An Improved Frequent Pattern Mining Algorithm Using Suffix Tree & Suffix Automata

MD. SABBIR AHMED Reg. No. : 10101020
MOHAMMAD RIAJUR RAHMAN Reg. No. : 10101006
MD. MOTAHER HOSSAIN Reg. No. : 10101024
MD. KHALID HASAN Reg. No. : 10101025

Submitted in partial fulfillment of the degree of Bachelor of Science, with Honours at University of Asia Pacific, Dhaka, Bangladesh.

29 May, 2014
DECLARATION

We, hereby, declare that the work presented in this thesis is the outcome of the investigation performed by us under the supervision of Md. Shiplu Hawlader, Lecturer, Department of Computer Science and Engineering, University of Asia Pacific. We also declare that no part of this thesis and thereof has been or is being submitted elsewhere for the award of any degree or diploma.

<table>
<thead>
<tr>
<th>Signature</th>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD. SABBIR AHMED</td>
<td>MOHAMMAD RIAJUR RAHMAN</td>
</tr>
<tr>
<td>Candidate</td>
<td>Candidate</td>
</tr>
<tr>
<td>Countersigned</td>
<td></td>
</tr>
<tr>
<td>Md. Shiplu Hawlader</td>
<td></td>
</tr>
<tr>
<td>Supervisor</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Signature</th>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD. MOTAHER HOSSAIN</td>
<td>MD. KHALID HASAN</td>
</tr>
<tr>
<td>Candidate</td>
<td>Candidate</td>
</tr>
</tbody>
</table>
The Thesis Report "An Improved Frequent Pattern Mining Algorithm Using Suffix Tree & Suffix Automata" submitted by MD. SABBIR AHMED, Reg.NO.10101020; MOHAMMAD RIAJUR RAHMAN, Reg.NO.10101006; MD. MOTAHER HOSSAIN, Reg.NO.:10101024, MD.KHALID HASAN, Reg.NO.:10101025 students of Spring-2010, to the Department of Computer Science & Engineering, University of Asia Pacific, has been accepted as satisfactory for the partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science & Engineering and approved as to its style and contents.

Approved as to the style & contents by

<table>
<thead>
<tr>
<th>Name</th>
<th>Position</th>
<th>Department</th>
<th>University</th>
</tr>
</thead>
<tbody>
<tr>
<td>Md. Firoz Mridha</td>
<td>Assistant Professor</td>
<td>Department of Computer Science &amp; Engineering, University of Asia Pacific</td>
<td></td>
</tr>
<tr>
<td>Dr. Md. Shahriar Rahman</td>
<td>Assistant Professor</td>
<td>Department of Computer Science &amp; Engineering, University of Asia Pacific</td>
<td></td>
</tr>
<tr>
<td>Md. Akhtaruzaman Adnan</td>
<td>Lecturer</td>
<td>Department of Computer Science &amp; Engineering, University of Asia Pacific</td>
<td></td>
</tr>
</tbody>
</table>
ACKNOWLEDGEMENTS

First of all, thanks to Almighty Allah for giving us the potency and energy to complete this thesis successfully.

We want to express out gratefulness towards our thesis supervisor Md. Shiplu Hawlader for his valuable advices and important suggestions. His regular and active supervision and erudite directions from the beginning to the end were the driving forces for the successful completion of the research work.

We would like to convey our thankfulness to all of our teachers at the Department of Computer Science and Engineering, University of Asia Pacific. Discussions with many of them have helped us tremendously in improving the quality of our work. We also thank the department for providing us with resources which were necessary for the preparation of the thesis.

And last but not the least, we would like to express thanks to our parents and family members for their tremendous support and inspiration.
ABSTRACT

As with the advancement of the IT technologies, the amount of accumulated data is also increasing. It has resulted in large amount of data stored in databases, warehouses and other repositories. Thus the Data mining comes into picture to explore and analyze the databases to extract the interesting and previously unknown patterns and rules known as association rule mining. Association Rule Mining is used to Finding frequent patterns, associations, correlations, or causal structures among sets of items or objects in transaction databases, relational databases, and other information repositories. Many algorithms have been proposed from last many decades for solving frequent pattern mining. Among different approaches to solve frequent pattern mining, a relatively new one is an improved frequent pattern mining algorithm using suffix tree & suffix automata. Previous researches we found which were based on prefix tree. In this paper, we have proposed a new frequent pattern mining algorithm which based on suffix tree & suffix automata. Experimental results on synthetic datasets show that the proposed algorithm provides better accuracy compared to previous algorithms.
## Contents

1 **Introduction**                                      1  
   1.1 Motivation ................................................................. 3  
   1.2 Aims & Objectives ....................................................... 4  
   1.3 Outlines of This Thesis ............................................... 4  

2 **Data Mining**                                     5  
   2.1 Basic Concept ............................................................. 5  
   2.2 Data Mining Applications ............................................. 6  
   2.3 The Primary Methods of Data Mining .............................. 8  

3 **Basic Concepts of Association Rule Mining**       9  
   3.1 Definition ................................................................. 10  
   3.2 Rule Evaluation Metrics .............................................. 10  
   3.3 Basic Concepts ............................................................ 11  
   3.4 Observations ............................................................... 12  
   3.5 Association Model ........................................................ 12  
   3.6 Problem of Association Rules ........................................ 12  

4 **Literature Review**                              13  
   4.1 The Apriori Algorithm ................................................ 13  
      4.1.1 The Apriori Algorithm in a Nutshell ......................... 14  
      4.1.2 The Apriori Algorithm: Pseudo code ......................... 15  
      4.1.3 Setting ............................................................... 15  
      4.1.4 Limitations .......................................................... 16  
      4.1.5 The Apriori Algorithm: Example .............................. 17  
      4.1.6 Methods to Improve Apriori’s Efficiency ................. 22  
      4.1.7 Principle ............................................................ 23  
      4.1.8 Flow chart of Apriori Algorithm ........................... 24  
      4.1.9 Hash based method of Apriori Algorithm .................. 24  
      4.1.10 Graph based approach ........................................... 25
# List of Tables

<table>
<thead>
<tr>
<th></th>
<th>Table Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-1</td>
<td>Example of Frequent Pattern</td>
<td>3</td>
</tr>
<tr>
<td>3-1</td>
<td>Definition of Association Rule</td>
<td>10</td>
</tr>
<tr>
<td>3-2</td>
<td>Example of Support &amp; Confidence</td>
<td>11</td>
</tr>
<tr>
<td>4-1</td>
<td>Apriori Example</td>
<td>17</td>
</tr>
<tr>
<td>4-2</td>
<td>FP-Growth Example</td>
<td>31</td>
</tr>
<tr>
<td>4-3</td>
<td>Mining the FP-Tree by creating conditional (sub) pattern bases</td>
<td>32</td>
</tr>
<tr>
<td>6-1</td>
<td>Complexity Analysis</td>
<td>46</td>
</tr>
</tbody>
</table>
List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-1</td>
<td>Data mining is the core of Knowledge discovery process</td>
<td>6</td>
</tr>
<tr>
<td>2-2</td>
<td>Data mining applications in 2008</td>
<td>7</td>
</tr>
<tr>
<td>4-1</td>
<td>First step of Apriori Algorithm</td>
<td>18</td>
</tr>
<tr>
<td>4-2</td>
<td>Second step of Apriori Algorithm</td>
<td>19</td>
</tr>
<tr>
<td>4-3</td>
<td>Third step of Apriori Algorithm</td>
<td>20</td>
</tr>
<tr>
<td>4-4</td>
<td>Principle of Apriori Algorithm</td>
<td>23</td>
</tr>
<tr>
<td>4-5</td>
<td>Flow chart of Apriori Algorithm</td>
<td>24</td>
</tr>
<tr>
<td>4-6</td>
<td>Transaction database (left), item frequencies (middle), and reduced transaction database with items in transactions sorted descending W.R.T. their frequency (right)</td>
<td>29</td>
</tr>
<tr>
<td>4-7</td>
<td>An FP-Tree that registers compressed, frequent pattern information</td>
<td>32</td>
</tr>
<tr>
<td>5-1</td>
<td>Example of Suffix Tree</td>
<td>39</td>
</tr>
<tr>
<td>5-2</td>
<td>Figure 5-2: (a) A deterministic finite automation A and (b) a deterministic automation recognizing $\Sigma^*L(A)$ where transitions labeled with $\phi$ are failure transitions</td>
<td>40</td>
</tr>
<tr>
<td>5-3</td>
<td>Contraction of Build Tree</td>
<td>43</td>
</tr>
<tr>
<td>5-4</td>
<td>Frequent Pattern is B(2)</td>
<td>44</td>
</tr>
<tr>
<td>5-5</td>
<td>Frequent Pattern is A(2)</td>
<td>44</td>
</tr>
<tr>
<td>5-6</td>
<td>Frequent Pattern is D(3), DA(2), DB(2)</td>
<td>44</td>
</tr>
<tr>
<td>5-7</td>
<td>Frequent Pattern is C(3), CD(3), CB(2), CA(2), CDA(2)</td>
<td>45</td>
</tr>
<tr>
<td>6-1</td>
<td>Comparison between Apriori, FP-Growth, CP-Tree &amp; Proposed Algorithm on T10I4D100K Data</td>
<td>48</td>
</tr>
<tr>
<td>6-2</td>
<td>Comparison between Apriori, FP-Growth, CP-Tree &amp; Proposed Algorithm on Mushroom Data Set</td>
<td>49</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

Data mining has enticed a great deal of attention in the information industry and in society as a whole in recent years, due to the extensive availability of huge amounts of data and the ensuing need for turning such data into useful information and knowledge. The information and knowledge gained can be used for applications ranging from market analysis, fraud detection, and customer retention, to production control and science exploration.

Data mining offers to extracting or “mining” knowledge from large amounts of data. The term is actually a misnomer. Remember that the mining of gold from rocks or sand is referred to as gold mining rather than rock or sand mining. Thus, data mining should have been more appropriately named “knowledge mining from data,” which is unfortunately somewhat long. “Knowledge mining,” a shorter term may not reflect the emphasis on mining from large amounts of data. Nevertheless, mining is a sprightly term characterizing the process that finds a small set of precious nuggets from a great deal of raw material. Thus, such a misnomer that carries both “data” and “mining” became a popular choice. Many other terms carry a similar or slightly different meaning to data mining, such as knowledge mining from data, knowledge extraction, data/pattern analysis, data archaeology, and data dredging. Many people treat data mining as a synonym for another popularly used term, Knowledge Discovery from Data, or KDD. KDD applications deliver measurable benefits, including reduced cost of doing business, enhanced profitability, and improved quality of service. Therefore Knowledge Discovery in Databases has become one of the most active and exciting research areas in the database community.

Cluster analysis or clustering is the task of grouping a set of objects in such a way that objects in the same group (called a cluster) are more similar (in some sense or another) to each other than to those in other groups (clusters). It is a main task of empiric data mining, and a common technique for statistical data analysis, used in many fields, including machine learning, pattern recognition, image analysis, information retrieval, and bioinformatics.
Cluster analysis itself is not one specific algorithm, but the general task to be solved. It can be achieved by various algorithms that differ significantly in their notion of what constitutes a cluster and how to efficiently find them. Popular notions of clusters include groups with small distances among the cluster members, dense areas of the data space, intervals or particular statistical distributions. Clustering can therefore be formulated as a multi-objective optimization problem. The appropriate clustering algorithm and parameter settings depend on the individual data set and intended use of the results. Cluster analysis as such is not an automatic task, but an iterative process of knowledge discovery or interactive multiple objective optimization that involves trial and failure. It will often be necessary to modify data reprocessing and model parameters until the result achieves the desired properties. So, the goal of clustering is to determine the intrinsic grouping in a set of unlabeled data. [2]

Frequent pattern mining has been a focused theme in data mining research for over a decade. Abundant literature has been dedicated to this research and tremendous progress has been made, ranging from efficient and scalable algorithms for frequent item-set mining in transaction databases to numerous research frontiers, such as sequential pattern mining, structured pattern mining, correlation mining, associative classification, and frequent pattern-based clustering, as well as their broad applications.

Frequent pattern mining was first proposed by Agrawal et al. (1993) for market basket analysis in the form of association rule mining. It analyses customer buying habits by finding associations between the different items that customers place in their “shopping baskets”. For instance, if customers are buying milk, how likely are they going to also buy cereal (and what kind of cereal) on the same trip to the supermarket? Such information can lead to increased sales by helping retailers do selective marketing and arrange their shelf space. [3]

Frequent pattern is an intrinsic and important property of data-sets. Foundation for many essential data mining tasks, some are given below:

- Association, correlation, and causality analysis.
- Sequential, structural (e.g., sub-graph) patterns.
- Pattern analysis in spatiotemporal, multimedia, time series, and stream data.
- Classification: discriminative, frequent pattern analysis.
- Cluster analysis: frequent pattern-based clustering.
- Data warehousing: iceberg cube and cube-gradient.
- Semantic data compression: fascicles.
- Broad applications.
A simple example of Frequent Pattern:

<table>
<thead>
<tr>
<th>TID</th>
<th>Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Bread, Milk.</td>
</tr>
<tr>
<td>2</td>
<td>Bread, Diaper, Beer, Eggs.</td>
</tr>
<tr>
<td>3</td>
<td>Milk, Diaper, Beer, Cock.</td>
</tr>
<tr>
<td>4</td>
<td>Bread, Milk, Diaper, Beer.</td>
</tr>
<tr>
<td>5</td>
<td>Bread, Milk, Diaper, Cock.</td>
</tr>
</tbody>
</table>

Table 1-1: Example of Frequent Pattern

Here Bread, Milk, Diaper is a frequent pattern.

In this research paper, we proposed an improved frequent pattern mining algorithm which can reduce the time & space complexity.

1.1 Motivation

Nowadays, large quantities of data are being congealed. The amount of data collected is said to be almost doubled every 9 months. Seeking knowledge from massive data is one of the most desired multiplications of Data Mining. Data could be large in two senses.

Usually there is a huge gap from the stored data to the knowledge that could be construed from the data. This transition won't occur automatically, that's where Data Mining comes into picture. In Exploratory Data Analysis, some initial knowledge is known about the data, but Data Mining could help in a more in-depth knowledge about the data.

Manual data analysis has been around for some time now, but it creates a bottleneck for large data analysis.

Fast developing computer science and engineering techniques and methodology generates new demands. Data mining techniques are now being applied to all kinds of domains, which are rich in data, e.g. Image Mining and Gene data analysis.
Data mining also face some challenges: Increasing data dimensionality and data size, various forms of data, new types of data like streaming data and multimedia data, Efficiency in data access and Information search methods, intelligent upgrade and integration methods. [4]

1.2 Aims & Objectives

Many frequent pattern mining algorithm exist for generating frequent pattern itemsets. After studying these algorithms we found that these algorithms have a limitation which is scanning the database in multiple times. So, it took more space & time for generating frequent pattern itemsets. Our aim is to develop a new algorithm which will scan the database at beginning time. As a result it reduces time & space complexity. For implementing our algorithm we use suffix tree (ukkonen’s) which reduces the time complexity and for reducing space complexity we use suffix automata.

Our main objective is to develop some algorithm like apriori, fp-tree, ukkonen’s which results the frequent pattern mining itemsets. Then compare these previous implemented algorithms with our proposed algorithm.

1.3 Outline of the Thesis

The rest of the chapters are organized as follows. Chapter 2 presents Basic Concepts of Data Mining and its applications. In Chapter 3, Basic Concepts of Association Rule Mining. In Chapter 4, Literature Review with Apriori Algorithm, FP-Growth Algorithm & basic concepts about Proposed Algorithm. In Chapter 5, The Proposed Algorithm has been described clearly. In Chapter 6 contains the Performance Analysis. Finally, Chapter 7 contains Conclusion & the Future Work of the new Algorithm.
Chapter 2

Data Mining

With the increase in Information Technology, the size of the databases created by the organizations due to the availability of low-cost storage and the evolution in the data capturing technologies is also increasing. These organization sectors include retail, petroleum, telecommunications, utilities, manufacturing, transportation, credit cards, insurance, banking and many others, extracting the valuable data, it necessary to explore the databases completely and efficiently. Knowledge discovery in databases (KDD) helps to identifying precious information in such huge databases. This valuable information can help the decision maker to make accurate future decisions.

2.1 Basic Concept

This is the important part of KDD. Data mining generally involves four classes of task; classification, clustering, regression, and association rule learning. Data mining refers to discover knowledge in huge amounts of data. It is a scientific discipline that is concerned with analyzing observational data sets with the objective of finding unsuspected relationships and produces a summary of the data in novel ways that the owner can understand and use. Data mining as a field of study involves the merging of ideas from many domains rather than a pure discipline the four main disciplines [15], which are contributing to data mining include:

- **Statistics**: it can provide tools for measuring significance of the given data, estimating probabilities and many other tasks (e.g. linear regression).
- **Machine learning**: it provides algorithms for inducing knowledge from given data (e.g. SVM).
- **Data management and databases**: since data mining deals with huge size of data, an efficient way of accessing and maintaining data is necessary.
- **Artificial intelligence**: it contributes to tasks involving knowledge encoding or search techniques (e.g. neural networks).
Figure 2-1: Data mining is the core of Knowledge discovery process. [16]

2.2 Data Mining Applications

Data mining has become an essential technology for businesses and researchers in many fields, the number and variety of applications has been growing gradually for several years and it is predicted that it will carry on to grow. A number of the business areas with an early embracing of DM into their processes are banking, insurance, retail and telecom. More lately it has been implemented in pharmaceutics, health, government and all sorts of e-businesses.
One describes a scheme to generate a whole set of trading strategies that take into account application constraints, for example timing, current position and pricing [18]. The authors highlight the importance of developing a suitable back testing environment that enables the gathering of sufficient evidence to convince the end users that the system can be used in practice. They use an evolutionary computation approach that favors trading models with higher stability, which is essential for success in this application domain.

Apriori algorithm is used as a recommendation engine in an E-commerce system. Based on each visitor’s purchase history the system recommends related, potentially interesting, products. It is also used as basis for a CRM system as it allows the company itself to follow-up on customer’s purchases and to recommend other products by e-mail[19].

A government application is proposed by [20]. The problem is connected to the management of the risk associated with social security clients in Australia. The
problem is confirmed as a sequence mining task. The action ability of the model obtained is an essential concern of the authors. They concentrate on the difficult issue of performing an evaluation taking both technical and business interestingness into account.

2.3 The Primary Methods of Data Mining

Data mining addresses two basic tasks: verification and discovery. The verification task seeks to verify user’s hypotheses. While the discovery task searches for unknown knowledge hidden in the data. In general, discovery task can be further divided into two categories, which are descriptive data mining and predicative data mining.

Descriptive data mining describes the data set in a summery manner and presents interesting general properties of the data. Predictive data mining constructs one or more models to be later used for predicting the behavior, of future data sets.

There are a number of algorithmic techniques available for each data mining tasks, with features that must be weighed against data characteristics and additional business requirements. Among all the techniques, in this research, we are focusing on the association rules mining technique, which is descriptive mining technique, with transactional database system. This technique was formulated by [21] and is often referred to as market-basket problem.
Chapter 3

Basic Concepts of Association Rule Mining

Association rules are one of the major techniques of data mining. Association rule mining finding frequent patterns, associations, correlations, or causal structures among sets of items or objects in transaction databases, relational databases, and other information repositories [13]. The volume of data is increasing dramatically as the data generated by day-to-day activities. Therefore, mining association rules from massive amount of data in the database is interested for many industries which can help in many business decision making processes, such as cross-marketing, Basket data analysis, and promotion assortment. The techniques for discovering association rules from the data have traditionally focused on identifying relationships between items telling some aspect of human behaviour, usually buying behaviour for determining items that customers buy together. All rules of this type describe a particular local pattern. The group of association rules can be easily interpreted and communicated.

Association Rule was first proposed by Agrawal et al in 1993. It is an important data mining model studied extensively by the database and data mining community. It assumes all data are categorical. It has no good algorithm for numeric data. Initially used for Market Basket Analysis to find how items purchased by customers are related.

Many studies have been conducted to address various conceptual, implementation, and application issues relating to the association rules mining task. Researcher in application issues focuses on applying association rules to a variety of application domains. For example: Relational Databases, Data Warehouses, Transactional Databases, and Advanced Database Systems (Object-Relational, Spatial and Temporal, Time-Series, Multimedia, Text, Heterogeneous, Legacy, Distributed, and WWW) [22].
3.1 Definition

Let us consider an item-set, as like below:

<table>
<thead>
<tr>
<th>TID</th>
<th>Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Bread, Milk.</td>
</tr>
<tr>
<td>2</td>
<td>Bread, Diaper, Beer, Eggs.</td>
</tr>
<tr>
<td>3</td>
<td>Milk, Diaper, Beer, Cock.</td>
</tr>
<tr>
<td>4</td>
<td>Bread, Milk, Diaper, Beer.</td>
</tr>
<tr>
<td>5</td>
<td>Bread, Milk, Diaper, Cock.</td>
</tr>
</tbody>
</table>

Table 3-1: Definition of Association Rule

An implication expression of the form \( X \rightarrow Y \), where \( X \) and \( Y \) are item-sets

Example: \{Milk, Diaper\} \( \rightarrow \) \{Beer\}.

3.2 Rule Evaluation Metrics

- **Support (s)**
  - Fraction of transactions that contain both \( X \) and \( Y \). Which is expressed: \( \text{Supp}(Y) = P(X \cup Y) \)

- **Confidence (c)**
  - Measures how often items in \( Y \) appear in transactions that contain \( X \).
  - Which is expressed: \( \text{Conf}(Y|X) = P(X \cup Y) / P(X) \).

Suppose *computer & Game CD a Movie DVD* with minimum confidence and support

- Support, \( s \), probability that a transaction contains \{Computer, Game CD, Movie DVD\}

- Confidence, \( c \), conditional probability that a transaction having \{Computer, Game CD\} also contains Movie *DVD*.

Example:
<table>
<thead>
<tr>
<th>Transaction ID</th>
<th>Items Bought</th>
</tr>
</thead>
<tbody>
<tr>
<td>2000</td>
<td>A,B,C</td>
</tr>
<tr>
<td>1000</td>
<td>A,C</td>
</tr>
<tr>
<td>4000</td>
<td>A,D</td>
</tr>
<tr>
<td>5000</td>
<td>B,E,F</td>
</tr>
</tbody>
</table>

Table 3-2: Example of Support & Confidence.

Let minimum support 50%, and minimum confidence 50%, then we have,

- A \Rightarrow C (50%, 66.6%).
- C \Rightarrow A (50%, 100%).

3.3 Basic Concepts

This section introduces the basic concepts of frequent pattern mining for discovery of interesting associations and correlations between item-sets in transactional and relational database. Association rule mining can be defined formally as follows:

I= \{i_1, i_2, i_3, \ldots, i_n\} is a set of items, such as products like (computer, CD, printer, papers, and so on). Let DB be a set of database transactions where each transaction T is a set of items such that T \subseteq I. Each transaction is associated with a unique identifier, transaction identifier (TID). Let X, Y be a set of items, an association rule has the form \(X \rightarrow Y\), where \(X\) is called the antecedent and \(Y\) is called the consequent of the rule where \(X,Y\) is a set of items is called as an item-set or a pattern.

An item-set (or a pattern) is frequent if its support is equal to or more than a user specified minimum support (a statement of generality of the discovered association rules). Association rule mining is to identify all rules meeting user-specified constraints such as minimum support and minimum confidence (a statement of predictive ability of the discovered rules). One key step of association mining is frequent item-set (pattern) mining, which is to mine all item-sets satisfying user specified minimum support. [23]

However a large number of these rules will be pruned after applying the support and confidence thresholds. Therefore the previous computations will be wasted. To avoid this problem and to improve the performance of the rule discovery algorithm, mining association rules may be decomposed into two phases:

- Discover the large item-sets, i.e., the sets of items that have transaction support above a predetermined minimum threshold known as frequent Item-sets.

- Use the large item-sets to generate the association rules for the database that have confidence above a predetermined minimum threshold.
The overall performance of mining association rules is determined primarily by the first step. The second step is easy. After the large item-sets are identified, the corresponding association rules can be derived in straightforward manner. Our main consideration of the thesis is First step i.e. to find the extraction of frequent item-sets.

3.4 Observations

- All the above rules are binary partitions of the same item-set.
- Rules originating from the same item-set have identical support but can have different confidence.
- Thus, we may decouple the support and confidence requirements. [5]

3.5 Association Model

- $I = \{i_1, i_2, ..., i_n\}$ a set of items
- $J = \mathcal{P}(I)$ set of all subsets of the set of items, elements of $J$ are called item-sets
- **Transaction $T$:** $T$ is subset of $I$
- **Data Base:** set of transactions
- An association rule is an implication of the form: $X \rightarrow Y$, where $X, Y$ are disjoint subsets of $I$ (elements of $J$)

3.6 Problem of Association Rules

Find rules that have support and confidence greater than user specified minimum support and minimum confidence.
Chapter 4

Literature Review

This chapter presents basic Apriori & FP-Growth algorithm with principles, generating process, limitation, pseudo code etc.

4.1 The Apriori Algorithm

Frequent pattern mining is a heavily researched area in the field of data mining with wide range of applications. Mining frequent patterns from large scale databases has emerged as an important problem in data mining and knowledge discovery community. A number of algorithms have been proposed to determine frequent pattern. Apriori algorithm is the first algorithm proposed in this field. With the time a number of changes proposed in Apriori to enhance the performance in term of time and number of database passes. In this paper three different frequent pattern mining approaches (Record filter, Intersection and Proposed Algorithm) are given based on classical Apriori algorithm. In these approaches Record filter approach proved better than classical Apriori Algorithm, Intersection approach proved better than Record filter approach and finally proposed algorithm proved that it is much better than other frequent pattern mining algorithm. In last we perform a comparative study of all approaches on dataset of 2000 transaction.

The Apriori Algorithm is an influential algorithm for mining frequent item-sets for Boolean association rules. [6]

Apriori is an algorithm for frequent item set mining and association rule learning over transactional databases. It proceeds by identifying the frequent individual items in the database and extending them to larger and larger item sets as long as those item sets appear sufficiently often in the database. The frequent item sets determined by Apriori can be used to determine association rules which highlight general trends in the database: this has applications in domains such as market basket analysis. [7]

Frequent pattern are patterns that appear in a dataset frequently. For example, a set of items, such as milk and bread that appear frequently together in a transaction data set is a frequent item set. Frequent patterns are prevalent in real-life data, such as sets of items bought together in a superstore. Frequent pattern mining has been successfully applied to association rule mining, pattern-based classification, clustering, finding correlated items, and has become an essential data mining task. [24]
Frequent item sets play an essential role in many data mining tasks that try to find interesting patterns from databases. The original motivation for searching frequent pattern came from the need to analyze so-called supermarket transaction data, that is, to examine customer behavior in terms of the purchased products. Frequent Pattern describe how often items are purchased together. Since their introduction in 1993 by Agrawal et al. [25], the frequent item set and association rule mining problems have received a great deal of attention. Within the past decade, hundreds of research papers have been published. We present a new algorithms or improvements on existing algorithms to solve these mining problems more efficiently. In this chapter, we explain the basic frequent item set mining problems.

4.1.1 The Apriori Algorithm in a Nutshell

- Find the frequent item-sets: the sets of items that have minimum support
  - A subset of a frequent item-set must also be a frequent item-set
    i.e., if \( \{AB\} \) is a frequent item-set, both \( \{A\} \) and \( \{B\} \) should be a frequent item-set
  - Iteratively find frequent item-sets with cardinality from 1 to \( k \) (\( k \)-item-set)
- Use the frequent item-sets to generate association rules.
4.1.2 The Apriori Algorithm: Pseudo code

The pseudo code for the algorithm is given below for a transaction database, and a support threshold of. Usual set theoretic notation is employed; though note that is a multi-set.

- Join Step: $C_k$ is generated by joining $L_{k-1}$ with itself
- Prune Step: Any $(k-1)$-item-set that is not frequent cannot be a subset of a frequent $k$-item-set
- Pseudo-code:

\[
C_k: \text{Candidate item-set of size } k \\
L_k: \text{frequent item-set of size } k
\]

\[
L_1 = \{\text{frequent items}\}; \\
\text{For } (k = 1; L_k \neq \emptyset; k++) \text{ do begin} \\
C_{k+1} = \text{candidates generated from } L_k; \text{ for each transaction } t \text{ in database} \\
\text{do} \\
\text{Increment the count of all candidates in } C_{k+1} \text{ that are contained in } t \\
L_{k+1} = \text{candidates in } C_{k+1} \text{ with min_support} \text{ end} \\
\text{Return } \bigcup_k L_k;
\]

4.1.3 Setting

Apriori is designed to operate on databases containing transactions (for example, collections of items bought by customers, or details of a website frequented). Other algorithms are designed for finding association rules in data having no transactions (Winepi and Minepi), or having no timestamps (DNA sequencing). Each transaction is seen as a set of items (an item-set). Given a threshold, the Apriori algorithm identifies the item sets which are subsets of at least transactions in the database.

Apriori uses a "bottom up" approach, where frequent subsets are extended one item at a time (a step known as candidate generation), and groups of candidates are tested
against the data. The algorithm terminates when no further successful extensions are found.

Apriori uses breadth-first search and a Hash tree structure to count candidate item sets efficiently. It generates candidate item sets of length from item sets of length K-1. Then it prunes the candidates which have an infrequent sub pattern. According to the downward closure lemma, the candidate set contains all frequent k-length item sets. After that, it scans the transaction database to determine frequent item sets among the candidates.

### 4.1.4 Limitations

Apriori, while historically significant, suffers from a number of inefficiencies or trade-offs, which have spawned other algorithms. Candidate generation generates large numbers of subsets (the algorithm attempts to load up the candidate set with as many as possible before each scan). Bottom-up subset exploration (essentially a breadth-first traversal of the subset lattice) finds any maximal subset $S$ only after all $2^{|S|}-1$ of its proper subsets.

Later algorithms such as Max-Miner try to identify the maximal frequent item sets without enumerating their subsets, and perform "jumps" in the search space rather than a purely bottom-up approach. [8]
4.1.5 The Apriori Algorithm: Example

- Consider a database, D, consisting of 9 transactions.

<table>
<thead>
<tr>
<th>TID</th>
<th>List of Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>T100</td>
<td>I1, I2, I5</td>
</tr>
<tr>
<td>T100</td>
<td>I2, I4</td>
</tr>
<tr>
<td>T100</td>
<td>I2, I3</td>
</tr>
<tr>
<td>T100</td>
<td>I1, I2, I4</td>
</tr>
<tr>
<td>T100</td>
<td>I1, I3</td>
</tr>
<tr>
<td>T100</td>
<td>I2, I3</td>
</tr>
<tr>
<td>T100</td>
<td>I1, I3</td>
</tr>
<tr>
<td>T100</td>
<td>I1, I2, I3, I5</td>
</tr>
<tr>
<td>T100</td>
<td>I1, I2, I3</td>
</tr>
</tbody>
</table>

Table 4-1: Apriori Example

- Suppose min. support count required is 2 (i.e. min-sup = 2/9 = 22 %)
- Let minimum confidence required is 70%.
- We have to first find out the frequent item-set using Apriori algorithm.
- Then, Association rules will be generated using min. support & min. confidence.
Step 1: Generating 1-item-set Frequent Pattern

The set of frequent 1-item-sets, $L_1$, consists of the candidate 1-item-sets satisfying minimum support.

In the first iteration of the algorithm, each item is a member of the set of candidate.
Step2: Generating 2-item-set Frequent Pattern

Figure 4-2: Second step of Apriori Algorithm.

- To discover the set of frequent 2-item-sets, \( L_2 \), the algorithm uses \( L_1 \) Join \( L_1 \) to generate a candidate set of 2-item-sets, \( C_2 \).

- Next, the transactions in \( D \) are scanned and the support count for each candidate item-set in \( C_2 \) is accumulated (as shown in the middle table).

- The set of frequent 2-item-sets, \( L_2 \), is then determined, consisting of those candidate 2-item-sets in \( C_2 \) having minimum support.

- Note: We haven’t used Apriori Property yet.

Step3: Generating 3-item-set Frequent Pattern
The generation of the set of candidate 3-item-sets, $C_3$, involves use of the Apriori Property.

In order to find $C_3$, we compute $L_2$ Join $L_2$.

$$C_3 = L_2 \text{ Join } L_2 = \{\{I_1, I_2, I_3\}, \{I_1, I_2, I_5\}, \{I_1, I_3, I_5\}, \{I_2, I_3, I_4\}, \{I_2, I_3, I_5\}, \{I_2, I_4, I_5\}\}.$$  

Now, Join step is complete and Prune step will be used to reduce the size of $C_3$. Prune step helps to avoid heavy computation due to large $C_k$.

Based on the Apriori property that all subsets of a frequent item-set must also be frequent, we can determine that four latter candidates cannot possibly be frequent.

For example, let’s take $\{I_1, I_2, \text{ and } I_3\}$. The 2-item subsets of it are $\{I_1, I_2\}, \{I_1, I_3\} \& \{I_2, I_3\}$. Since all 2-item subsets of $\{I_1, I_2, \text{ and } I_3\}$ are members of $L_2$, We will keep $\{I_1, I_2, \text{ and } I_3\}$ in $C_3$.

Let’s take another example of $\{I_2, I_3, I_5\}$ which shows how the pruning is performed. The 2-item subsets are $\{I_2, I_3\}, \{I_2, I_5\} \& \{I_3, I_5\}$.

BUT, $\{I_3, I_5\}$ is not a member of $L_2$ and hence it is not frequent violating Apriori Property. Thus We will have to remove $\{I_2, I_3, I_5\}$ from $C_3$.

Therefore, $C_3 = \{\{I_1, I_2, I_3\}, \{I_1, I_2, I_5\}\}$ after checking for all members of result of Join operation for Pruning.

Now, the transactions in $D$ are scanned in order to determine $L_3$, consisting of
Step 4: Generating 4-item-set Frequent Pattern

- The algorithm uses \( L_3 \ Join \ L_3 \) to generate a candidate set of 4-item-sets, \( C_4 \). Although the join results in \( \{\{I_1, I_2, I_3, I_5\}\} \), this item-set is pruned since its subset \( \{I_2, I_3, I_5\} \) is not frequent.
- Thus, \( C_4 = \emptyset \), and algorithm terminates, having found all of the frequent items. This completes our Apriori Algorithm.

These frequent item-sets will be used to generate strong association rules (where strong association rules satisfy both minimum support & minimum confidence).

Step 5: Generating Association Rules from Frequent Item-sets

- Procedure:
  - For each frequent item-set “\( I \)”, generate all nonempty subsets of \( I \).
  - For every nonempty subset \( s \) of \( I \), output the rule “\( s \subseteq (I-s) \)” if \( \text{support}_\text{count}(I) / \text{support}_\text{count}(s) \geq \text{min}_\text{conf} \) where \( \text{min}_\text{conf} \) is minimum confidence threshold.

- Back To Example:

  We had \( L = \{\{I_1\}, \{I_2\}, \{I_3\}, \{I_4\}, \{I_5\}, \{I_1,I_2\}, \{I_1,I_3\}, \{I_1,I_5\}, \{I_2,I_3\}, \{I_2,I_4\}, \{I_2,I_5\}, \{I_1,I_2,I_3\}, \{I_1,I_2,I_5\}\} \).
  - Let’s take \( I = \{I_1,I_2,I_5\} \).
  - It’s all nonempty subsets are \( \{I_1,I_2\}, \{I_1,I_5\}, \{I_2,I_5\}, \{I_1\}, \{I_2\}, \{I_5\} \).
  - Let minimum confidence threshold is , say 70%.
  - The resulting association rules are shown below, each listed with its confidence.

  - \( R_1: I_1 \wedge I_2 \not\subseteq I_5 \)
    - Confidence = \( \text{sc}\{I_1,I_2,I_5\}/\text{sc}\{I_1,I_2\} = 2/4 = 50% \)
    - \( R_1 \) is Rejected.
R2: I1 ^ I5 ⊈ I2
   - Confidence = \( \frac{\text{sc}\{I1, I2, I5\}}{\text{sc}\{I1, I5\}} = \frac{2}{2} = 100\% \)
   - R2 is Selected.

R3: I2 ^ I5 ⊈ I1
   - Confidence = \( \frac{\text{sc}\{I1, I2, I5\}}{\text{sc}\{I2, I5\}} = \frac{2}{2} = 100\% \)
   - R3 is Selected.

R4: I1 ⊈ I2 ^ I5
   - Confidence = \( \frac{\text{sc}\{I1, I2, I5\}}{\text{sc}\{I1\}} = \frac{2}{6} = 33\% \)
   - R4 is Rejected.

R5: I2 ⊈ I1 ^ I5
   - Confidence = \( \frac{\text{sc}\{I1, I2, I5\}}{\text{sc}\{I2\}} = \frac{2}{7} = 29\% \)
   - R5 is Rejected.

R6: I5 ⊈ I1 ^ I2
   - Confidence = \( \frac{\text{sc}\{I1, I2, I5\}}{\text{sc}\{I5\}} = \frac{2}{2} = 100\% \)
   - R6 is Selected.

In this way, We have found three strong association rules.

4.1.6 Methods to Improve Apriori’s Efficiency

- Hash-based item-set counting: A \( k \)-item-set whose corresponding hashing bucket count is below the threshold cannot be frequent.
- Transaction reduction: A transaction that does not contain any frequent \( k \)-item-set is useless in subsequent scans.
- Partitioning: Any item-set that is potentially frequent in DB must be frequent in at least one of the partitions of DB.
- Sampling: mining on a subset of given data, lower support threshold + a method to determine the completeness.
- Dynamic item-set counting: add new candidate item-sets only when all of their subsets are estimated to be frequent.

4.1.7 Principle

Downward closure property.
✓ If an item-set is frequent, then all of its subsets must also be frequent
✓ If an item-set is not frequent, any of its super set is never frequent

Figure 4-4: Principle of Apriori Algorithm. [9]

4.1.8 Flow chart of Apriori Algorithm
4.1.9 Hash based method of Apriori Algorithm

\[
\text{Repeat // for each transaction of the database} \\
\{ \\
D = \{ \text{set of all possible k-item-sets in the ith transaction} \} \\
\text{For each element of } D \\
\{ \\
\text{Find a unique integer uniq_int using the hash function for } k\text{-item-set} \\
\text{Increment } \text{freq}[\text{uniq_int}] \\
\text{Increment } \text{trans_pos} \\
\text{// Moves pointer to next transaction until end_of_file} \\
\text{For } (\text{freq_ind}=0; \text{freq_ind}<\text{length_of_the_array(two_three_freq[]); freq_ind++}) \\
\{ \\
\text{if } (\text{freq[freq_ind]} \geq \text{required support}) \\
\text{mark the corresponding } k\text{-item-set} \\
\} \\
\}
\]

4.1.10 Graph based approach
Procedure FrequentItemGraph (Tree, F)
{
    scan the DB once to collect the frequent 2-itemsets and their support ascending;
    add all items in the DB as the header nodes
    for each 2-itemset entry (top down order) in freq2list
        do
            if (first item = item in header node) then
                create a link to the corresponding header node i=3
                for each i-item-sets entry in the tree
                    do
                        call buildsubtree (F)
                        end
                Procedure buildsubtree (F)
                    If (first i-1 item-set = item-sets in their respective header nodes) then
                        create a link to the corresponding header node i=i+1
                        repeat buildsubtree (F)
                        end
                    end

4.1.11 Performance

To assess the relative performance of the algorithms for discovering large sets, we performed several experiments on an IBM RS/6000 530H workstation with a CPU clock rate of 33 MHz, 64 MB of main memory, and running AIX 3.2. The data resided in the AIXle system and was stored on a 2GB SCSI3.5" drive, with measured sequential throughput of about 2 MB/second. [11]

We give an overview of the AIS [12] and SETM [13] algorithms against which we compare the performance of the Apriori and Apriori Tid algorithms.

We then describe the synthetic datasets used in the performance evaluation and show the performance results. Finally, we describe how the best performance features of Apriori and Apriori Tid can be combined into an Apriori Hybrid algorithm and demonstrate its scale-up properties.

4.1.12 Problem
The problem is usually decomposed into two sub problems.

- One is to find those item sets whose occurrences exceed a predefined threshold in the database; those item sets are called frequent or large item sets.
- The second problem is to generate association rules from those large item sets with the constraints of minimal confidence.

4.1.13 Discussion

- In order to be able to continue with the hashing method, we need a perfect hash function \( h(e_1, e_2, \ldots, e_k) \), this hash function can be obtained by one of the following methods:

\[ h(e_1,e_2,\ldots,ek) = \text{prm}(1)^{e_1} + \text{prm}(2)^{e_2} + \ldots + \text{prm}(k)^{e_k} \]

Where \( \text{prm} \) is the set of prime numbers, \( \text{prm} = \{2, 3, 5, 7\ldots\} \) Although this hash function guarantee a unique key for every item-set, but it requires an irrational memory space, for example, consider an original item set \( X \) with only 10 items, and the following \( T \) hashed item set, \( T = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\} \), consider a 4-itemset \( (1, 2, 3, 10) \), this item set will be hashed to the value “282475385”, this will result in reserving large memory space without being used effectively.

- Other perfect hash functions, used in hashing strings, are not applicable here, because the input variables are limited to 26, which is the number of alphabets, while the number of items in a certain database can be very larger than this.

Association rule mining has a wide range of applicability such as market basket analysis, medical diagnosis/research, website navigation analysis, homeland security and so on. In this paper, we surveyed the list of existing association rule mining techniques and compare these algorithms with our modified approach. The conventional algorithm of association rules discovery proceeds in two and more steps but in our approach discovery of all frequent item will take the same steps but it will take the less time as compare to the conventional algorithm. We can conclude that in this new approach, we have the key ideas of reducing time. As we have proved above how the proposed Apriori algorithm take less time than that of classical Apriori algorithms. That is really going to be fruitful in saving the time in case of large database. This key idea is surely going to open a new gateway for the upcoming researcher to work in the filed of the data mining. [26]
4.2 FP-Growth Algorithm

One of the currently fastest and most popular algorithms for frequent item set mining is the FP-growth algorithm [27]. It is based on a prefix tree representation of the given database of transactions (called an FP-tree), which can save considerable amounts of memory for storing the transactions. Our thesis work is based on FP-Growth Algorithm a Suffix tree representation.

The basic idea of the FP-growth algorithm can be described as a recursive elimination scheme: in a preprocessing step delete all items from the transactions that are not frequent individually, i.e., do not appear in a user-specified minimum number of transactions. Then select all transactions that contain the least frequent item (least frequent among those that are frequent) and delete this item from them.

On return, remove the processed item also from the database of all transactions and start over, i.e., process the second frequent item etc. In these processing steps the prefix tree, which is enhanced by links between the branches, is exploited to quickly find the transactions containing a given item and also to remove this item from the transactions after it has been processed.

The FP-Growth Algorithm is an alternative way to find frequent item-sets without using candidate generations, thus improving performance. For so much it uses a divide-and-conquer strategy. The core of this method is the usage of a special data structure named frequent-pattern tree (FP-tree), which retains the item-set association information.

In simple words, this algorithm works as follows: first it compresses the input database creating an FP-tree instance to represent frequent items. After this first step it divides the compressed database into a set of conditional databases, each one associated with one frequent pattern. Finally, each such database is mined separately. Using this strategy, the FP-Growth reduces the search costs looking for short patterns recursively and then concatenating them in the long frequent patterns, offering good selectivity.

In large databases, it's not possible to hold the FP-tree in the main memory. A strategy to cope with this problem is to firstly partition the database into a set of smaller databases (called projected databases), and then construct an FP-tree from each of these smaller databases.
4.2.1 Preprocessing

Similar to several other algorithms for frequent item set mining, like, for example, Apriori, FP-growth pre-processes the transaction database as follows: in an initial scan the frequencies of the items (support of single element item-sets) are determined. All infrequent items—that is, all items that appear in fewer transactions than a user-specified minimum number—are discarded from the transactions, since, obviously, they can never be part of a frequent item set.

In addition, the items in each transaction are sorted, so that they are in descending order W.R.T. their frequency in the database. Although the algorithm does not depend on this specific order, experiments showed that it leads to much shorter execution times than a random order. An ascending order leads to a particularly slow operation in my experiments, performing even worse than a random order.

This pre-processing is demonstrated in Table 5-1, which shows an example transaction database on the left. The frequencies of the items in this database, sorted descendingly, are shown in the middle of this table. If we are given a user specified minimal support of 3 transactions, items f and g can be discarded. After doing so and sorting the items in each transaction descending W.R.T. their frequencies we obtain the reduced database shown in Table 5-1 on the right.
Of course, this is not the only way in which the initial FP-tree can be built. At first sight it may seem to be more natural to build it by inserting transaction after transaction into an initially empty FP-tree, creating the necessary nodes for each new transaction. Indeed, such an approach even has the advantage that the transaction database need not be loaded in a simple form (for instance, as a list of integer arrays) into main memory. Since only one transaction is processed at a time, only the FP-tree representation and one new transaction are in main memory. This usually saves space, because an FP-tree is often a much more compact representation of a transaction database.

4.2.2 Applications

- Market basket analysis,
- cross-marketing,
- catalog design,
- sale campaign analysis,
- Web log (click stream) analysis, and
- DNA sequence analysis.
4.2.3 FP-Growth Method: Construction of FP-Tree

- First, create the root of the tree, labeled with “null”.
- Scan the database D a second time. (First time we scanned it to create 1-itemset and then L).
- The items in each transaction are processed in L order (i.e. sorted order).
- A branch is created for each transaction with items having their support count separated by colon.
- Whenever the same node is encountered in another transaction, we just increment the support count of the common node or Prefix.
- To facilitate tree traversal, an item header table is built so that each item points to its occurrences in the tree via a chain of node-links.

Now, the problem of mining frequent patterns in database is transformed to that of mining the FP-Tree.

4.2.4 Mining the FP-Tree by Creating Conditional (sub) pattern bases

Steps:

- Start from each frequent length-1 pattern (as an initial suffix pattern).
- Construct its conditional pattern base which consists of the set of prefix paths in the FP-Tree co-occurring with suffix pattern.
- Then, Construct its conditional FP-Tree & perform mining on such a tree.
- The pattern growth is achieved by concatenation of the suffix pattern with the frequent patterns generated from a conditional FP-Tree.
- The union of all frequent patterns (generated by step 4) gives the required frequent item-set.
4.2.5 FP-Growth Method: An Example

- Consider a database, D, consisting of 9 transactions.

<table>
<thead>
<tr>
<th>TID</th>
<th>List of Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>T100</td>
<td>I1, I2, I5</td>
</tr>
<tr>
<td>T100</td>
<td>I2, I4</td>
</tr>
<tr>
<td>T100</td>
<td>I2, I3</td>
</tr>
<tr>
<td>T100</td>
<td>I1, I2, I4</td>
</tr>
<tr>
<td>T100</td>
<td>I1, I3</td>
</tr>
<tr>
<td>T100</td>
<td>I2, I3</td>
</tr>
<tr>
<td>T100</td>
<td>I1, I3</td>
</tr>
<tr>
<td>T100</td>
<td>I1, I2, I3, I5</td>
</tr>
<tr>
<td>T100</td>
<td>I1, I2, I3</td>
</tr>
</tbody>
</table>

Table 4-2: FP-Growth Example

- Consider the same previous example of a database, D consisting of 9 transactions.
- Suppose min. support count required is 2 (i.e. min_sup = 2/9 = 22 %)
- The first scan of database is same as Apriori, which derives the set of 1-itemsets & their support counts.
- The set of frequent items is sorted in the order of descending support count.
- The resulting set is denoted as L = \{I2:7, I1:6, I3:6, I4:2, I5:2\}
FP-Growth Method: Construction of FP-Tree

Figure 4-7: An FP-Tree that registers compressed, frequent pattern information [45]

<table>
<thead>
<tr>
<th>Item Id</th>
<th>Sup Count</th>
<th>Node-link</th>
</tr>
</thead>
<tbody>
<tr>
<td>I2</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>I1</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>I3</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>I4</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>I5</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

Table 4-3: Mining the FP-Tree by creating conditional (sub) pattern bases

<table>
<thead>
<tr>
<th>Item</th>
<th>Conditional pattern base</th>
<th>Conditional FP-Tree</th>
<th>Frequent pattern generated</th>
</tr>
</thead>
<tbody>
<tr>
<td>I5</td>
<td>{(I2 I1: 1), (I2 I1 I3: 1)}</td>
<td>&lt;I2: 2, I1: 2&gt;</td>
<td>I2 I5: 2, I1 I5: 2, I2 I1 I5: 2</td>
</tr>
<tr>
<td>I4</td>
<td>{(I2 I1: 1), (I2: 1)}</td>
<td>&lt;I2: 2&gt;</td>
<td>I2 I4: 2</td>
</tr>
<tr>
<td>I3</td>
<td>{(I2 I1: 1), (I2: 2), (I1: 2)}</td>
<td>&lt;I2: 4, I1: 2&gt;, &lt;I1: 2&gt;</td>
<td>I2 I3: 4, I1, I3: 2, I2 I1 I3: 2</td>
</tr>
<tr>
<td>I2</td>
<td>{(I2: 4)}</td>
<td>&lt;I2: 4&gt;</td>
<td>I2 I1: 4</td>
</tr>
</tbody>
</table>
4.2.6 Advantages of FP-Growth

- Only 2 pass over data-set.
- “Compresses” data-set.
- No candidate generation.
- Much faster than Apriori.

4.2.7 Disadvantages of FP-Growth

- FP-Tree may not fit in memory.
- FP-Tree is expensive to build.

4.2.8 Why Frequent Pattern Growth Fast

- Performance study shows
  - FP-growth is an order of magnitude faster than Apriori, and is also faster than tree-projection

- Reasoning
  - No candidate generation, no candidate test
  - Use compact data structure
  - Eliminate repeated database scan
  - Basic operation is counting and FP-tree building
4.3 Prefix Tree

In computer science, a trie, also called digital tree and sometimes radix tree or prefix tree (as they can be searched by prefixes), is an ordered tree data structure that is used to store a dynamic set or associative array where the keys are usually strings. Unlike a binary search tree, no node in the tree stores the key associated with that node; instead, its position in the tree defines the key with which it is associated. All the descendants of a node have a common prefix of the string associated with that node, and the root is associated with the empty string. Values are normally not associated with every node, only with leaves and some inner nodes that correspond to keys of interest.

The term trie comes from retrieval. This term was coined by Edward Fredkin. However, other authors pronounce it /ˈtreet/ "try", in an attempt to distinguish it verbally from "tree".[29][30]

In the example shown, keys are listed in the nodes and values below them. Each complete English word has an arbitrary integer value associated with it. A trie can be seen as a deterministic finite automaton without loops. Each finite language is generated by a trie automaton, and each trie can be compressed into a DAFSA.

It is not necessary for keys to be explicitly stored in nodes. (In the figure, words are shown only to illustrate how the trie works.)

Though tries are most commonly keyed by character strings, they don't need to be. The same algorithms can easily be adapted to serve similar functions of ordered lists of any construct, e.g., permutations on a list of digits or shapes. In particular, a bitwise trie is keyed on the individual bits making up a short, fixed size of bits such as an integer number or memory address.

4.4 Suffix Tree

In computer science, a suffix tree (also called PAT tree or, in an earlier form, position tree) is a compressed trie containing all the suffixes of the given text as their keys and positions in the text as their values. Suffix trees allow particularly fast implementations of many important string operations.
The construction of such a tree for the string $S$ takes time and space linear in the length of $S$. Once constructed, several operations can be performed quickly, for instance locating a substring in $S$, locating a substring if a certain number of mistakes are allowed, locating matches for a regular expression pattern etc. Suffix trees also provided one of the first linear-time solutions for the longest common substring problem. These speedups come at a cost: storing a string's suffix tree typically requires significantly more space than storing the string itself.

### 4.4.1 Definition

The suffix tree for the string $S$ of length $n$ is defined as a tree such that: [32]

- The tree has exactly $n$ leaves numbered from 1 to $n$.
- Except for the root, every internal node has at least two children.
- Each edge is labeled with a non-empty substring of $S$.
- No two edges starting out of a node can have string-labels beginning with the same character.
- The string obtained by concatenating all the string-labels found on the path from the root to leaf $i$ spells out suffix $S[i..n]$, for $i$ from 1 to $n$.

Since such a tree does not exist for all strings, $S$ is padded with a terminal symbol not seen in the string (usually denoted $\$$). This ensures that no suffix is a prefix of another, and that there will be $n$ leaf nodes, one for each of the $n$ suffixes of $S$. Since all internal non-root nodes are branching, there can be at most $n - 1$ such nodes, and $n + (n - 1) + 1 = 2n$ nodes in total ($n$ leaves, $n - 1$ internal non-root nodes, 1 root).

Suffix links are a key feature for older linear-time construction algorithms, although most new algorithms, which are based on Farach's algorithm, dispense with suffix links. In a complete suffix tree, all internal non-root nodes have a suffix link to another internal node. If the path from the root to a node spells the string $X\alpha$, where $X$ a single is character and $\alpha$ is a string (possibly empty), it has a suffix link to the internal node representing $\alpha$. See for example the suffix link from the node for ANA to the node for NA in the figure above. Suffix links are also used in some algorithms running on the tree.
4.4.2 Applications

Suffix trees can be used to solve a large number of string problems that occur in text-editing, free-text search, computational biology and other application areas. Primary applications include: [33]

- **String search**, in $O(m)$ complexity, where $m$ is the length of the sub-string (but with initial $O(n)$ time required to build the suffix tree for the string)
- Finding the longest repeated substring
- Finding the longest common substring
- Finding the longest palindrome in a string

Suffix trees are often used in bioinformatics applications, searching for patterns in DNA or protein sequences (which can be viewed as long strings of characters). The ability to search efficiently with mismatches might be considered their greatest strength. Suffix trees are also used in data compression; they can be used to find repeated data, and can be used for the sorting stage of the Burrows–Wheeler transform. Variants of the LZW compression schemes use suffix trees (LZSS). A suffix tree is also used in suffix tree clustering, a data clustering algorithm used in some search engines. [34]

4.4.3 Functionality

A suffix tree for a string $S$ of length $n$ can be built in $\Theta(n)$ time, if the letters come from an alphabet of integers in a polynomial range (in particular, this is true for constant-sized alphabets). [35] For larger alphabets, the running time is dominated by first sorting the letters to bring them into a range of size $O(n)$; in general, this takes $O(n \log n)$ time. The costs below are given under the assumption that the alphabet is constant.

Assume that a suffix tree has been built for the string $S$ of length $n$, or that a generalized suffix tree has been built for the set of strings $D = \{S_1, S_2, \ldots, S_K\}$ of total length $n = |n_1| + |n_2| + \cdots + |n_K|$. You can:

- Search for strings:
  - Check if a string $P$ of length $m$ is a substring in $O(m)$ time. [36]
Find the first occurrence of the patterns $P_1, \ldots, P_q$ of total length $m$ as substrings in $O(m)$ time.

Find all occurrences of the patterns $P_1, \ldots, P_q$ of total length $m$ as substrings in $O(m + z)$ time.

Search for a regular expression $P$ in time expected sub linear in $n$. [37]

Find for each suffix of a pattern $P$, the length of the longest match between a prefix of $P[i \cdots m]$ and a substring in $D_{in} \Theta(m)$ time. This is termed the matching statistics for $P$.

Find properties of the strings:

Find the longest common substrings of the string $S_i$ and $S_j$ in $\Theta(n_i + n_j)$ time.

Find all maximal pairs, maximal repeats or super maximal repeats in $\Theta(n + z)$ time.

Find the Lempel–Ziv decomposition in $\Theta(n)$ time.

Find the longest repeated substrings in $\Theta(n)$ time.

Find the most frequently occurring substrings of a minimum length in $\Theta(n)$ time.

Find the shortest strings from $\Sigma$ that do not occur in $D$, in $O(n + z)$ time, if there are $z$ such strings.

Find the shortest substrings occurring only once in $\Theta(n)$ time.

Find, for each $i$, the shortest substrings of $S_i$ not occurring elsewhere in $D_{in} \Theta(n)$ time.

The suffix tree can be prepared for constant time lowest common ancestor retrieval between nodes in $\Theta(n)$ time. [11] One can then also:

Find the longest common prefix between the suffixes $S_i[p..n_i]$ and $S_j[q..n_j]$ in $\Theta(1)$.
- Search for a pattern $P$ of length $m$ with at most $k$ mismatches in $O(kn + z)$ time, where $z$ is the number of hits.[13]

- Find all $z$ maximal palindromes in $\Theta(n)$, or $\Theta(n)$ time if gaps of length $g$ are allowed, or $\Theta(kn)$ if $k$ mismatches are allowed.[15]

- Find all $z$ tandem repeats in $O(n \log n + z)$, and $k$-mismatch tandem repeats in $O(kn \log(n/k) + z)$.

- Find the longest substrings common to at least $k$ string in $D$ for $k = 2, \ldots, K$ in $\Theta(n)$ time.

- Find the longest palindrome substring of a given string (using the suffix trees of both the string and its reverse) in linear time. [38]

### 4.5 Related Work

Mining of frequent itemsets is an important phase in association mining which discovers frequent itemsets in transactions database. It is the core in many tasks of data mining that try to find interesting patterns from datasets, such as association rules, episodes, classifier, clustering and correlation, etc [43]. Many algorithms are proposed to find frequent itemsets, but all of them can be catalogued into two classes: candidate generation or pattern growth.

Apriori [44] is a representative the candidate generation approach. It generates length ($k+1$) candidate itemsets based on length ($k$) frequent itemsets. The frequency of itemsets is defined by counting their occurrence in transactions. FP-growth, is proposed by Han in 2000, represents pattern growth approach, it used specific data structure (FP-tree). FP-growth discover the frequent itemsets by finding all frequent in 1-itemsets into condition pattern base , the condition pattern base is constructed efficiently based on the link of node structure that association with FP-tree. FP-growth does not generate candidate itemsets explicitly.
Chapter 5

The Proposed Algorithm

In this paper we proposed a frequent pattern mining algorithm using suffix automata, which is a modified algorithm of the Ukkonen’s Algorithm.

5.1 Suffix Tree

Suffix tree is a compressed trie containing all the suffixes of the given text as their keys and positions in the text as their values. Suffix tree allows a particularly fast implementation of many important string operations. It is more faster than prefix tree data structure. [39]

Example:

Suffix tree for the text BANANA

![Figure 5-1: Example of Suffix Tree](40)
5.2 Ukkonen's algorithm

In computer science, Ukkonen's algorithm is a linear-time, online algorithm for constructing suffix trees, proposed by Esko Ukkonen in 1995.

The algorithm begins with an implicit suffix tree containing the first character of the string. Then it steps through the string adding successive characters until the tree is complete. This order addition of characters gives Ukkonen's algorithm its "on-line" property. Earlier algorithms proceeded backward from the last character to the first one, let it be from the longest to the shortest suffix or from the shortest to the longest suffix.[3] The naive implementation for generating a suffix tree requires $O(n^2)$ or even $O(n^3)$ time, where $n$ is the length of the string. By exploiting a number of algorithmic techniques, Ukkonen reduced this to $O(n)$ (linear) time, for constant-size alphabets, and $O(n \log n)$ in general. [41]

5.3 Suffix automata

Suffix automata and factor automata are efficient data structures for representing the full index of a set of strings. They are minimal deterministic automata representing the set of all suffixes or substrings of a set of strings. The suffix automation of a string $u$ is the minimal deterministic finite automation recognizing the set of suffixes of $u$. Its size is linear in the length of $u$, $n$.

![Figure 5-2: (a) A deterministic finite automation $A$ and (b) a deterministic automation recognizing $\Sigma^*L(A)$ where transitions labeled with $\phi$ are failure transitions.]

More precisely, its number of states between $n & 2n-1$ and its number of transitions between $n+1 & 3n-2$. This automation can be obtained by minimizing the suffix trie of $u$. A crucial advantage of suffix automata is that, unlike suffix trees they do not require the use of compact transitions for the size to be linear to $|u|$. [42]
5.4 Algorithm

- Create prefix paths for a particular suffix node. This is done by gathering all the paths containing a particular suffix node. Any path that ends with this suffix is examined.

- Using the prefix path tree determine whether the suffix is frequent. This is done by adding the support counts associated with the node and if the number is greater than or equal to the min_sup (minimum support) the node is frequent. If the node isn’t frequent the analysis ends for this suffix.

- Convert the prefix paths into a conditional FP-tree.
  
  I. Update the support counts along the prefix paths to reflect the actual number of transactions containing the item-set
  
  II. Truncate the prefix paths by removing the nodes of the chosen suffix
  
  III. Remove items that may no longer be frequent (if the support count of a particular node is less than min_sup it is no longer frequent and should be pruned).

  IV. Repeat I →III for all prefix paths for the chosen suffix.

- Repeat Steps 1-3 for all suffix nodes to determine the frequent item-set for the dataset.

5.5 Procedure

Step 1: Given word sequence to be searched is tokenized first.
Step 2: Initialized n to 1
  
  Search the nodes of root for first token
  
  If there is a match
    
    Performs step3
  
  Else
    
    If n is last node of root+1
      
      Return search failed
    
    Else
      
      Increment n and again search with next node of root

Step 3: Consider only the level 1 nth node sub tree
Compare next token with level 2 first node
Do{
    If there is a match
        I. Perform depth first search traversal on the tree comparing with remaining tokens
        II. Applying k- mismatch retrieve the documents that contain the word sequence
    Else
        If all the level 1 nth node sub tree nodes are traversed word sequence is not present
        Else
            Apply k- mismatch, perform DFS traversal and compare with the next token
}
While all the tokens of word are not completed or entire level 1 nth node sub tree nodes are not traversed.
Example:

Let, Minimum Support 2

T1: CDAB
T2: CDB
T3: CDA

Figure 5-3: Contraction of Build Tree
Sub-tree of B

Figure 5-4: Frequent Pattern is B(2).

Sub-tree of A

Figure 5-5: Frequent Pattern is A(2).

Sub-tree of D

Figure 5-6: Frequent Pattern is D(3), DA(2), DB(2).
Figure 5-7: Frequent Pattern is C(3), CD(3), CB(2), CA(2), CDA(2).
Chapter 6

Performances Analysis

This chapter presents results and performance comparison with this algorithm and some previous algorithms. Before the performance analysis complexity analysis will be discussed briefly.

6.1 Complexity Analysis

As described before, we see that naive implementation for generating a suffix tree or a prefix tree requires $O(n^2)$ time, where $n$ is the length of the string. But we use Ukkonen’s algorithms techniques which reduce the time complexity to $O(n)$ time.

<table>
<thead>
<tr>
<th>Data structure</th>
<th>Time complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Suffix (naïve)</td>
<td>$O(n^2)$</td>
</tr>
<tr>
<td>Prefix</td>
<td>$O(n^2)$</td>
</tr>
<tr>
<td>Suffix(Ukkonen)</td>
<td>$O(n)$</td>
</tr>
</tbody>
</table>

Table 6-1: Complexity Analysis

6.2 Environments of Experiments

All the experiments are performed on a Intel ® Core™ Duo CPU 2.93GHz PC machine with 2GB main memory, running on Microsoft Windows 7. All the programs are written in Java. Notice that we do not directly compare our absolute number of runtime with those in some published reports running on the RISC workstations because different machine architectures may differ greatly on the absolute runtime for the same algorithms. Instead, we implement their algorithms to the best of our knowledge based on the published reports on the same machine and compare in the same running environment. Please also note that run time used here means the total execution time, that is, the period between input and output, instead of CPU time measured in the experiments in some literature. We feel that run time is a more comprehensive measure since it takes the total running time consumed as the measure of cost, whereas CPU time considers only the cost of the CPU resource. The experiments are pursued on both synthetic and real data sets. The synthetic data sets which we used for our experiments
were generated using the procedure described in Agrawal and Srikant (1994). We refer readers to it for more details on the generation of data sets.

We report experimental results on two synthetic data sets. First one is T10I4D100K. In this data set, the size of data is 3.83MB which contains 100000 transactions and 870 items. Second one is Mushroom. In this data set, the size of the data is 0.83MB which contains 8124 transactions and 119 items.

In the first experiment on T10I4D100K, time required to construct Proposed Algorithm, Apriori Algorithm, CP-Tree and FP-Growth is compared. These experiments have been carried out on above datasets whose results are shown in Figure:6-1 respectively. We saw that our proposed algorithm has less construction time and more efficient than Apriori, FP-Growth and CP-Tree.
In the second experiment on Mushroom, time required to construct Proposed Algorithm, Apriori Algorithm, CP-Tree and FP-Growth is compared. These experiments have been carried out on above datasets whose results are shown in Figure:6-2 respectively. We saw that our proposed algorithm has less construction time and more efficient than Apriori, FP-Growth and CP-Tree.
Figure 6-2: Comparison between Apriori, FP-Growth, CP-Tree & Proposed Algorithm on Mushroom Data Set.
Chapter 7

Conclusion

Mining frequent item-sets for the association rule mining from the large transactional database is a very crucial task. There are many approaches that have been discussed; nearly all of the previous studies were using Apriori approach and FP-Tree approach for extracting the frequent item-sets, which have scope for improvement. Thus the goal of this research was to find a scheme for pulling the rules out of the transactional data sets considering the time and the memory consumption. This chapter summarizes the work done in this thesis and then the future scope is given.

In this thesis, we considered the following factors for creating our new scheme, which are the time and the memory consumption, these factors are affected by the approach for finding the frequent item-sets. Work has been done to develop an algorithm which is an improvement over Apriori and FP-tree with using an approach of improved Apriori and FP-Tree algorithm for a transactional database. According to our observations, the performances of the algorithms are strongly depends on the support levels and the features of the data sets (the nature and the size of the data sets). Therefore we employed it in our scheme to guarantee the time saving and the memory in the case of sparse and dense data sets. It is found that for a transactional database where many transaction items are repeated many times as a super set in that type of database maximal Apriori (improvement over classical Apriori) is best suited for mining frequent item-sets. The item-sets which are not included in maximal super set are treated by FP-tree for finding the remaining frequent item-sets. Thus this algorithm produces frequent item-sets completely. This approach doesn’t produce candidate item-sets and building FP-tree only for pruned database that fit into main memory easily. Thus it saves much time and space and considered as an efficient method as proved from the results. For both data sets the running time and memory consumption of our new scheme outperformed Apriori. Whereas the running time of our scheme performed well over the FP-growth on the collected data set at the lower support level where probability of finding maximal frequent item-sets is large and at higher lever running time is approximately same as the FP-Tree. The memory consumption is also approximately same as the FP-Tree at higher support and performed well at lower support.
**The main contributions of this Thesis:**

We can summarize the main contribution of this research as follows:

- To study and analyze various existing approaches to mine frequent item-sets.
- To devised a new better scheme than classical Apriori and FP-tree alone using maximal Apriori and FP-tree as combined approach for mining frequent item-sets.

### 7.1 Future Trends

There are a number of future work directions based on the work presented in this thesis.

- Using constraints can further reduce the size of item-sets generated and improve mining efficiency.

- This scheme was applied in retailer industry application, trying other industry is an interesting field for future work.

- This scheme use Maximal Apriori and FP-Tree. We can use other combination to improve this approach.
References

Wesley Publishers”. 2006


[19] Data mining Concepts and Techniques” by By Jiawei Han, Micheline Kamber, Morgan Kaufmann Publishers, 2006


[27] J. Han, H. Pei, and Y. Yin. Mining Frequent Patterns without Candidate Generation. In: Proc. Conf. on the Management of Data (SIGMOD’00, Dallas, TX). ACM Press, New York, NY, USA 2000


[29] Franklin Mark Liang (1983). *Word Hy-phen-a-tion By Com-put-er* (Doctor of


[34] First introduced by Zamir & Etzioni (1998).

[35] Barsky, Marina; Stege, Ulrike; Thomo, Alex; Upton, Chris (2009), "Suffix trees for very large genomic sequences", CIKM '09: Proceedings of the 18th ACM Conference on Information and Knowledge Management, New York, NY, USA: ACM


[45] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. SIGMOD' 00.