Find class vector algorithm* is run in Step 2.3.1.3 to find the correct class vector.

After the new feature matrix is derived, the *Selecting feature set algorithm* is then executed again to find an appropriate feature set. For the above example, the *2*-feature matrix M^2 is then input to the *Selecting feature set algorithm* and the feature set $FS = \{C_2, C_4\}$ are found and returned as the solution.

After the above method is executed, the feature set *FS* to classify the given data set *T* is generated. *FS* may be over-fitting or under-fitting for the problem since they are derived only according to the current data set. These features are then evaluated and modified by domain experts. They thus serve as the candidates for the experts to have a good initial standpoint.

7.3 Complexity Analysis and Experiments

The time and space complexities of the proposed algorithms are analyzed in this section. Let *n* be the number of records, *m* be the number of features and *c* be the number of classes. Also define *i* as the maximum possible number of features in a feature set, *j* as the maximum number of possible values of a feature, and *s* as the number of iterations. The time complexity and space complexity of each step in the *Find class vector algorithm* is shown in Table 7.6.

Table 7.6: The time and space complexities of the *Find class vector algorithm*

Step No	Time Complexity	Space Complexity
Step 1	***O*(1)**	***O*(*c*)**
Step 2	***O*(*jc*)**	***O*(*jc*)**
Step 3	***O*(1)**	***O*(*c*)**
Total	***O*(*jc*)**	***O*(*jc*)**

The time and space complexities of each step in the *Create cleansing tree algorithm* is shown in Table 7.7. Note that the maximum amount of nodes within a *Ctree* is *n*.

Table 7.7: The time and space complexities of the *Create cleansing tree algorithm*

Step No	Time Complexity	Space Complexity
Step 1	***O*(1)**	***O*(1)**
Step 2	***O*(1)**	***O*(1)**
Step 3	***O*(1)**	***O*(1)**
Step 4	***O*(*nmj*)**	***O*(*n*)**
Step 5	***O*(*mj*)**	***O*(*n*)**
Step 6	***O*(1)**	***O*(1)**
Step 7	***O*(1)**	***O*(1)**
Total	***O*(*nmj*)**	***O*(*n*)***

The time and space complexities of each step in the *Find span order algorithm* is shown in Table 7.8:

Table 7.8: The time and space complexities of the *Find span order algorithm*

Step No	Time Complexity	Space Complexity
Step 1	$O(m)$	$O(m)$
Step 2	$O(cm)$	$O(cm)$
Step 3	$O(c\mathrm{lg}c)$	$O(c)$
Step 4	$O(1)$	$O(1)$
Total	$O(Max(cm, clgc))$	$O(cm)$

The time and space complexities of each step in the *Cleansing feature matrix algorithm* is shown in Table 7.9:

Table 7.9: The time and space complexities of the *Cleansing feature matrix algorithm*

Step No	Time Complexity	Space Complexity
Step 1	$O(mj)$	$O(mj)$
Step 2	$O(1)$	$O(1)$
Total	$O(mj)$	$O(mj)$

The time and space complexities of each step in the *Selecting feature set algorithm* is shown in Table 7.10.

Table 7.10: The time and space complexities of the *Selecting feature set algorithm*

Step No	Time Complexity	Space Complexity
Step 1	$O(1)$	$O(1)$
Step 2	$O(m^s j^s)$	$O(1)$
Step 3	$O(1)$	$O(1)$

Step 4	$O(j^s)$	$O(1)$
Step 5	$O(1)$	$O(1)$
Step 6	$O(c)$	$O(c)$
Step 7	$O(1)$	$O(1)$
Total	$O(m^s j^s)$	$O(c)$

The time and space complexities of each step in the *Calculating next matrix algorithm* is shown in Table 7.11:

Table 7.11: The time and space complexities of the *Calculating next matrix algorithm*

Step No	Time Complexity	Space Complexity
Step 1	$O(m^s j^s)$	$O(m^s j^s)$
Step 2	$O(mj)$	$O(mj)$
Step 3	$O(1)$	$O(j)$
Total	$O(m^s j^s)$	$O(m^s j^s)$

To evaluate the performance of the proposed method, we compare it with other feature selection methods. Our target machine is a Pentium III 1G Mhz processor system, running on the Microsoft Windows 2000 multithreaded OS. The system includes 512K L2 cache and 256 MB shared-memory.

Several datasets from the UCI Repository [60] are used for the experiments. These datasets have different characteristics. Some have known relevant features (such as Monks), some have many classes (such as SoybeanL), and some have many instances (such Mushroom). In addition, a large real data set about endowment insurances from a world-wide financial group is used to examine the usability of the proposed method. Experimental results show the proposed method can discover the desired feature sets and can thus help the enterprise to build a CBR system for their loan promotion function of customer relationship management system. The data set of insurance data uses 27 condition features to describe the states of 3 different insurance types. Different types of attribute values including date/time, numeric and symbolic data exist. They are all transformed into the symbolic type by some clustering methods. Six of them have missing values.

The characteristics of the above datasets are summarized in Table 7.12.

Table 7.12: The datasets used in the experiments

Database Name	Class No.	Condition Feature No.	Record No.	Missing Features
Monk1	2	6	124	no
Monk2	2	6	169	no
Monk3	2	6	122	no
Vote	2	16	300	no
Mushroom	2	22	8124	Yes
SoybeanL	19	35	683	Yes
Insurance	3	27	35000	Yes

In the experiments, the accuracy, the number of selected features, and the time will be compared between our method and the traditional rough set method. The accuracy is measured by the classification results of the target table. If the selected feature set can solve the problem without any error, 100% accuracy is reached; otherwise the accuracy is calculated by the number of correctly classified records over the total number of records. Experimental results show both methods can reach 100% accuracy. We then compare the feature sets found by these two approaches. The results are shown in Table 7.13. Obviously, the accuracy of all datasets is 100% since both of these two method discover the minimal feature sets.

Table 7.13: The selected feature sets found by the two approaches.

	Traditional Rough Set Approach	**Bitmap-based Approach**	**Accuracy**
Dataset	**Feature Set**	**Feature Set**	**100%**
Monk1	C1, C2, C5	C1, C2, C5	**100%**
Monk2	C1-C6	C1-C6	**100%**
Monk3	C1, C2, C4, C5	C1, C2, C4, C5	**100%**
Vote	C1-C4, C9, C11, C13, C16	C1-C4, C9, C11, C13, C16	**100%**
Mushroom	C3, C4, C11, C20	C3, C4, C11, C20	**100%**
SoybeanL	Need too much computation time.	C14, C20, C26, C27, C29, C30, C31, C32, C33, C34, C35	**100%**
Insurance	C4, C15, C17, C20, C22, C25	C4, C15, C17, C20, C22, C25	**100%**

Note that there may be more then one solution for the selected features. In Table 7.13, only the first selected feature set (in the alphabetical order) is listed. It is easily seen that the selected feature sets of our proposed approach and the traditional rough set approach are the same except for the SoybeanL problem. The SoybeanL problem needs too much computation time by the traditional rough set approach.

The numbers of the selected features by the two approaches are shown in Table 7.14. Both methods get the same numbers for all problems except for SoybeanL.

Table 7.14: The number of the selected features found by the two approaches.

Dataset	Traditional RS	Bitmap-based
Monk1	3	3
Monk2	6	6
Monk3	4	4
Vote	8	8
Mushroom	4	4
SoybeanL	11	11
Insurance	6	6

At last, the computation time is compared. The data sets are first loaded into the memory from the hard disk and the processing times are measured. The time is rounded to 0 if the real time is less than 0.001 seconds. The results are shown in Table 7.15.

Table 7.15: The CPU times needed by the two approaches

Dataset	Traditional RS	Bitmap-based
Monk1	0.07	0
Monk2	0.351	0.01
Monk3	0.141	0
Vote	428.19	1.923

Mushroom	4911.32	27.91
SoybeanL	>1000000	247805
Insurance	468656	2435.66

Consistent with our expectation, the proposed approach is much faster than the traditional rough set approach. Especially for the Insurance data, our approach needs only about 40 minutes, but the traditional rough set approach needs much more computation time.

In this chapter, we have proposed a bit-based feature selection approach to discover optimal feature sets for the given table(dataset). In this approach, the feature values are first encoded into bitmap indices for searching the optimal solutions efficiently. Also, the corresponding indexing and selecting algorithms are described in details for implementing the proposed approach. Experimental results on different data sets have also shown the efficiency and accuracy of the proposed approach.

The traditional rough-set approach has two very time-consuming parts, combination of features and comparison of upper/lower approximations. In this method, we use the single-time-clock bit-wise operations to shorten the computation time of the comparison part. Moreover, the workload in the combination part is highly reduced since the new levels of combination can be generated via the pervious ones. The bit-wise operations are also used to speed up the combination generation. The proposed feature-selection approach also adopts appropriate meta-data structures to take advantages of the computational power of the bit-wise operations.

The feature selection problem is generally an NP-complete problem. Although the proposed approach can process a larger amount of features than the traditional rough-set approach, it still becomes unmanageable especially when the number of features is huge or when the number of possible values of features is large. In the future, we will continuously investigate and design efficient heuristic approaches to manage huge amounts of features and possible values. We will also attempt to integrate different feature selection approaches to automatically select an appropriate one for optimal or near-optimal solutions according to the characteristics of given data sets.

Chapter 8
Using BWI Indexing in Semiconductor Manufacturing Defect Detection Systems

In this chapter, an unsupervised-learning data-driven data mining system of a production-level defect detection system in an intelligent engineering data analysis (*iEDA*) system in Taiwan Semiconductor Manufacturing Company Ltd. (TSMC) is introduced. The bit-wise indexing methods (including Sample, Encapsulated and Compact Bit-wise Indexing Methods), Data Mining Technologies, and Statistic Methods are hybridly used in this application in order to generate the possible root-cause candidate list for the given manufacturing details of an individual low-yield situation event. Also, some critical issues about of applying a data mining solution for manufacturing defects detection system in semiconductor manufacturing domain will be discussed and reviewed. Finally, we will propose the system framework of the next-generation data mining solution in the future for providing a more knowledgeable, reasonable, reliable and flexible solution for data mining solution in the semiconductor manufacturing domain.

8.1 Problem Description

In recent years, the procedures of manufacturing have become increasingly complex [16][17][18]. To meet high expectations regarding yield targets, rapidly identifying the root causes of defects is essential for meeting high expectations regarding yield targets. Therefore, the technologies of process control, statistical analysis and experiment design are used to establish a solid base for well tuned manufacturing processes. However, identifying root cause remains extremely difficult due to multi-factor and nonlinear interactions in this intermittent problem. Traditionally, the process of identifying root cause of defects is costly. The semiconductor manufacturing industry provides an example. With a huge amount of semiconductor engineering data stored in the database and versatile analytical charting and reporting in production and development, the CIM/MES/EDA systems in most semiconductor manufacturing companies help users analyze the collected data to achieve the goal of yield enhancement. However, semiconductor manufacturing procedures are sophisticated, and thus multi-dimensional and large volumes of data are required to be collected for these procedures. Data mining technologies [4][3][9][33] are employed to deal with such large amounts of high-dimensional data [6][16][17][29][41][51][52][59]. In this chapter, we propose a data mining system and describe the experience of applying such systems for discovering the root causes of low-yield situations in TSMC [16][17]. Additionally, the evaluation of applying such a mining system for manufacturing defect detection in the semiconductor manufacturing domain is discussed and reviewed. Finally, a new architecture for a reasonable, reliable and flexible defect detection platform based on the data mining approach is briefly described.

8.2 DM Project for Yield Enhancement

In June 2002, a research project on data mining techniques was triggered by the Manufacturing Information Technology Division of Taiwan Semiconductor Manufacturing Company. Five test cases were conducted, including partial lot-based information, WIP information, CP information, In-line metrology results, WAT results and some manufacturing parameters. Each case represents a low-yield situation with an already discovered root cause related to some manufacturing procedure; however, all of the cases require extensive trouble-shooting time. Based on the given cases, a prototype of the data-driven data mining system is required to discover the possible root causes for the subject cases. Since a large amount of data on this company exists, the data mining system only discovers the killer machines for the cases that were prepared by product engineers in the event of an abnormal manufacturing situation. Additionally, the attribute weights in the given cases are initially treated as equal because of the lack of previous built-in knowledge. Also, this engine is required to be noise-insensitive since noise is difficult to filter in semiconductor yield enhancement applications.

After discussing this project, the data mining system should be designed according to the following criteria:

1. Platform criterion: The data mining system needs to be executed in both server-end and client-end applications according to the functional specification of an iEDA (Intelligent Engineering Data Analysis) system in TSMC.
2. Development environment criterion: The data mining system should be developed as some independent functional modules due to the system integration and platform issue; and a prototype system integrating all proposed modules should be provided for testing and evaluation via TSMC.
3. Given data set criterion: Since the EDA system involves a vast and still growing quantity of data, it seems impossible to analyze all manufacturing data in the EDA system via the data mining system. The data mining system is designed for

analyzing a pre-generated data set in the event of a low-yield situation. Restated, the input data for the data mining system should be generated as a low-yield situation case. Some lot-based manufacturing information is involved in this low-yield situation case, and each case comprises a maximum of six segments, including basic lot information, WIP information, CP information, in-line metrology results, WAT results and other manufacturing parameter segments, and a unique decision feature used to classify the high and low yield group of given lots. As mentioned above, the data mining system is designed as a data-driven solution, and no previous knowledge is built to recognize the attribute catalog and type, with the attributes of all given cases that are processed by the data mining system being named according to the pre-defined naming rules. Furthermore, the user-prepared data files are acceptable only if the naming rules of attributes are followed.

4. Accuracy criterion: In this data mining project, the accuracy rate should exceed 80% in all cases. The percentage of hit cases thus should exceed 80%, where a hit case means that the real root cause ranks within the top five rankings on the possible root cause ranking list.
5. Efficiency criterion: The procedure of the mining engine should be completed within one minute using the benchmark case involving 300 lots and 13000 attributes for each lot.

The above criteria are incorporated into a data mining system scenario through the following procedures:

1. Data preparation procedure: The raw data of cases are first retrieved from the EDA database and then transformed into Bit-wise Indexing (BWI) matrixes [10] to accelerate the subsequent mining procedure. Figure 8.1 illustrates three major functional modules, including the Data Quality Analysis, Cutting- Point Calculating and Data Dispatcher modules, in this processing phase. Since semiconductor manufacturing processes have become increasingly sophisticated, data collection problems also have become increasing serious, particularly when using advanced technologies. Generally, in a spit lot situation, sparse and null data issues may seriously impact the accuracy of the data mining results. The Data Quality Analysis module is then employed to check quality of a given data set based upon our proposed quality indicators. This function also provides lot and attribute merging mechanisms in order to help users for combining the spit lot or procedure step in the given data set. When the quality of the given data set is confirmed by the user, a decision feature is required for judging the lot information within the whole data set. The decision feature of this data set is used to separate all given lots into two independent groups, called normal and abnormal lot group. After the decision

feature is selected, the Cutting-Point Calculating module is executed to determine whether the normal lot group is located at the right-hand (larger than) or left-hand (smaller than) side of the given critical point. Certainly, users can define these two parameters by themselves based on different situations. Since decision feature and cutting-point are selected, the Data Dispatcher module has been used to dispatch some individual data segments for data mining according to the naming rules, and the corresponding BWI matrixes thus are generated.

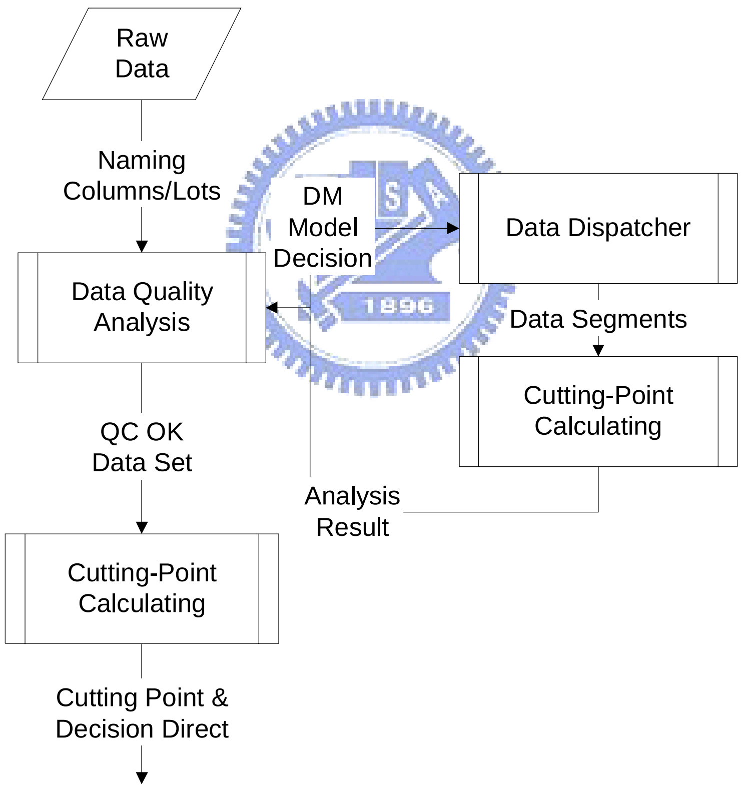


Figure 8.1: the flowchart of data preparation procedure

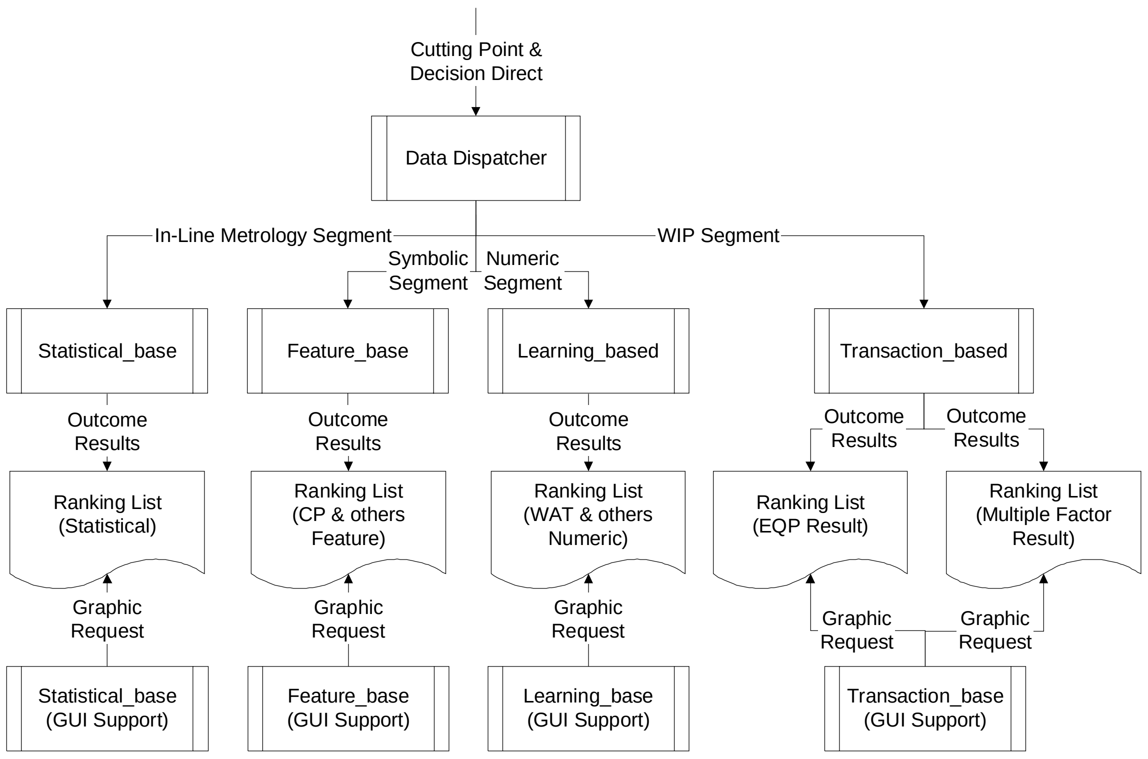

Cutting Point & Decision Direct
Data Dispatcher
In-Line Metrology Segment
WIP Segment
Symbolic Segment
Numeric Segment
Statistical_base
Feature_base
Learning_based
Transaction_based
Outcome Results
Outcome Results
Outcome Results
Outcome Results
Outcome Results
Ranking List (Statistical)
Ranking List (CP & others Feature)
Ranking List (WAT & others Numeric)
Ranking List (EQP Result)
Ranking List (Multiple Factor Result)
Graphic Request
Graphic Request
Graphic Request
Graphic Request
Graphic Request
Statistical_base (GUI Support)
Feature_base (GUI Support)
Learning_base (GUI Support)
Transaction_base (GUI Support)

Figure 8.2: the flowchart of data mining procedure

2. Data mining procedure: Once the target BWI matrixes are fully prepared and the data quality is verified, the data mining procedure is triggered to analyze the content of cases and discover the root causes for the target cases. Figure 8.2 briefly describes four major data mining modules, including the Transaction-based, Learning-based, Feature-based and Statistical-base modules, as presented below:

i) Transaction-based module: Generally, over 80% of low-yield situations in the semiconductor manufacturing result from machine failure, and it is extremely difficult to determine the degree to which each machines contributes to failure during the manufacturing procedure. The root causes for production of low-yield wafers are hard to determine, since yield can not be qualified during the manufacturing process. Generally, product engineers require some data analysis methods for identifying evidence regarding possible root cause. According to the experience of domain experts, methods based on single variable analysis usually have seldom null-value tolerant ability. Therefore, these methods are not quite suitable for seeking the root cause machine for the semiconductor manufacturing domain. To solve the above problem, the Transaction-based module, including equipment and multiple factor mining function, is applied to analyze the WIP data segment to discover each killer machine through a hybrid data mining method. The equipment mining function is used to rank all possible killer machines in a given WIP data segment based on the confidence of mining result [16][17]. That is, this function is used to discover abnormal machine behavior by analyzing the manufacturing and machine logs. Moreover, semiconductor wafers usually have one silicon subtract and several metal and dielectric layers. This arrangement implies that some steps may be repeatedly executed by a killer machine which influences the yield of all bypass wafers. Therefore, the multiple factor mining function is proposed to handle the case of equipment failure related to the descending yield for repeated manufacturing using a single machine. Since the mining method used in this module integrates some data mining methods of transaction analysis, it is named the Transaction-based module.

ii) Learning-based module: This module is used to process all of the numerical data except the in-line metrology measurement part in the given data set to identify the abnormal behavior of all numeric data in the given data segment. Initially, each attribute in the give data segment is separated into normal/abnormal groups according to the decision feature. A learning procedure then is triggered for identifying the behavior pattern for each attribute based on the distribution and trend of the normal group for this attribute. Once the attribute behavior pattern is learned, the degree of abnormality of the corresponding abnormal lots group is judged and highlighted. Finally, the possible ranking list and corresponding mining result charts are obtained. Since the resource of learning procedure only includes the given data segment, the over or under fitting problems remain unsolved.

iii) Feature-based module: this module is used to process the symbolic and data/time data in the given data set to determine the root cause of some recipes, programs and tester changes. All attribute values of a given attribute in the given segments are partitioned into normal/abnormal groups according to the distribution of corresponding groups of decision features. Since the corresponding groups are separated, the similarity degree can be calculated with the proposed feature similarity calculation method and the ranking list is thus proposed to users.

iv) Statistical-base module: This module is used to process the in-line metrology measurement segment of the given data set. Since the measurement results of in-line metrology are randomly sampled and only three or five wafers are measured in each metrology, the existing lots of null values may influence the accuracy of the data mining results. Therefore, statistical correlation analysis is used to process the data in the sparse data set, and may reduce the impact of quality issues on the given data. The list of attributes in this data set is proposed and ranked based on the resulted correlation degrees.

3. Possible root cause ranking list generation procedure: After the execution of the appropriate data mining modules, the possible root cause ranking lists are generated and the corresponding evaluation indexes obtained. Furthermore, the corresponding charts of the result of each module are provided to help product engineers realize the results of the data mining system.

8.3 Evaluation Result for the Yield Enhancement DM Project

From June to September 2002, the proposed data mining system had successfully highlighted the root causes within top five ranking in the generated ranking list for 15 of the 19 real cases evaluated by proposed data mining system within a maximum of 40 seconds. Two cases were useless due to the data preparation (Case 10 - too few lots available for mining) and multiple/combination killer machines (Case 12 – a combined machine issue) issues. The accuracy rate is approximately 88%. According to these excellent results, TSMC decided that the data mining module should be embedded into a new function, called *Yield Explorer*, of the iEDA system in TSMC. After the new function was released in September 2002, five of the newly received 23 cases could not be processed due to the out-of-mining-scenario problem (Case 20 – queuing-time issue) and multiple/combination killer machines issues (Case 20, 21, 22 and 23 – multiple machines/steps issue), two cases are still undergoing further investigation, and ten hit cases were obtained from the remaining 16 cases, as listed in Table 8.1. The performance evaluation about using BWI Indexing is listed in Table 8.2. The accuracy rate decreases from 88% to 63%, and the reasons for this lower accuracy rate are briefly described below:

1. Data Preparation Problem：Before the announcement of the data mining solution at TSMC, all testing cases were carefully reviewed from the perspectives of both data preparation and quality. The hit rate of all testing cases in this stage is extremely high. After the *Yield Explorer* function of

iEDA system was released, the data preparation and quality tasks for data mining system can be executed by all product engineers in TSMC. Even with enough training of this function, the concept of data mining remains difficult for all users to understand so quickly. From our observation, 70% of failure cases result from inappropriate data preparation procedures, which raise the problem of data preparation. The following interesting problems should be discussed.

a. What is a case? - The input of the proposed data mining is a lot data set with a single root cause, and this data set is judged by a single decision feature to distinguish the normal and abnormal lot groups for further mining. Product engineers attempt to identify the reasons for low-yield situations by examining the situation itself or analyzing the appropriate data set. In this situation, the task of generating a suitable data set becomes important for low-yield situation analysis through data mining systems. (For example, in response to a low-yield situation that occurred on October 15, product engineers prepared all lot-based data between October 1 and October 30). It implies that the low yield lots in the prepared data set may comprise not only the root cause affected lots, but also the regular low yield lots, since the duration of the given data set is not evaluated carefully, and it is extremely difficult to differentiate between the above two varieties lot without any meta-knowledge. Consequently, the result of data mining systems may be incorrect. On the other hand, the product engineer can prepare a suitable data set only when the root cause of low yield situation is most likely discovered. However, discovering the root cause through data mining becomes unimportant in this situation.

b. What is a root cause? - The root cause of a low-yield situation is the major reason for the low-yield situation in a regular manufacturing procedure, for example, the root cause is first defined as the machine that affects more than 50% of the low-yield lots in the given data set. In our experience, it seems not possible to prepare such "perfect" data set before the root cause still unknown. Therefore, the definition of root cause is further modified to be the machine that affects the most low-yield lot in the given data set. Even that, the root cause machine involved root of Case 12 and 33 are only 15% and 20%, respectively, in the prepared data sets from the corresponding product engineers. Moreover, it is very difficult to prepare a suitable data containing just one root cause from the perspective of the product engineers, and thus the real root cause may not be highlighted correctly.

c. What is the time duration? – When a low-yield situation occurs, the product engineer must determine the most likely time duration required to generate a suitable data set for the data mining system. Generally, the time duration of a low-yield situation is defined by the product engineer according to their personal experience. Since data mining system is highly sensitive for prepared data set, time duration becomes a problem.

Table 8.1: The evaluation cases in TSMC data mining project

No	Lot Number	Column Number	Problem	Root Cause	Rank
1	77	1397	CP Low Yield	Tool Issue	1
2	51	3402	CP Bin Wafer-edge Fail	Inline Metrology	2
3	154	3953	CP Low Yield	Tool Issue	1
4	34	3362	CP Bin Fail	CP Test Program	1
5	277	3356	FT Bin Fail	Tool Issue	3
6	272	2491	CP Low Yield	Tool Issue	X
7	146	3183	CP Low Yield	Tool Issue	1
8	141	3135	CP Low Yield	Tool Issue	3
9	54	3149	CP Low Yield	Tool Issue	2
10	4	2719	CP Wafer-Ring Fail	Tool Issue	X
11	8	1653	CP Bin Fail	Tool Issue	4
12	54	2376	CP Low Yield	Tool Issue	X
13	116	2884	CP Bin Fail	Tool Issue	1
14	313	3369	WAT Fail	Tool Issue	1
15	53	2462	CP Bin Fail	Tool Issue	5
16	484	2903	CP Bin Fail	Tool Issue	2
17	189	2809	CP Bin Fail	Tool Issue	X

18	106	2616	CP Bin Fail	Tool Issue	2
19	13	2071	CP Low Yield	Tool Issue	4
20	168	1797	WAT fail	Tool Issue	10
21	371	1983	CP Bin Fail	Tool Issue	15
22	72	2469	CP Bin Fail	Tool Issue	16
23	60	2183	CP Low Yield	Tool Issue	19
24	91	2511	WAT Fail	Unknown	Unknown
25	77	2447	CP Bin Fail	Tool Issue	1
26	40	1659	CP Low Yield	Tool Issue	3
27	72	2137	CP Bin Fail	Tool Issue	2
28	23	3500	CP Bin Fail	Queue-time Issue	X
29	59	1259	CP Bin Fail	Testing-tool Issue	1
30	133	1389	CP Bin Fail	Tool Issue	3
31	74	1239	CP Bin Fail	Tool Issue	1
32	168	4172	CP Bin Fail	Unknown	Unknown
33	102	2744	CP Bin Fail	Tool Issue	11
34	102	2744	CP Bin Fail	Tool Issue	3
35	136	1197	CP Bin Fail	WAT Parameter	5
36	33	3877	CP Bin Fail	WAT Parameter	1
37	167	1642	CP Bin Fail	WAT Parameter	X
38	65	1189	CP Bin Wafer-edge Fail	Tool Issue	1
39	50	1095	CP Bin Wafer-center Fail	Tool Issue	27
40	48	1290	CP Low Yield	Tool Issue	1
41	92	1198	CP Low Yield	Tool Issue	X
42	68	1203	CP Bin Fail	Tool Issue	17

Table 8.2: The performance evaluation of all TSMC cases in this data mining project

No	Total Cells	Storage/Query Solution Processing Time (Seconds)							Time Saving			
		Database Solution		In-Memory Computing Solution		BWI Indexing Structure			v.s. DB Solution		v.s. Memory Solution	
		Storage Access Time	Query Time	Storage Access Time	Query Time	Storage Access Time	Query Time	BWI Build-up Time (Sample)	Secs	Percent	Secs	Percent
1	107569	107	663	45	104	2	4	19	745	96.75%	124	83.22%
2	173502	173	1222	67	152	3	6	24	1362	97.63%	186	84.93%
3	608762	482	3150	190	419	10	19	35	3568	98.24%	545	89.49%
4	114308	102	706	32	70	2	4	19	783	96.91%	77	75.49%
5	929612	759	3939	247	529	15	28	171	4484	95.44%	562	72.42%
6	677552	724	4230	159	333	10	19	29	4896	98.83%	434	88.21%
7	464718	482	4184	108	225	7	13	56	4590	98.37%	257	77.18%
8	442035	552	5913	88	179	6	11	60	6388	98.81%	190	71.16%
9	170046	181	1642	35	73	2	4	24	1793	98.35%	78	72.22%
10	10876	11	106	2	4	0	0	5	112	95.73%	1	16.67%
11	13224	13	122	3	5	0	0	5	130	96.30%	3	37.50%
12	128304	131	1194	23	44	2	3	41	1279	96.53%	21	31.34%
13	334544	399	2183	52	98	5	8	32	2537	98.26%	105	70.00%
14	1054497	1368	6596	149	278	15	25	115	7809	98.05%	272	63.70%
15	130486	151	834	17	32	2	3	20	960	97.46%	24	48.98%
16	1405052	1628	11035	196	362	20	33	380	12230	96.58%	125	22.40%
17	530901	843	5440	81	151	8	13	64	6198	98.65%	147	63.36%
18	277296	387	4099	38	69	4	6	35	4441	99.00%	62	57.94%
19	26923	38	367	3	6	0	1	7	397	98.02%	1	11.11%
20	301896	481	4519	36	63	4	7	7	4982	99.64%	81	81.82%
21	735693	1400	10106	88	154	11	17	18	11460	99.60%	196	80.99%
22	177768	323	3041	21	36	3	4	5	3352	99.64%	45	78.95%
23	130980	191	2952	14	23	2	3	2	3136	99.78%	30	81.08%
24	228501	255	5435	28	49	3	5	10	5672	99.68%	59	76.62%
25	188419	244	4609	20	35	3	4	4	4842	99.77%	44	80.00%
26	66360	78	1598	7	11	1	1	1	1673	99.82%	15	83.33%

27	153864	158	2558	15	24	2	3	4	2707	99.67%	30	76.92%
28	80500	66	1791	8	13	1	2	1	1853	99.78%	17	80.95%
29	74281	52	1367	7	11	1	2	1	1415	99.72%	14	77.78%
30	184737	153	3604	19	32	3	4	3	3747	99.73%	41	80.39%
31	91686	91	1707	9	14	1	2	2	1793	99.72%	18	78.26%
32	700896	660	10767	79	133	11	17	18	11381	99.60%	166	78.30%
33	279888	318	4815	28	46	4	6	8	5115	99.65%	56	75.68%
34	279888	381	3604	27	44	4	6	7	3968	99.57%	54	76.06%
35	162792	249	2719	17	28	2	4	4	2958	99.66%	35	77.78%
36	127941	215	1799	12	19	2	3	3	2006	99.60%	23	74.19%
37	274214	416	2947	23	37	4	6	7	3346	99.49%	43	71.67%
38	77285	95	682	6	10	1	2	2	772	99.36%	11	68.75%
39	54750	77	409	5	8	1	1	1	483	99.38%	10	76.92%
40	61920	76	406	5	8	1	1	1	479	99.38%	10	76.92%
41	110216	159	798	8	13	2	2	2	951	99.37%	15	71.43%
42	81804	125	379	6	10	1	2	2	499	99.01%	11	68.75%
Avg	**291107**	**352**	**3101**	**48**	**94**	**4**	**7**	**30**	**3412**	**98.68%**	**101**	**69.31%**

As we can see, the processing time, including storage access time and query time, of BWI indexing solution using 1/12 time to compete the data mining procedure of the proposed data mining engine rather then In-memory computing solution, the time cost of three storage/query solutions are shown in Figure 8.3 .

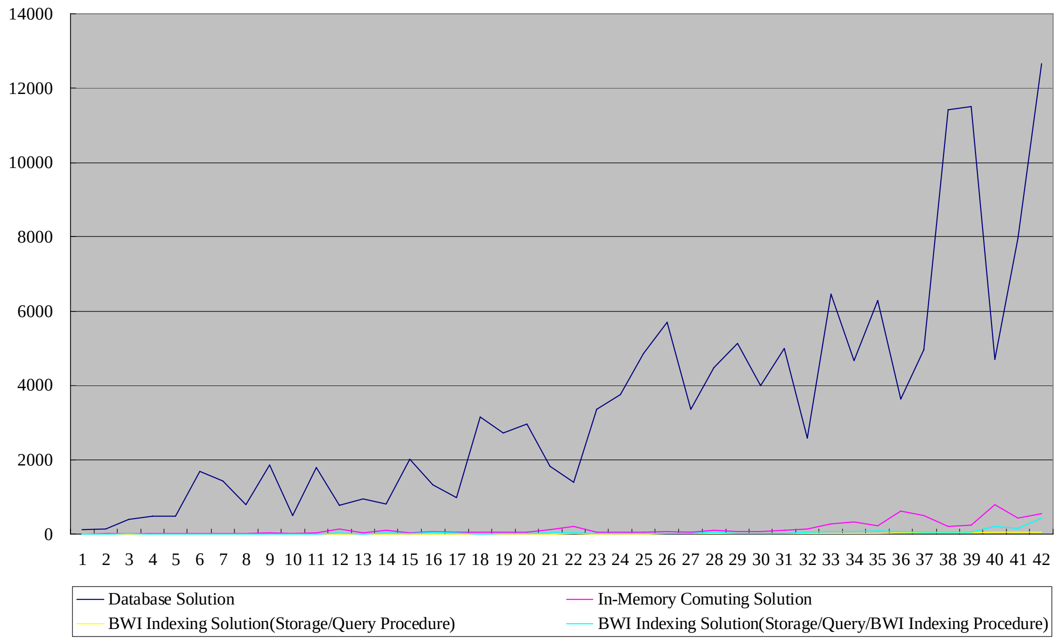


Figure 8.3: The Processing Time of all computing solutions

To solve the above problems, the data preparation problem becomes the key issue in the next-generation data mining system, even if some tradeoff relations exist among these problems.

2. Null value issue: Among the 72 lots involved in case 22, only 21 contain values in the decision feature(yield attribute), while four are abnormal. Since null-value situation is frequent, the proposed data mining system is best designed for null-value tolerance. Based on the experience of the TSMC project, the null value issue should be handled for the next-generation data mining system.

3. Spit lot/step issue: It is known that the spit lot/step issue causes numerous null values, the data mining system will produce unacceptable results. Essentially,

each record in the lot-based data set represents all wafers within a lot. On occurrence of the spit lot/step issue, some lots are separated into several sub-lots, and the lot-based data set are no longer qualified to represent the real status accordingly. Since the spit lot/step issue becomes more important for the 130 and 90 *nm* manufacturing procedures, the lot-based information should be drilled down to the wafer-based record; however, it is difficult to retrieve the wafer-based data using the EDA system.

4. Ratio of the Antithesis-group: As mentioned above, the behavior patterns discovered by the learning procedure of the learning-based module in the normal lot group are used to challenge the abnormal lot group. Ideally, a ratio of these two antithesis groups of 70% to 80% lots in the normal lot group is recommended for data mining. However, it is extremely difficult for product engineers to maintain this ratio for all given data sets by retrieving suitable data in every low-yield situation. For example, the root cause machines in Cases 21 and 22 are ranked 1 and 3, respectively, if the ratio of antithesis group is set as 80%.

8.4 Intelligent Yield Enhancement System for Semiconductor Manufacturing

Future research will focus mining engine enhancement, mining platform construction and knowledge engineering consultation issues.

1. Data mining system enhancement issues:

i) Multiple tools/factors mining scenario – In this scenario, we would like to develop a new scenario that can discover not only a single, but also a set of killer machines. In the last year, the root cause of one collected cases is issued by this situation. Since the failure contribution degree of each tool is hard to be determined, it is difficult to find out the single machine failure only using the manufacturing data much less the combination problem of multiple killer machines; however, such problems become increasingly frequent in advanced manufacturing procedures implies that developing an efficient and faithful mining scenario for discovery of multiple tools/factors is also important.

ii) Queue-time mining scenario –This scenario would like to develop a new scenario for identifying the low-yield situation for workflow resulting from abnormal waiting time. It means, the wafers becomes low-yield due to the material oxygenized by the delay during some critical stages. In 2002, some cases are affected by this situation were treated as numeric data segments and thus processed via the Learning-based module. Since the results were inadequate, a new scenario is developing to handle this situation.

iii) Secondary root cause scenario – In this scenario, we will develop a new scenario for discovering the second root cause. According to our pervious experience, several root causes are found to be involved in a single case, and some case preparation guidelines are proposed to avoid such problems. Therefore, a new scenario is required to handle such situations.

iv) The cross-scenario estimation mechanism – As mentioned above, the results generated by the proposed modules are ranked independently. A cross-scenario estimation mechanism to evaluate all ranking lists based on an overall weighting mechanism will be proposed and the combination issues for the secondary root causes scenario will be solved accordingly.

2. Mining platform construction issue: Accordingly, a knowledge platform, called MDDS (Manufacturing Defect Detection System) platform, is proposed to integrate all developed engines with considering all above enhancement issues in order to construct a complete yield enhancement platform. Three major parts of the proposed platform are described as follows:

i) Information Collection part: Regular meetings are recommended to discuss domain knowledge about semiconductor manufacturing and the related concepts and information properties of EDA, CIM and MES systems, including the data format, data amount, and data requirements of data mining solutions for yield enhancement purposes, which then can be formally and clearly described. Subsequently, a functional specification with a given dataflow scenario for the suitable preprocessing and data mining requirements should be proposed for each type of low-yield situation. Consequently, the Wrapper System and Wafermap Analyzing System are used to gather all related information, such as the database resources and other data files. For the architecture of Information Collection part shown in Figure 8.4, the corresponding algorithm Information_Collection Algorithm is presented below.

Algorithm 8.1 - *Information_Collection algorithm*

Input: a set of solved low-yield cases $C=\{c_1, c_2, \ldots, c_n\}$ and $c_k=\{l_{k,1}, l_{k,2}, \ldots, l_{k,|c_k|}, rc_k\}$ such that $l_{k,i}$ is the i-th covered lot of case c_k and rc_k is the known root cause of case c_k for $1 \le k \le n$,$1 \le i \le |c_k|$ and $n \ge 1$.

Output: 1. The corresponding data set $DS=\{ds_1, ds_2, \ldots, ds_n\}$ and wafermap set $WF=\{wf_1, wf_2, \ldots, wf_n\}$ of C.

2. A set of BWI matrixes BWI={bwi_1, bwi_2, …, bwi_n}, where bwi_k={bwi_k^D, bwi_k^W} such that bwi_k^D and bwi_k^W are the corresponding BWI matrixes of ds_k and wf_k, respectively, for case c_k in C and $1 \leq k \leq n$.

Step 1. For each case c_i in C, do the following sub-steps:

Step 1.1. For all covered lots {$l_{i,1}$, $l_{i,2}$, …, $l_{i,|c_i|}$} in c_i, retrieve data set ds_i from all underlying databases, including CIM, MES and EDA databases, via Wrapper System.

Step 1.2. For case c_i, retrieve wafermaps wf_k from the storage of corresponding wafermaps for all covered lots {$l_{i,1}$, $l_{i,2}$, …, $l_{i,|c_i|}$} in c_i.

Step 2. For each data set in DS and WF, do the following sub-steps:

Step 2.1. For data set ds_i in DS, transform ds_i to corresponding BWI matrix bwi_i^D via Wrapper System.

Step 2.2. For wafermaps wf_i in WF, transform wf_i to corresponding BWI matrix bwi_i^W via Wafer Analysis System.

Step 3. Return BWI and then store it in BWI Indexing Server.

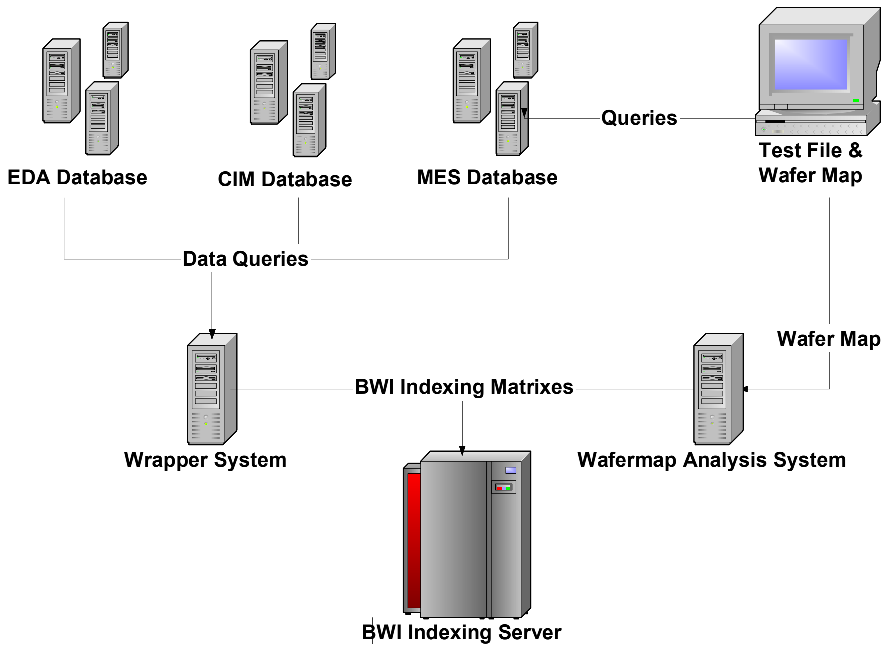


Figure 8.4: The architecture of information collection part in MDDS Knowledge Platform

The function of all sub-systems in Figure 8.4 is summarized below:

1. Wrapper System：This system is in charge of information collection for all analysis and mining requirements. Three information resources, CIM, MES and EDA databases, must be accessed. For each database resource, the product engineers can retrieve the related information via a data query function, and these query results are then transformed into a BWI matrix based on the data format requirements of the proposed dataflow scenario. Moreover, the corresponding text files, such as the WAT and CP testing resulting, are also retrieved via some text file processing ability of the database and the corresponding BWI matrixes then are obtained.

2. Wafermap Analyzing System：This system is in charge of information

collection for all related wafermaps in the Wrapper System. After wafermap retrieved, each wafermap is transformed to a corresponding BWI matrix for further analysis.

3. Bit-Wise Indexing Server：This server is used for storing all transformed BWI matrixes. For all stored BWI Matrixes, this server can provide OLAP and indexing similarity computing to support parallelized, scalable, high performance data query for all stored BWI Matrixes.

ii) Learning and analyzing part: After executing the data collection procedure, a Rule-Learning System is used for mining association rules from the BWI matrix server based on predefined data relationships, and a Model-Learning System is used to model learning among all manufacturing machines. Furthermore, the corresponding BWI matrixes of wafermaps are classified and analyzed using the Wafermap Analysis System to extract some of the wafermap patterns among them. Finally, all wafermap patterns, learned rules and machine models are judged by the domain experts. After the verification procedure, the suitable wafermap patterns, learned rules and machine models are stored in the Wafermap Gallery, Knowledge Base and Machine Model Base, respectively. For the architecture of Learning and analyzing part shown in Figure 8.5, the corresponding algorithm Learning_and_Analysis Algorithm is presented below.

Algorithm 8.2 - *Learning_and_Analysis algorithm*

Input: a set of solved low-yield cases C and a set of BWI matrixes $BWI=\{bwi_1, bwi_2, \ldots, bwi_n\}$ and $bwi_k=\{bwi_k^D, bwi_k^W\}$, for case c_k in C and $1 \leq k \leq n$.

Output:
1. The verified wafermap patterns $WP=\{p_1, p_2, \ldots, p_i\}$ of C.
2. The verified association rules $AR=\{r_1, r_2, \ldots, r_j\}$ of C.
3. The verified neural-network machine models $NM=\{nn_1, nn_2, \ldots, nn_j\}$ of C.

Step 1. For each BWI matrix bwi_k in C, do the following sub-steps:

Step 1.1. For wafermaps BWI matrixes bwi^W and the corresponding rc_*, analyze the wafermap patterns $WP=\{p_1, p_2, \ldots, p_i\}$ via Wafer Analysis System.

Step 1.2. For data set BWI matrixes bwi^D and the corresponding rc_*, where rc_* is the root cause rcs in C, mining the association rules $AR=\{r_1, r_2, \ldots, r_j\}$ via Rule-Learning System.

Step 1.3. For data set BWI matrixes bwi^D and the corresponding rc_*, where rc_* is the root cause rcs in C, learning the corresponding neural-network machine models $\{nn_1, nn_2, \ldots, nn_j\}$ via Model-Learning System.

Step 2. For WP, AR and MN, do the following sub-steps:

Step 2.1. Require the domain experts for results verification.

Step 2.2. Remove the unqualified patterns, rules and machine models from *WP*, *AR* and *MN*, respectively.

Step 3. Return *WP*, *AR* and *MN* and then store them in Wafermap Gallery, Knowledge Base and Machine Model Base, respectively.

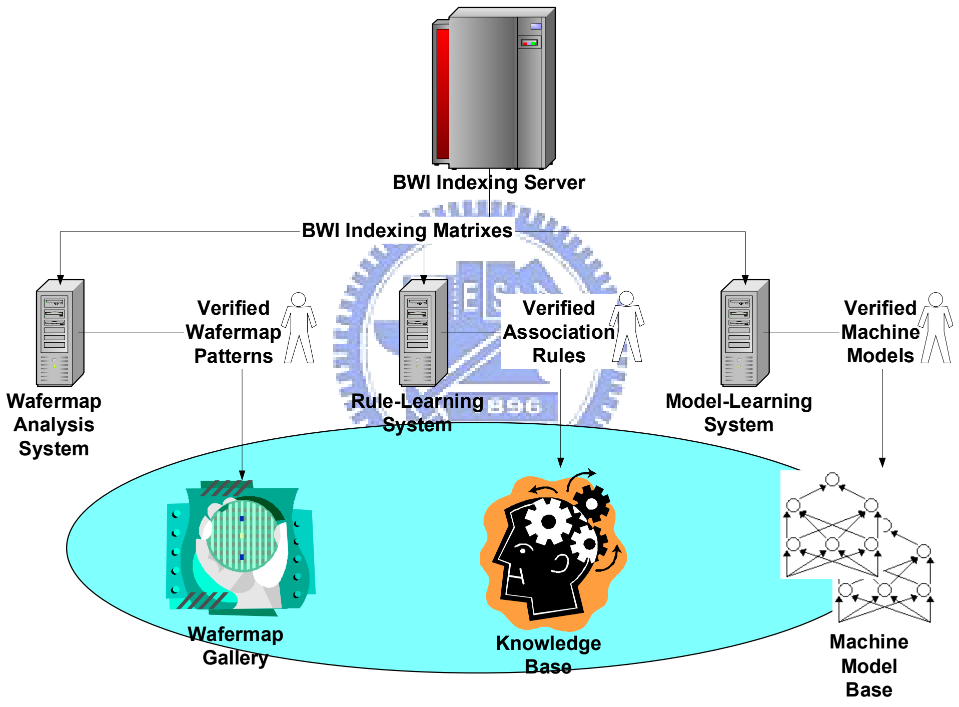


Figure 8.5: The architecture of Learning and Analyzing part in MDDS Knowledge Platform

The functions of the sub-systems in Figure 8.5 are described below:

1. Wafermap Analyzing System：This system analyzes all of the related wafermaps and identifies useful wafermap patterns. All input wafermaps are

first classified into systematic and randomized categories. For wafermaps in the systematic category, the system identifies possible wafermap patterns and delivers the discovered patterns to the domain experts for verification. Subsequently, all verified patterns and their indicated root cause are stored in the Wafermap Gallery for further investigation.

2. Rule-Learning System：This system is applied to analyze all related information for each solved low-yield situation to discover the correlation among attributes via the Transaction-based module. Initially, the correlation between the decision feature and other attributes in the given data is calculated via the Learning-based module. Subsequently, the highly related features are treated as the new decision features, and the Transaction-based Module is thus triggered for mining the possible root causes. After completing the mining procedure, possible root cause lists of both the original root cause and highly related features are transformed to the transaction log and processed through an association rule mining procedure. Finally, the discovered association rules are delivered to the domain experts for verification and all verified association rules are stored in the Knowledge Base for further investigation.

3. Model-Learning System：This system manages individual machine behavior learning based on predefined data relationships. Initially, a predefined

perceptron neural networking should be defined for each individual machine in the semiconductor manufacturing fab. The input nodes of this perception consist of manufacturing parameters, temperature, air pressure and sensor information for the target machine. A half WIP (wafer-in-process) data related to this machine is used as a training instance to learn the machine model using neural network technologies, and the other half is used to verify the perceptron. If the result is satisfied, the machine model is stored in the Machine Model base; otherwise, the unqualified machine model is verified by some domain experts for further examination.

4. Wafermap Gallery, Knowledge Base and Machine Module Base: These are the storage of wafermap patterns, verified manufacturing rules and individual machine modules, respectively.

iii) Application part: The learning results obtained through the incremental learning procedures in the Learning and Analysis part are used to examine and monitor the in-line and off-line procedures for yield enhancement. The Model Monitoring System, like an advanced APC system, is responsible for in-line monitoring and delivering alarm messages to both in-line monitoring workstations and mining engines in the event of abnormal behavior, based on the related machine information from the manufacturing and measuring tools. Additionally, the verified mining rules can be applied to examine the in-line information and identify abnormal situations using the Intelligent Reasoning System. Once the low-yield situation happens, the related wafermaps are matched and some suspected machine IDs are thus delivered to the Mining System for recommendation. The data mining results then are evaluated and ranked via the cross-scenario estimation mechanism according to the recommendations from the Wafermap Analysis, Intelligent Reasoning and Model systems. Finally, a data mining report for the given low-yield situation is delivered to the corresponding product engineers for further study. For the architecture of Application part shown in Figure 8.6, the corresponding algorithm Application Algorithm is presented below.

Algorithm 8.3 - ***Application algorithm***

Input: a newly arrived low-yield cases $nc=\{l_{nc,1}, l_{nc,2}, \ldots, l_{nc,|nc|}\}$ where $l_{nc,i}$ is the i-th covered lot of case nc and $1 \leq i \leq |nc|$.

Output: The ranked possible root causes list rcl of nc.

Step 1. For newly arrived low-yield cases, call Information-Collection Algorithm and the corresponding ds_{nc}, wf_{nc}, bwi_{nc}^{D} and bwi_{nc}^{W} are thus returned.

Step 2. For wafermap set wf_{nc} and BWI matrix bwi_{nc}^{W}, match the existing wafermap patterns *WP* in Wafermap Gallery via Wafermap Analyzing System and the root cause set rc_{WP} is thus returned, where rc_{wf} is the root causes of all matched patterns in *WP*.

Step 3. For data set ds_{nc} and BWI matrix bwi_{nc}^{D}, trigger inference procedure for all mined rules *AR* in Knowledge Base via Intelligent Reasoning System and the root cause set rc_{AR} is thus returned, where rc_{AR} is the root causes of all trigger rules in *AR*.

Step 4. For data set ds_{nc} and BWI matrix bwi_{nc}^{D}, trigger computing procedure for all built neural-network-based machine model *NM* in Machine Model Base via Model Monitoring System and the abnormal machine set ab_{NM} is thus returned, where ab_{NM} is the set of machines that return all abnormal alerts.

Step 5. For data set ds_{nc} and BWI matrix bwi_{nc}^{D}, trigger the Mining System for discovering the possible root causes and the ranked list of root cause *rcl* is thus found.

Step 6. For root cause in rc_{WP}, rc_{AP} and ab_{NM}, if the root cause exists in *rcl*, enhance the ranking weight via the cross-scenario estimation mechanism.

Step 7. Rank the *rcl* according to the new ranking weight.

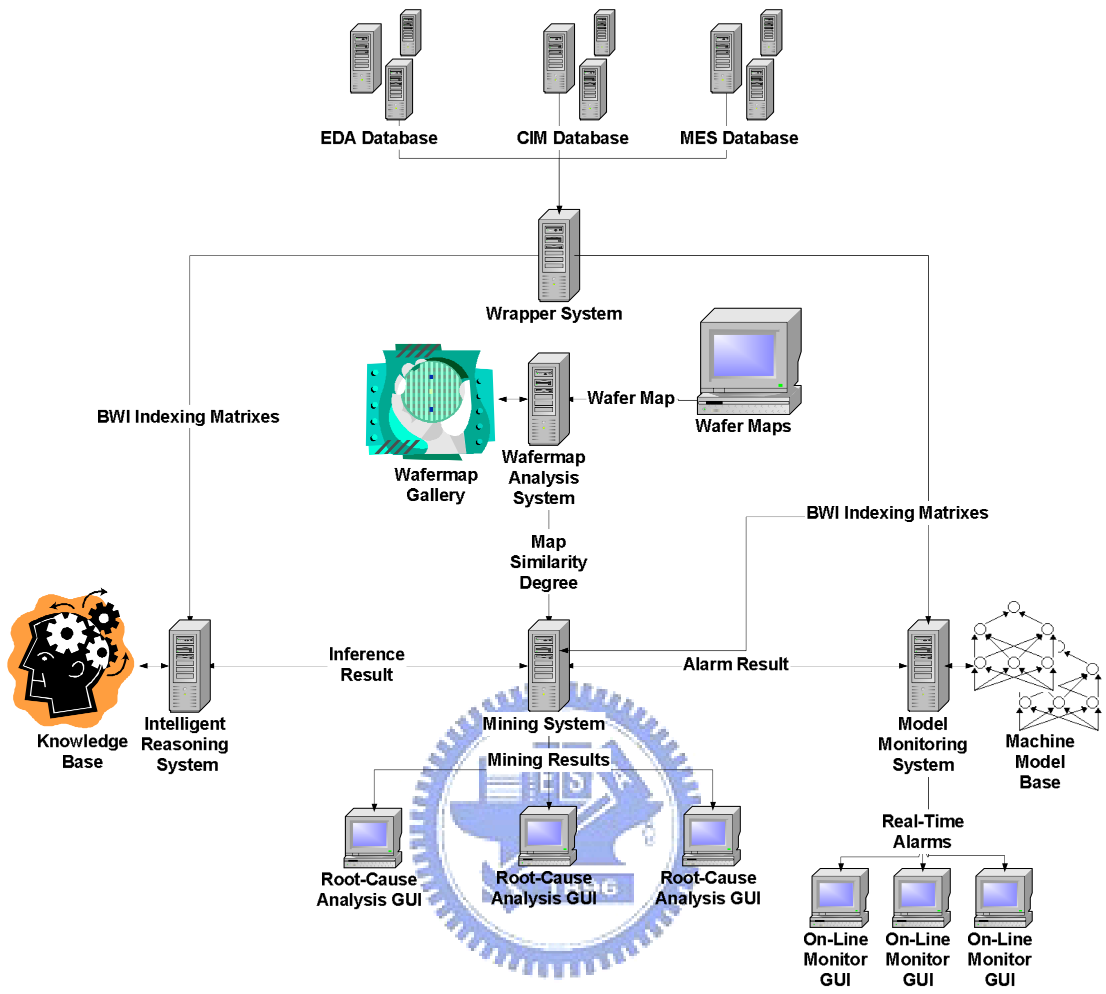


Figure 8.6: The architecture of Application part in MDDS Knowledge Platform

From Figure 8.6, all sub-systems are described in the following:

1. Wafermap Analyzing System：For each new wafer map, the Wafermap Analyzing System identifies the matching patterns within the Wafermap Gallery. Once similar patterns are discovered, the corresponding root causes are delivered to the Mining System for re-weighting recommendation

2. Intelligent Reasoning System： Following verification by the Learning

System, mining rules can be applied to identify abnormal situations and pre-examine the given parameters. Once the new case is entered into the Intelligent Reasoning System, all possible facts obtained are forwarded to the Mining System for further examination. Similar the Model System, reasoning results or the confidence and support for triggered rules can be provided through the explanation function of the Intelligent Reasoning and Learning Systems.

3. Model Monitoring System： After all machine models are tuned or optimized by the Learning System, these models can be used to monitor the in-line manufacturing and measuring tools. After entering a new case into the Model Monitoring System, the Wrapper System is triggered to collect all related information of this case. Since then, the machine related parameters are calculated using the corresponding neural-network machine model. For abnormal computational results, the corresponding machine ID and some alarm messages are sent to the Mining System and the users, respectively. If required, the Model Monitoring System can provide the related evidence supported by the Model Analyzing System for further explanation.

4. Mining Monitoring System：This system is in charge of data mining procedure. Once the low-yield situation happens, the related information is delivered to

the Wafermap Analysis, Model Monitoring and Intelligent Reasoning Systems for re-weighting recommendations. Simultaneously, the corresponding mining scenarios are triggered. After the results of Wafermap Analyzing, Model Monitoring, Intelligent Reasoning and Mining Systems are generated, they are overall evaluated and ranked via the cross-scenario estimation mechanism. Finally, the data mining report for the given low-yield situation is delivered to the users for further investigation, and explanations and evidence for each corresponding result are also provided.

5. Wafermap Gallery, Knowledge Base and Machine Module Base: These systems are used for the storage of wafermap patterns, verified manufacturing rules and individual machine modules, respectively.

3. Knowledge engineering consultation task: In our data mining project, since domain experts and IT persons in semiconductor manufacturing domain are usually not familiar with data mining concept, we will continually help them realize the concept of data mining correctly, Also, we will take the opportunity of deploying the MDDS platform to help the semiconductor manufacturing people understand the esprit of data mining systems.

Currently, the semiconductor manufacturing is becoming increasingly complex as market demand drives higher circuit density. This trend requires new and more sophisticated processing tools, longer process flows, and more detailed sampling of metrology data to verify process controls. Moreover, this trend also implies large amounts and high complexity of manufacturing data, and tight time-to-market for advanced devices. Therefore, an intelligent and efficient yield enhancement system is desired to deal with the low-yield situation and efficiently increase the yield trend. In this section, we have proposed a data mining system which had been successfully applied in Taiwan Semiconductor Manufacturing Company (TSMC) for discovering the root causes of low-yield situations. Also, the evaluation of our mining system for manufacturing defects detection in semiconductor manufacturing domain has been done and several important issues have been fully discussed. Finally, a new architecture of a reasonable, reliable and flexible defect detection platform using data mining approach has been described.

Chapter 9
Conclusions and Future Work

The Fields of knowledge Discovery Systems and Data Mining have rapidly grown in the past 10 years. Research, applications, and tool development in business, science, government, and academia are becoming increasingly popular. Since the amount of data is continuously and rapidly growing in most knowledge systems, discovering the useful information correctly and efficiently is becoming a significant issue. In this thesis, an efficient indexing technology, called Bit-wise Indexing Technology, and three indexing methods for different applications were proposed. Furthermore, the corresponding indexing and matching algorithms for each indexing model were described in detail. To demonstrate the suitability, flexibility and efficiency of the proposed indexing methods, they were applied in four knowledge system applications, including reinforcement learning, pattern matching, supervised learning and unsupervised-learning data mining applications. In the first application, the Sample Bit-Wise Indexing Method was used to encode the defect status of manufacturing product in order to accelerate the data preprocessing procedure. In the second application, the Encapsulated Bit-wise Indexing Method was used to encode the networking activity to accelerate the data preparation procedure. The third application used Compact Bit-wise Indexing method in a Rough-set-based Feature Selection Method to encode the feature and class relationships efficiently for reducing the processing time of the feature selection procedure. The proposed feature selection method had been used in a KA project to discover the desired feature sets to help an endowment insurance department of a world-wide financial group builds a CBR system for their loan promotion function of a customer relationship management system. In the last application, the bit-wise indexing methods, Data Mining Technologies, and Statistic Methods were hybridly combined to construct an unsupervised-learning data-driven data mining system for production-level defect detection in a engineering data analysis system. This application was officially applied in Yield Explorer Function of Intelligent Engineering Data Analysis system (*iEDA*) in Taiwan Semiconductor Manufacturing Corporation (TSMC) for root cause detection and yield enhancement.

In the future, a product-level bit-wise indexing server will be constructed and the maintenance issue, such as record/table/relation insertion/deletion/modification, will be further investigated. Also, the suitable indexing models will be proposed for various knowledge systems, such as a rule-based expert system, case-based reasoning system or neural net system. Moreover, the system platform of the next-generation data mining solution that proposed at the end of Chapter 8 will be further investigated and constructed and to provide a knowledgeable, reasonable, reliable and flexible data mining solution in semiconductor manufacturing domain. Additionally, the proposed bit-wise indexing method will be applied to different application domains. For instances, the proposed method is currently being applied to an intelligent clinical trial management system (iCTMS), to enhance the accuracy and performance of the knowledge acquisition and validation procedures.

References

[1] ADCOM Technology Inc, “Sonic wall”, *http://www.adcom.com.tw/product/sonicw/index.htm*, 2000

[2] H. Almullim et al. “Learning with many irrelevant features,” in *Proceedings of 9th National Conference on Artificial Intelligent*, 1991, pp. 547-552.

[3] R. Agrawal, T. Imielinski and A. Swami, “Mining association rules between sets of items in large database”, *ACM SIGMOD Conference*, 207-216, 1993.

[4] R. Agrawl and R. Srikant, “Fast Algorithm for Mining Association rules”, *ACM VLDB Conference*, 487-499, 1994.

[5] R. Barletta, "An introduction to case-based reasoning", *AI Expert*, Vol. 6, No.8, pp.42-49, 1991.

[6] D. Braha and A. Shmiloviei, “Data Mining for Improving a Cleaning Process in the Semiconductor Industry”, *IEEE Transactions on semiconductor manufacturing*, 15 (1), 2002.

[7] G. Brassard G. et al. *Fundamentals of Algorithm*, Prentice Hall, New Jersey, 1996.

[8] N. Cercone, A. An, and C. Chan, “Rule-induction and case-based reasoning:

hybrid architectures appear advantageous”, *IEEE Transactions on Knowledge and Data Engineering*, 11(1), 166-174, 1999.

[9] M. S. Chen, J. Han and P. S. Yu, “Data Mining: An Overview from a Database Perspective”, *IEEE Transactions on Knowledge and Data Engineering*, 8 (6), 1996.

[10] C. P. Chen, A. Shyu, P. Liou, R. Q. Leu, K. Huang, J. Y. Lin, T. H. Yang, H. C. Liu, M. I. Ting and Y. C. Shih, “A novel methodology of critical dimension statistical process control”, *3rd International Workshop on Statistical Metrology*, 104-1087, 1998.

[11] W. C. Chen, S. S. Tseng, J. H. Chen, M. F. Jiang, "A framework of feature selection for the case-based reasoning", in *Proceeding of IEEE International Conference on Systems, Man, and Cybernetics*, 2000, CD-ROM.

[12] W. C. Chen; S. S. Tseng; L. P. Chang, M. F. Jiang, “A similarity indexing Method for the data warehousing - bit-wise indexing method,” in *Lecture Notes in Artificial Intelligent*, Vol. 2035, 2001, pp. 525-537

[13] W. C. Chen; S. S. Tseng; L. P. Chang, T. P. Hong, “A parallelized indexing method for large-scale case-based reasoning,” *Expert System with Applications*, Vol. 23(2), 2002, pp.95-102.

[14] W. C. Chen; M. C. Yang; S. S. Tseng, “A high-speed feature selection method for

large dimensional data set," *in Proceeding of International Computer Symposium*, 2002, CD-ROM.

[15] W. C. Chen; M. C. Yang; S. S. Tseng, "The bitmap-based feature selection method," *in 18th ACM Symposium on Applied Computing (SAC), Data Mining Track*, 2003, CD-ROM.

[16] W. C. Chen, S. S. Tseng, K. R. Hsiao and C. C. Liu, "A Data Mining Project for Solving Low-yield Situations of Semiconductor Manufacturing", *IEEE/SEMI Advanced Semiconductor Manufacturing Conference*, 2004.

[17] W. C. Chen, S. S. Tseng and C. Y. Wang, "A Novel Manufacturing Defect Detection Method Using Data Mining Approach", *17th International Conference on Industrial & Engineering Applications of Artificial Intelligence & Expert Systems*, 2004.

[18] C. Chou, K. Yang, G. Chiang and R. Shiao "EPIS (equipment and process information system) a new prospect of EDA and SPC system", *Semiconductor Manufacturing Technology Workshop*, 163-166, 1998.

[19] S. K. Choubey et al, "On feature selection and effective classifiers," *Journal of ASIS*, Vol. 49(5), 1998, pp.423-434.

[20] Cisco, *Cisco PIX firewall manual*, http://mail.ht.net.tw/~erik/doc/cisco/pix.zip, 1999

[21] CLDP, *CLDP Firewall How to*, http://freebsd.ntu.edu.tw/cldp/ Firewall-HOWTO, 2000

[22] J. Daengdej, D. Lukpse, E. Tsui, P. Beinat, and L. Prophet, “Combining case-based reasoning and statistical method for proposing solution in RICAD”, *Knowledge-Based Systems*, 10, 153-159, 1997.

[23] H. Dai, *Discovery of Cases for Case-Based Reasoning in Engineering*, 89-96, 1997.

[24] J. Doak, “An evaluation of feature selection methods and their application to computer security,” Technical Report, University of California, 1992.

[25] S. Dutta, B. Wierenga, and A. Dalebout, "Case-based reasoning systems: from automation to decision-aiding and stimulation", *IEEE Transactions on Knowledge and Data Engineering*, 9(6), 911-922, 1997.

[26] FEYA TECHNOLOGIES CO., “Border Ware 6.0”, *http://www.feya.com.tw/ security/borderware.html*, 2000

[27] E. Fukuda,; S. Harakawa,; Ikeda, M.; “Advanced process control system description of an easy-to-use control system incorporating pluggable modules”, *IEEE International Symposium on Semiconductor Manufacturing Conference*, 321 -324, 1999.

[28] D. Gardingen, and I. Watson,”A web based CBR system for heating ventilation

and air conditioning systems sales support”, *Knowledge-Based Systems*, 12, 207-214.

[29] M. Gardner and J. Bieker “Data Mining Solves Tough Semiconductor Manufacturing Problems,” *ACM KDD Conference*, 2000.

[30] A. Gonzalez, R. Perez, "Selection of relevant features in a fuzzy genetic learning," *IEEE Transaction on SMC-Part B*, Vol. 31(3), 2001, pp. 417-425.

[31] K. M. Gupta, and A. R. Montazemi, “Empirical evaluation of retrieval in case-based reasoning systems using modified cosine matching function”, *IEEE Transactions on Systems, Man, and cybernetics-Part A: Systems and Humans*, 27 (5), 601-612, 1997

[32] A. J. Gonzalez, L. Xu, and U. M. Gupta, “Validation techniques for case-based reasoning systems”, *IEEE Transactions on Systems, Man, and Cybernetic-Part A: Systems And Humans*, 28 (4), 465-477, 1998.

[33] J. Han and M. Kamber. *Data Mining: Concepts and Techniques*, Morgan Kaufmann Publishers, 2001.

[34] R. J. Hilderman and H. J. Hamilton, “Measuring the interestingness of discovered knowledge: A principled approach,” *Intelligent Data Analysis*, Vol. 7, No. 4, 347 - 382, 2003.

[35] M. Y. Huang, R. J. Jasper and T. M. Wicks "A large scale distributed intrusion

detection framework based on attack strategy analysis", *Computer Network*, 31, pp. 2465-2475, 1999

[36] C. C. Huang and B. Tseng, "Rough Set Approach to Case-Based Reasoning," *Expert Systems with Applications*, Vol. 26, No. 3, pp. 369-385, April 2004.

[37] K. Ilgun, “USTAT: A Real-Time Intrusion Detection system for UNIX”, *IEEE Symposium on Research on Security and Privacy*, 1993

[38] K. Ilgun, R.A. Kemmerer, P. A. Porras “State Transition Analysis: A Rule-Based Intrusion Detection System”, *IEEE Transactions on Software Engineering*, 21 (3), 1995

[39] G. H. John et al. “Irrelevant feature and the subset selection problem,” in *Proceedings of 11th International Conference on Machine Learning*, 1994, pp. 121-129.

[40] R. Kemmerer, *NSTAT: A Model-based Real-time Network Intrusion Detection System” Technical Report TRCS-97-18*, Department of Computer Science, University of California, Santa Barbara, 1997

[41] A. Kuiak. “Rough Set Theory: A Data Mining Tool for Semiconductor Manufacturing.”, *IEEE Transaction on electronics packing manufacturing*, 24 (1), 2001.

[42] M. Last, A. Kandel, O. Maimon, "Information theoretic algorithm for feature

selection," *Pattern Recognition Letter*, Vol. 22, 2001, pp.799-811.

[43] H. M. Lee, C. M. Chen, J. M. Chen, Y. L. Jou, "An efficient fuzzy classifier with feature selection based on fuzzy entropy," *IEEE Transaction on SMC-Part B,* Vol. 27(2), 1997, pp. 426-432.

[44] L. Li and L. X. “Knowledge-based problem solving: an approach to health assessment”, *Expert Systems with Applications*, 16, 33-42, 1999.

[45] T. Li, S. Zhu and M. Ogihara, “Algorithms for clustering high dimensional and distributed data,” *Intelligent Data Analysis*, 7 (4), 305 - 326, 2003.

[46] U. Lindqvist and P. A.Porras, “Detecting computer and network misuse through the production-based expert system toolset (P-BEST)”, *IEEE Symposium on Security and Privacy*, 146 – 161, 1999.

[47] H. Liu et al, “A probabilistic approach to feature selection – a filter solution,” in *Proceedings of 13th International Conference on Machine Learning*, 1996, pp. 319-327

[48] H. Liu, R. Setiono, "Incremental feature selection," *Applied Intelligence*, Vol. 9, 1998, pp.217-230.

[49] Megasoft Corporation, *http://www.taipeisoft.com/Products/WinR/winr.html*, 2000

[50] D. Q. Miao et al, “A heuristic algorithm of reduction for knowledge,” *Journal of Computer Research and Development*, Vol. 36(6), 1999, pp. 681-684.

[51] F. Mieno, T. Santo, Y. Shibuya, K. Odagiri, H. Tsuda and R. Take. “Yield Improvement Using Data Mining System,” *IEEE Semiconductor Manufacturing Conference*, 1999.

[52] R. Mike Gardener, J. Bieker and S. Elwell. “Solving Tough Semiconductor Manufacturing Problems Using Data Mining,” IEEE/SEMI Advanced Semiconductor Manufacturing Conference, 2000.

[53] S. Mittal, K. Lubic, P. McNally, “Use of Inline Defect Monitors to Drive P852 Yield Improvement” Intel Manufacturing Excellence Conference, 1995.

[54] P. O'Neil, and D. Quass, “Improved query performance with variant indexes”, *The SIGMOD Conference*, 1997

[55] Z. Pawlak, "Rough set," *International Journal of Computer and Information Sciences*, 1982, pp.341-356,.

[56] Z. Pawlak. *Rough Sets, Theoretical Aspects of Reasoning about Data*, Kluwer Academic Publishers, Boston, 1991.

[57] P. Porras, *STAT – A State Transition Analysis Tool for Intrusion Detection*, Master’s thesis, Computer Science Department, University of California, Santa Barbara, 1992

[58] J. Quinlan, "Introduction of decision trees," *Machine Learning*, Vol.1(1), 1986, pp.81-106.

[59] V. Raghavan. “Application of Decision Trees for Integrated Circuit Yield Improvement”. IEEE/SEMI Advanced Semiconductor Manufacturing Conference, 2002.

[60] M. L. Raymer, W. F. Punch, E. D. Goodman, L. A. Kuhn, "Dimensionality reduction using genetic algorithm," *IEEE Transaction on Evolutionary Computation*, Vol. 4(2), 2000, pp.164-171.

[61] E. Rolland, “Abstract Heuristic Search Method for Genetic Algorithm” Ph.D dissertation, The Ohio State University Columbus, 1991.

[62] M. Sarfaty, A. Shanmugasundram, A. Schwarm, J. Paik, J. Zhang, R. Pan, M. J. Seamons, H. Li, R. Hung, S. Parikh,” Advance Process Control solutions for semiconductor manufacturing”; *IEEE/SEMI Advanced Semiconductor Manufacturing Conference*, 2002 Page(s): 101 -106

[63] J. C. Schlimmer et al, “Efficiently inducing determinations: A complete and systematic search algorithm that uses optimal pruning,” in *Proceedings of 10th International Conference on Machine Learning*, 1993, pp. 284-290.

[64] A. Skowron, C. Rauszer, "The discernibility matrices and functions in information systems," *Intelligent Decision Support*, 1992, pp.331-362.

[65] S. P. Shieh and V. D. Gligor, “A pattern-oriented intrusion-detection model and its applications”, *EEE Computer Society Symposium on Research in Security and*

Privacy, 327 –342, 1991.

[66] S. P. Shieh and V. D. Gligor, "On a pattern-oriented model for intrusion detection", *IEEE Transactions on Knowledge and Data Engineering*, 9 (3), 661 -667, 1997.

[67] K. S. Shin, and I. Han, "Case-based reasoning supported by genetic algorithms for corporate bond rating", *Expert Systems with Applications*, 16, 85-95, 1999.

[68] M. S. Suh,, W. C. Jhee,, Y. K. Ko, and A. Lee, "A case-based expert system approach for quality design", *Expert Systems With Applications*, 15, 181-190, 1998.

[69] SYSWARE Corp., "CheckPoint 2000", *http://firewall.sysware.com.tw/*, 2000

[70] B. Tseng, M. C. Jothishankar and T. Wu, "Quality Control Problem in Printed Circuit Board Manufacturing - an Extended Rough Set Theory Approach," *Journal of Manufacturing Systems*, Vol. 23, No. 1, pp. 56-72, 2004.

[71] UCI Repository : http://www.ics.uci.edu/~mlearn/MLRepository.html

[72] G. Vigna and R. A. Kemmereer "NetSTAT: A Network-based Intrusion Detection Approach", *IEEE Computer Security Applications Conference*, 25-34, 1998.

[73] I. Waston, "Case-based reasoning is a methodology not a technology", *Knowledge-Based Systems*, 12, 303-308, 1999.

[74] K. L. Wu and P. S. Yu, "Range-based bitmap indexing for high cardinality

attributes with skew”, *The 22nd Annual International Conference on Computer Software and Applications*, 1998.

[75] M. C. Wu, and A.P. Buchmann, “Encoded bitmap indexing for data warehouses”, *The 14th International Conference on Data Engineering*, 1998.

[76] F. B. Wu et al. “An inductive learning method based on rough set theory expressed knowledge system,” *Control and Decision*, Vol. 14(3), 1999, pp. 206-211.

[77] Y. Yang, T. C. Chiam, "Rule discovery based on rough set theory," in *Proceedings of the Third International Conference on FUSION*, Vol. 1, 2000, pp. TuC4_11 -TuC4_16.

[78] H. Yu et al, “Rough set based knowledge reduction algorithms,” *Computer Science*, Vol. 28(5), 2001, pp.31-34.

[79] N. Zhong, J. Dong, S. Ohsuga, "Using rough sets with heuristics for feature selection", *Journal of Intelligent Systems*, Vol. 16, 2001, pp. 199-214.

Index