P-1642

IMPLEMENTATION OF SPF AND INVERTED MATRIX ALGORITHMS FOR MINING FREQUENT DATA PATTERNS

A PROJECT REPORT

Submitted by

SANDHYA.B 71202205039

M.SARAN PRIYA 71202205040

In partial fulfillment for the award of the degree
Of
BACHELOR OF TECHNOLOGY
in
INFORMATION TECHNOLOGY

KUMARAGURU COLLEGE OF TECHNOLOGY, COIMBATORE

ANNA UNIVERSITY: CHENNAI 600025

APRIL 2006

ANNA UNIVERSITY: CHENNAI 600025

BONAFIDE CERTIFICATE

Certified that this project report **"IMPLEMENTATION OF SPF AND INVERTED MATRIX ALGORITHMS FOR MINING FREQUENT DATA PATTERNS"** is the bonafide work of **"SANDHYA.B, M.SARAN PRIYA"** who carried out the project work under my supervision.

SIGNATURE	**SIGNATURE**
Dr.G.Gopalsamy	**Ms.M.Lavanya**
HEAD OF THE DEPARTMENT	**SUPERVISOR**
Department of	Lecturer
Information Technology	Information Technology
Kumaraguru College of Technology	Kumaraguru College of Technology
Coimbatore-641006	Coimbatore-641006

The candidates with University Register Nos. 71202205039, 71202205040 were examined by us in the project viva-voce examination held on 26/04/2006

INTERNAL EXAMINER **EXTERNAL EXAMINER**

ACKNOWLEDGEMENT

We extend our sincere thanks to our Principal **Dr.K.K.Padmanabhan, Ph.D.,** Kumaraguru College of Technology, Coimbatore, for his incredible support for all our toil regarding the project.

We are deeply obliged to **Dr.G.Gopalsamy, PhD**, Head of the Department of Information Technology for his concern and implication during the project course.

No words to express our gratitude to **Mr.K.R.Baskaran, B.E. M.S,** Asst. Professor, Department of Information Technology, who has helped us with the most minute details.

We articulate our thankfulness to our guide **Ms.M.Lavanya, M.Tech,** Lecturer, Department of Information Technology, for lending a hand throughout the project path.

We thank **Mrs.N.Suganthi, M.E.,** Senior Lecturer and **Mr.P.C.Thirumal, M.S,** Lecturer, Department of Information Technology, for their moral support in completing the project.

Our thanks are also to **Ms.S.Vani, M.E,** Lecturer, Department of Information Technology, for helping us overcome the perplexity while choosing the project.

We thank all the **Teaching and Non-teaching staffs** of our department for providing us the technical support for our project.

We also thank our **friends and family** who helped us to complete this project fruitfully.

DECLARATION

We,

Sandhya.B 71202205039

M.Saran Priya 71202205040

Declare that the project entitled **"IMPLEMENTATION OF SPF AND INVERTED MATRIX ALGORITHMS FOR MINING FREQUENT DATA PATTERNS"**, submitted in partial fulfillment to Anna University as the project work of Bachelor Of Technology (Information Technology) Degree, is a record of original work done by us under the supervision and guidance of Ms.M.Lavanya, Lecturer, Department of Information Technology, Kumaraguru College of Technology, Coimbatore.

Place: Coimbatore
Date: 25/04/2006

[Sandhya.B]

[M.Saran Priya]

Project Guided by

25/04/06

[Ms.M.Lavanya]
Lecturer.

ABSTRACT

Irrespective of the field of study, data mining is playing a very imperative role in the analysis of the patterns in which the data occur. The generation of frequent items from these patterns helps us to make critical assessments, be it for sales, marketing or any other stream.

In this project, two data mining algorithms namely, **Segmented Progressive Filter and Inverted Matrix** have been implemented. These algorithms are anti-Apriori algorithms which are mainly developed to overcome the repetitive I/O disk scans and huge computation and communication required by the Apriori-based algorithms during the candidacy generation.

Segmented Progressive Filter algorithm involves two steps that is;

- Segmentation
- Progressive filtering

In segmentation, the database is partitioned into sub-databases based on time. In progressive filtering, frequent candidate 2-itemsets are filtered from the partitions with their number of occurrences as the key parameter.

In Inverted Matrix algorithm, a new database layout called Inverted Matrix is created. This structure depicts the relationship between the transactions and the items in them with the help of pointers.

Also, a small independent tree called COFI tree may be built by summarizing the co-occurrences and generating frequent patterns by applying a simple non-recursive mining process.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

S.No	ABBREVIATION	EXPANSION
1.	SPF	Segmented and Progressive Filtering
2.	MCP	Maximum Common Exhibition Period
3.	COFI	Co-occurrence Of Frequent Items

1. INTRODUCTION

1.1 EXISTING SYSTEM AND ITS LIMITATIONS

Data mining is the knowledge discovery in databases, and is the non-trial extraction of implicit unknown and potentially useful information from the data. Data mining is required for us to craft a number of proactive and knowledge-driven assessments depending upon the pattern of occurrences of the items in the database, irrespective of its category. One of the most intricate tasks in data mining is the discovery of association rules.

A number of algorithms are on hand for this purpose. Most of the existing algorithms are Apriori based. These algorithms used downward closure property for mining the frequent item sets. These algorithms are proven to be non-scalable for the reason that

- It requires repetitive I/O disk scans
- Massive memory size and
- Huge computation and communication in the candidacy generation.
-

These pitfalls has motivated to implement two new algorithms namely, Segmented Progressive Filtering and Inverted matrix, both of which are anti-Apriori and help us to overcome the drawbacks faced by the previous techniques.

1.2 PROPOSED SYSTEM AND ITS ADVANTAGES

The proposed system which includes the implementation of two algorithms that is, Segmented Progressive Filtering and Inverted matrix, is developed for mining the frequent patterns from the given collection of enormous, heterogeneous data.

Inverted matrix algorithm is a parallel association rule mining method that makes use of minimum inter-processor communication while mining massive transactional data sets.

It involves the formation of a new database layout called Inverted Matrix. Optionally a relatively small independent tree may be constructed, summarizing co-occurrences and a simple, non-recursive mining process facilitated for the candidacy generation. This algorithm has an upper hand over other Apriori based algorithms in terms of

- Non-repetitive I/O scans
- Less memory size requisite
- Minimal inter-node communication cost.

Segmented Progressive Filtering, the abbreviation of SPF, is used to mine items that are frequent only for a particular span of time. For example, water melons may be frequent only for the months between March and July. While mining the entire database, this item becomes infrequent. To evade such omissions, the SPF algorithm is implemented. It involves two phases namely, *Segmentation* and *Progressive filtering*. The database is divided into sub-databases referred to as partitions based on time and the filtering process is carried out to every partition to generate the frequent patterns. The SPF algorithm outperforms other algorithms in terms of

- Execution time
- Scalability

2. SYSTEM REQUIREMENTS AND ANALYSIS

System study is an activity that encompasses most of the tasks that we have collectively called computer system engineering. The study is conducted with the following objectives:

- Identify the needs
- Evaluate the system concept for feasibility
- Perform economic and technical analysis
- Allocate functions to hardware, software, people, and other system elements
- Create a system definition that forms the foundation for all subsequent engineering works

2.1 PRODUCT DEFINTION

the main purpose of this project is to implement algorithms which would mine frequent patterns from the large databases with maximum ease and making least use of the available resources like memory capacity, CPU, response time etc. irrespective of the application.

2.2 PROJECT PLAN

In the analysis phase, the available technologies have been learnt and the most suitable technology has been identified for the project implementation. From this study, the most suitable language preferred is Java. Differentiation of modules, the user interface design and other crucial aspects were identified and designed as necessary. Other ambiguities which may exist are identified and taken care of.

After the analysis phase, the design phase commences in which the various modules and its functionalities are identified. The complete system flow of control and data are depicted in the form of diagrams.

Next the implementation phase converts the design into executable code. Each module is coded separately and finally integrated to form the entire system. Care is taken to make the code easily understandable by future users by adding apt comments and using appropriate variable names.

In the testing phase, each module is tested thoroughly and finally the integrated modules are tested together to ensure the correct working of the entire system. Testing is also done to ensure that the product satisfies the specified requirements and set criteria.

Table 2.1 PROJECT PLAN

WORK	DURATION
• Feasibility Analysis • Abstract Preparation • Requirements Gathering	One week
• Collecting required software • Installing the software and learning Them	Two weeks
• Coding-implementing the segmentation algorithm	One week
• Coding-implementing the filtering algorithm	One week
• Analysis of the Inverse matrix algorithm	One week
• Implementing the algorithm	Two week
• Testing of the system	One week

2.3 SOFTWARE REQUIREMENTS SPECIFICATION

2.3.1 INTRODUCTION

2.3.1.1 PURPOSE

The purpose of this project is to implement the two anti-Apriori algorithms, Segmented Progressive Filtering and Inverted matrix. It describes the techniques of mining the database easily with maximum efficiency and minimal use of resources. The document bridges the gap between the customer and the analyst.

2.3.1.2 SCOPE

SRS forms the essentials for agreement between the clients and the supplier and what the product will do. It also provides a reference for the validation of the final project. Any changes made to the SRS in future will have to go through formal change approval process.

2.3.2 DEFINITION

2.3.2.1 CUSTOMER

A customer is the person or organization, internal or external to the producing organization, who takes financial responsibility for the system. In a large system this may not be the end user. The customer is the ultimate recipient of the developed product and its artifacts.

2.3.2.2 USER

The user is the person who will use the system that is developed. All processes and functionalities are considered for his ease of use and other benefits.

2.3.2.3 ANALYST

The analyst details the specification of the system's functionalities by describing the requirements aspect and other supporting software requirements.

2.3.3 GENERAL DESCRIPTION

2.3.3.1 PRODUCT OVERVIEW

This system aims to provide implementation to two algorithms namely, Segmented Progressive Filtering and Inverted matrix in which the database is taken into consideration and it is mined to generate the frequent item sets with the help of non-repetitive I/O scans, less memory size requisite and minimal inter-node communication cost. The database maybe a one holding transaction data of any nature like grocery stores, medical shops, etc.

2.3.3.2 USER CHARACTERISTICS

The system that is developed is very user-friendly and does not require much of any knowledge from the user's perspective. However, the knowledge of the type of mining to be done would be an added advantage if the user needs the best result.

2.3.3.3 GENERAL CONSTRAINTS

The system has a single GUI, which provides three options for the user to select the mining algorithm. The system consists of three major modules-

- SPF
- Inverted matrix
- COFI-tree

Each of these modules are divided into sub-modules as-

SPF-

- Segmentation
- Progressive filtering

COFI-tree-

- Tree generation
- Tree mining

2.3.3.4 GENERAL ASSUMPTIONS

The database must be converted to the form mineable by the algorithm. Also the user must be aware that the mining he/she requires is based on the database transactions with respect to time or the entire database.

2.3.4. SPECIFIC REQUIREMENTS

2.3.4.1 INPUTS AND OUTPUTS

There are only two inputs to the system. One is the button that the user clicks for the system to carry out the respective algorithm and the second input is the database itself that the algorithm makes use for the purpose of mining. All the mining results are only with respect to the input database. The output is the frequent itemsets generated by the algorithm. This output is also displayed in the textbox present in the GUI itself.

2.3.4.2 FUNCTIONAL REQUIREMENTS

- The system should be able to construct all the sub-structures according to the values of the database. That is, it must be a dynamic process, rather than a static process.
- The system should be capable of producing the results to the users without any problem.

2.3.4.3 SYSTEM REQUIREMENTS

2.3.4.3.1 HARDWARE REQUIREMENTS

Processor – Pentium IV

RAM Size – 128 MB RAM

Hard Disk Capacity – 20 GB

2.3.4.3.2 SOFTWARE REQUIREMENTS

Operating System – Windows

Language – Java

Programming suite (front end) – Net Beans IDE 3.6

Database (back end) – MS Access 7.0

2.3.5 PERFORMANCE CONSTRAINT

The system must execute correctly as long as it gets the correct input from the user and the database formats are intact as defined for the system.

3. SYSTEM STUDY

3.1 DATA MINING

Data mining is the process of discovery meaningful, new correlation patterns and trends by shifting through large amount of data stored in repositories, using pattern recognition techniques as well as mathematical and statistical techniques.

Data mining, the extraction of hidden predictive information from large databases, is a powerful new technology with great potential to help companies focus on the most important information in their data warehouses. Data mining tools predict future trends and behaviors, allowing businessmen to make proactive, knowledge-driven decisions. Data mining can be done with a variety of algorithms.

In this project, two algorithms SPF and Inverted Matrix are implemented which are used for mining frequent data patterns from the large database. These are anti-priori algorithms. Apriori-based algorithms used the downward closure property for mining the frequent patterns. But this is no longer valid.

3.2 SEGMENTED AND PROGRESSIVE FILTER

In SPF, a new model of mining general temporal association rules from large databases IS EXPLORED, where the exhibition periods of the items are allowed to be different from one to another. The basic idea behind this algorithm is to first segment the database into sub-databases in such a way that items in each sub-database will have either the common starting time or the common ending time.

Then, for each sub-database, SPF progressively filters candidate 2-itemsets with cumulative filtering thresholds either forward or backward in time. This feature allows SPF of adopting the scan reduction technique by generating all candidate k-item sets ($k > 2$) from candidate 2-itemsets directly.

The experimental results show that algorithm SPF significantly outperforms other schemes which are extended from prior methods in terms of the execution time and scalability.

3.3 INVERTED MATRIX

Inverted Matrix, a new disk-based parallel association rule mining algorithm achieves its efficiency by applying three new ideas-

- First, transactional data is converted into a new database layout called Inverted Matrix that prevents multiple scanning of the database during the mining phase, in which finding globally frequent patterns could be achieved in less than a full scan with random access. This data structure is replicated among the parallel nodes.
- Second, for each frequent item assigned to a parallel node, a relatively small independent tree is built summarizing co-occurrences.
- Finally, a simple and non-recursive mining process reduces the memory requirements as minimum candidacy generation and counting is needed, and no communication between nodes is required to generate all globally frequent patterns.

3.4 JAVA

3.4.1 INTRODUCTION

Java is an object-oriented programming language with a built-in application programming interface (API) that can handle graphics and user interfaces and that can be used to create applications and applets. Because of its rich set of API's, similar to Macintosh and Windows, and its platform independence, Java can also be thought of as a platform in itself. Java also has standard libraries for doing mathematics.

Java is a general purpose programming language with a number of feature that make the language well suited for use on the World Wide Web. Small Java applications are called Java applets and can be downloaded from a Web server and run on your computer by a Java- compatible Web browser, such as Netscape Navigator or Microsoft Internet Explorer.

The inventors are Java wanted to design a language, which could offer solution to some of the problems encountered in modern programming. They wanted the language to be not only reliable, portable and distributed.

Although the above appears to be a list of buzzwords, they aptly describe the full potential of the language. These features have made Java the first application language of the World Wide Web. Java will also become the primer language for general-purpose stand-alone applications.

3.4.2 PROPERTIES OF JAVA

- Compiled and Interpreted
- Platform-Independent and Portable
- Robust and Secure
- Multithreading and Interactive
- Dynamic and Extensible

3.4.2.1 COMPILED AND INTERPRETED

Usually a computer language is either compiled or interpreted. Java combines both these approaches thus making Java a two-stage system. First, Java compiler translates source code into what is known as byte code instructions.

3.4.2.2PLATFORM-INDEPENDENT AND PORTABLE

The most significant contribution of Java over other languages is its portability. Java programs can be easily moved from one computer system to another, anywhere anytime. Changes and upgrades in operating systems, processors and system resources will not force any changes in Java programs. This is the reason why Java has become a popular language for programming on Internet, which inter connects different kinds of system worldwide. We can download a Java applet from a remote computer on to our local system via Internet an extension of the user's basic system providing practically unlimited number of accessible applets and applications.

3.4.2.3 ROBUST AND SECURE

Security becomes an important issue for a language that is used for programming on Internet. Threat of viruses and abuse of resources is everything. Java systems not only verify all memory access but also ensure that no viruses are communicated with the applet. The absence of pointers in Java ensures that programs cannot gain access to memory locations without proper authorization.

3.4.2.4 DISTRIBUTED

Java is designed as a distributed language for creating applications on networks. It has the ability to share both the data programs. Java applications can open and access remote objects on Internet as easily as they can in the local system. This enables multiple programmers at multiple locations to collaborate and work together on a single object.

3.4.2.5 MULTITHREADED

Multithreaded means handling multiple tasks simultaneously. Java supports multithreading programs. This means that we need not wait for the application to finish one task before beginning the other. This feature greatly improves the interactive performance of graphical applications.

3.4.2.6 DYNAMIC AND EXTENSIBLE

Java is a dynamic language. Java is capable of dynamically linking in new class libraries methods and objects. Java can also determine the type of class through a query, making it possible to either dynamically link or abort the program, depending on the response.

3.4.3 ADVANTAGES OF JAVA

Java has gained enormous popularity since it first appeared. Its rapid ascension and wide acceptance can be traced to its design and programming features, particularly in its promise that we can write a program once, and run it anywhere. Java was chosen as the programming language for network computers (NC) and has been perceived as a universal front end for the enterprise database. As started in Java language white paper by Sun Microsystems: "Java is a simple, object-oriented, distributed, interpreted, robust, secure, architectural neutral, portable, multithreaded and dynamic".

3.5 NET BEANS

Java programmers need to be productive and this is the focus of the NetBeans Integrated Development Environment (IDE). The NetBeans IDE is a fast and feature full tool for developing Java software. It is standards compliant and runs on any operating system where a Java Virtual Machine is available. The NetBeans IDE is open-source and is developed right here on the netbeans.org website. It can be used free of charge for any type of software development. The NetBeans IDE runs on any platform. While the IDE is modular (its functionality can be extended by plug- ins), the focus of netbeans.org is on providing all the tools a Java developer needs in one download, with no additional set-up or configuration needed to begin productive work with the IDE.The NetBeans IDE 3.6 includes the following new features and functionality:

- A brand new windowing system, providing native look and feel on Windows platforms
- Enhanced navigation and workflow
- 2-tier J2EE 1.4 support (servlet 2.4 and JSP 2.0)

- Improved JSP debugging with JSR-45 Support
- Available co-bundled with Sun Java System Application Server, Platform Edition 8
- Code editor enhancements such as Smart Brackets and Code Folding
- Integrated testing support through JUnit
- Improved online help with Java Help 2.0 integration
- Improved CVS Support
- Task List window
- Basic Support for J2SE 1.5 (Tiger) Beta
- Available co-bundled with J2SE 1.4.2

4. DESIGN DOCUMENT

4.1 INPUT DESIGN

The system takes two inputs for its functioning. One is the database, and the other is the choice of algorithm selected by the user.

The database consists of the transactions and the items involved in the different transactions. In case of the SPF algorithm, the date is also considered in order to partition the database based on time.

To facilitate the selection of the required algorithm by the user, command buttons are provided at the GUI interface. Clicking on the required button commands the system to carry out that algorithm.

Both the algorithms SPF and Inverted Matrix make use of the same database.

4.2 PROCESS DESIGN

The steps for implementation of the two non-apriori algorithms are given along the following lines-

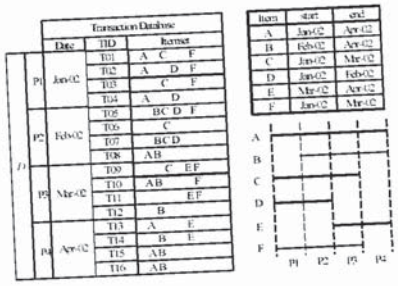
4.2.1 SEGMENTED PROGRESSIVE FILTERING

4.2.1.1 SEGMENTATION

In the segmentation algorithm, the database is divided into sub-databases known as partitions. The number of partitions is optimized in such a way that the number is neither too large nor too small because both cases result in ineffective mining results. The precise steps are described below-

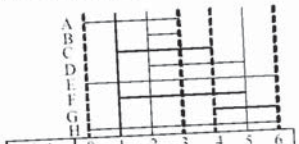
- Partitions of the database are made based on the MCP values. The MCP is nothing but the exhibition periods, which is the difference between the last occurrence and the first occurrence time of the particular item.
- Following is an illustration of the MCP values of each item.

Fig 4.1 AN ILLUSTRATIVE DATABASE WHERE THE ITEMS HAVE INDIVIDUAL EXHIBITION PERIODS.



- For each transition of the partition, the value of the direction is reversed. This would enable us to mine in two directions, so that the process is done fast and efficiently.
- The flag values at each partition is given in the following figure-

Fig 4.2 AN EXAMPLE ILLUSTRATING THE EXECUTION OF *PROCSG*.



index	0	1	2	3	4	5	6
flag[i][L]		T	T	T	T	F	
flag[i][R]		F	F	T	T	T	
direction	-1	L	L	-1	-1	R	

- The values of head, index and direction are stored in the segmented matrix and returned.
- The algorithm for segmentation is given as follows-

Procedure ProcSG(n)

1. $SM = \emptyset$; $direction = -1$; $head = 1$;
2. **for** ($index = 0$ **to** n)
3. $flag[i][L] = false$; $flag[i][R] = false$;
4. **for** (each item $i \in \mathcal{I}$) {
5. $(p, q) = MCP(i)$;
6. $flag[p-1][L] = true$;
7. $flag[q][R] = true$;
8. }
9. **for** ($i = 1$ **to** $n-1$) {
10. **if** ($direction == -1$) {
11. **if** ($flag[i][L] == true$ **and** $flag[i][R] == true$) {
12. $SM = SM \cup \{(head, index, direction)\}$;
13. $head = index + 1$;
14. } **elseif** ($flag[i][L] == true$ **and** $flag[i][R] == false$)
15. $direction = L$;
16. **elseif** ($flag[i][L] == false$ **and** $flag[i][R] == true$)
17. $direction = R$;
18. } **elseif** ($direction == L$) {
19. **if** ($flag[index][R] == true$) {
20. $SM = SM \cup \{(head, index, direction)\}$;
21. $head = index + 1$;
22. $direction = -1$;
23. }
24. } **elseif** ($direction == R$) {
25. **if** ($flag[i][L] == true$) {
26. $SM = SM \cup \{(head, index, direction)\}$;
27. $head = index + 1$;
28. $direction = -1$;
29. }
30. }
31. }
32. $SM = SM \cup \{(head, n, direction)\}$;
33. **return** SM;

4.2.1.2 PROGRESSIVE FILTERING

The steps for the progressive filtering phase are

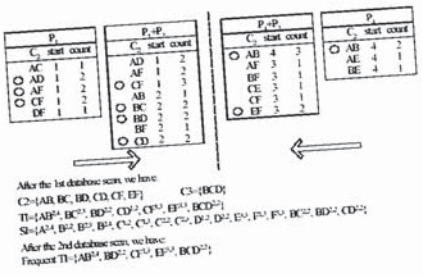
- The value of the direction is checked. If it is left, the starting partition is the head flag and the ending partition becomes the tail flag.
- The count of all the 2-itemsets in the first partition is calculated. The count becomes the support count of the itemsets.
- A min_support value of 2 is set for the filtering process, which may be changed according to user needs.
- All the 2-itemsets, whose support is more than 2, are moved to the next partition.
- The procedure is repeated for all the partitions to get the required frequent itemsets.
- The algorithm for filtering is given as follows-

Procedure ProcPF($p, q, direction$)

1. $PS = \emptyset$;
2. **if** ($direction == left$)
3. $head = p$; $tail = q$;
4. **else**
5. $head = q$; $tail = p$;
6. **for** ($h = head$ **to** $tail$)
7. **for** (each 2-itemset X_2 in P_h)
8. **if** ($X_2 \notin PS$) {
9. $X_2.count = N_{P_h}(X_2)$;
10. $X_2.start = h$;
11. **if** ($X_2.count \geq min\_supp * |P_h|$)

12. $PS = PS \cup X_2$;
13. } **else** {
14. $X_2.count = X_2.count - N_{P_h}(X_2)$;
15. **if** ($X_2.count < \lfloor min\_supp * \sum_{m=X_2.start}^{h} |P_m| \rfloor$)
16. $PS = PS - X_2$;
17. }
18. **return** PS;

Fig 4.3 PROGRESSIVELY FILTERING 2-ITEMSETS IN EACH PARTITION



4.2.2 INVERTED MATRIX ALGORITHM

In this algorithm, a new data structure known as Inverted Matrix is created by making use of a layout known as the Inverted Matrix layout. This layout is a combination of two layouts, namely vertical layout and horizontal layout.

4.2.2.1 HORIZONTAL AND VERTICAL LAYOUTS

Consider the following transaction database-

Fig 4.4 TRANSACTION DATABASE

T#	Items				
T1	A	G	D	C	B
T2	B	C	H	E	D
T3	B	D	E	A	M
T4	C	E	F	A	N
T5	A	B	N	O	P
T6	A	C	Q	R	G
T7	A	C	H	I	G
T8	L	E	F	K	B
T9	A	F	M	N	O
T10	C	F	P	J	R
T11	A	D	B	H	I
T12	D	E	B	K	L
T13	M	D	C	G	O
T14	C	F	P	Q	J
T15	B	D	E	F	I
T16	J	E	B	A	D
T17	A	K	E	F	C
T18	C	D	L	B	A

The horizontal layout relates all the items on the same transaction together. In this approach, the ID plays the role of the key for the transaction table. The downside of this method is that it duffers some superfluous processing due to the absence of index on the items.

The vertical approach, on the other hand, combines all the transactions in which a particular item is present. The benefit of this approach is that it reduces the effect of large data sizes and also it does not require re-scanning of database. However, it has its own pitfalls, like tedious frequent itemsets generation and the presence of the candidacy generation phase, to mention a few.

4.2.2.2 INVERTED MATRIX LAYOUT

The inverted matrix combines both the methods to make the best use of their individual advantages. The idea of this method is to associate each transaction with its respective items, and to associate all the items with all the transactions it occurs. The pointer in this method consists of two parameters, one is the address of a line and the second is the address of a column.

Given below is the inverted matrix structure for the transaction database considered previously-

Fig 4.5 INVERTED MATRIX

loc	Index	Transactional Array										
		1	2	3	4	5	6	7	8	9	10	11
1	R,2	2,1	3,2									
2	Q,2	12,2	3,3									
3	P,3	4,1	9,1	9,2								
4	O,3	5,2	5,3	5,3								
5	N,3	13,1	17,4	6,2								
6	M,3	14,2	13,3	12,4								
7	L,3	5,1	8,2	15,9								
8	K,3	13,2	14,9	13,7								
9	J,3	13,4	13,5	14,7								
10	I,3	11,2	11,3	13,8								
11	H,3	14,1	12,3	15,4								
12	G,4	15,1	16,4	16,5	16,5							
13	F,7	14,3	14,4	18,7	16,8	16,8	14,6	14,6				
14	E,6	15,2	15,3	16,3	17,5	15,5	15,7	15,9	16,9			
15	D,9	16,1	16,2	17,3	17,6	17,7	16,7	17,8	17,9	16,10		
16	C,10	17,1	17,2	18,3	18,5	18,6	ø,ø	ø,ø	ø,ø	18,10	17,10	
17	B,10	18,1	ø,ø	18,2	18,4	ø,ø	18,8	ø,ø	ø,ø	18,9	18,11	
18	A,11	ø,ø	ø,ø	ø,ø	ø,ø	ø,ø	ø,ø	ø,ø	ø,ø	ø,ø	ø,ø	ø,ø

Each line in the matrix has an address and is prefixed by the items it represents with its frequency in the database. The lines are ordered in ascending order of the frequency of items they represent.

The inverted matrix is built in two phases-

- In the first phase, the database is scanned once to find the frequency of each item and orders them in ascending order.
- In the second phase, the database is scanned once again to sort each transaction into ascending order according to the frequency of each item and then fills the matrix appropriately.

4.2.2.3 MATRIX CONSTRUCTION PROCESS ILLUSTRATION

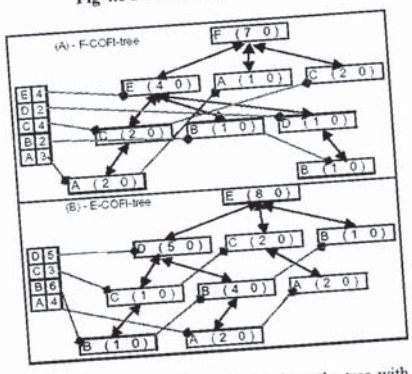
Consider the first transaction from the table in the Fig 4.4. That is, A G D C B. This transaction is sorted into (G, D, C, B, A) based on the item frequencies process. Item G has the physical location line 12 in the Inverted Matrix in Fig 4.5, D has the location line 15, the location of C is line 16, B is in line 17 and finally A is in line 18. This is according to the vertical approach. Item G has a link to the first empty slot in the transactional array of item D that is 1. Consequently, (15, 1) entry is added in the first slot of item G to point o the first empty location in the transactional array of D. At the First empty location of D (15, 1) an entry is added to point to the first empty location of the next item C that is (16, 1). The same process occurs for all items in the transaction. The last item of the transaction, item A produces an entry with pointer null (ø. ø). The same is performed for every transaction.

4.2.2.4 COFI TREE - DESIGN AND CONSTRUCTION

COFI (Co-occurrence Frequent Item) tree generation is another approach to parallel association rule mining. These trees have a header with ordered frequent items and horizontal pointers pointing to a succession of nodes containing the same frequent item, a counter that computes the occurrences of this item in the tree and the prefix tree per-se with paths representing sub-transactions. However, the COFI-trees have bidirectional links in the tree allowing bottom-up scanning as well, and the nodes contain not only the item label and a frequency counter, but also a participation counter. A COFI-tree for a given frequent item x contains only nodes labeled with items that are more frequent than or as frequent as x.

In our example, if we need to mine the Inverted Matrix in Fig 4.5 using a two-processor machine with $\sigma > 4$, the starting line for each parallel node would be location 13, as it is the location of the first frequent item. Processor 1 finds all frequent patterns related to items at locations 13, 15, and 17 which are items (F, D, and B). Processor 2 would generate all frequent patterns related to items at locations 14, and 16 which are for items E and C. The first Co-Occurrence Frequent Item-tree is built for item F. In this tree for F, all frequent items which are more frequent than F and share transactions with F participate in building the tree. The tree starts with the root node containing the item in question, F. For each sub-transaction containing item F with other frequent items that are more frequent than F, a branch is formed starting from the root node F. If multiple frequent items share the same prefix, they are merged into one branch and a counter for each node of the tree is adjusted accordingly. Fig 4.6 illustrates COFI-trees for frequent items F and E.

Fig 4.6 F AND E COFI TREES



In Fig 4.6, the rectangle nodes are nodes from the tree with an item label and two counters. The first counter is a support for that node while the second counter, called *participation-count*, is initialized to 0 and is used by the mining algorithm discussed later. The nodes have also pointers: a horizontal link which points to the next node that has the same *item-name* in the tree, and a bi-directional vertical link that links a child node with its parent and a parent with its child. The bi-directional pointers facilitate the mining process by making the traversal of the tree easier. The squares in the figures are actually cells from the header table as with the FP-Tree. This is a list made of all frequent items that participate in building the tree structure sorted in ascending order of their global support. Each entry in this list contains the *item-name*, *item counter*, and a *pointer* to the first node in the tree that has the same item-name.

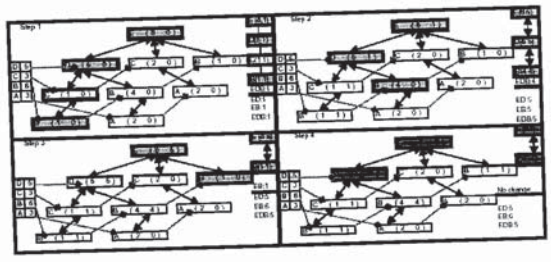
To explain the COFI-tree building process, we shall consider the building steps for the F-COFI-tree in Fig 4.6. Frequent item F is assigned to processor 1 that reads from the Inverted Matrix in Figure 4.5 all sub-transactions that starts with item F. The first sub-transaction has items FECA, and consequently 4 nodes are created, as FECA: 1 forms one branch with support = 1 for each node in the branch. The second sub-transaction has FEB. Nodes for F and E already exist and only new node for B is created as another child for E. The support for all these nodes is incremented by 1. B becomes 1, E and F become 2. FA is read then, and a new node for A is created with support = 1, and the F support is incremented by 1 to become 3. FC is read twice, and a new node for item C is created with support =2, and F support becomes 5. FEDB is read after that, FE branch already exists and a new child branch for DB is created as a child for E with support = 1. The support for E nodes becomes 3, F becomes 6. Finally the last sub-transaction FECA is read, its branch already exists, and only the counters for all nodes are incremented by 1. The participation count for each node is initialized to 0. The header in the F-COFI-tree, like with FP-Trees, constitutes a list of all frequent items to maintain the location of first entry for each item in the COFI-Tree. A link is also made for each node in the tree that points to the next location of the same item in the tree if it exists. The COFI-tree for a frequent item x contains only nodes labeled with items that are more frequent or as frequent as x, and share at least one transaction with x. Based on this definition, if A has support greater than B then the B COFItree is most likely larger than the COFI-tree for item A as more items would participate in building the B COFI-tree. In other words, the higher the support of a frequent item, the smaller its COFI-tree is. We use this observation to improve the load balance between processors.

After ordering the frequent items by their support, starting from the least frequent, each processor successively receives one item. The process is repeated until all items are distributed. In other words, if we have m processors, and n COFI-trees need to be built, assuming $m < n$ then processor 1 builds the COFI-tree for the least frequent item, processor 2 builds the COFI-tree for the next least frequent item, and so on up to processor m. After that processor 1 takes item $m+1$ and so on until all n items are distributed.

4.2.2.5 MINING THE COFI TREES

The mining process is done for each tree independently with the purpose of finding all frequent k-itemsets patterns that the item on the root of the tree participates in. Steps to produce frequent patterns related to the E item for example, are illustrated in Fig 4.7. From each branch of the tree, using the support count and the participation count, candidate frequent patterns are identified and stored temporarily in a list. The non-frequent ones are discarded at the end when all branches are processed. Figure 4.7shows the frequent itemsets containing E discovered assuming a support threshold greater than 4. Mining the "COFI-tree of item E" starts by identifying all non-frequent items with respect to the E item.

Fig 4.7 STEPS NEEDED TO GENERATE FREQUENT PATTERNS RELATED TO ITEM E



Items A, and C occur 3 times with item E, and consequently they cannot form a frequent pattern with item E. All nodes with labels A, and C will be discarded during the mining process. After eliminating non-frequent items in this tree the mining process starts from the most frequent item in the tree, which is item B. Item B exists in three branches in the E COFI-tree which are (B: 1, C: 1, D:5 and E:8), (B:4, D:5, and E:8) and (B:1, and E:8). The frequency of each branch is the frequency of the first item in the branch minus the participation value of the same node. Item B in the first branch has a frequency value of 1 and participation value of 0 which makes the first pattern EDB frequency equal to 1. The participation values for all nodes in this branch are incremented by 1, which is the frequency of this pattern. In the first pattern EDB: 1, we need to generate all sub-patterns that item E participates in which are ED:1 EB:1 and EDB:1. The second branch that has B generates the pattern EDB: 4, as the frequency of B on this branch is 4 and its participation value equals to 0.

All participation values on these nodes are incremented by 4. Sub-patterns are also generated from the EDB pattern which are ED: 4 , EB: 4, and EDB: 4. All patterns already exist with support value equals to 1, and only updating their support value is needed to make it equal to 5. The last branch EB:1 will generate only one pattern which is EB:1, and consequently its value will be updated to become 6. The second frequent item in this tree, "D" exists in one branch (D: 5 and E: 8) with participation value of 5 for the D node. Since the participation value for this node is equal to its support value, then no patterns can be generated from this node. Finally all non-frequent patterns are omitted leaving us with only frequent patterns that item E participates in which are ED:5, EB:6 and EBD:5. The COFI-tree of Item E can be removed at this time and another tree can be generated and tested to produce all the frequent patterns related to the root node. The same process is executed by all processors to generate all frequent patterns.

4.3 OUPUT DESIGN

After choosing the particular algorithm, the frequent items will be displayed in the text area in the GUI interface.

The output of the segmentation module of the SPF algorithm is the head, index and direction in which, the head denotes the pointed partition, index denotes the number of partitions and direction denotes the direction in which the mining will be done (Right to Left or Left to Right). The progressive filtering module mines the partitions and furnishes the frequent items as the output.

The output of the Inverted matrix algorithm is the Inverted matrix structure from which we may choose the frequent items.

The output of the COFI tree is frequent patterns too, that are obtained by mining the tree structure formed for each frequent item.

4.4 DATABASE DESIGN

The database in our system consists of the following fields-

Table 4.1 DATABASE DESIGN

Field name	Data type	Description
ID	AutoNumber	Gives the transaction number
DATE	Text	The date on which the transaction is made
It1	Text	Represents item 1
It2	Text	Represents item 2
It3	Text	Represents item 3
It4	Text	Represents item 4
It5	Text	Represents item 5
It6	Text	Represents item 6

5. PRODUCT TESTING

Testing can be separated into Verification & Validation and Functional & Structural Testing.

5.1 UNIT TESTING

This is the lowest level of testing a product. It involves individually testing each small unit of code to ensure that it works on its own, independent of the other units. It tells the developers that an application's pieces are working as designed. The developer establishes the test cases, test procedure, and test data for testing the software corresponding to each software unit.

Each module of our project were executed separately and checked for the correct working of the modules. The intended output for the algorithms were thus obtained after the unit testing.

5.2 VERIFICATION

Verification methods are used to ensure the system (software, hardware) compiles with an organization standard and processes relying on review. In our system the modules were tested for correct functionality.

5.3 VALIDATION TESTING

Validation ensures that a system operates according to the plans by executing real-life function. Validation is done for modules. This test is done to check the validity of the entire input.

5.4 FUNCTIONAL TESTING

Functional testing is a Black Box testing because of no knowledge. They verify all validation tests and inspect how the system performs. Each module is tested for expected output, without tracing the path of code. This testing methodology views each program as a mathematical function that maps its inputs to its outputs. It uncovers any errors that occur in implementing requirements or in design specifications. The outputs of the reports were verified for correctness and completeness.

5.5 STRUCTURAL TESTING

Structural Testing is called White Box Testing because the internal logic of the system is known. The purpose of this testing is to check out whether the logical operation of the system works well. Here the processing of the input is considered.

5.6 INTEGRATION TESTING

The purpose of this integration testing is to test the project as a whole after combining the individual modules. Here, the tested modules are combined into sub-systems, which are then tested. This is done to test ion the modules can be integrated properly, emphasizing on interface between modules.

The software was subjected to Integration testing and the different modules were linked together and executed.

The verification and validation of a software system is an abiding process through each phase of the software development process. Testing plays a vital role in determining the reliability and efficiency of the software and for this reason is a very imperative stage in software development.

6. FUTURE ENHANCEMENT

We have done mining for a sample database considering only a few items and handful transactions. Our future enhancement is to mine the frequent patterns from any database having numerous item sets and including a lot of transactions. Later data mining can also be performed from the any table and consequently making sure of designing a database that occupies less memory when considering large number of transactions.

7. CONCLUSION

Thus using Segmented Progressive Filtering and Inverted Matrix algorithms, we have mined the frequent patterns. Several algorithms are developed for mining the frequent items in the database. But, due to some drawbacks, we have implemented these algorithms.

Accordingly in Inverted Matrix algorithm, an inverted matrix layout is formed which summarizes the frequent patterns from the database. A COFI tree is built summarizing the co-occurrences and a mining process is done for each tree independently and thus the frequent patterns are identified. This algorithm as a result reduces the memory size and the I/O disk scans and therefore less communication and computation involved during the candidacy generation.

In SPF algorithm, we mine using general temporal association rule from large databases where the exhibition periods of the items are allowed to be different from one to another. Thus the SPF outperforms itself in terms of the execution time and scalability.

8. APPENDIX

8.1 SAMPLE CODE

```
import java.sql.*;
import java.util.*;
import java.lang.reflect.*;

public class Datamining extends javax.swing.JFrame
{
Statement stmt;
ResultSet rs;
ResultSet rs1;
int noofitems=0,nooftrans=0;
String index[][]=new String[1000][];
static int lastind=0;
/** Creates new form Datamining */
public Datamining()
{
initComponents();
}
private void initComponents()
{//GEN-BEGIN:initComponents
java.awt.GridBagConstraints gridBagConstraints;
jPanel1 = new javax.swing.JPanel();
jLabel1 = new javax.swing.JLabel();
jPanel2 = new javax.swing.JPanel();
jButton2 = new javax.swing.JButton();
jButton3 = new javax.swing.JButton();
jButton1 = new javax.swing.JButton();
jPanel3 = new javax.swing.JPanel();
jScrollPane1 = new javax.swing.JScrollPane();
jTextArea1 = new javax.swing.JTextArea();
jLabel2 = new javax.swing.JLabel();
addWindowListener(new java.awt.event.WindowAdapter()
{
public void windowClosing(java.awt.event.WindowEvent evt)
{
exitForm(evt);
}
});
```



```
jLabel1.setFont(new java.awt.Font("Tahoma", 1, 12));
jLabel1.setText("Mining Temporal Association Rule");
jPanel1.add(jLabel1);
getContentPane().add(jPanel1, java.awt.BorderLayout.NORTH);
jPanel2.setLayout(new java.awt.GridBagLayout());
jButton2.setText("SPF");
jButton2.addActionListener(new java.awt.event.ActionListener()
{
public void actionPerformed(java.awt.event.ActionEvent evt)
{
jButton2ActionPerformed(evt);
}
});
gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridwidth = java.awt.GridBagConstraints.REMAINDER;
gridBagConstraints.gridheight = 5;
gridBagConstraints.insets = new java.awt.Insets(31, 3, 31, 3);
jPanel2.add(jButton2, gridBagConstraints);
jButton3.setText("Inverse Matrix");
jButton3.addActionListener(new java.awt.event.ActionListener()
{
public void actionPerformed(java.awt.event.ActionEvent evt)
{
jButton3ActionPerformed(evt);
}
});
gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridwidth = java.awt.GridBagConstraints.REMAINDER;
gridBagConstraints.insets = new java.awt.Insets(4, 0, 4, 0);
jPanel2.add(jButton3, gridBagConstraints);
jButton1.setText("COFI tree");
jButton1.addActionListener(new java.awt.event.ActionListener()
{
public void actionPerformed(java.awt.event.ActionEvent evt)
{
jButton1ActionPerformed(evt);
}
});
gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridwidth = java.awt.GridBagConstraints.REMAINDER;
gridBagConstraints.gridheight = 5;
```

```
gridBagConstraints.insets = new java.awt.Insets(24, 0, 24, 0);
jPanel2.add(jButton1, gridBagConstraints);
getContentPane().add(jPanel2, java.awt.BorderLayout.WEST);
jPanel3.setLayout(null);
jScrollPane1.setViewportView(jTextArea1);
jPanel3.add(jScrollPane1);
jScrollPane1.setBounds(40, 30, 380, 260);
jLabel2.setFont(new java.awt.Font("Tahoma", 1, 12));
jLabel2.setText("Frequent Itemsets");
jPanel3.add(jLabel2);
jLabel2.setBounds(110, 10, 260, 15);
getContentPane().add(jPanel3, java.awt.BorderLayout.CENTER);
java.awt.Dimension screenSize =
java.awt.Toolkit.getDefaultToolkit().getScreenSize();
setBounds((screenSize.width-600)/2, (screenSize.height-400)/2, 600, 400);
}//GEN-END:initComponents
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt)
{//GEN-FIRST:event_jButton1ActionPerformed
Vector vec=new Vector();
int swapval=0;
try
{
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
String url="jdbc:odbc:spf";
String str=null;
int count=0;
Connection con =DriverManager.getConnection(url,"", "");
stmt = con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
ResultSet.CONCUR_UPDATABLE);
ResultSetMetaData rm;
rs = stmt.executeQuery("SELECT * FROM spf");
// Get the rowcount column value.int
rm=rs.getMetaData();
noofitems=rm.getColumnCount();
String tempstr=null;
rs.last();
int support[][]=new int[noofitems][2];
nooftrans=rs.getRow();
noofitems=noofitems-2;
//support count array generation and frequent set calculation with threshold value
4
```



```
for(int i=0;i<noofitems;i++)
{
for(int j=0;j<nooftrans;j++)
{
str="select * from spf where id="+(j+1);
rs=stmt.executeQuery(str);
rs.first();
if(rs.getString(i+3).equals("1"))
{
count++;
}
}
support[i][0]=i+1;
support[i][1]=count;
if(count>4)
{
vec.add(tempstr.valueOf(i+1));
}
count=0;
}//end of support count array generation and frequent set calculation
/***********support count 1 items set of each transaction********/
Vector tot=new Vector();
for(int i=0;i<nooftrans;i++)
{
Vector items=new Vector();
str="select * from spf where id="+(i+1);
rs=stmt.executeQuery(str);
rs.first();
for(int j=0;j<noofitems;j++)
{
if(rs.getString(j+3).equals("1"))
items.add(new String().valueOf(j+1));
}
int[] newitems=new int[items.size()];
for(int k=0;k<items.size();k++)
{
newitems[k]=Integer.parseInt(items.get(k).toString());
}
for(int j1=0;j1<newitems.length;j1++)
{
for(int j2=j1+1;j2<newitems.length;j2++)
{
```



```
if(support[(newitems[j1])-1][1]>support[(newitems[j2])-1][1])
{
swapval=newitems[j1];
newitems[j1]=newitems[j2];
newitems[j2]=swapval;
}
}
}
for(int s=0;s<newitems.length;s++)
{
System.out.print(" "+newitems[s]);
}
System.out.println("\n");
tot.add(newitems);
}//end of support count 1 itemset tble
// cofi tree generation
int[] treeitems=new int[vec.size()];
Vector cofitree=new Vector();
Vector subcofitree=new Vector();
Vector maintree=new Vector();
int sss=0;
for(int j=0;j<vec.size();j++)
{
treeitems[j]=Integer.parseInt(vec.get(j).toString());
System.out.print(" "+treeitems[j]);
}
for(int j1=0;j1<treeitems.length;j1++)
{
for(int j2=j1+1;j2<treeitems.length;j2++)
{
if(support[(treeitems[j1])-1][1]>support[(treeitems[j2])-1][1])
{
swapval=treeitems[j1];
treeitems[j1]=treeitems[j2];
treeitems[j2]=swapval;
}
}
}
int[] nodedetail1=new int[3];
Vector vecc=new Vector();
Vector tvecc=new Vector();
```



```
boolean status=false;
int tempindex=0;
int newindex=0;
int[][] subitem1=new int[100][];
int[][] subitem3=new int[100][100];
int[] test=new int[5];
int subcount=0;
int[] counterval=new int[10];
boolean mark=false;
for(int i=0;i<treeitems.length-1;i++)
{
newindex=0;
for(int i1=0;i1<tot.size();i1++)
{
int length=java.lang.reflect.Array.getLength(tot.get(i1));
int[] subitem=new int[length];
int[] subitem11=new int[length];
String strval=null;
for(int k=0;k<length;k++)
{
subitem[k]=Integer.parseInt(java.lang.reflect.Array.get(tot.get(i1), k).toString());
subitem11[k]=Integer.parseInt(java.lang.reflect.Array.get(tot.get(i1),
k).toString());
}
int ass=treeitems[i];
if( searching(subitem,treeitems[i])==true)
{
int[] nodedetail=new int[3];
String [] node=new String[3];
int[] itemindex=new int[subitem.length];
for(int jj=0;jj<subitem.length;jj++)
{
System.out.print(" "+subitem[jj]);
}
System.out.println(treeitems[i]+"\n");
if(status==true)
{
status=false;
counterval[newindex]=subcount;
newindex++;
subcount=0;
}
```

```
for(int k1=0;k1<subitem.length;k1++)
{
if(k1==0 && subitem[k1]!=treeitems[i])
{
k1++;
mark=true;
}
if(i1==0)
{
int[] temp=new int[node.length];
String tempchec=new String().valueOf(subitem[k1]);
strval=tempchec;
node[0]=strval;
node[1]="1";
node[2]="0";
for(int ii=0;ii<node.length;ii++)
{
temp[ii]=Integer.parseInt(node[ii]);
}
subcofitree.add(temp);
status=true;
subitem1[newindex]=new int[subitem.length];
System.out.println(subcofitree.lastIndexOf(temp));
System.out.println(java.lang.reflect.Array.get(subcofitree.get(subcofitree.lastIndex
Of(temp)),0));
subitem3[newindex][subcount]=subcofitree.lastIndexOf(temp);
subcount++;
}
else if(vectorsearch(subcofitree,subitem[k1])==true)
{
xx: for(int s=0;s<subcofitree.size();s++)
{
int len=java.lang.reflect.Array.getLength(subcofitree.get(s));
int[] arr=new int[len];
for(int s1=0;s1<len;s1++)
{
arr[s1]=Integer.parseInt(java.lang.reflect.Array.get(subcofitree.get(s),
s1).toString());
}
int tempcheck=Integer.parseInt(java.lang.reflect.Array.get(arr, 0).toString());
                    int tempcheck1=subitem[k1];
```



```
yy: if(Integer.parseInt(java.lang.reflect.Array.get(arr, 0).toString())==subitem[k1]
)
{
if(k1==0 || mark==true)
{
if(mark==true)
{
mark=false;
}
int tempint=Integer.parseInt(java.lang.reflect.Array.get(subcofitree.get(s),
1).toString());
tempint=tempint+1;
nodedetail[0]=Integer.parseInt(java.lang.reflect.Array.get(subcofitree.get(s),
0).toString());
nodedetail[1]=tempint;
nodedetail[2]=0;
int[] temp=new int[node.length];
for(int ii=0;ii<node.length;ii++)
{
temp[ii]=nodedetail[ii];
}
subcofitree.setElementAt(temp, s);
tempindex=s;
break xx;
}
else if(tempindex+1==s)
{
int tempint=Integer.parseInt(java.lang.reflect.Array.get(subcofitree.get(s),
1).toString());
tempint=tempint+1;
nodedetail[0]=Integer.parseInt(java.lang.reflect.Array.get(subcofitree.get(s),
0).toString());
nodedetail[1]=tempint;
nodedetail[2]=0;
int[] temp=new int[node.length];
for(int ii=0;ii<node.length;ii++)
{
temp[ii]=nodedetail[ii];
}
subcofitree.setElementAt(temp, s);
tempindex=s;
```



```
break xx;
}
else
{
nodedetail[0]=subitem[k1];
nodedetail[1]=1;
nodedetail[2]=0;
int[] temp=new int[node.length];
for(int ii=0;ii<node.length;ii++)
{
temp[ii]=nodedetail[ii];
}
subcofitree.add(temp);
if(k1==subitem.length-1)
{
status=true;
for(int isub=0;isub<subitem.length;isub++)
{
subitem3[newindex][subcount]=(subcofitree.lastIndexOf(temp)-isub);
subcount++;
}
}
break xx;
}
}
else if(s!=subcofitree.size())
{
break yy;
}
}
}
else
{
nodedetail[0]=subitem[k1];
System.out.println(nodedetail[0]);
int sun=subitem[k1];
nodedetail[1]=1;
nodedetail[2]=0;
int[] temp=new int[node.length];
```



```
for(int ii=0;ii<node.length;ii++)
{
temp[ii]=nodedetail[ii] ;
}
subcofitree.add(temp);
if(k1==subitem.length-1)
{
status=true;
for(int isub=0;isub<subitem.length;isub++)
{
subitem3[newindex][subcount]=(subcofitree.lastIndexOf(temp)-isub);
subcount++;
}
}
}
int s=subitem3.length;
}
}
System.out.println("xx"+newindex);
Vector subtree=new Vector();
for(int ii=0;ii<newindex;ii++)
{
for(int k3=0;k3<counterval[ii];k3++)
{
int len=java.lang.reflect.Array.getLength(subcofitree.get(subitem3[ii][k3]));
System.out.println(subitem3[ii][k3]+"
"+java.lang.reflect.Array.get(subcofitree.get(subitem3[ii][k3]),0));
int[] arr=new int[len];
for(int s1=0;s1<len;s1++)
{
arr[s1]=Integer.parseInt(java.lang.reflect.Array.get(subcofitree.get(subitem3[ii][k3
]), s1).toString());
}
nodedetail1[0]=Integer.parseInt(java.lang.reflect.Array.get(arr, 0).toString());
System.out.println(nodedetail1[0]);
nodedetail1[1]=Integer.parseInt(java.lang.reflect.Array.get(arr, 1).toString());
nodedetail1[2]=Integer.parseInt(java.lang.reflect.Array.get(arr, 2).toString());
int[] temp=new int[nodedetail1.length];
```

```
for(int st=0;st<temp.length;st++)
{
temp[st]=nodedetail1[st];
}
subtree.addElement(temp);
}
Vector vecswap=new Vector();
vecswap=(Vector)subtree.clone();
maintree.addElement(vecswap);
subtree.removeAllElements();
sss++;
}
cofitree.addElement(maintree);
subcofitree.removeAllElements();
}
System.out.println("size"+cofitree.size());
jTextArea1.append("COFItree mining"+"\n");
for(int i=0;i<cofitree.size();i++)
{
Vector itemsets=new Vector();
Vector tree=new Vector();
tree=(Vector)cofitree.get(i);
Vector treesub=new Vector();
System.out.println(tree.size());
for(int j=0;j<tree.size();j++ ) {
jTextArea1.append("\n");
treesub=(Vector)tree.get(j);
System.out.println(treesub.size());
for(int k=0;k<treesub.size();k++)
{
System.out.println((java.lang.reflect.Array.get(treesub.get(k), 0).toString()));
System.out.println(support[3][1]);
if(support[Integer.parseInt(java.lang.reflect.Array.get(treesub.get(k),
0).toString())][1]>=4)
{
itemsets.add(java.lang.reflect.Array.get(treesub.get(k), 0).toString());
}
}
int t1=Integer.parseInt(java.lang.reflect.Array.get(treesub.get(treesub.size()-1),
1).toString());
int t2=Integer.parseInt(java.lang.reflect.Array.get(treesub.get(treesub.size()-1),
2).toString());
```



```
for(int ii=1;ii<itemsets.size();ii++)
{
String str1=itemsets.get(0).toString();
String str2=itemsets.get(ii).toString();
jTextArea1.append("  "+str1+" "+str2+":"+(t1-t2));
}
System.out.println("itemsize"+itemsets.size());
for(int s=0;s<itemsets.size();s++)
{
if((t1-t2)!=0)
{
nodedetail1[0]=Integer.parseInt(java.lang.reflect.Array.get(treesub.get(s),
0).toString());
nodedetail1[1]=Integer.parseInt(java.lang.reflect.Array.get(treesub.get(s),
1).toString());
nodedetail1[2]=Integer.parseInt(java.lang.reflect.Array.get(treesub.get(s),
2).toString());
nodedetail1[2]=nodedetail1[2]+(t1-t2);
int val=nodedetail1[0];
int[] temp=new int[nodedetail1.length];
for(int it=0;it<temp.length;it++)
{
temp[it]=nodedetail1[it];
}
int pos=indexsearch(treesub,temp[0]);
int[] temp1=new int[nodedetail1.length];
for(int ii=0;ii<nodedetail1.length;ii++)
{
temp1[ii]=nodedetail1[ii];
}
treesub.setElementAt(temp1, pos);
}
}
tree.setElementAt(treesub,j);
itemsets.clear();
}
}
}
catch(ClassNotFoundException e)
{
}
```



```
catch(java.sql.SQLException e)
{
}
}//GEN-LAST:event_jButton1ActionPerformed
private void jButton3ActionPerformed(java.awt.event.ActionEvent evt)
{//GEN-FIRST:event_jButton3ActionPerformed
Vector vec=new Vector();
Try
{
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
String url="jdbc:odbc:spf";
String str=null;
int count=0;
Connection con =DriverManager.getConnection(url,"", "");
stmt = con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
ResultSet.CONCUR_UPDATABLE);
ResultSetMetaData rm;
rs = stmt.executeQuery("SELECT * FROM spf");
// Get the rowcount column value.int
rm=rs.getMetaData();
noofitems=rm.getColumnCount();
String tempstr=null;
rs.last();
int support[]=new int[noofitems];
String cval[]=new String[noofitems-2];
int support1[]=new int[noofitems];
for(int i=3;i<=noofitems;i++)
{
cval[i-3]=rm.getColumnName(i);
}
nooftrans=rs.getRow();
noofitems=noofitems-2;
Vector index=new Vector();
for(int i=0;i<noofitems;i++)
{
for(int j=0;j<nooftrans;j++)
{
str="select * from spf where id="+(j+1);
rs=stmt.executeQuery(str);
rs.first();
if(rs.getString(i+3).equals("1"))
```



```
{
count++;
}
}
support[i]=count;
support1[i]=count;
if(count>4)
{
vec.add(tempstr.valueOf(i+1));
}
count=0;
}
String temp1=null;
int temp=0;
for(int i=0;i<6;i++)
{
for(int j=i+1;j<6;j++)
{
if(support1[i]>support1[j])
{
temp=support1[j];
temp1=cval[j];
cval[j]=cval[i];
cval[i]=temp1;
support1[j]=support1[i];
support1[i]=temp;
}
}
}
String[][] va=new String[nooftrans][7];
String[][] va1=new String[nooftrans][7];
ResultSet rsv=null;
Statement stmt1;
for(int j=1;j<nooftrans;j++)
{
try
{
str="select * from spf where id="+(j);
stmt1 =
con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCU
R_UPDATABLE);
rsv=stmt1.executeQuery(str);
```

```
rsv.first();
String[] sv=new String[2];
sv=cval[0].split("t");
String[] sv1=new String[2];
sv1=cval[1].split("t");
String[] sv2=new String[2];
sv2=cval[2].split("t");
String[] sv3=new String[2];
sv3=cval[3].split("t");
String[] sv4=new String[2];
sv4=cval[4].split("t");
String[] sv5=new String[2];
sv5=cval[5].split("t");
va[j][1]=rsv.getString(3);
va[j][2]=rsv.getString(4);
va[j][3]=rsv.getString(5);
va[j][4]=rsv.getString(6);
va[j][5]=rsv.getString(7);
va[j][6]=rsv.getString(8);
}
catch(Exception e)
{
System.out.println(e);
}
}
 int[][] tarray=new int[100][100];
 int[][] tarray1=new int[100][100];
 for(int i=1;i<nooftrans;i++)
 {
 int counter=0;
 for(int j=1;j<=noofitems;j++)
 {
 if(va[i][j].equals("1"))
 {
 Object obj;
 obj=(Object)new String().valueOf(j);
 index.add(obj);
 counter++;
 }
 }
 String[] arr=new String[index.size()];
```



```
for(int v=0;v<index.size();v++)
{
arr[v]=(index.get(v)).toString();
}
String tempv=null;
for(int v=0;v<index.size();v++)
{
for(int v1=v+1;v1<index.size();v1++)
{
String a=arr[v];
String a1=arr[v1];
if(support[Integer.parseInt(a)-1]>support[Integer.parseInt(a1)-1])
{
tempv=arr[v];
arr[v]=arr[v1];
arr[v1]=tempv;
}
}
}
for(int v=0;v<index.size()-1;v++)
{
System.out.print(arr[v]);
}
System.out.println();
if(counter>=2)
{
for(int v=0;v<index.size()-1;v++)
{
String[] sva=new String[6];
int a=Integer.parseInt(arr[v]);
int b=Integer.parseInt(arr[v+1]);
String[] sv=new String[2];
sv=cval[0].split("t");
sva[0]=sv[1];
String[] sv1=new String[2];
sv1=cval[1].split("t");
sva[1]=sv1[1];
String[] sv2=new String[2];
sv2=cval[2].split("t");
sva[2]=sv2[1];
String[] sv3=new String[2];
```


```
sv3=cval[3].split("t");
sva[3]=sv3[1];
String[] sv4=new String[2];
sv4=cval[4].split("t");
sva[4]=sv4[1];
String[] sv5=new String[2];
sv5=cval[5].split("t");
sva[5]=sv5[1];
int counter1=0,counter2=0;
for(int counterv1=0;counterv1<6;counterv1++)
{
if(sva[counterv1].equals(arr[v]))
{
counter1=counterv1;
}
if(sva[counterv1].equals(arr[v+1]))
{
counter2=counterv1;
}
}
int cv=0;
while(!(tarray[counter1][cv]==0))
{
cv++;
}
int cv1=0;
while(!(tarray[counter2][cv1]==0))
{
cv1++;
}
tarray[counter1][cv]=counter2+1;
tarray1[counter1][cv]=cv1+1;
if(v==(index.size()-2))
{
tarray[counter2][cv1]=9;
tarray1[counter2][cv1]=9;
}
}
}
index.removeAllElements() ;
}
jTextArea1.append("Inverted Matrix\n");
```



```
for(int i=0;i<6;i++)
{
System.out.print(i+1+" "+cval[i]+" "+support1[i]+" ");
jTextArea1.append(i+1+"\t"+cval[i]+" "+support1[i]+" ");
for(int j=0;j<6;j++)
{
System.out.print("\t"+tarray[i][j]+"  "+tarray1[i][j]);
jTextArea1.append("\t"+tarray[i][j]+"  "+tarray1[i][j]);
}
System.out.println();
jTextArea1.append("\n");
}
}
catch(ClassNotFoundException e)
{
}
catch(java.sql.SQLException e)
{
}
}//GEN-LAST:event_jButton3ActionPerformed
private String[][] jButton2ActionPerformed(java.awt.event.ActionEvent evt)
{//GEN-FIRST:event_jButton2ActionPerformed
String SM[][]=new String[1000][];
int direction=-1,vectorpos=0;
int head=1,pos=0;
int[][] itemvector;
int ind=0;
boolean flag[][] =new boolean[1000][];
boolean set=false;
String query=null;
int s1;
try
{
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
String url="jdbc:odbc:spf";
Connection con =DriverManager.getConnection(url,"", "");
stmt = con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
ResultSet.CONCUR_UPDATABLE);
ResultSetMetaData rm;
rs = stmt.executeQuery("SELECT * FROM spf");
rm=rs.getMetaData();
noofitems=rm.getColumnCount();
```

```
rs.last();
nooftrans=rs.getRow();
noofitems=noofitems-2;
System.out.println("temcount"+noofitems);
rs.first();
String temp=null;
rs.beforeFirst();
while(rs.next())
{
if(temp==null)
{
temp=rs.getString(2);
pos++;
index[ind]=new String[2];
index[ind][0]=String.valueOf(pos);
index[ind][1]=rs.getString(2);
ind++;
}
else if (!(temp.equals(temp=rs.getString(2))))
{
pos++;
index[ind]=new String[2];
index[ind][0]=String.valueOf(pos);
index[ind][1]=temp;
ind++;
}
}
lastind=ind;
rs.first();
}
catch(ClassNotFoundException e)
{
System.out.println("exception in driver");
}
catch(SQLException e)
{
System.out.println("exception in sql");
}
for(int i=0;i<ind;i++)
{
flag[i]=new boolean[2];
flag[i][0]=false;
```



```
flag[i][1]=false;
}
for(int i=0;i<noofitems;i++)
{
int p=0,q=0;
itemvector=new int[noofitems][2];
String[] datestring=new String[2];
datestring=MCP(i+3);
for(int j=0;j<ind;j++)
{
if(index[j][1]!=null)
{
if(index[j][1].equals(datestring[0]))
{
p=Integer.parseInt(index[j][0]);
}
else if(index[j][1].equals(datestring[1]))
{
q=Integer.parseInt(index[j][0]);
}
}
}
}
String str=null;
for(int i=1;i<=ind-1;i++) {
if(direction==-1)
{
if(flag[i][1]==true && flag[i][2]==true)
{
SM[vectorpos]=new String[3];
SM[vectorpos][0]=str.valueOf(head);
SM[vectorpos][1]=str.valueOf(i);
SM[vectorpos][2]=str.valueOf(direction);
head=i+1;
vectorpos++;
}
flag[i][0]==true && flag[i][1]==false) {
direction=1;
}
```



```
else if(flag[i][0]==false && flag[i][1]==false)
{
direction=2;
}
}
else if(direction==1)
{
if(flag[ind][2]==true)
{
SM[vectorpos]=new String[3];
SM[vectorpos][0]=str.valueOf(head);
SM[vectorpos][1]=str.valueOf(i);
SM[vectorpos][2]=str.valueOf(direction);
head=i+1;
direction=-1;
vectorpos++;
}
}
else if(direction==2)
{
if(flag[i][1]==true)
{
SM[vectorpos]=new String[3];
SM[vectorpos][0]=str.valueOf(head);
SM[vectorpos][1]=str.valueOf(i);
SM[vectorpos][2]=str.valueOf(direction);
head=i+1;
direction=-1;
vectorpos++;
}
}
}
SM[vectorpos]=new String[3];
SM[vectorpos][0]=str.valueOf(head);
SM[vectorpos][1]=str.valueOf(ind);
SM[vectorpos][2]=str.valueOf(direction);
vectorpos++;
for(int i=0;i<vectorpos;i++)
{System.out.println(SM[i][0]);
System.out.println(SM[i][1]);
System.out.println(SM[i][2]);
}
```



```
Procfilter(1,2,"left");
return SM;
}//GEN-LAST:event_jButton2ActionPerformed
/** Exit the Application */
private void exitForm(java.awt.event.WindowEvent evt)
{//GEN-FIRST:event_exitForm
System.exit(0);
}//GEN-LAST:event_exitForm
public static void main(String args[])
{
new Datamining().show();
}
public String[] MCP(int index)
{
String[] first=new String[15];
String[][] arr=new String[4][2];
String[][] arr1=new String[4][2];
int count=0;
String item,tempstr; String item1;
String record=null;
String record1=null;
String record2=null;
try
{
String temp=null,temp1=null;
int ind=0,ind1=0;
rs.beforeFirst();
if(index==7)
{
System.out.println("");
}
while(rs.next())
{
record=rs.getString(index);
record2=rs.getString(2);
if(Integer.parseInt(record)==1)
{
if( temp==null )
{
temp=record2;
arr[ind][0]= temp;
arr[ind][1]="1";
```

```
ind++;
}
else if(temp.equals(record2)==false)
{
temp=record2;
if((arr[ind-1][0]).equals(temp)==false)
{
arr[ind][0]=temp;
arr[ind][1]="1";
ind++;
}
}
}
if(Integer.parseInt(record)==1)
{
if( temp1==null )
{
temp1=record2;
arr1[ind1][0]=temp1;
arr1[ind1][1]="1";
ind1++;
}
else if(temp1.equals(record2)==false)
{
temp1=record2;
if((arr1[ind1-1][0]).equals(temp1)==false)
{
arr1[ind1][0]=temp1;
arr1[ind1][1]="1";
ind1++;
}
}
}
}
for(int i=0;i<ind;i++)
{
if(first[0]==null)
first[0]=arr[i][0];
else
first[1]=arr[i][0];
}
}
```



```
catch(SQLException e)
{
System.out.println(e);
}
return first;
}
public void Procfilter(int p, int q, java.lang.String direction)
{
Vector ps=new Vector();
int head=0,tail=0;
String str=null;
if(direction.equals("left")==true)
{
head=p;
tail=q;
}
else
{
head=q;tail=p;
}
for(int h=head;h<=tail;h++)
{
}
Vector vec1=new Vector();
Vector tempvec=new Vector();
int finalcount=0,threshold=2;
for(int i=0;i<lastind;i++)
{
tempvec=arrayset(index[i][1].toString());
vec1.addElement(tempvec);
}
int twoitem[][]=new int[100][2];
int index2=0;
jTextArea1.append("frequent item sets:"+"\n");
try
{
for(int i=0;i<vec1.size();i++)
{
Vector obj=(Vector)vec1.get(i);
int arr[]=new int[obj.size()];
```



```
for(int j=0;j<obj.size();j++)
{
arr[j]=Integer.parseInt(obj.get(j).toString());
}
for(int i1=0;i1<arr.length;i1++)
{
for(int j1=i1+1;j1<arr.length;j1++)
{
twoitem[index2][0]=arr[i1];
twoitem[index2][1]=arr[j1];
index2++;
}
}
for(int k1=0;k1<index2;k1++)
{
for(int k=1;k<=nooftrans;k++)
{
str="select * from spf where id="+k;
rs=stmt.executeQuery(str);
rs.first();
if(rs.getString(twoitem[k1][0]).equals("1")==true &&
rs.getString(twoitem[k1][1]).equals("1")==true )
{
finalcount++;
}
}
if(finalcount>=threshold)
{
jTextArea1.append(twoitem[k1][0]+" ,"+twoitem[k1][1]+"  ");
}
}
jTextArea1.append("\n");
index2=0;
}
}
catch(SQLException e)
{
System.out.println(e);
}
}
public Vector arrayset(java.lang.String date)
{
```



```
String str=null,temp=null;
Vector vec=new Vector();
int count=0;
int index=0;
try
{
str="select * from spf where date="+"'"+date+"'";
rs1=stmt.executeQuery(str);
while(rs1.next())
{
String strval=rs1.getString(3);
for(int i=0;i<noofitems;i++)
{
if(rs1.getString(i+3).equals("1"))
{
if(index==0)
{
vec.addElement(temp.valueOf((i+3)-2));
}
else
{
for(int i1=0;i1<vec.size();i1++)
{
if(vec.get(i1).toString().equals(temp.valueOf((i+3)-2)))
{
count++;
}
}
if(count==0)
{
vec.addElement(temp.valueOf((i+3)-2));
}
else
{
count=0;
}
}
}
}
index++;
}
}
```

```
catch(SQLException e)
{
System.out.println(e);
}
return vec;
}
public int[] sorting(int[] items)
{
Arrays.sort(items);
return items;
}
public boolean searching(int[] values, int key)
{
boolean status=false;
for(int i=0;i<values.length;i++)
{
if(values[i]==key)
status= true;
}
return status;
}
public boolean vectorsearch(Vector source, int key)
{
boolean result=false;
for(int i=0;i<source.size();i++)
{
int sourceval=Integer.parseInt(java.lang.reflect.Array.get(source.get(i),
0).toString());
if(sourceval==key)
result=true;
}
return result;
}
public int indexsearch(Vector source, int index)
{
for(int i=0;i<source.size();i++)
{
if(Integer.parseInt(java.lang.reflect.Array.get(source.get(i),0).toString())==index)
{
return i;
}
}
```



```
return 0;
}
// Variables declaration - do not modify//GEN-BEGIN:variables
private javax.swing.JButton jButton1;
private javax.swing.JButton jButton2;
private javax.swing.JButton jButton3;
private javax.swing.JLabel jLabel1;
private javax.swing.JLabel jLabel2;
private javax.swing.JPanel jPanel1;
private javax.swing.JPanel jPanel2;
private javax.swing.JPanel jPanel3;
private javax.swing.JScrollPane jScrollPane1;
private javax.swing.JTextArea jTextArea1;
// End of variables declaration//GEN-END:variables
}
```


8.2 SAMPLE OUTPUT

Fig 8.1 SPF OUTPUT

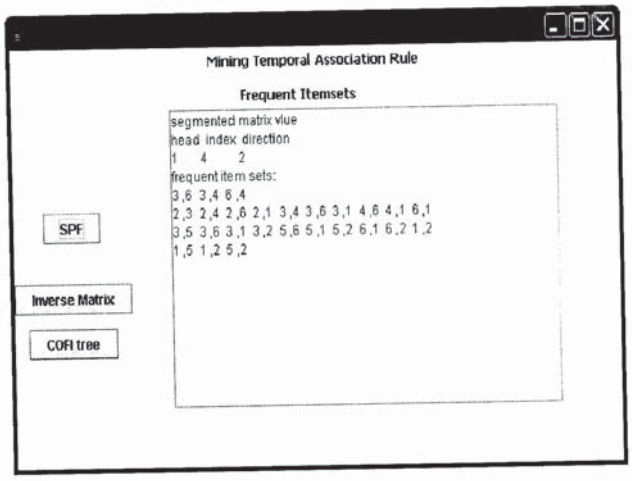


Fig 8.2 INVERTED MATRIX OUTPUT

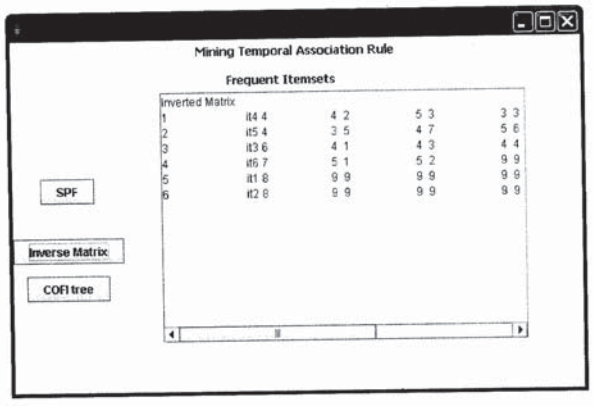
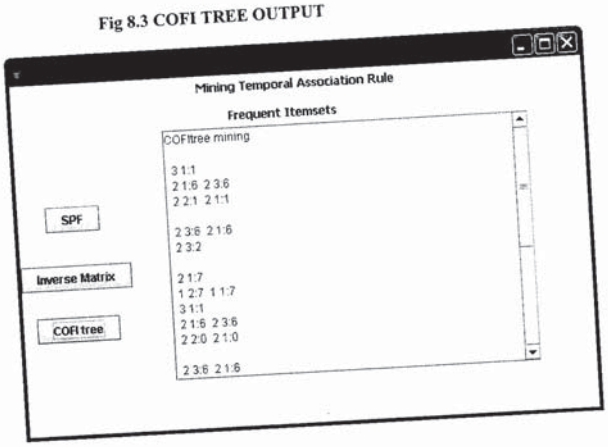


Fig 8.3 COFI TREE OUTPUT



9. REFERENCES

- M. El-Hajj and O. R. Za¨ýane. Inverted matrix: Efficient discovery of frequent items in large datasets in the context of interactive mining, university of Alberta, technical report 08-03, March 2003.
- Arun K Pujari "Data Mining Techniques ", Universities Press, 6 th Edition.
- J. Han and M. Kamber. Data Mining: Concepts and Techniques. Morgan Kaufman, San Francisco, CA, 2001.
- R. Agarwal, C. Aggarwal, and V. Prasad. A Tree Projection Algorithm for Generation of Frequent Item sets. Journal of Parallel and Distributed Computing (Special Issue on High Performance Data Mining), 2000.
- Dolf Zantinge and Pietri Adrians "Data Mining", Pearson Education, 2003.