

**Data Structures and Algorithms**  
**for Efficient Cycle Mining of Association Rules**

Thesis Submitted to

The College of Arts and Sciences

University of Dayton

In Partial Fulfillment of the Requirements for

The Degree

Master of Science in Computer Science Department

by

Bin Gao

University of Dayton

Dayton, Ohio

April 2003

**APPROVED BY**

---

~~James P. Buckley, Ph.D.~~  
~~Advisory Committee, Chairman~~

---

~~Jennifer Seitzer, Ph.D.~~

---

~~Dale E. Courte, Ph.D.~~

# **ABSTRACT**

## **DATA STRUCTURES AND ALGORITHMS FOR EFFICIENT CYCLE MINING OF ASSOCIATION RULES**

Name: Gao, Bin

University of Dayton

Advisor: Dr. J.P. Buckley

Data mining focuses on developing efficient algorithms by which patterns or rules in large databases are discovered to meet specified thresholds. Association analysis is one of the most important tasks of data mining. It is the discovery of association rules. In fact, there exist very interesting patterns under these association rules. Cyclic structure is one of those patterns. The discovery of the cyclic structures is called cycle mining, which determines if there is a cycle among the association rules and what cycles are.

This thesis presents a number of approaches for performing cycle mining. First, the properties and representation of association rules are defined and described. The relationship among association rules can be described as a directed graph. Then, cycle mining is defined to find all cycles from the directed graph of association rules. Efficient algorithms are provided to perform the task. A hash table with chaining is modified to represent the directed graph. It is efficient for cycle mining since the mining procedure

may involve a great deal of searching and deletion. A program has been written to implement the data structure and algorithms in JAVA. It is based on client-server architecture, with a MySQL database server as the backend. Empirical analysis is performed to determine how efficient and scalable the data structure and algorithms are.

## ACKNOWLEDGEMENTS

I am grateful for support from many people. Foremost I would like to thank my advisor, Dr. James Buckley, for giving me guidance. Special thanks to Dr. Dale Courte and Dr. Jennifer Seitzer, the members of my thesis committee. They spent a lot of time reviewing my thesis draft with Dr. Buckley. Without their suggestions and encouragement, I can hardly imagine how I can complete my thesis.

I also would like to thank all faculties in the computer science department of University of Dayton. What impressed me most are their respectable personality as well as their expertise. They are Dr. James Buckley, Dr. Barbara Smith, Dr. Dale Courte, Dr. Jennifer Seitzer, Dr. John Loomis, Dr. R. Sritharan, Dr. Joseph Lang, Dr. Raghava Gowda, Ms. Rebecca Lamb, Ms. Vanessa Starkey, and Fr. Thomas Schoen. Thanks also go to Denise Brown and Halter Dean for giving me a lot of help without any reservation. I'd like to thank my friends Liming Shen, Chunwe Lo, Yih Horng, Huilan Wu and Xun Li, who greatly encouraged me during my study and thesis writing.

A sincere note of thanks to my parents and parents-in-law for their unconditional care and help.

Finally, special thanks to my dear wife, Zhenmei Zhang, for her love and support. Her love has given me more energy and driven me to work hard. While I am working on my thesis at University of Dayton, she is making an unbelievable effort. At Penn State University, she has to take care of our newborn son, and write her dissertation.

# TABLE OF CONTENTS

Abstract .....	iii
Acknowledgements .....	v
Table of Contents .....	vi
List of Figures .....	viii
List of Tables.....	ix
Introduction .....	1
1 Data Mining and Association Rules.....	3
1.1 Introduction .....	3
1.2 KDD and Data Mining .....	3
1.3 Data Mining Models and Tasks .....	6
1.4 Association Rules .....	6
1.5 Association Rule Mining Algorithms .....	8
1.6 Properties Related to Pruning Techniques.....	9
1.7 Related Work.....	10
2 Cyclic Structure among the Association Rules .....	11
2.1 Introduction .....	11
2.2 Definition of a Cycle .....	12
2.3 Cycles among Association Rules.....	13
2.4 Properties of Cycles of Association Rules.....	13
2.5 Near Cycle .....	14
3 Cycle Mining of Association Rules .....	16

3.1 Introduction .....	16
3.2 Architecture of Cycle Mining System .....	17
3.3 Framework of Cycle Mining of Association Rules .....	19
3.4 Algorithms for Cycle Mining .....	20
<b>4 Data Structure for Cycle Mining.....</b>	<b>31</b>
4.1 Introduction .....	31
4.2 Data for Cycle Mining .....	31
4.3 Data Representation.....	33
4.4 Future Study .....	37
<b>5 Implementation of Cycle Mining of Association Rules.....</b>	<b>38</b>
5.1 Environments.....	38
5.2 Implementation.....	39
5.3 Graphical User Interface.....	41
5.4 Examples .....	41
5.5 Performance Analysis.....	43
<b>6 Summary and Conclusions.....</b>	<b>46</b>
6.1 Outline of the Contributions of This Work.....	46
6.2 Conclusions and Discussion .....	46
<b>Appendix 1: Data Definition Using SQL.....</b>	<b>48</b>
<b>Appendix 2: List of files in Software Package.....</b>	<b>50</b>
<b>Bibliography.....</b>	<b>52</b>

## LIST OF FIGURES

Figure 1. 1 KDD Process (from Fayyad, 2001) .....	4
Figure 1. 2 Data Mining Models and Tasks (from Dunham, 2002) .....	5
Figure 2. 1 Structures Hidden in Example Association Rules .....	12
Figure 3. 1 Architecture of Cycle Mining System .....	17
Figure 3. 2 Example Digraph .....	21
Figure 3. 3 Pruning Strategies for Cycle Mining .....	23
Figure 3. 4 Algorithm for Cycle Mining: MiningAllCycleTD() .....	25
Figure 3. 5 Algorithm for Cycle Mining: MiningAllCycleDT() .....	27
Figure 3. 6 Algorithm: cycleDetect() .....	28
Figure 3. 7 Algorithm: unmark() .....	29
Figure 4. 1 Referential Integrity Constraints Displayed on Tables .....	32
Figure 4. 2 Hash Table with Chaining .....	34
Figure 4. 3 ADT of Graph for Cycle mining .....	34
Figure 4. 4 Close Look of Node A .....	35
Figure 5. 1 Uses Relations among Classes .....	39
Figure 5. 2 Performance Chart for 40 Association Rules .....	44
Figure 5. 3 Performance Chart for 10 Itemsets .....	44
Figure 5. 4 Performance Chart for 100 Association Rules with 50 Itemsets .....	45



## LIST OF TABLES

Table 2. 1 A Set of Example Association Rules .....	11
Table 3. 1 Evaluation of Cycles by a Generic Algorithm .....	22
Table 4. 1 Definitions of the Tables Related to Cycle Mining.....	32
Table 4. 2 ADT Specification of a Cycle .....	35
Table 4. 3 ADT Specification of a Digraph .....	36
Table 4. 4 ADT Specification of a Node of Digraph .....	36
Table 5. 1 Environments of Implementation.....	38
Table 5. 2 Graphical User Interface .....	40
Table 5. 3 Examples of Cycle Mining via the Program .....	41
Table 5. 4 Parameters for Performance Analysis .....	43

## INTRODUCTION

Data mining focuses on developing efficient algorithms by which patterns or rules in large databases are discovered to meet specified thresholds. Association analysis is one of the most important tasks of data mining. It is the discovery of association rules. Once association analysis is completed, the association rules are explored and stored back into the relational database. In some sense, the association rules can be considered as plain data. Thus, we also can discover patterns hidden in the association rules.

This thesis will focus on finding the cyclic structures underlying the association rules because cyclic structures are very common and important in the real world. Such structures are known as meta-patterns [Seitzer et al, 1999], which are patterns of extracted rules. This task is called *cycle mining*.

To perform the task, we use knowledge in graph theory and combinational algorithms. There are three subtopics discussed in the thesis in chapter 2, 3 and 4. Firstly, we describe the properties of association rules. Each association rule can be represented using two vertices and one line with an arrowhead. We get a *directed graph* by graphically putting all association rules together. Hence, the directed graph can be used to represent the relationship among the association rules. Secondly, cycle mining is positioned as exploring all cycles existing in a directed graph. A backtracking algorithm using depth

first searching is used to perform the discovery. As to the efficiency of cycle mining, two strategies with pseudocode are provided. Thirdly, we compose a kind of data structure to implement the cycle mining algorithm. This data structure is based on a hash table with chaining. It is more efficient than adjacency matrix and adjacency linked list.

As the product of the research, a program has developed to perform cycle mining. The program has been written using Java programming language. MySQL database management system is selected as the back-end of the program. The program is based on the client-server architecture, so it can access database over networking connection.

In this introduction, we just briefly describe the topic and organization of this thesis because we also write introduction sections for most of chapters. Please refer to them for detailed information.

# CHAPTER 1

## DATA MINING AND ASSOCIATION RULES

### 1.1 Introduction

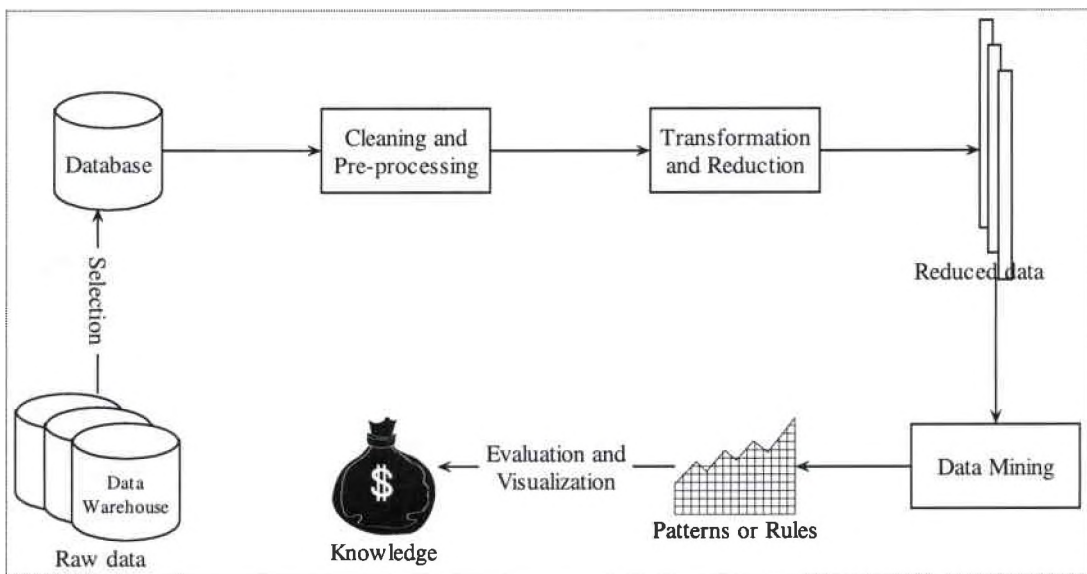
It is estimated that the amount of information in the world doubles every 20 months [Frawley et al, 1991]. The information has been and is being stored into databases in the form of computer files, which makes databases very large, amounting to gigabytes and even terabytes of data. At the same time, the users of these data are not satisfied with the traditional data analysis, whether OLTP or statistical analysis. They are expecting more sophisticated information from the databases in the form of understandable summaries. It is known that the kind of information is usually significant and meaningful patterns with strong possibility, but these patterns are not detected during traditional database management.

### 1.2 KDD and Data Mining

KDD is short for Knowledge Discovery in Databases, which is a powerful tool that is developing to solve the problem as mentioned in the previous section. KDD was defined as [Fayyad et al, 1996]:

*Knowledge Discovery in Databases is the non-trivial process of identifying valid, novel, potentially useful, and ultimately understandable structure in the data.*

Here, *data* is a set of facts that are collected from real world and stored into databases. *Structure* refers to patterns or rules [Elmasri et al, 2000]. There is not an absolutely strict definition for the term *Knowledge*. According to [Frawley et al, 1991], *knowledge* is a pattern that is interesting and certain enough. This depends on the user, specific domain, and is determined by whatever functions and thresholds the user chooses. KDD is a well-defined process, which involves many sequential steps including data preparation, data reduction, data modeling, data mining, knowledge evaluation, and solution analysis. KDD process can clearly be illustrated by Figure 1.1.

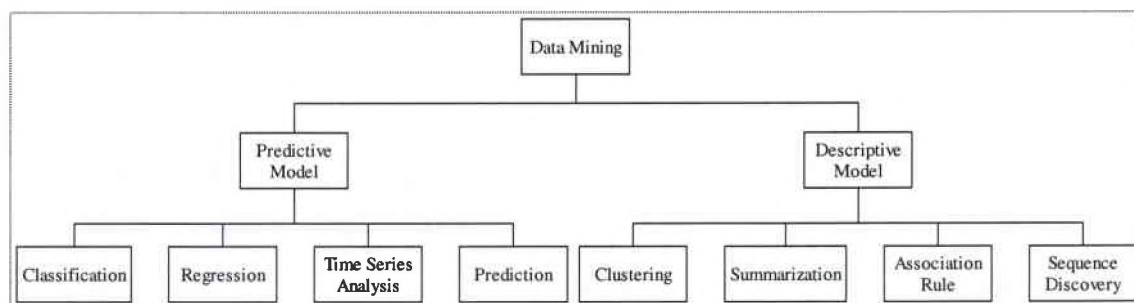


**Figure 1. 1 KDD Process (from Fayyad, 2001)**

According to the definition of KDD and the figure above, data mining is the core step in the KDD process [Džeroski, 2001]. It is concerned with discovering patterns or rules hidden in the data by applying computational techniques. Meanwhile, other steps in the process are considered as preparing data for the data mining as well as evaluating the discovered patterns or rules. Data mining is formally defined as [Fayyad et al, 1996]:

*Data mining is a step in the KDD process consisting of applying computational techniques that, under acceptable computation efficiency limitations, produce a particular enumeration of patterns (or models) over the data.*

In a sense, data mining is defined to focus on developing efficient algorithms by which patterns or rules in the large databases are discovered to meet specified thresholds.



**Figure 1. 2 Data Mining Models and Tasks (from Dunham, 2002)**

### **1.3 Data Mining Models and Tasks**

In the usage of data mining, many algorithms are involved to perform different tasks. All of the algorithms attempt to fit a model to the data in the database [Dunham, 2002] or discover a model from the data. In other words, data mining can be divided into two categories: prediction and description [Weiss et al, 1998; Dunham, 2002]. The description model accomplishes knowledge discovery. In this category, data can be identified, classified, summarized and optimized. Patterns or rules can be discovered through this kind of model. Prediction refers to the problems in term of specified goals, which are related to existing records with known answers. Prediction usually follows the knowledge discovery, where information is not enough to be used as prediction. On the other hand, knowledge discovery is complementary to prediction. In Figure 1.2, the tasks of predictive model and descriptive model are listed. [Dunham, 2002; Džeroski, 2001; Garofalakis et al, 1999] provides the definitions, usage and techniques of these tasks.

### **1.4 Association Rules**

Association analysis is one of the most important tasks of data mining as shown in the Figure 1.2. It is the discovery of association rules, which were first introduced in [Agrawal et al, 1993]. It is well known that market-basket analysis is the most common example of association analysis, and retail stores frequently use association rules to assist in marketing. [Agrawal et al, 1993; Dunham, 2002] give the formal definition of association rules and related terms. Because they are the basis of further work discussed later, it is important to show them here:

- **Association rule:** Given a set of items  $I = \{I_1, I_2, \dots, I_m\}$  and a database of transactions  $D = \{t_1, t_2, \dots, t_n\}$  where  $t_i = \{I_{i1}, I_{i2}, \dots, I_{ik}\}$  and  $I_{ik} \in I$ , an association rule is an implication of the form  $X \Rightarrow Y$  where  $X, Y \subset I$  are sets of items called itemsets and  $X \cap Y = \emptyset$ . Here,  $X$  is called *antecedent* of the association rule, and  $Y$  *consequent*.
- **Support:** The support for an association rule  $X \Rightarrow Y$  is the percentage of transactions in the database that contain both  $X$  and  $Y$ . A certain threshold is called *minsupport* represented by  $s$ .
- **Confidence:** The confidence for an association rule  $X \Rightarrow Y$  is the ratio of the number of transactions that contain both  $X$  and  $Y$  to the number of transactions that only contain  $X$ . A certain threshold is called *minconfidence* represented by  $\alpha$ .
- **Association analysis:** Given a set of items  $I = \{I_1, I_2, \dots, I_m\}$  and a database of transactions  $D = \{t_1, t_2, \dots, t_n\}$  where  $t_i = \{I_{i1}, I_{i2}, \dots, I_{ik}\}$  and  $I_{ik} \in I$ , the association analysis is to identify all association rules  $X \Rightarrow Y$  with minimum support and confidence. These values  $(s, \alpha)$  are given as input to the analysis.
- **Frequent itemset:** Frequent itemset is an itemset whose number of occurrences is above *minsupport*. The *minsupport* is usually given by domain experts.

Association rules specify the correlations between frequent itemsets. They can be directly derived from the existing data in the database, which is different from the



predictive rules. For predictive rules, patterns of the form “IF conditions THEN conclusion” can be used to accomplish a prediction for the target variable’s value, which may not be in the database.

## 1.5 Association Rule Mining Algorithms

As mentioned earlier, data mining is defined to focus on developing efficient algorithms. In the same sense, the research on the association rules attempts to develop efficient algorithms that can uncover implications beyond the data or items. The task of these algorithms are typically performed in two parts:

1. Find all frequent itemsets, where an itemset is frequent only if the number of its occurrences is above the *minsupport* ( $s$ ).
2. Generate rules from frequent itemsets, where the confidence of every rule passes the *minconfidence* ( $\alpha$ ).

The solution to the second part is pretty straightforward if the frequent itemsets have been determined. Hence, most of association rule mining algorithms focus on discovering frequent itemsets. The first algorithm is called Apriori algorithm developed by [Agrawal et al, 1994]. The algorithm uses *downward closure property* to optimize the memory usage. Since then, many techniques have been applied to develop the association rule mining algorithms. A summary and comparison of the existing algorithms is found in [Hipp et al, 2000; Zaki et al, 2001; Dunham, 2002].

## 1.6 Properties Related to Pruning Techniques

Careful pruning strategies can save a significant amount of execution time and space. There are some properties that can be used with association rule mining or other data mining techniques. Here, we just describe downward closure property and transitive property.

**Downward Closure Property:** For association rule mining, a frequent itemset is said to be downward closure if an itemset satisfies the minimum support requirement, so do all of its nonempty subset. Namely, all nonempty subsets of a frequent itemset must be frequent themselves. Let  $ABC^*$  be an itemset, where  $A, B, C$  are items. Its nonempty subsets are  $A, B, C, AB, AC, BC$  and  $ABC$ . If  $ABC$  is frequent, then so is each of these subsets. So only frequent  $k$ -itemsets that contain  $k$  items are used to build  $(k+1)$ -itemsets. Using the property, mining algorithms can prune the search space.

**Transitive property of Association Rules:** The definition of association rule  $X \Rightarrow Y$  was given above. It is binary because it consists of only two members: *antecedent*  $X$  and *consequent*  $Y$ .  $X$  and  $Y$  are sets of items. From the set theory, we can observe and prove that the relation on  $X$  and  $Y$  is transitive, that is,

$$\text{if } X \Rightarrow Y \text{ and } Z \subset Y, \text{ then } X \Rightarrow Z.$$

Here,  $Z$  is a subset of  $Y$ . In its formal definition, it should be written as

$$\text{if } X \Rightarrow Y \text{ and } Y \Rightarrow Z, \text{ then } X \Rightarrow Z.$$

---

\* For simplicity, we use  $ABC$  to mean a set  $\{A, B, C\}$ ,  $AB$   $\{A, B\}$  and  $A$   $\{A\}$ .

However, the way conflicts with the definition of association rule, since association rule requires  $X \cap Y = \emptyset$  and  $Z \cap Y = \emptyset$ . The minor modification still keeps the relation on  $X$  and  $Y$  transitive in the context of association rules. The property also can be the basis of the pruning techniques of efficient association rule mining algorithms. If the algorithms can find the association rule  $X \Rightarrow Y$  at first, then it is unnecessary to use complete mining process to discover the rule  $X \Rightarrow Z$ . In fact, there exist a number of itemsets like  $Z$  in the frequent itemsets discovered by generic association analysis algorithm [Elmasri et al, 2000].

## 1.7 Related Work

Of note is some recent work at University of Dayton computer science department. Based on Apriori algorithm, [Chen, 2001] proposed two association-rule mining algorithms: one offers high search speed with hash technique, and another uses less memory without hashing. Moreover, a data mining system, ACMS (Active Cycle Mining System) [Chen, 2001], was designed and implemented. Under UNIX and Oracle 8i database, the system was developed with Java programming language. It can be integrated with computing system INDED [Seitzer et al, 1999].

## CHAPTER 2

### CYCLIC STRUCTURE AMONG THE ASSOCIATION RULES

#### 2.1 Introduction

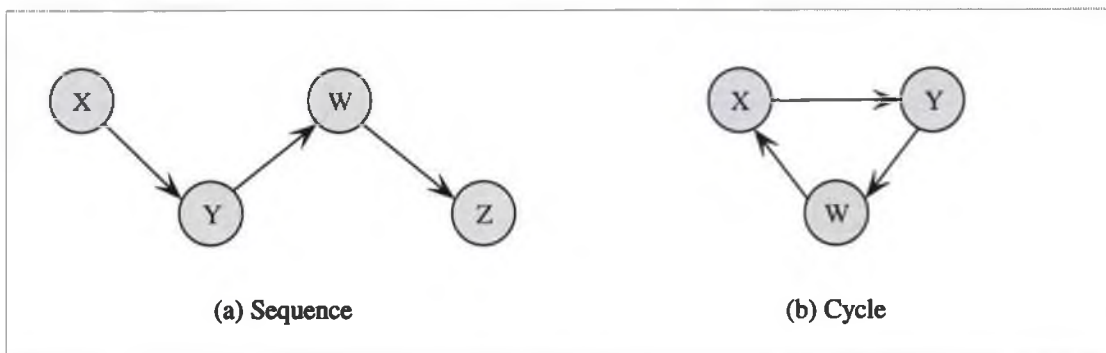
The first chapter described the association rule and its generation through computational techniques. If the association rules have been identified, the next step is to determine if there exist patterns or rules uncovered from the association rules. Table 2.1 shows a set of binary association rules derived from databases, where four items (W, X, Y, Z) are found to form the association rules and four rules are discovered.

**Table 2. 1 A Set of Example Association Rules**

Antecedent	Consequent	Association Rule
Y	W	$Y \Rightarrow W$
W	X	$W \Rightarrow X$
X	Y	$X \Rightarrow Y$
W	Z	$W \Rightarrow Z$

Examination of the rules in table 2.1 reveals two types of structures as illustrated in Figure 2.1. The combination of two nodes and one edge represents an association rule. The two structure types are sequential and cyclic. Such structures are known as meta-

patterns [Seitzer et al, 1999], which are patterns of extracted rules. In real problems, cyclic structure is common [Buckley et al, 1999] and important [Seitzer et al, 1999]. However, a cycle is considered as an undesirable thing in some cases, since it may mean deadlock or infinite loop. Hence, it is very important to detect the cycles hidden in the association rules too.



**Figure 2. 1 Structures Hidden in Example Association Rules**

## 2.2 Definition of a Cycle

A cycle is a path that begins and ends at the same vertex and contains at least one edge. It is formally defined as [Cormen et al, 2001]:

*A path of length  $k$  from a vertex  $u$  to a vertex  $u'$  in a graph  $G = (V, E)$  is a sequence  $\langle v_0, v_1, \dots, v_k \rangle$  of vertices such that  $u = v_0$ ,  $u' = v_k$  and  $(v_{i-1}, v_i) \in E$  for  $i = 1, 2, \dots, k$ . In a directed graph, a cycle is a path  $\langle v_0, v_1, \dots, v_k \rangle$ , when  $v_0 = v_k$ . The cycle is simple if  $v_0, v_1, \dots, v_k$  are distinct. A self-loop is a cycle of length 1.*

This is a generic definition for a graph. In the case of association rules, its cycles have several specific properties that will be discussed in section 2.4.

### 2.3 Cycles\* among Association Rules

Considering the variable time, we can classify cycles found in association rules into two categories: one is related to time and another not. A study [Özden et al, 1998; Ramaswamy et al, 1998] notes that association rules display regular cyclic variation over time. For instance, if association rules over monthly sales data are computed, “we may observe seasonal variation where certain rules are true at approximately the same month each year”. This is known as Cyclic Association Rule.

On the other hand, the cycles are directly derived from the association rules without temporal variable consideration. Only objects in the real world consist in the cycles, for instance, the cycles of the itemsets in market basket. This can be illustrated by the example in the section 2.1. We will focus on this type of cyclic relation among association rules in future work.

### 2.4 Properties of Cycles of Association Rules

The basic properties of a cycle among association rules can be summarized as follows:

- The cycle formed by association rules is directed. Association rule consists of one *antecedent* and one *consequent*.

---

\* For simplification, the term cycle is used as a synonym for cyclic structure or cyclic relation.

- A cycle of length 1 cannot exist in the context of association rules. The *antecedent* and *consequent* in an association rule are distinct, so a self-loop is impossible.
- The maximum length of the cycle is less than and equal to the number of the frequent itemsets.
- The cycle of association rules is *simple*. It encounters only one vertex twice, which is not only the start point but also end point of the cyclic path, and no other vertex more than once.
- The support of a cycle is the minimum support value of any the constituent rule forming the cycle. The confidence of a cycle is the minimum confidence value of any the constituent rules forming the cycle. Minimum support of a cycle is used to categorize the fuzzy cycles [Buckley et al, 1999]. So far, there is no detailed discussion about the implications and applications of both support and confidence of a cycle.
- The starting point of a cycle is defined as the antecedent of the rule that has maximum support of all constituent rules forming the cycle [Seitzer et al, 1999].

## 2.5 Near Cycle

The cycle described above is called a complete cycle [Buckley et al, 1999]. Similar to a complete cycle, there exists a near-cycle in a graph. Informally, the example\* shown in Figure 2.1-(a) is a near cycle. If we have an edge from Z to X, the cycle is complete; on

---

\* From the personal discussion with Dr. Buckley

the other hand, if we take away an edge from Figure 2.1-(b), then the cycle will become a near-cycle. “To discover all near-cycles from a graph” can be translated into “to explore trees from a graph”. The algorithms discussed in chapter 3 can be used with modifications, and we also can borrow algorithms that generate all spanning trees from a graph [Reingold et al, 1977]. Hence, this completes our discussion of near cycles.



## CHAPTER 3

### CYCLE MINING OF ASSOCIATION RULES

#### 3.1 Introduction

In the previous chapter, cycles of association rules were discussed. Discovering the cycles naturally becomes a primary task in our current research. Applications appear in many fields, such as scheduling problem in operations research, network analysis in electrical engineering, program segmentation in computer science [Reingold et al, 1977], and coding of ring compounds in organic chemistry [Deo et al, 1982]. In fact, solutions to these types of problems involve graph algorithms, which will be discussed below. The algorithms find all simple cycles of a graph.

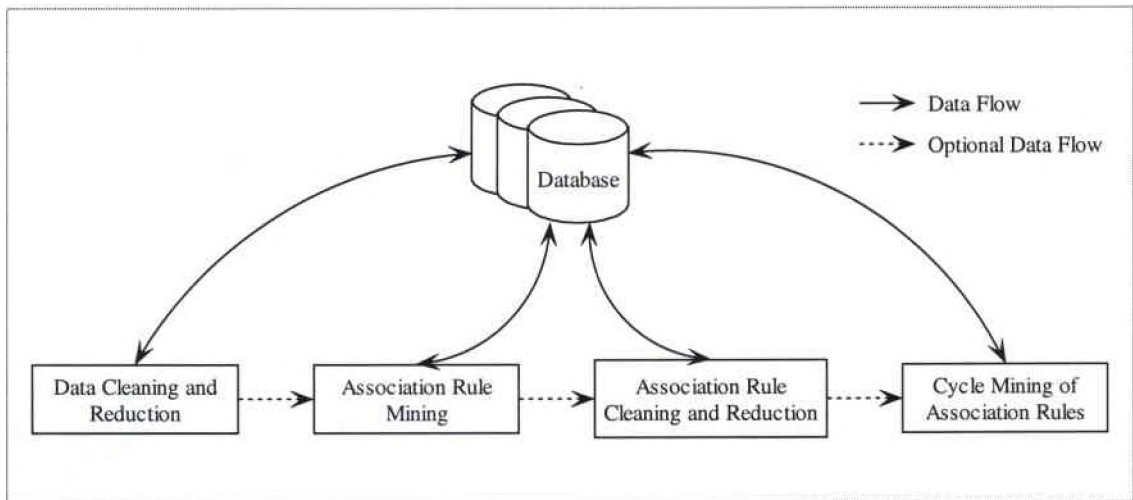
In the 1960's and 1970's, there were many articles describing algorithms to find all cycles of an undirected graph [Gotlieb et al, 1967; Paton, 1969; Gibbs, 1975]. It is harder to determine all cycles of a directed graph, also known as a digraph. Efficient algorithms have been proposed to enumerate all cycles in digraph [Tieran, 1969; Weinblatt, 1972; Johnson, 1975; Szwarcfiter et al, 1976], bounding the time-complexity to an exponential in the number of vertices. The algorithms that find all cycles of any graph can benefit from backtracking algorithms using depth-first search (DFS) [Reingold et al, 1977]. From

another perspective, cycle vector space algorithms exist that enumerate all cycles of a planar graph [Syslo, 1981; Dogrusöz et al, 1995]. Unlike backtracking algorithms, cycle vector space algorithms for cycle discovery are suitable for parallel computing.

Based on the previous discussion of cycles in association rules, we can clearly find that association rules can form a digraph. Thus, cycle mining of association rules can be limited to find all simple cycles of a digraph.

### 3.2 Architecture of Cycle Mining System

Before starting to discuss cycle mining in detail, we attempt to build the architecture of cycle mining system through combining cycle mining with KDD process. Figure 3.1 shows the system schematically.



**Figure 3. 1 Architecture of Cycle Mining System**

The system contains five processes and two types of data flow path, which are represented with solid line and dotted line. The solid line indicates that the data flow is

required and all processes must access the database to retrieve or back up data. The dotted line means that current process can directly pass results to the next process. It may speed the total accesses, but it is optional since all data can be retrieved from the database and backed up to the database.

- The first process performs data cleaning and reduction. It must obtain raw data from the database. After cleaning and reduction, it saves resulting data to the database. Meanwhile it optionally passes the data to the next process.
- The second process mines association rules from the data provided by the previous step. The process has an interface with the database that has been populated using association rules.
- Association rules needs to be cleaned and reduced in order to meet specified requirements.
- The process of cycle mining will discover the cyclic structures hidden in the association rules. Finally, the cycles must be stored in the database. The following sections will focus on the details of this step.
- All four processes above access the database for data retrieve or backup. The database may be centralized or distributed.

In this architecture, two of four processes are related to data cleaning and reduction. The arrangement may be helpful to keep the data quality high and improve the performance in the following steps. Moreover, the four processes must be carefully designed and implemented in order to be integrated with a well-defined database.

### 3.3 Framework of Cycle Mining of Association Rules

[Seitzer et al, 1999] proposed a four-step framework for cycle discovery. Based on this framework, the INDED system was implemented using inductive logic programming techniques. Here, we attempt to modify this framework to mine cycles among association rules. The framework consists of two steps:

1. Build an initial digraph to represent the logic relations among association rules from the database;
2. Generate all cycles from the digraph.

The first step solves two problems: determination of how to represent the relations among association rules, and the data structures to be used. We will present this in chapter four in more details.

In the second step, an algorithm must be used to traverse the digraph to determine all cycles. Here, the size of digraph varied with the process of cycle generation through increasing or decreasing the vertices. Both standalone sequential and parallel techniques can be used to determine all cycles. However, we will use a backtracking algorithm with depth-first search to perform the task here, instead of a parallel computing technique. The resulting cycles must be stored back in the database.

### 3.4 Algorithms for Cycle Mining

In the first section, we described that cycle mining of association rules can be limited to find all simple cycles of a digraph. Why to discover the cycle from a digraph, instead of a hypergraph? In a hypergraph, each of its hyperedges connects an arbitrary subset of vertices, rather than two vertices [Cormen et al, 2001]. Both *antecedent* and *consequent* in an association rule represent subsets of items, called frequent itemsets. If the items are the direct research objects, we must mine the cycle from a hypergraph. But once the association analysis has been completed, association rules can be considered as atomic elements. Hence, each of frequent itemsets can be represented as a vertex of a digraph, and the relationship between two frequent itemsets can be represented as an edge of a digraph. Hence, the terminology of graph theory will be used to describe algorithms in the following section.

#### 3.4.1 Requirements

Any effective algorithm for cycle mining must meet all the proposed requirements below.

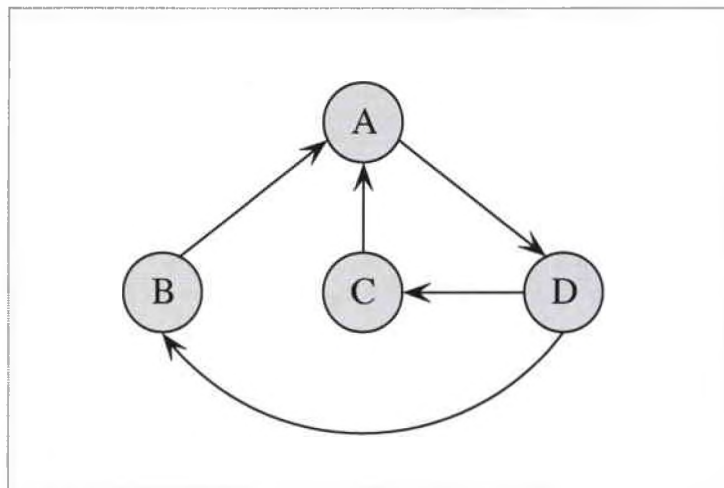
If any cannot be met, the algorithm will not be considered efficient.

1. Must discover ALL existing cycles from the association rules.
2. Must prevent a search that infinitely traverses cycles.
3. Must complete in reasonable time in  $n$ , the total number of *Consequents* and *Antecedent* of association rules. It has been shown that there exist more than  $(n - 1)!$  cycles if the graph is a complete digraph defined in [Robinson et al, 1980]. Thus, we can hardly expect an algorithm to be polynomial in  $n$ .

4. Every cycle should be simple, meaning every vertex occurs once at most.
5. Each cycle discovered should be unique, and not simply a permutation of cycles discovered earlier.

### 3.4.2 Generic Algorithms for Cycle Mining

In this type of algorithm, the initial graph is never subject to change once it is built. And depth-first search is used to find all cycles. There are two strategies. One is to search the cycles with a given length. The allowable range of this length is between 2 and the number of vertices in initial graph. The set of cycles with each length should cover all the cycles that exist in the association rules. Another method is to search the cycles related to a starting vertex. When the cycles starting at each vertex are found, we can know all cycles have been discovered.



**Figure 3. 2 Example Digraph**

A digraph is shown in Figure 3.2. There are four distinct vertices {A, B, C, D} and five edges. Let us only examine how the second strategy above. The complete evaluation procedure is summarized in Table 3.1, where six cycles are discovered.

In fact, we just can observe there are only two distinct cycles in Figure 3.2. Four occurrences of the cycles are duplications — cycle 1 (ADCA), cycle 4 (CADC) and cycle 5 (DCAD) are same, and cycle 2 (ADBA) is same as cycle 3 (BADB) and cycle 6 (DBAD). We will not evaluate the first strategy since it also causes the duplications in the same way. Neither approach can meet the fourth requirement above, therefore is neither efficient.

**Table 3.1 Evaluation of Cycles by a Generic Algorithm**

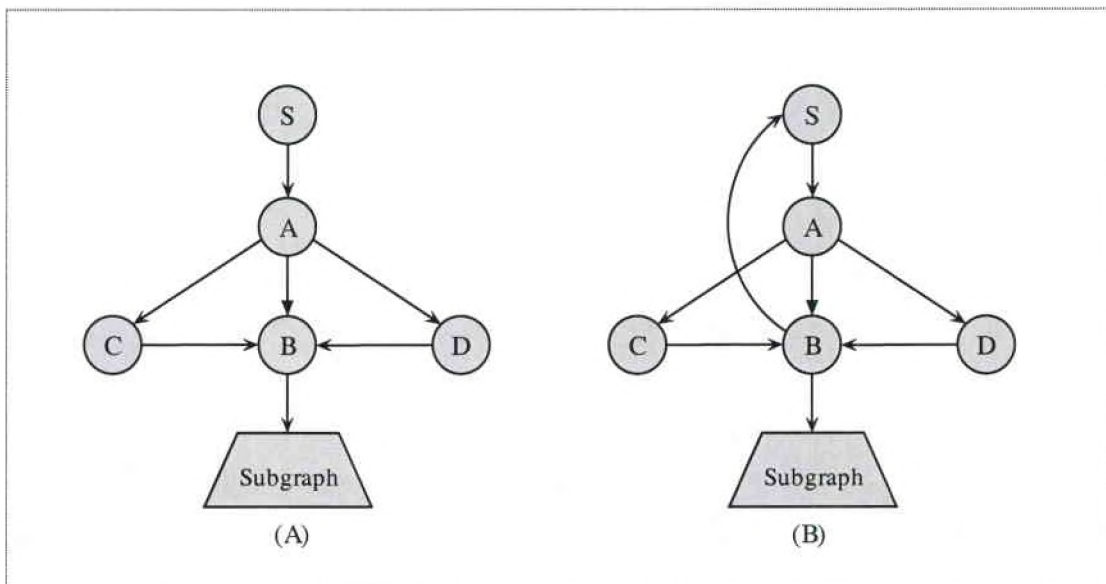
Step	Path	Action	Step	Path	Action
1	A	Start at vertex A	14	CA	
2	AD		15	CAD	Report cycle 4: CADC
3	ADC	Report cycle 1: ADCA	16	CA	Backtrack
4	AD	Backtrack	17	C	No cycle, clear path
5	ADB	Report cycle 2: ADBA	18	D	Start at vertex D
6	AD	Backtrack	19	DC	
7	A	No cycle, clear path	20	DCA	Report cycle 5: DCAD
8	B	Start at vertex B	21	DC	Backtrack
9	BA		22	D	Backtrack
10	BAD	Report cycle 3: BADB	23	DB	
11	BA	Backtrack	24	DBA	Report cycle 6: DBAD
12	B	No cycle, clear path	25	DB	Backtrack
13	C	Start at vertex C	26	D	No cycle, clear path

Note: “path” is a container that holds the visited vertices. Usually, it can be represented by a stack.

### 3.4.3 Pruning Strategies for Cycle Mining

Whether an algorithm for cycle mining is efficient or not depends on whether it adopts any pruning technique or not. If an algorithm uses a strategy to reduce useless work, it may be more efficient than the generic algorithm. There are two kinds of pruning techniques that can be used in cycle mining applications. Below, we will discuss and examine the two strategies.

**Strategy 1: mark vertex that cannot generate any cycle.** Suppose the current searching path is (S, A, B) and the next edge discovered is (B, Subgraph) shown in Figure 3.3-(A). Here, Subgraph may contain many of vertices and edges.



**Figure 3. 3 Pruning Strategies for Cycle Mining**

In this path, every vertex gets a flag that makes it unavailable. If there does not exist any vertex such that (S, A, B, Subtree) is a cycle, then we can ignore Subtree in the following



discovery. Since there are no discovered edges starting at vertex B, we back up to vertex A. If B led to a cycle, it can be reset so that it can be available again. Otherwise, it is left unavailable. In Figure 3.3-(B), B is reset to be available when we back up to A since B led to a cycle  $S \rightarrow A \rightarrow B \rightarrow S$ . Then we can explore the cycles:  $S \rightarrow A \rightarrow C \rightarrow B \rightarrow S$  and  $S \rightarrow A \rightarrow D \rightarrow B \rightarrow S$ . However, in Figure 3.3-(A), B is unavailable when we back up to A. Neither path (S, A, C) nor path (S, A, D) can be expended through visiting B and traversing Subgraph. We avoid useless traversal, as there is no a vertex E such that edge (E, S) is truth in Subgraph.

**Strategy 2: delete vertex to which all cycles related have been discovered.** In a cycle, vertices are not in an absolute order, but rather in relative order. Thus, whenever one vertex is the start of a traversal, all cycles involving the vertex can be explored, that is, no new cycles for the vertex can be found when the searching path starts at another vertex. Hence, an efficient algorithm must screen the vertex so that it cannot be visited repeatedly. Our strategy is to delete the vertex and the edges out from it as soon as the search starting at it is done. In Figure 3.3-(B), vertex S and edge (S, A) are deleted after the discovery. Edge (B, S) will similarly be detected when B becomes starting point.

### 3.4.4 Efficient Algorithms for Cycle Mining

Based on the discussion above, we will present two efficient algorithms for cycle mining of association rules. For easy readability, the algorithms use Java-based [Baase et al, 2000; Goodrich et al, 1998] pseudocode with detailed comments, rather than strict Java.

---

**Input:** digraph  $G=(V, E)$  represented by a suitable data structure.  
 The digraph covers all vertices and edges initially.  
**Output:** all cycles that exist in digraph without any duplicate

```

MiningAllCycleTD(Digraph digraph)

1.  //create a stack to record the search path
2.  path = new stack();
3.
4.  //assign the first available vertex to start
5.  start = digraph.firstVertex();
6.
7.  //explore all cycles starting at every vertex in digraph
8.  while(start or  $|V|>1$ ) // $|V|$  is the number of vertices
9.    //reset every vertex in digraph available
10.   for(vertex  $\in$  digraph.vertice())
11.     avail(vertex) = true;
12.     B(vertex).empty();
13.
14.   //discover all cycles starting at vertex "start"
15.   cycleDetect(start, start, flag);
16.
17.   //delete start after search according to strategy 2
18.   digraph.delete(start);
19.
20.   //extract next available vertex
21.   start = digraph.nextVertex();
22.
23.  return;

```

---

**Figure 3. 4 Algorithm for Cycle Mining: MiningAllCycleTD()**

For the first algorithm shown in Figure 3.4, the input is a digraph that has been built to cover all vertices and edges. The algorithm is called MiningAllCycleTD, short for Mining All Cycle using Top-Down approach. This recursive algorithm finds all cycles starting at a specified vertex, implemented through a procedure cycleDetect() shown in Figure 3.6. This procedure was initially developed in [Johnson, 1976]. After executing

the procedure, the initial digraph is pruned as demonstrated in **Strategy 2**. When only one vertex remains in the digraph, we can say that all cycles are explored. In this algorithm, there are two variables, `digraph` and `path`. They are visible to procedure `MiningAllCycleTD()` and `cycleDetect()`. In line 11, function `avail(vertex)` is used to indicate the availability of each vertex. In line 12, the  $B(\text{vertex})$  is a set that can be defined as

$$B(w) = \{v \in V \mid (v, w) \in E, v \text{ is unavailable and not on the current path}\}$$

Here,  $V$  is the set of all vertices in a digraph,  $E$  is the set of all edges in a digraph, and  $w \in V$ . In plain English,  $B(w)$  is the set of unavailable predecessors of  $w$ .  $B(\text{vertex})$  is visible to both `MiningAllCycleTD()` and `cycleDetect()`.

The next algorithm is shown in Figure 3.5. Unlike the first algorithm, the input is an empty digraph. The algorithm is called `MiningAllCycleDT`, short for Mining All Cycle using Down-Top approach. Recursive procedure `cycleDetect()` in Figure 3.6 is used here as well. All notations are same as those in first algorithm.

When a new edge is inserted into the digraph, procedure `cycleDetect()` then finds all cycles including the edge. Until there are no new edges, all cycles contained in the current digraph are discovered. During the mining process, the number of vertices in digraph decreased in the first algorithm through removing vertices shown in line 18, while the second one increases the number by 1 once. However, the two algorithms are similar in that they both attempt to prune the traversals of digraph.

---

**Input:** digraph  $G=(V, E)$  represented by a suitable data structure.  
 The digraph doesn't include any vertex and edge initially.  
**Output:** all cycles that exist in digraph without any duplicate

```

MiningAllCycleDT(Digraph digraph)

24.  //create a stack to record the search path
25.  path = new stack();
26.
27.  get a new edge (head, end) from input;
28.  edge = new edge(head, end);
29.
30.  //explore all cycles starting at every vertex in digraph
31.  while(edge)
32.    //add new edge into digraph
33.    digraph.addEdge(edge);
34.
35.    //assign the end of new edge as start point
36.    start = edge.head();
37.
38.    //reset every vertex in digraph available
39.    for(vertex  $\in$  digraph.vertice())
40.      avail(vertex) = true;
41.      B(vertex).empty();
42.
43.    //discover all cycles starting at vertex "start"
44.    cycleDetect(start, edge.end(), flag);
45.
46.    get a new edge (head, end) from input;
47.    edge = new edge(head, end);
48.
49.  return;

```

---

**Figure 3.5** Algorithm for Cycle Mining: MiningAllCycleDT()

Obviously, algorithm cycleDetect() is the core of cycle mining for association rules. It can explore all cycles starting at a specified vertex or edge. While it uses recursive depth-first search, it also performs the pruning through **Strategy 1**. Function avail(vertex)

in line 51 makes a vertex unavailable when it is pushed into the traversal path, thus avoiding duplicate cycles. On the other hand, algorithm unmark() makes vertices that can generate cycles available again. For more details, see the algorithms and their comments.

---

**Input:** starting vertex of a cycle, any vertex in digraph that can be added into path, and flag that used to indicate if there is a cycle discovered. The type of start and vertex is **Object**, a generic type.

**Output:** all cycles starting at the specified vertex

Boolean cycleDetect(**Object** start, **Object** vertex)

```

50.   flag = false;
51.   avail(vertex) = false;
52.
53.   //add vertex into the search path
54.   path.push(vertex);
55.
56.   while(w ∈ Adjacency(vertex))
57.       if(w == start)
58.           output cycle consisting of vertices in path;
59.           flag = true;
60.       else if(avail(w) == true)
61.           flag2 = cycleDetect(start, w);
62.           flag = flag || flag2;
63.
64.   //make it available again if vertex can generate a cycle
65.   if(flag == true)
66.       unmark(vertex);
67.   else
68.       while(w ∈ Adjacency(vertex))
69.           B(w).add(vertex);
70.
71.   //delete vertex from the search path;
72.   path.pop(vertex);
73.
74.   return flag;

```

---

Figure 3. 6 Algorithm: cycleDetect()

---

**Input:** a vertex in digraph. Its type can be any that you want.

**Output:** make the input vertex available again, and make its predecessors available again if they are not in the current path

```

unmark(Object vertex)

75.  avail(vertex) = true;
76.
77.  for(w ∈ B(vertex))
78.    B(vertex).delete(w);
79.
80.    //make w available again if it isn't in current search path
81.    if(avail(w) == false)
82.      unmark(w);
83.
84.  return;

```

---

Figure 3. 7 Algorithm: unmark()

### 3.4.5 Analysis of Algorithms for Cycle Mining

It has been proven that at most  $O(|V| + |E|)^*$  time can elapse in a call to cycleDetect() before either the call returns or a cycle is reported [Johnson, 1976]. Thus, the algorithms listed in section 3.4.4 executes in  $O((|V| + |E|)(c + 1))$ , where  $c$  is the number of cycles in the digraph. For a complete digraph with  $n$  vertices, the bound can be  $O(n^2 \cdot (c + 1))$ . The upper bound of the number of the number of cycles,  $c$ , can be

$$\sum_{i=2}^n \binom{n}{i} (i-1)! > (n-1)!$$

---

\*  $|V|$  represents the number of vertices in a digraph, and  $|E|$  is the number of edges.

Thus, the speed of the algorithms completely depends on the number of cycles existing in the target digraph. If the number of cycles is known to be small, the algorithm is very efficient.

## CHAPTER 4

### DATA STRUCTURE FOR CYCLE MINING

#### 4.1 Introduction

As discussed earlier, data mining is different from the OLTP and statistical analysis because of the large quantity of data being processed. It is most common that the input to data mining algorithm is only one large table. In some sense, we cannot assume that the size of association rules is small. When the values of *minsupport* and *minconfidence* are very small, there are still a relative large number of association rules to be discovered. Hence, it is important to choose a data structure for cycle mining that minimizes memory requirements. In addition to memory management, the structures must provide fast search and efficient deletion and insertion operations due to the prominence of those operations in any data mining algorithm.

#### 4.2 Data for Cycle Mining

From the perspective of a database management system, the input to cycle mining primarily consists of four tables. Market-basket database serves as a typical example of this data format. The definitions are listed in Table 4.1. Of these, cycle mining is concerned only with the *association rule* table as its input.



Table 4. 1 Definitions of the Tables Related to Cycle Mining

Table Name	Definition of table			
<i>Item</i>	<u>Item ID</u>	Item Description	...	
<i>Transaction</i>	<u>Transaction ID</u>	<u>Item ID</u>	...	
<i>Frequent Itemset</i>	<u>Itemset ID</u>	<u>Item ID</u>	Support	
<i>Association Rule</i>	<u>Rule ID</u>	Antecedent	Consequent	Confidence
<i>Cycle</i>	<u>Cycle ID</u>	<u>Rule ID</u>	OrderNum	

Note: underlined words are the primary keys of the corresponding tables; ellipsis represents the attributes that may participate in the relations.

The column *OrderNum* enumerates vertices in a cycle in relative order. For instance, the cycle is  $A \rightarrow B \rightarrow C \rightarrow A$ . The order of A is 1, order of B is 2, and order of C is 3. Figure 4.1 shows referential integrity constraints displayed on the relation definitions in Table 4.1. Both *Antecedent* and *Consequent* in table *association rule* reference the Itemset ID of table *frequent itemset*.

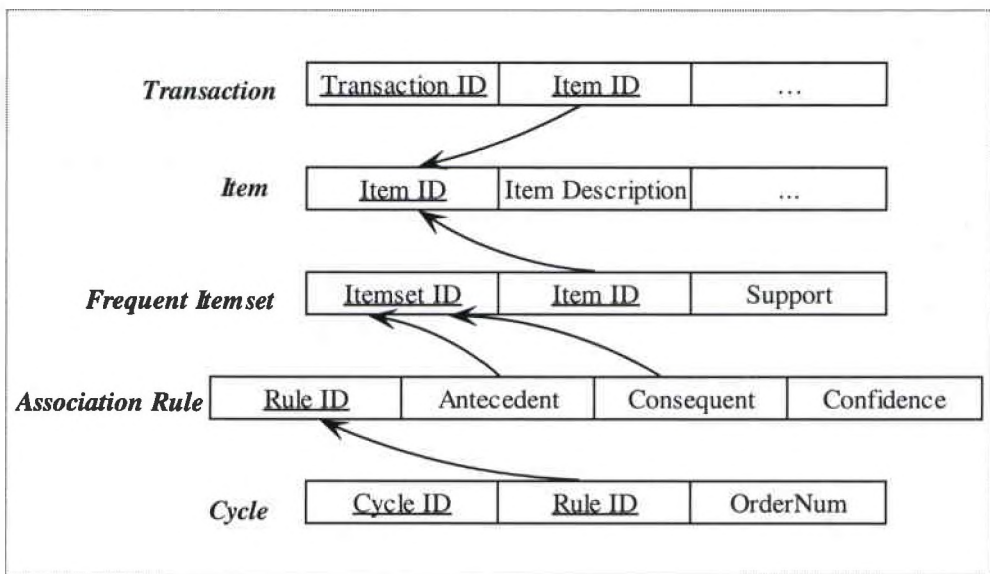


Figure 4. 1 Referential Integrity Constraints Displayed on Tables

## 4.3 Data Representation

Chapter 2 described how the relations among the association rules could be described as a digraph. Either an adjacency matrix or adjacency linked list usually is used to represent the graph structure. However, both of the representations cannot be used effectively with association rules. The first reason is that both may waste a lot of memory resource since the matrix usually is sparse, meaning many nodes have no out edges. Second, the keys of elements usually are not same as the indices of an array, so the search operation executes at  $O(n)$  time. Hence, we may need others representations. Hash table and hash tree are the candidates. The hash techniques support constant average-time search, insertion and deletion operations [Cormen et al, 2001; Weiss, 1998]. The performance is desired by cycle mining.

### 4.3.1 Hash Table with Chaining

Figure 4.2 is a hash table with chains. The number of bucket is controllable, and overflows in general graph representations are handled by chaining. Moreover, chaining can avoid collisions in a hash table.

In the hash table, each node in chains has its structure as shown in Figure 4.2. Except that a node *Antecedent* can point to next node, it also can link with a chain. The chain stores the key of nodes *Consequents* that are directly reached from *Antecedent*. For this representation, the ADT looks like that shown in Figure 4.3.

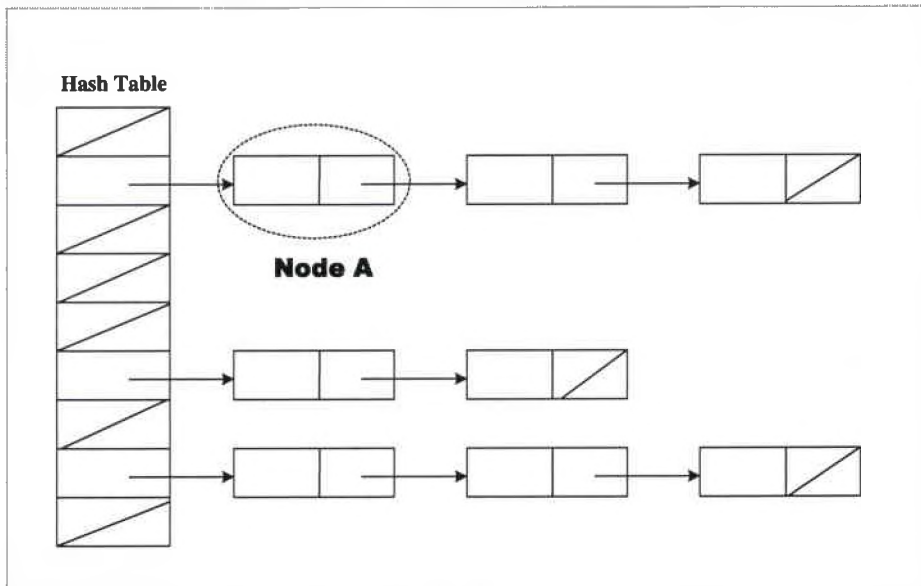


Figure 4. 2 Hash Table with Chaining

The example shown in Table 2.1 can allow us to give a close look at **Node A** in Figure 4.2. Suppose the information of **Node A** is *W*. Figure 4.4 can make it clear. *Antecedent* *W* in **Node A** has two *consequents*, *X* and *Z*, which are stored in a linked list. Moreover, the node has a pointer to the next node that has same hash key as **Node A**.

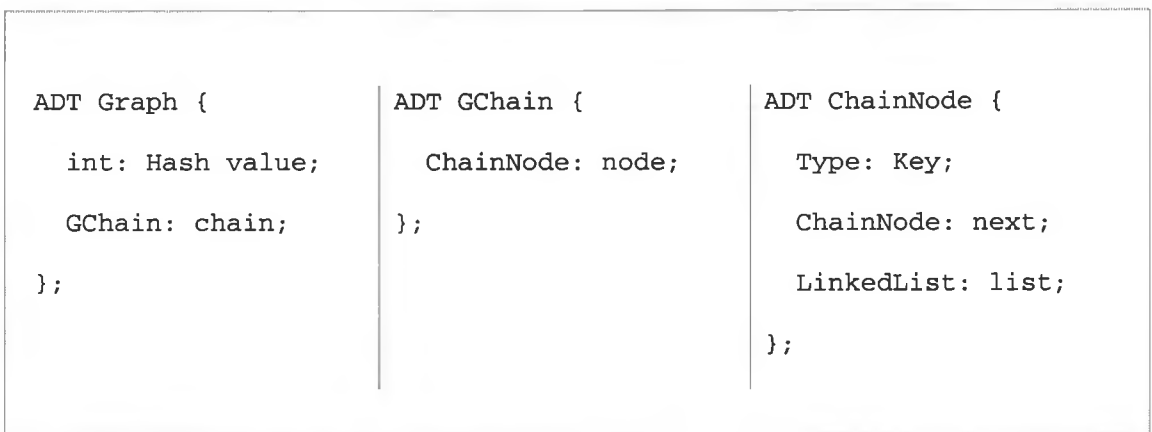
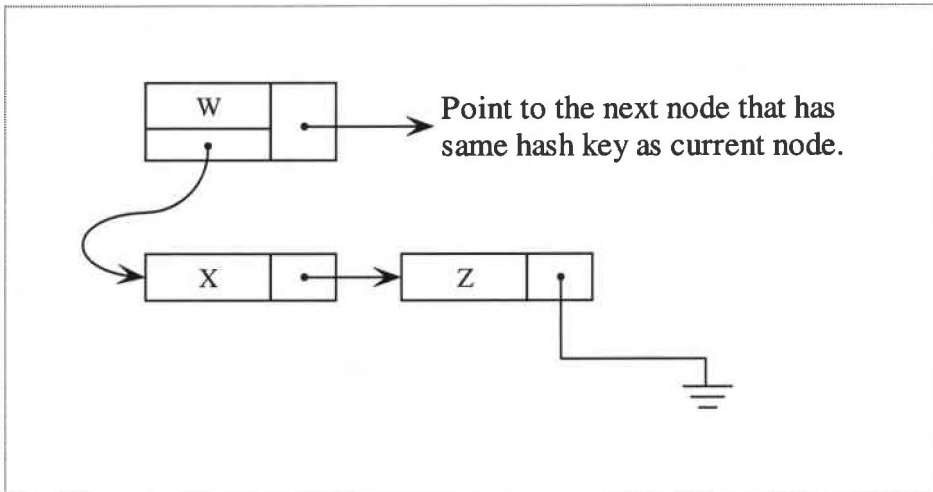


Figure 4. 3 ADT of Graph for Cycle mining



**Figure 4. 4 Close Look of Node A**

### 4.3.2 ADT Specifications of Cycle Mining

Below, we list the ADT specifications for cycle mining. Here, the types and arguments of operations are omitted for simplification. We directly use well-known data structures, such as Stack, HashTable and LinkedList. Those specifications are Java-based.

**Table 4. 2 ADT Specification of a Cycle**

---

```

ADT Cycle{
  Instances:
    Digraph digraph;
    private Stack path;
  Operations:
    buildInitialDigraph(); //build the initial digraph for cycle mining
    cycleDetect(); //examine if there exists cycles in digraph
    isCycle(); //determine if the current path is a cycle
    outputCycle(); //output a cycle
    lengthOfCycle(); //compute the length of a cycle
    numOfCycles(); //compute how many cycles are explored
};

```

---

---

**Table 4. 3 ADT Specification of a Digraph**


---

```

ADT Digraph{
  Instances:
    HashTable hashTable;
  Operations:
    insert(); //insert a new vertex into digraph
    delete(); //delete the specified vertex from digraph
    search(); //search the specified vertex
    numOfVertices(); //compute the number of vertices in digraph
    numOfEdges(); //compute the number of edges in digraph
    outputDigraph(); //output the digraph
};

```

---



---

**Table 4. 4 ADT Specification of a Node of Digraph**


---

```

ADT DigraphNode{
  Instances:
    Object info; // info of a digraph node
    LinkedList adjacencyNode; //info of nodes adjacent to current node
  Operation:
    create(); //create a new node
    addEdge(); //add a new edge out from current node
    deleteEdge(); //delete an edge
    isEdage(); //determine if the specified edge exists
    outputNode(); //output current node with its edges
};

```

---

#### **4.4 Future Study**

It is very common that a hash tree is adopted to implement association rule mining. The Hash tree provides an effective way to store, access, and count itemsets. It is able to efficiently search, insert and delete itemset. A hash tree is a multiway search tree. At each level the candidate itemsets are stored in a hash tree by applying a hash function. Like association rule mining, cycle mining involves search, insertion and deletion. Hence, the hash tree may become a candidate data structure for cycle mining. In order to be useful, such a hash tree must be well balanced.

# CHAPTER 5

## IMPLEMENTATION OF CYCLE MINING OF ASSOCIATION RULES

### 5.1 Environments

As a product of this paper, a cycle mining system has been developed and tested on the two systems as detailed below. We adopt MySQL database server as the backend of the program, because MySQL is one lightweight and open source database software. Moreover, the DBMS provides a very friendly interface with Java 2 SDK.

**Table 5. 1 Environments of Implementation**

<b>System I</b>	
CPU	Intel Pentium II 300MHz
Memory	128MB
Operating System	Windows XP
Programming Language	Java 2 SDK 1.4.1
JDBC	MySQL Connector/J 3.0
Database Management System	MySQL 4.0
<b>System II</b>	
CPU	AMD Athlon 800MHz
Memory	384MB
Operating System	Windows XP
Programming Language	Java 2 SDK 1.4.1
JDBC	MySQL Connector/J 3.0
Database Management System	MySQL 3.23

## 5.2 Implementation

The program has been written in JAVA programming language. It implements the first algorithm listed in Chapter 3 using a hash table with chaining. In addition to the operations described in the algorithm, many database operations are involved. Although the operations can degrade the performance of the program, it is necessary to sort the *association rule* table by *Antecedent* and then *Consequent*, in order to easily create a digraph node with all edges. The package involves eleven classes, five of which are used to implement a graphical user interface. Figure 5.1 demonstrates the uses relations among those classes.

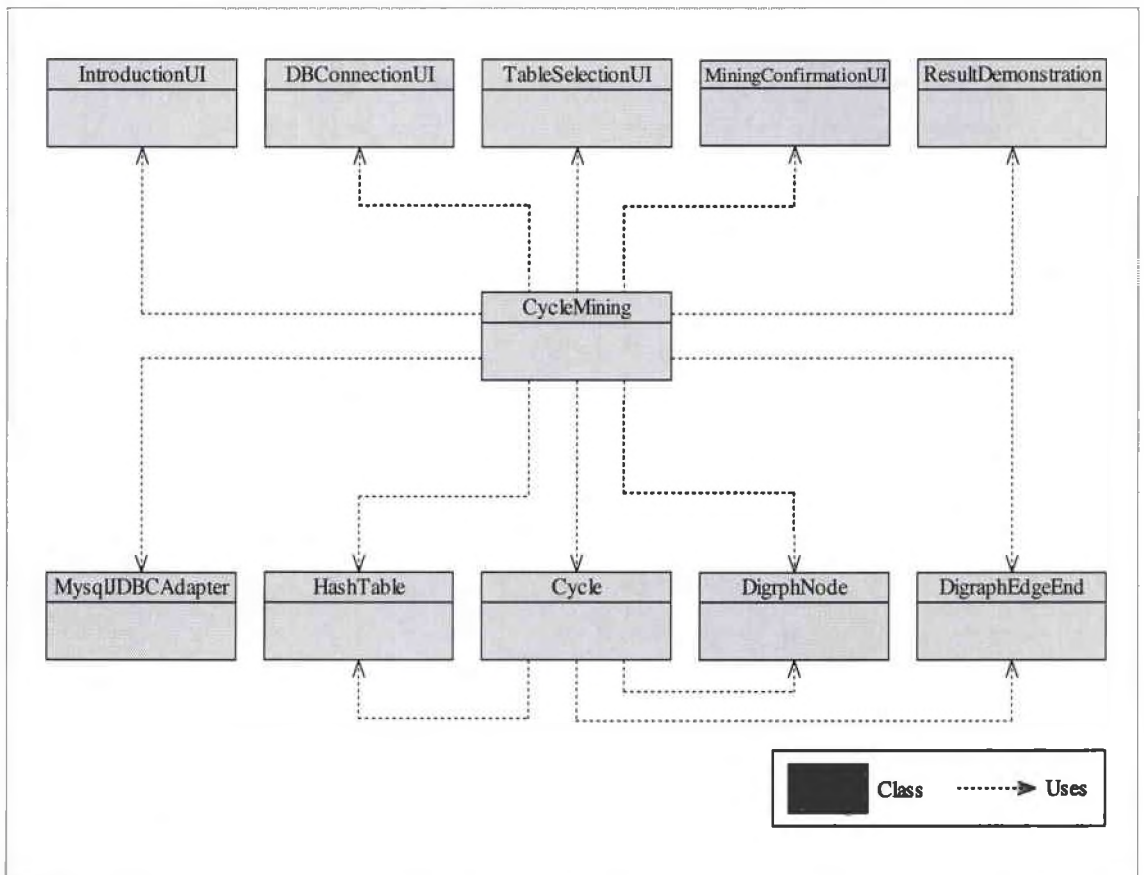


Figure 5.1 Uses Relations among Classes

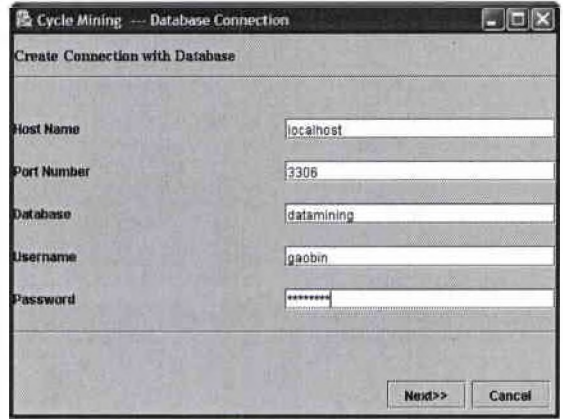


Table 5.2 Graphical User Interface

Step 1. Introduction



Step 2. Database Connection



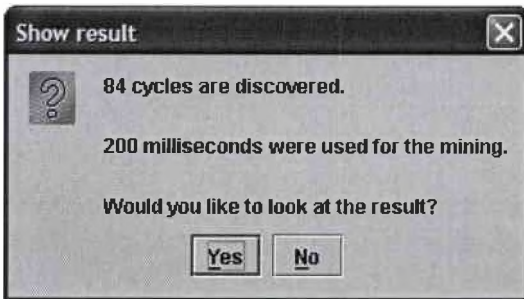
Step 3. Table Selection



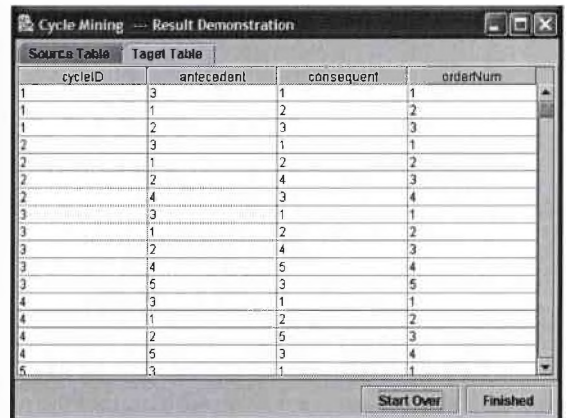
Step 4. Mining Confirmation



Option Dialog



Step 5. Result Demonstration



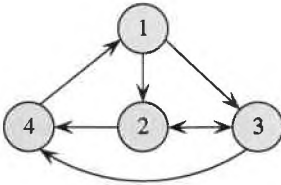
### 5.3 Graphical User Interface

The graphical user interface consists of five sequential pages and several option dialogs. They provide the information about the program and database, and prompt users to enter required information or choose desired options. The five pages are respectively named as introduction, database connection, table selection, mining confirmation, and result demonstration. They are presented step-by-step. Table 5.2 lists the interface partially.

### 5.4 Examples\*

The results of cycle mining are stored in a table of the database, and users can also check the result through the log file or console. Four small examples are listed in Table 5.3 as reference.

**Table 5.3 Examples of Cycle Mining via the Program**

Graphic Representation	Discovered Cycles	Cycles in Database																																																																				
	<p>[3,2], [3,2,4,1], [3,4,1,2], [3,4,1], [1,2,4]</p>	<table border="1"> <thead> <tr> <th>ID</th> <th>A</th> <th>C</th> <th>O</th> </tr> </thead> <tbody> <tr><td>1</td><td>3</td><td>2</td><td>1</td></tr> <tr><td>1</td><td>2</td><td>3</td><td>2</td></tr> <tr><td>2</td><td>3</td><td>2</td><td>1</td></tr> <tr><td>2</td><td>2</td><td>4</td><td>2</td></tr> <tr><td>2</td><td>4</td><td>1</td><td>3</td></tr> <tr><td>2</td><td>1</td><td>3</td><td>4</td></tr> <tr><td>3</td><td>3</td><td>4</td><td>1</td></tr> <tr><td>3</td><td>4</td><td>1</td><td>2</td></tr> <tr><td>3</td><td>1</td><td>2</td><td>3</td></tr> <tr><td>3</td><td>2</td><td>3</td><td>4</td></tr> <tr><td>4</td><td>3</td><td>4</td><td>1</td></tr> <tr><td>4</td><td>4</td><td>1</td><td>2</td></tr> <tr><td>4</td><td>1</td><td>3</td><td>3</td></tr> <tr><td>5</td><td>1</td><td>2</td><td>1</td></tr> <tr><td>5</td><td>2</td><td>4</td><td>2</td></tr> <tr><td>5</td><td>4</td><td>1</td><td>3</td></tr> </tbody> </table>	ID	A	C	O	1	3	2	1	1	2	3	2	2	3	2	1	2	2	4	2	2	4	1	3	2	1	3	4	3	3	4	1	3	4	1	2	3	1	2	3	3	2	3	4	4	3	4	1	4	4	1	2	4	1	3	3	5	1	2	1	5	2	4	2	5	4	1	3
ID	A	C	O																																																																			
1	3	2	1																																																																			
1	2	3	2																																																																			
2	3	2	1																																																																			
2	2	4	2																																																																			
2	4	1	3																																																																			
2	1	3	4																																																																			
3	3	4	1																																																																			
3	4	1	2																																																																			
3	1	2	3																																																																			
3	2	3	4																																																																			
4	3	4	1																																																																			
4	4	1	2																																																																			
4	1	3	3																																																																			
5	1	2	1																																																																			
5	2	4	2																																																																			
5	4	1	3																																																																			

\* All examples and test cases are randomly generated by the program.

	<p>[3,2,4]</p>	<table border="1"> <thead> <tr> <th>ID</th> <th>A</th> <th>C</th> <th>O</th> </tr> </thead> <tbody> <tr> <td>1</td> <td>3</td> <td>2</td> <td>1</td> </tr> <tr> <td>1</td> <td>2</td> <td>4</td> <td>2</td> </tr> <tr> <td>1</td> <td>4</td> <td>3</td> <td>3</td> </tr> </tbody> </table>	ID	A	C	O	1	3	2	1	1	2	4	2	1	4	3	3
ID	A	C	O															
1	3	2	1															
1	2	4	2															
1	4	3	3															
	<p>No Cycle</p>																	
<p>(Complete Digraph)</p>	<p>[1,2,4,5], [1,2,4], [1,2,5,4], [1,2,5], [1,2], [1,4,2,5], [1,4,2], [1,4,5,2], [1,4,5], [1,4], [1,5,2,4], [1,5,2], [1,5,4,2], [1,5,4], [1,5], [2,5], [3,1,2,4,5], [3,1,2,4], [3,1,2,5,4], [3,1,2,5], [3,1,2], [3,1,4,2,5], [3,1,4,2], [3,1,4,5,2], [3,1,4,5], [3,1,4], [3,1,5,2,4], [3,1,5,2], [3,1,5,4,2], [3,1,5,4], [3,1,5], [3,1], [3,2,1,4,5], [3,2,1,4], [3,2,1,5,4], [3,2,1,5], [3,2,1], [3,2,4,1,5], [3,2,4,1], [3,2,4,5,1], [3,2,4,5], [3,2,4], [3,2,5,1,4], [3,2,5,1], [3,2,5,4,1], [3,2,5,4], [3,2,5], [3,2], [3,4,1,2,5], [3,4,1,2], [3,4,1,5,2], [3,4,1,5], [3,4,1], [3,4,2,1,5], [3,4,2,1], [3,4,2,5,1], [3,4,2,5], [3,4,2], [3,4,5,1,2], [3,4,5,1], [3,4,5,2,1], [3,4,5,2], [3,4,5], [3,4], [3,5,1,2,4], [3,5,1,2], [3,5,1,4,2], [3,5,1,4], [3,5,1], [3,5,2,1,4], [3,5,2,1], [3,5,2,4,1], [3,5,2,4], [3,5,2], [3,5,4,1,2], [3,5,4,1], [3,5,4,2,1], [3,5,4,2], [3,5,4], [3,5], [4,2,5], [4,2], [4,5,2], [4,5]</p>	<p>There are 84 cycles discovered, and 320 rows are used to store the all cycles in database. The size is so large that they cannot be listed here.</p>																

Note: In the fourth row, ID, A, C and D are short for cycleID, antecedent, consequent and OrderNum respectively. For definition of the table, refer to table “cycle\_lazy” in Appendix 1.

## 5.5 Performance Analysis

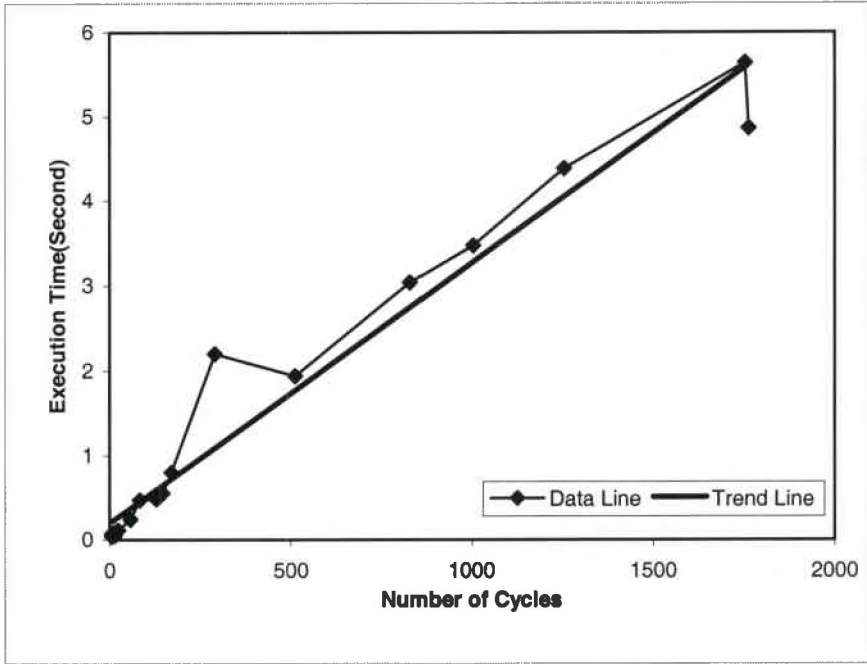
The program also provides a function that can be used to randomly generate association rules. There are three factors that can influence the performance of the algorithm described in Chapter 3. They are the number of association rules, the number of itemsets, and the number of cycles. The first two of them can be controlled, but we cannot control the third one. In Table 5.4, the parameters for performance analysis are listed. For the first case, the number of association rules is set to 40 and the number of itemsets varies between 7 and 25.

**Table 5. 4 Parameters for Performance Analysis**

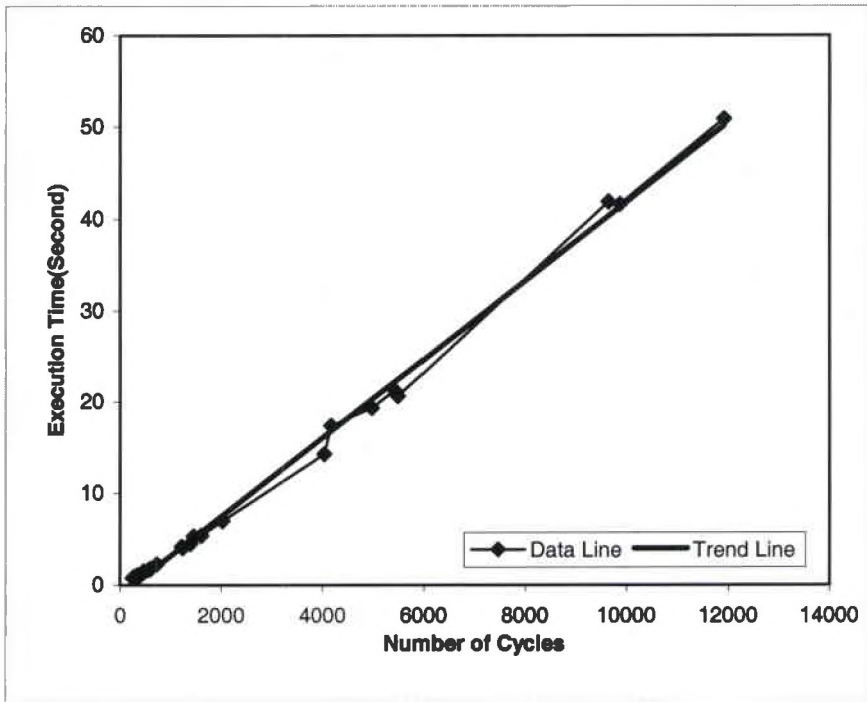
Case	Number of association rules	Number of Itemsets	Performance Chart
1	40	Between 7 and 25	Figure 5.2
2	Between 35 and 55	10	Figure 5.3
3	100	50	Figure 5.4

Note: System II in Table 5.1 is used to make those performance tests.

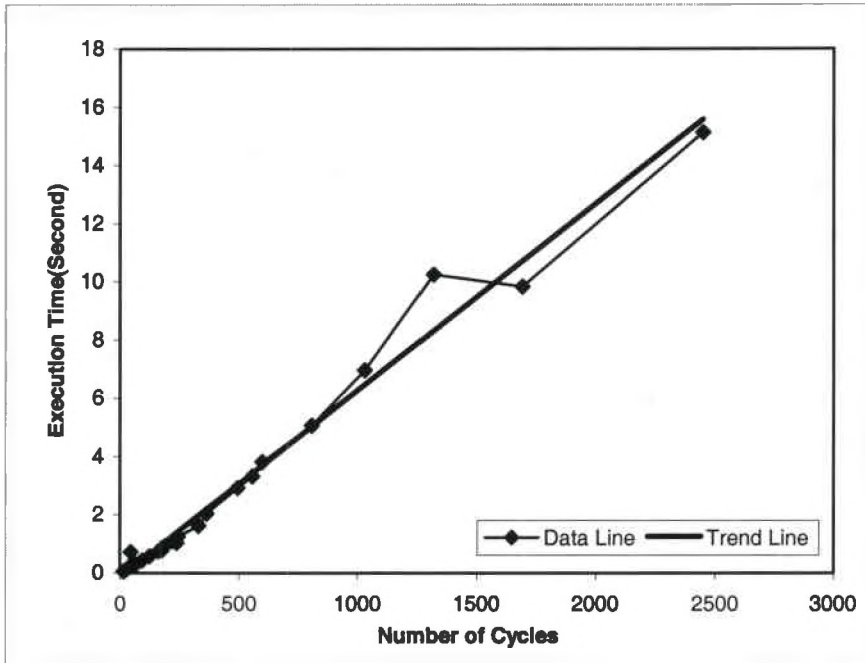
From Figure 5.2, 5.3 and 5.4, we can observe that the execution time is related to the number of cycles linearly. The observation is consistent with which has been discussed. The speed of the algorithms completely depends on the number of cycles existing in the target digraph. For example, we have 40 itemsets and 1560 association rules, and then the number of cycles may be more than  $39!$  ( $=2.0397882081197443358640281739903e+46$ ). In fact, we can never obtain all cycles of this example. Suppose that we could get them, we can never understand them. Hence, there exist many limits for cycle mining, which can be resulted from the algorithms or the problem of cycle mining itself.



**Figure 5.2 Performance Chart for 40 Association Rules**



**Figure 5.3 Performance Chart for 10 Itemsets**



**Figure 5. 4 Performance Chart for 100 Association Rules with 50 Itemsets**

## CHAPTER 6

### SUMMARY AND CONCLUSIONS

#### 6.1 Outline of the Contributions of This Work

- Representing the relationship among association rules as a digraph;
- Surveying several graph algorithms, and introducing them to cycle mining with modification;
- Composing a data structure based on the hash table with chaining to represent a digraph;
- Defining the database schema for association rule mining and cycle mining;
- Introducing MySQL database management system to our research;
- Developing a program to explore the cycles among the association rules.

#### 6.2 Conclusions and Discussion

Based on the research and simulation on cycle mining, the most important conclusion is that the speed of the algorithms completely depends on the number of cycles existing among association rules.

The second conclusion is that the current research result can efficiently work on the cases in which the number of cycles ( $c$ ) is small. The number may be more than the factorial of

the number of frequent itemsets ( $i$ ). If there are many such cases in the real applications, our research will be a futile effort since cycles are in such a large quantity that we can hardly understand them. However, how small is the number? Experiment in Figure 5.3 shows that the program can enumerate 11916 cycles within 50.884 seconds and the execution time is related to the number linearly. So we can roughly estimate the execution time through the number of association rules ( $a$ ), and  $i$ . The closer to  $i \cdot (i-1) a$  is, the larger  $c$  will be, and then the longer the execution time will be.

When the number of frequent itemsets ( $i$ ) is larger, the execution time also increases. In an experiment where  $a = 87710$  and  $i = 65000$ , the program spent about 421 seconds discovering all cycles and there are just 3 cycles. Though the execution time is related to the number of itemsets, the program still can be performed in reasonable time.

Based on the analysis above, we may hope that the cycle mining works on the case with a small amount of cycles. This is a limit of the current cycle mining. In the future study on cycle mining, it is an important topic how to discover useful and meaningful cycles from association rules. On the other hand, divide-and-conquer can be used to improve the performance of cycle mining when the dataset of association rules are partitioned recursively.



## APPENDIX 1: DATA DEFINITION USING SQL

```
#####  
#                                                                 #  
#   File Name: create_table.sql                                  #  
#   Author:    Bin Gao                                          #  
#   Date:     March 1, 2003                                     #  
#                                                                 #  
#   The definition of database is used to association rule mining #  
#   and cycle mining. The SQL statements are initially written for #  
#   MySQL DBMS, but they can be immigrated to other SQL databases #  
#   through deleting words "TYPE=InnoDB". The reason to use the #  
#   words is because MySQL only applies "references" to the InnoDB #  
#   table type at the time of this writing. InnoDB table is #  
#   transactional safe with row locking.                         #  
#                                                                 #  
#####  
  
# create a new empty database, and make it active  
create database dataMining;  
use dataMining;  
  
# create the item table that record the information of items  
drop table item;  
create table item(  
    itemID      int not null auto_increment,  
    itemDesc    varchar(100) not null,  
    primary key (itemID),  
    index (itemID)  
)TYPE=InnoDB;  
  
# create transaction table that is used to contain all transactions  
drop table transaction;  
create table transaction(  
    transactionID int not null auto_increment,  
    itemID        int not null references item (itemID),  
    primary key (transactionID, itemID),  
    index (transactionID, itemID)  
)TYPE=InnoDB;
```

```

# create freqItemset table where association rule mining
# can store the frequent itemsets
drop table freqItemset;
create table freqItemset(
    itemsetID    int not null auto_increment,
    itemID       int not null references item (itemID),
    support      decimal(5,2) not null,
    primary key  (itemsetID, itemID),
    index (itemsetID, itemID)
)TYPE=InnoDB;

# create assocRule table in which association rule mining
# will store the association rules discovered from frequent itemsets
drop table assocRule;
create table assocRule(
    ruleID       int not null auto_increment,
    antecedent   int not null references freqItemset (itemsetID),
    consequent   int not null references freqItemset (itemsetID),
    confidence   decimal(5,2) not null,
    primary key  (ruleID),
    index (ruleID)
)TYPE=InnoDB;

# create cycle table, and cycle mining will output all cycles
# into the tables. This is a formal definition.
drop table cycle;
create table cycle_formal(
    cycleID      int not null,
    ruleID       int not null references assocRule (ruleID),
    orderNum     int not null default 0,
    primary key  (cycleID, ruleID),
    index (cycleID, ruleID)
)TYPE=InnoDB;

# create cycle table, and cycle mining will output all cycles
# into the tables. This is a lazy definition without references.
drop table cycle;
create table cycle_lazy(
    cycleID      int not null,
    antecedent   int not null references freqItemset (itemsetID),
    consequent   int not null references freqItemset (itemsetID),
    orderNum     int not null default 0,
    primary key  (cycleID, antecedent, consequent)
)TYPE=InnoDB;

```

## APPENDIX 2: LIST OF FILES IN SOFTWARE PACKAGE

In Chapter 5, we introduce the software package that implements cycle mining of association rules. The software package has been written using Java programming language. The package contains two folders and three files. In fact, you can run the software if you get the three files and start MySQL database server.

**DOC:** is a folder that contains the documentations of all APIs as HTML files.

**SRC:** is a folder that contains all source code of the software. There are thirteen individual files as below:

- **Create\_tab.sql:** defines the database and tables used to association rule mining and cycle mining.
- **Cycle.java:** defines the methods to generate all cycles starting at the specified vertex.
- **CycleMining.java:** provides GUI to perform cycle mining algorithm and connect MySQL database management system.
- **DBConnectionUI.java:** provides GUI to receive the information related to MySQL database management system.
- **DigraphEdgeEnd.java:** defines an edge for a digraph.
- **DigraphNode.java:** defines a node or vertex for a digraph.

- **HashTable.java**: defines a hash table with chaining. This can be used to represent a digraph. The class is little different than class **Hashtable** in JSDK
- **IntroductionUI.java**: provides GUI to introduce the software to users.
- **MiningConfirmationUI.java**: provides GUI to prompt users to start cycle mining.
- **MysqlJDBCAdapter.java**: provides a JDBC adapter so that MySQL database can be connected and accessed.
- **Populate\_tab.sql**: includes three sets of sample test data.
- **ResultDemonstration.java**: provides the interface to show the results of cycle mining.
- **TableSelectionUI.java**: provides GUI to prompt users to select valid tables for input and output.

**CycleMining.bat**: a batch file that can start the cycle mining package. Under Windows, the users just need to click the icon in order to run the software.

**CycleMining.jar**: a zipped file that archives all class files of the software. Also the file includes the source code and documentation inside.

**MysqlJDBC.jar**: the file providing the official JDBC driver for MySQL. The file is licensed under the GPL. For more details about the file, visit <http://www.mysql.com/downloads/api-jdbc-stable.html>.

## BIBLIOGRAPHY

- 1 Agrawal, R. and Srikant, R. 1994. Fast Algorithms for Mining Association Rules in Large Databases. Proceedings of the International Very Large Databases Conference, 487-499.
- 2 Agrawal, R., Imielinshi, T. and Swami, A. 1993. Mining Association Rules between Sets of Items in Large Databases. Proceedings of the ACM International Conference on Management of Data, 207-216.
- 3 Aho, A.V. and Ullman, J.D. 1995. The Foundations of Computer Science (C edition). New York, NY: Computer Science Press, 451-528.
- 4 Atkinson, L. 2002. Core MySQL. Upper Saddle River, NJ: Prentice-Hall, INC, 118-431.
- 5 Baase, S. and Gelder, A.V. 2000. Computer Algorithms: Introduction to Design and Analysis. Reading, MA: Addison-Wesley, 1-688.
- 6 Buckley, J.P. and Seizer, J. 1999. A Paradigm for Detecting Cycles in Large Database Sets via Fuzzy Mining. IEEE Workshop on Knowledge and Data Engineering Exchange, 68-75.
- 7 Chen, L. 2001. Active Cycle Mining System. Project for Master Degree, Computer Science Department, University of Dayton, 1-32.
- 8 Cherkassky, B.V. and Goldberg, A.V. 1996. Negative-Cycle Detection Algorithms. European Symposium on Algorithms, 349-363.
- 9 Cormen, T.H., Leiserson, C.E., Rivest, R.L. and Stein, C. 2001. Introduction to Algorithms (2<sup>nd</sup> ed). Cambridge, MA: MIT Press, 197-318, 1057-1126.
- 10 Deitel, H.M. and Deitel, P.J. 1999. Java: How to Program (3<sup>rd</sup> ed). Upper Saddle River, NJ: Prentice-Hall, INC, 1-1344.
- 11 Deo, N., Prabhu, G. and Krishnamoorthy, M.S. 1982. Algorithms for Generating Fundamental Cycles in a Graph. ACM Transactions on Mathematical Software, 8(1): 26-42.
- 12 Deogun, J., Raghavan, V., Sarhar, A. and Sever, H. 1997. Data Mining: Trends in Research and Development. In T.Y. Lin and N. Cercone, editors. Rough Sets and Data Mining: Analysis for Imprecise Data. Boston, MA: Kluwer Academic Publishers, 9-45.

- 13 Dogrusöz, U. and Krishnamoorthy, M. S. 1995. Cycle vector space algorithms for enumerating all cycles of a planar graph. Technical Report 95-5, Rensselaer Polytechnic Institute, Department of Computer Science.
- 14 Dunham, M.H. 2002. Data Mining: Introductory and Advanced Topics. Upper Saddle River, NJ: Prentice-Hall, INC, 1-314.
- 15 Džeroski, S. 2001. Data Mining in Nutshell. In S. Džeroski and N. Lavrač, editors. Relational Data Mining. Berlin, German: Springer-Verlag, 1-27.
- 16 Elmasri, R. and Navathe, S.B. 2000. Fundamentals of Database System (3<sup>rd</sup> ed). Reading, MA: Addison Wesley Longman, INC, 841-880.
- 17 Fayyad, U. 2001. Knowledge Discovery in Databases: An Overview. In S. Džeroski and N. Lavrač, editors. Relational Data Mining. Berlin, German: Springer-Verlag, 28-47.
- 18 Fayyad, U., Piatetsky-shapiro, G. and Smyth, P. 1996. From Data Mining to Knowledge Discovery in Databases: An Overview. In U. Fayyad, G. Piatetsky-shapiro, P. Smyth and R. Uthurusamy, editors. Advances in Knowledge Discovery and Data Mining. Cambridge, MA: MIT Press, 1-34.
- 19 Frawley, W., Piatetsky-shapiro, G. and Matheus, R. 1991. Knowledge Discovery in Databases: An Overview. In G. Piatetsky-shapiro and W. Frawley, editors. Knowledge Discovery in Databases. Cambridge, MA: MIT Press, 1-27.
- 20 Garofalakis, M.N., Rastogi, R., Seshadri, S. and Shim, K. 1999. Data Mining and the Web: Past, Present and Future. Proceedings of the Second International workshop on Web Information and Data Management, 43-47.
- 21 Gibbs, N. E. 1975. Algorithm 491: Basic Cycle Generation [H]. Communications of the ACM, 18(5): 275-276.
- 22 Goodrich, M. T. and Tamassia, R. 1998. Data Structures and Algorithms in Java. New York, NY: John Wiley & Sons, INC, 1-738.
- 23 Gotlieb, C.C and Corneil, D.G. 1967. Algorithms for Finding a Fundamental Set of Cycles for an Undirected Linear Graph. Communications of the ACM, 10(12): 780-783.
- 24 Gunopulos, D., Hardon, R., Mannila, H. and Toivonen, H. 1997. Data mining, Hypergraph Transversals, and Machine Learning (Extended Abstract). Symposium on Principles of Database Systems, 209-216.

- 25 Hand, D., Mannila, H. and Smyth, P. 2001. Principles of Data Mining. Cambridge, MA: MIT Press, 427-448.
- 26 Hipp, J., Güntzer, U. and Nakhaeizadeh, G. 2000. Algorithms for Association Rule Mining: A General Survey and Comparison. SIGKDD, 2(1): 58-64.
- 27 Johnson, D.B. 1975. Finding All the Elementary Circuits of a Directed Graph. SIAM Journal on Computing, 4(1): 77-84.
- 28 Mateti, P. and Deo, N. 1976. On Algorithms for Enumerating All Circuits of a Graph. SIAM Journal on Computing, 5(1): 90-99.
- 29 Özden, B., Ramaswamy, S. and Silbershatz, A. 1998. Cyclic Association Rules. Proceedings of the IEEE International Conference on Data Engineering, 412-421.
- 30 Paton, K. 1969. An algorithm for Finding a Fundamental Set of Cycles of a Graph. Communications of the ACM, 12(9): 514-518
- 31 Ramaswamy, S., Mahajan, S. and Silbershatz, A. 1998. On the Discovery of Interesting Patterns in Association Rules. Proceedings of the 24<sup>th</sup> Very Large Databases Conference, 368-379.
- 32 Read, R.C. and Tarjan, R.E. 1975. Bounds on Backtrack Algorithms for Listing Cycles, Paths, and Spanning Trees. Networks, 5: 237-252.
- 33 Reese, G., Yarger, R.J. and King, T. 2002. Managing and Using MySQL (2<sup>nd</sup> ed). Sebastopol, CA: O'Reilly & Associates, INC, 1-425.
- 34 Reingold, E.M., Nievertgelt, J. and Deo, N. 1977. Combinatorial Algorithms: Theory and Practice. Englewood, NJ: Prentice-Hall, INC, 318-400.
- 35 Robinson, D.F. and Foulds, L.R. 1980. Digraphs: Theory and Techniques. New York, NY: Gordon and Breach Science Publishers, INC, 13-69, 201-225.
- 36 Seitzer, J., Buckley, J.P. and Monge, A. 1999. Meta-Pattern Extraction: Mining Cycle. Proceedings of the Florida Artificial Intelligence Research Society International Conference, 466-470.
- 37 Szwarcfiter, J. L. and Lauer, P. E. 1976. A Search Strategy for the Elementary Cycles of a Directed Graph. BIT 16: 192-204.
- 38 Tarjan, R.E. 1972. Depth-first Search and Linear Graph Algorithms. SIAM Journal of Computing, 1(2): 146-160.

- 39 Tiernan, J. C. 1970. An Efficient Search Algorithm to Find the Elementary Circuits of a Graph. *Communications of the ACM*, 13(12): 722-726.
- 40 Weinblatt, H. 1972. A New Search Algorithm for Finding the Simple Cycles of a Finite Directed Graph. *Journal of the ACM*, 19(1): 43-56.
- 41 Weiss, M.A. 1998. *Data Structures and Problem Solving Using Java*. Reading, MA: Addison Wesley Longman, INC, 367-408.
- 42 Weiss, S.M. and Indurkha, N. 1998. *Predictive Data Mining: a Practical Guide*. San Francisco, CA: Morgan Kaufmann Publishers, INC, 1-228.
- 43 Zaki, M.J. and Hsiao, C. 2001. CHARM: An Efficient Algorithm for Closed Itemset Mining. *Proceedings of the Second SIAM International Conference on Data Mining*, 457-473.