Incremental Frequent Pattern Mining

Abstract

Data Mining (DM) is a process for extracting interesting patterns from large volumes of data. It is one of the crucial steps in Knowledge Discovery in Databases (KDD). It involves various data mining methods which mainly falls into predictive and descriptive models. Descriptive models look for patterns, rules, relationships and associations within data. One of the descriptive methods is Association rule analysis which represents co-occurrence of items or events. Association rules are commonly used in market basket analysis. An association rule is in the form of $X \rightarrow Y$ and it shows that $X$ and $Y$ co-occur with a given level of support.

Having petabytes of data finding its way into data storages in perhaps every minute –if not seconds, made everyone look for efficient methods for analyzing these large datasets. Among others, Association rule mining is a common technique used in discovering interesting frequent patterns in data acquired in various application domains. Many algorithms have been proposed for searching for frequent patterns. The search space combinatorically explodes as the size of source data increases. Incremental algorithms further improve the efficiency of frequent pattern mining. Simply using more powerful computers, or even super-computers to handle the large data sets is not sufficient. However, commodity hardware and a distributed data storage environment allows parallel computations using Map Reduce framework in discovering frequent patterns in large datasets and it is also scalable.

In this proposal we will explore and devise incremental algorithms with parallel computation using Map Reduce as well as other Hadoop Eco-System tools (such as Spark). We also plan to use the devised algorithms in exploring emerging patterns in data accumulated over periods of time in increments.
<table>
<thead>
<tr>
<th>Contents</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chapter 1</td>
<td>4</td>
</tr>
<tr>
<td>Introduction</td>
<td>4</td>
</tr>
<tr>
<td>Why Data Mining?</td>
<td>5</td>
</tr>
<tr>
<td>How data is collected</td>
<td>6</td>
</tr>
<tr>
<td>Data Mining (DM) (definition)</td>
<td>7</td>
</tr>
<tr>
<td>What is not considered Data Mining?</td>
<td>10</td>
</tr>
<tr>
<td>Data Mining Models (types) and Tasks</td>
<td>10</td>
</tr>
<tr>
<td>Descriptive models</td>
<td>11</td>
</tr>
<tr>
<td>Predictive models</td>
<td>12</td>
</tr>
<tr>
<td>Other Models</td>
<td>13</td>
</tr>
<tr>
<td>Outlier analysis</td>
<td>14</td>
</tr>
<tr>
<td>KDD Process</td>
<td>14</td>
</tr>
<tr>
<td>Step 1 Data</td>
<td>15</td>
</tr>
<tr>
<td>Step 2 Pre-processing</td>
<td>15</td>
</tr>
<tr>
<td>Step 3 Integrated data</td>
<td>16</td>
</tr>
<tr>
<td>Step 4 Data mining step</td>
<td>16</td>
</tr>
<tr>
<td>Step 5 Models and patterns</td>
<td>16</td>
</tr>
<tr>
<td>Step 6 Analysis, interpretation and evaluation</td>
<td>16</td>
</tr>
<tr>
<td>Step 7 Knowledge</td>
<td>16</td>
</tr>
<tr>
<td>What are the challenges of Data Mining?</td>
<td>17</td>
</tr>
<tr>
<td>Chapter 2</td>
<td>18</td>
</tr>
<tr>
<td>Frequent Pattern Mining and Association Rule Mining (FPM-ARM)</td>
<td>18</td>
</tr>
<tr>
<td>Frequent Pattern Mining –FPM (Definition)</td>
<td>18</td>
</tr>
<tr>
<td>Association Rule Mining - ARM (Definitions)</td>
<td>18</td>
</tr>
<tr>
<td>FPM-ARM Application Areas</td>
<td>18</td>
</tr>
<tr>
<td>Market Basket Analysis</td>
<td>19</td>
</tr>
<tr>
<td>FPM Methods</td>
<td>19</td>
</tr>
<tr>
<td>Basic Concepts and Terminology</td>
<td>20</td>
</tr>
<tr>
<td>Why We Use Support and Confidence:</td>
<td>22</td>
</tr>
<tr>
<td>How many frequent patterns?</td>
<td>22</td>
</tr>
<tr>
<td>Methods for finding association rules</td>
<td>24</td>
</tr>
</tbody>
</table>
Frequent Pattern Mining (FPM) Algorithms ................................................................. 26
   Major Algorithms ........................................................................................................ 26
   What are the issues with Apriori? ................................................................. 34
   Extensions of Apriori ................................................................................................. 34
   Eclat (Zaki M. J., 1997) ....................................................................................... 36
   Maximal Itemset Mining (Max-patterns): ................................................... 38
   MAFIA ....................................................................................................................... 38
   Closed Itemsets: ........................................................................................................ 39
   FP-growth .................................................................................................................. 40
   Other algorithms and improvements .............................................................. 42
Chapter 3 ..................................................................................................................... 43
Sequential Pattern Mining ....................................................................................... 43
   Problem .................................................................................................................... 43
   Terms ......................................................................................................................... 43
   Example ..................................................................................................................... 44
   Algorithms: .............................................................................................................. 47
   Vertical based approach ....................................................................................... 47
   Pattern Growth based ............................................................................................. 48
   Others ......................................................................................................................... 48
Chapter 4 ..................................................................................................................... 49
Incremental Algorithms .......................................................................................... 49
   Apriori-based incremental algorithms .......................................................... 51
   Tree based algorithms: .......................................................................................... 59
   Summary .................................................................................................................. 61
Chapter 5 ..................................................................................................................... 63
Parallel Techniques, Parallel Algorithms and Issues ............................................ 63
   Parallel Techniques ................................................................................................. 63
   Parallel Algorithms ................................................................................................. 68
   Earlier Approaches: ............................................................................................... 68
   New approaches– A Spark Approach: ............................................................ 71
   A MapReduce -Based Approach ......................................................................... 73
   Issues in Parallel programming Environment: ............................................. 78
Chapter 1

Introduction
Why Data Mining?
Since capturing, transferring and storing data has become both technologically and economically feasible, data has been pouring into storage systems and has been accumulating in the last decade at faster rates than ever. Existing data is more than doubling every two years. The data growth rate has moved from terabytes to petabytes.

While stored data is huge and still growing exponentially, very few effective tools were created to analyze and find useful information in this data. Businesses, such as supermarkets, routinely collect data through loyalty cards, and always want to know the purchasing behavior of their customers in order to market their products more efficiently and effectively. Nowadays, the competitive pressure is very high in some industries. For companies in these industries, it is no longer optional to use data mining techniques since many competitors have been using them already. Therefore, to stay competitive, all of the companies within the same market segment have to take a look into advances in these technologies and adapt them and use them accordingly.

Businesses now use every means to provide better and more customized services and cutting edge technologies to their customers or clients. Data mining is one of the major methods to support and customize these services and marketing strategies. The ones which cannot adapt or cannot take advantage of ever evolving technologies, can lose their customer base very quickly to their competitors.

Nowadays, customers in general are very sophisticated. They know what a good product is, what its fair price is, what brands, companies, services and products are good or not. All of this information is readily available to anyone who has an access to a computer and internet and have the basic skills of using one of the search engines. Hence, to satisfy their needs a simple query such as the ones created with SQL or google search –which simply gets the subset of stored facts and information from databases, is usually not valuable enough for many users, because they can get this information themselves. What would be valuable for many of these same people nowadays is to access to not obvious, hidden information which typically resides
inside of mass amount of raw data. Some companies analyze all of the stored transactions and the browsing history of their customers in their databases, analyze the relationship, associations and correlations among data, interpret, deduce new information from these and use it as a service for their customers.

Amazon.com, for example, uses recommender systems (aka recommendation system), which uses data mining tools and methods as well as machine learning techniques, to explore, associate and extract data from very large datasets and recommends similar products to its customers. The Amazon application looks at the history, current clicks, and site navigation of a customer, and then recommends customers with similar products that other customers looked at or bought. The recommendation system and product suggestions are supported by the ratings given by their customers. Hence, seeing what others bought and how happy they are with their purchases can make a customer comfortable with making a similar purchase very quickly.

Data mining can explore data and extract valuable information which can be potentially very useful for businesses. It has been used to help increase or decrease inventories and cut business costs in retail businesses as well as used in government data analysis, law enforcement agencies in their handling and dealing with crime.

This survey is about Data Mining: why we mine or need to mine data, how and where it accumulates, why traditional approaches are not sufficient, what the issues and challenges there are to effectively extract useful information from large amounts of data.

**How data is collected**
Data used in data mining comes from many sources and in many different forms or shapes – such as tables from databases, simple text from documents, spreadsheets, graphs, charts, web documents, audio or video from multi-media files, XML documents and time dimensional data from temporal data files. Data can be generated by industry or government, within scientific or
social domains, during online and offline shopping or transactions, or possibly by using barcode scanners. Data can be collected or generated by many automated or non-automated tools or techniques. Many automatic data capturing tools and technologies, such as radio frequency Identification devices (RFID), barcode scanners, smart phones, GPS devices, optical character recognition (OCR), remote sensor devices, surveillance cameras, voice recognition systems, biometrics measurements, and others, identify and capture data.

Fast data transfer rates make it possible for this captured data to be transferred, securely stored, and accessed from a remote location in real time. All of these, and other automated data collection tools, and the integration of computers in every part of society such as in social networks, digital media, recommendation systems and scientific studies have contributed to the data explosion. In addition, scientific research, experiments and simulations, as well as universally accessible data via the internet, including both the download and upload functionality of many internet sites, such as youtube.com, message boards, forums, blogs, social networks, business transactions and trading data has created a lot of activity and interaction among people, government, computer systems and networks. These generate huge amounts of data in terms of e-commerce transactions, government transactions as well as social network interactions and other forms of data. Furthermore, the price decrease of computers, storage and network systems and increase in the power of the computers as well as innovations in technology and networks caused the availability of computers in all walks of life and became another catalyst for data increase and accumulation from every part of the society. (M-S. Chen, 1996) (H.Dunham, 2002) (Tan, Introduction to Data Mining, 2005).

**Data Mining (DM) (definition)**

DM is one of the steps in the process known as Knowledge Discovery in Databases (KDD). It is a step by step process for extracting, discovering and analyzing previously unknown, non-intuitive, implicit, hidden, interesting, non-trivial, potentially useful information, correlations, patterns or relationships from different or non-obvious perspectives. It can use semi-automatic or automatic means from a collection of typically static, very large data sets, databases or data
warehouses and summarizing or generalizing (aggregations) into useful information by combining techniques from Artificial Intelligence such as Neural networks and Machine Learning, Statistics, database management, information retrieval, algorithms, high performance computing (HPC), user applications, visualizations, and others. Some other terms for data mining are: “data dredging, knowledge discovery (KD), knowledge extraction, knowledge mining, exploratory data analysis, data archeology, data driven discovery, information harvesting, unsupervised pattern recognition, data fishing, data snooping, and business intelligence”. (M-S. Chen, 1996) (H.Dunham, 2002) (Tan, Introduction to Data Mining, 2005) (J. Han, 2005) (G.Brookshear, 2009)

Figure 1: The techniques that data mining uses. Modified and adapted from (Tan, Introduction to Data Mining, 2005) (J. Han, 2005).

DM is an important tool in several fields including retail, marketing, risk analysis, stock market and investment analysis, fraud analysis and detection, surveillance, credit risk analysis, inventory management, weather forecasting, bio-informatics, scientific discovery and in many others. (J. Han, 2005) (G.Brookshear, 2009).
DM can be classified depending on the source of data (databases, flat files, transactions, web navigation patterns); the type of databases (relational databases, OO databases, temporal databases, spatial databases, multimedia databases); the type of information to be extracted (trends, patterns, deviations, associations, relationships, clustering information, classification features, characteristics, deviation analysis, pattern recognition, data summarization and generalization, data characterization); the type of DM techniques (generalization based, statistical based, math based, pattern based and combination of two or more of these techniques); the type of data miners (interactive miners, data oriented miners, query oriented miners, autonomous miners); and the kind of DM approaches (database approach, Statistical approach, machine learning approach) (M-S. Chen, 1996).

**Database querying vs. Data Mining**

<table>
<thead>
<tr>
<th></th>
<th>Database querying Perspective</th>
<th>Data Mining Perspective</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data</td>
<td>Data is in tables</td>
<td>Data is in any medium, flat files, database tables, in charts, graphs.</td>
</tr>
<tr>
<td>Data currency</td>
<td>Current or operational data, as is, original, currently used in the production environment.</td>
<td>Static, cleaned, processed, modified and integrated.</td>
</tr>
<tr>
<td>Query</td>
<td>Well defined queries, such as find all students whose GPA &gt; 3.0.</td>
<td>Not well defined, data miner is not necessarily sure what exactly to find. For example a query like this: find those successful students who have similar studying habits (clustering). Find items which are frequently bought with other items. (Association</td>
</tr>
</tbody>
</table>
rules). Find all students who will likely to fail in their computer science courses who are non-majors. (classification)

| Output          | Exact, clear output. Subset of database. | Not clear, hidden but potentially useful information. It is not merely just a subset of data. It is probably a result of analysis of data. |

Table 1: Database processing vs. Data Mining processing. Adapted from (H.Dunham, 2002).

**What is not considered Data Mining?**
Traditional approaches to data inquiry or interrogation or a simple query or search are not considered data mining. These include methods such as writing SQL queries to extract information from database tables, searching a term in Google or any other search engine, looking into a dictionary, an encyclopedia, or looking in a book, magazine or library for information. These are generally not considered data mining because they merely get a subset of stored data or facts. Data mining on the other hand does exhaustive, iterative search to analyze data and extract useful information and patterns? (Tan, Introduction to Data Mining, 2005) (G.Brookshear, 2009). The following table shows how data mining differs from database querying.

**Data Mining Models (types) and Tasks** (H.Dunham, 2002) (G.Brookshear, 2009)
Data mining methods and models can be broken into three broad categories:
As we can see in Figure 2, we can think of the various data mining methods as being separated into predictive, descriptive, and other models. We discuss these in detail below.

**Descriptive models**
look for patterns, rules, relationships and associations within data. Existing data is analyzed looking for relationships. Rules or decisions based on these analyses are made. Clustering, summarization, association rules, sequence discovery, deviation detection (outlier analysis), database segmentation, dependency modeling algorithms fall into this category.

*Clustering*, also referred as segmentation, partitioning or unsupervised learning where classes are not known in advance, groups data based on a distance measure. It is another form of DM which searches and looks to identify classes (H.Dunham, 2002) (G.Brooksheer, 2009). Clusters are not pre-defined so a domain expert might be needed
to identify and group the data into clusters. (H.Dunham, 2002). The goal is to maximize the similarity among the elements within a class and minimize the similarity among the elements from different classes.

**Summarization**, sometimes referred to as characterization or generalization, extracts or describes the data among some commonality which represents the whole.

**Association rules**, also called link analysis or affinity analysis, looks into relationships, links, co-occurrences between data groups (itemsets). (H.Dunham, 2002) (G.Brookshear, 2009). It seeks to answer questions such as, what itemsets are frequently grouped together, and how these patterns can be found efficiently if the dataset is very large.

**Sequence Analysis**, Sequential pattern analysis looks to identify patterns of occurrences over time. (G.Brookshear, 2009). Thus, relationships among data are based on time and are organized in some order or sequence. For example, first a person buys a computer then, buys RAM or first a person buys a house then buys furniture. (J. Han, 2005)

**Predictive models** make a prediction about future based on the historical data. Classification, regression, inductive logic, neural networks, decision trees, time series analysis, prediction algorithms fall into this category.

**Classification**

Classification is also known as supervised learning because classes are predefined, classification groups data into predefined segments or classes. Two common forms are class description and class discrimination. Class description looks for properties to identify groups, while class discrimination looks for features to separate two groups from each other. (H.Dunham, 2002) (G.Brookshear, 2009). Typical applications include credit worthiness determination, marketing based on income level, education level, or cars according to their efficiency or gas usage per mile etc. (J. Han, 2005). Pattern recognition “is a type of classification where an input pattern is classified into one of several classes based on its similarity to these predefined classes.” (Tan, Introduction to Data Mining, 2005).
**Time series analysis**

This method captures and examines data within some predetermined time interval or period such that the data is evaluated over a longer period of time.

**Prediction**

Current and past data and results are used to predict future, unknown or missing data or results. Typical applications include credit assessment, targeted marketing, medical analysis and others. Regression is a major method of prediction. It determines a function that will be able to predict how to get to the target data items or results.

**Other Models**

*Graph mining* is a method for finding associations among nodes or sub-graphs. It looks for similarities or frequent sub graphs. Both web mining and network analysis or mining are type of graph mining:

*Web mining* is a type of graph mining that is a field of data mining which looks for patterns in the web including page ranks, how often a page is visited, who are the frequent people or visitors, how long they use the pages, what is their opinion (opinion mining). Web mining can be divided further into three groups: “Web usage mining, Web content mining and Web structure mining.” (Tan, Introduction to Data Mining, 2005) (J. Han, 2005)

*Network analysis* is a type of graph mining where algorithms look for computer networks to find patterns and interconnections among various connection capable machines. For example, a person might belong to a single (homogenous) network or many networks (heterogeneous) networks, such as alumni network, friends, family network, a forum, message board, blog or colleague network. (J. Han, 2005). Interconnections among computers and servers for these activities reveal this type of information.

*Spatial Mining* is a data mining field which searches for patterns, similar objects, within a space, location, maps, and other distance related data. This spatial data is stored in
spatial databases. For example, finding all the subway sandwich stores in a particular zip code area would fall into spatial data mining.

Outlier analysis is about finding or capturing a data item or itemset which is not conforming to the rest of the data.

KDD Process
KDD is “The nontrivial process of identifying valid, novel, potentially useful and ultimately understandable patterns in data.” (U. Fayyad, 1996)
Figure 3: KDD process. Adapted from (M-S. Chen, 1996) (H.Dunham, 2002) (U. Fayyad, 1996).

The arrows to the previous steps show that we may need to go back iteratively.

A step by step diagram of the KDD process is presented in Figure 3. There are seven basic steps in this process.

**Step 1 Data:** Data comes from many different sources, including Web or paper files or documents, electronic or non-electronic sources, legacy systems, different schemas, different DBMS systems or database models, flat files, different storage mediums, and data structures, to name a few. Data is heterogeneous and inconsistent. Data mining deals with huge datasets, sometimes terabytes and petabytes large. For smaller datasets we can use statistical models.

**Step 2 Pre-processing:** In this step, only the relevant, non-duplicate data or features of the data are selected. Data is cleaned and noise is removed or fixed. The data is transformed into a common format or standard and static state. In the selection process either the entire dataset or a sample of the dataset is selected for processing. This may require writing conversion programs or using ETL software to map the old data to a new, standard format. At this stage a data warehouse can also be constructed where all data selected from different departments or functions (such as marketing, sales, accounting) can be consolidated, integrated, transformed and stored.
**Step 3 Integrated data**: After the pre-processing step, data is consolidated, transformed and stored in its static state. Data is typically stored in a data warehouse or transformed directly into standard forms or data structures that are usable for data mining operations. It may be stored into smaller components called Data Marts according to their particular functional areas. The integrated and consolidated data are the inputs for the Data Mining algorithms. Alternatively the data could be split so that results could be analyzed from one part and validated (cross validation) on the another part of the data.

**Step 4 Data mining step**: Here, intelligent pattern recognition, or data intensive algorithms, techniques, methods, tools or programs are used to extract useful patterns, associations, correlations, relationships or information from the consolidated and integrated data. The data mining models used depends on the desired task and the goal to be accomplished.

**Step 5 Models and patterns**: The result or output of the data mining process is not merely a subset of the data but models and patterns. These are extracted by going over data iteratively and analytically many times. Still, the results need to be further evaluated, interpreted and analyzed. It may require a domain expert to analyze the results.

**Step 6 Analysis, interpretation and evaluation**: This is post processing step. In this step the results or patterns will be evaluated, interpreted and selected. Many patterns, associations or correlations can be found and it would be impossible and also meaningless to keep and try to use them all. Therefore, all of the uninteresting, insignificant patterns or results which may occur by chance are eliminated. Only the significant, interesting results, pattern or rules identified are kept.

**Step 7 Knowledge**: In this step, patterns, deviations, associations, relationships, outlier information, clustered, classified data, characteristics, trends or any other information that was discovered or extracted is presented to the user or domain expert. This presentation needs to be in a way that makes sense for the user. In this step, visual aids, and other GUI types of presentations could be used. (J. Han, 2005)
What are the challenges of Data Mining?
The increasing size, format, complexity and dimensions of data pose problems for data mining. The quantity of data being produced is large and dramatically increasing every day. It is heterogeneous, including text and multimedia data such as images, audio, and video. Data comes from many different sources and in many different formats using different compression and encryption methods. It may have thousands of dimensions which increases its complexity. It may be incomplete, inconsistent, ununiformed (noisy). It may require field knowledge to explore and understand data. It may be sensitive data which may require privacy preserving data mining techniques. Data may also be increasing periodically, which may require incremental data mining techniques. (J. Han, 2005).

Security and legal issues are other challenges. Many people as well as governments, are more aware of the privacy and security of the data and the requirements needed to protect it, Therefore, security of the data, and privacy of the individual needs to be ensured while collecting and processing data.

Another challenge presents itself in the long running time of the data mining algorithms. Since the amount of data is large, new, scalable and efficient algorithms need to be developed to ensure that the mined data is still relevant, running time is practical and the overall results are meaningful. (M-S. Chen, 1996)
Chapter 2

Frequent Pattern Mining and Association Rule Mining (FPM-ARM)

Association rule mining is typically decoupled into two separate processes: frequent pattern mining and association rule mining. The following are some of the brief statements or definitions about each term:

**Frequent Pattern Mining – FPM (Definition)**
- Frequent pattern is a set of items, subsequences, substructures that occurs frequently in a database.
- Frequent pattern mining looks at the historical data and gives the probability of similar occurrences in the future.
- It is a method to find frequent patterns in a dataset.
- Frequent pattern mining is about finding associations, correlations or frequently occurring patterns.
  
  (J. Han, 2005), (R. Agrawal, 1993).

**Association Rule Mining - ARM (Definitions)**
- “Given a set of transactions, ARM finding the rules that will predict the occurrence of an item based on the occurrences of other items in the transaction.”
- ARM tries to learn what items to be grouped together.
- It can tell us what sets of products (or itemsets) are frequently bought together.
- It is a probabilistic implication or co-occurrence, (not cause-effect relationship), between products. Given X, what is the probability of occurrence of Y. If then Else relationship.
  
  (Tan, Introduction to Data Mining, 2005)

**FPM-ARM Application Areas** Some of the important application areas of FPM-ARM include market basket analysis, sale campaign analysis, promotions, product placement, cross marketing, catalog design, recommendation systems, click stream analysis (web log), web page
or web link analysis (such as co-occurrence of keywords on webpages), medical data analysis (for example, what symptoms are associated with a particular disease), genome analysis, drug design, DNA sequence analysis and others. (J. Han, 2005) (C.Borgelt) (Saleeb, 2005)

**Market Basket Analysis:** FPM-ARM originally appeared in terms of a method of market basket analysis. In market basket analysis, FPM-ARM searches and finds purchasing habits or behavior of customers and accordingly applies this information to its marketing strategies to increase sales and reduce costs through targeted marketing to be more effective, efficient and competitive. For example, using this information a retailer can decide what products should be or not to be put on sale, whether the products should be bundled together, where should the products be placed in the store (product shelf layout), whom to send coupons and advertisements. For example, if there is a high correlation between buying coffee and milk, then, putting both of the items on sale at the same time, from a business point of view, wouldn’t be a good idea because there is high probability that the customers who bought one item will buy the other one too. Putting both of the items on sale would mean loss of revenue. It makes more sense to put only one item on sale and sell the other item on a regular price. (J. Han, 2005).

In Market Basket Transaction data we are interested in is finding out whether an item exists in a transaction or not – i.e. presence or absence of an item in a transaction. While quantity, price and other information could also be valuable and are different dimensions of data and transactions, they are considered different problems and in this study they are not covered.

**FPM Methods:** FPM methods include frequent itemset mining, frequent sequence mining, frequent tree mining and frequent graph mining.
Basic Concepts and Terminology

Given the following dataset, let \( I = \{i_1, \ldots, i_d\} \) be a set of items in the dataset. Each set of items forms one transaction.

<table>
<thead>
<tr>
<th>TID</th>
<th>Items purchased</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>1, 3, 5</td>
</tr>
<tr>
<td>002</td>
<td>2, 3, 5</td>
</tr>
<tr>
<td>003</td>
<td>1, 4, 5</td>
</tr>
<tr>
<td>004</td>
<td>1, 3</td>
</tr>
<tr>
<td>005</td>
<td>2, 3, 4, 5</td>
</tr>
</tbody>
</table>

Table 2: An example database

**Itemset**: one or more items in a set which are a subset of the items in \( I \). For example, \( \{2\} \) or \( \{3, 4, 5\} \) are both itemsets.

**k-itemset**: If an itemset has \( k \) items in it, we call it a \( k \)-itemset. For example itemset \( i = \{2\} \) is a \( 1 \)-itemset itemset \( \{\text{Apple, Orange, Milk}\} \) is a \( 3 \)-itemset and so on.

**Support count** (\( \sigma(X) \)): it is the number of transactions which contain an itemset. This is called absolute support. For the above transactions support count is as on the left:

<table>
<thead>
<tr>
<th>Itemset</th>
<th>Support Count</th>
<th>Support %</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>60</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>40</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>80</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>40</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>80</td>
</tr>
</tbody>
</table>

**Support** (or relative support) denoted as \( \sigma \) (itemset), is the number of occurrence of an itemset divided by the number of transactions in the database. For example 1-itemsets \( \{2\} \) and \( \{4\} \) has support of 40% while itemset \( \{3\} \) and \( \{5\} \) has support of 80%.
Table 3: Support count and relative support (support) of 1-itemsets

**Confidence**: Confidence is the conditional probability $c$, such that a given transaction which contains $X$, will also contain $Y$. In the above example the association rule $\{1, 3\} \rightarrow \{5\}$ has a support count for $X \cup Y = 1$ and a support count for $X = 2$.
Therefore, the confidence $c = \frac{\sigma (X \cup Y)}{\sigma (X)} = 1/2 = 50%$.
Confidence is not symmetric because the value of the divisor changes. So the association rules $X \rightarrow Y$ and $Y \rightarrow X$ can have completely different confidence values. In the above rule ($X \rightarrow Y$), the confidence $= 50\%$ but if we look at the rule $Y \rightarrow X$, then $X \cup Y / Y$ is $1/3 = 33.3\%$.

**Minimum support (Minsup)**: a user defined threshold which states the minimum support for an itemset to show that it is significant.

**Minimum Confidence (Minconf)**: a user defined threshold which states the minimum confidence value needed to show that an association rule is reliable. For the example above, if minconf $= 45\%$ then while $X \rightarrow Y$ is a confident rule as it has the confidence $50\%$, $Y \rightarrow X$ won’t be a confident rule because its confidence is $33\%$ which is below the minconf threshold.

**Transaction database** $D = \{T_1, T_2, ..., T_N\}$ is called a database where each transaction $T_i$ has a unique transaction id and contains subset of elements from $I$.

**Candidate itemset**: Selected or generated itemset which is not exposed to minimum support test yet or its support is not known yet.

**Frequent itemset**: an itemset which has support equal to or greater than the minimum support threshold (minsup) is called a frequent itemset. In other words, it is a particular pattern or patterns which occur frequently. A pattern is co-occurrence, sequential occurrence, substructure (as in graphs) or correlation.

**Frequency** is the occurrence of the pattern or correlation equal or greater than the support and confidence level.

**Association rule**: A rule is represented as follows: $\{X\} \rightarrow \{Y\}$ where both $X$ and $Y$ are itemsets, their intersection is empty set i.e. $X \subseteq I$, and $X \cap Y = \emptyset$ (disjoint sets) and both $X$ and $Y$ are part of
the transactions in the transaction set. In the above example, the support of the 1-itemsets is as follows: \{\{1\} = 3, \{2\} = 2, \{3\} = 4, \{4\} = 2, and\{5\} = 4\} if the minsup is 60% and minconf = 60%, then items \{2, 4\} will be discarded because each of them has 40 % support. So, from the remaining items 1, 3, 5 which are greater than or equal to both minsup and minconf, we can create the following association rules:

\{3\} \rightarrow \{5\} \ (60 \% \ support, 75\% \ confidence) \\
\{3, 5\} \rightarrow \{1\} \ (20\% \ support, 33\% \ confidence) so, this rule doesn’t pass support level and also confidence level. It will be discarded.

Support is the occurrence count of XUY which is 1 divided by the number of transactions is 5. So, support, \( s = \sigma (XUY)/N = 1/5 = 20\% \). So, since support is based on the union of all of the itemsets involved in the association rules, it is a symmetric measure. For example \( X \rightarrow Y \) and \( Y \rightarrow X \) would give the same identical support level.

There could be more association rules given our simple example of a few transactions. All association rules with at least minimum support and confidence level are strong association rules. In other words, strength of an association rule is measured by support and confidence metrics. These rules are considered interesting.

**Why We Use Support and Confidence:** We use support because support eliminates uninteresting rules which may simply occur accidentally. Confidence on the other hand tells us how reliable the inferred rule(s) is (are) It is the conditional probability of the occurrence of Y given X. It is can also be thought of as the fraction of the transactions that have X which also have Y or more formally, \( P (Y \mid X) \).

**How many frequent patterns?**
There are \( 2^d - 1 \) possible candidate itemsets (excluding the null itemset). So, potentially there could be \( 2^d - 1 \) frequent patterns.
Let $N =$ Number of transactions,  
$d =$ Number of items in the dataset  
$M =$ Number of Candidate Itemsets. $M = 2^d$  
$w$ as the maximum number of items in a transaction known as the maximum transaction width  
$NM =$Number of comparisons  
$R =$ Number of rules

Association rule discovery is interested in finding all rules which have support greater or equal to minsup and confidence greater than or equal to minconf. Given this expectation, there could be exponentially many rules. The number of rules can be calculated with the following formula: $R = 3^d - 2^{d+1} +1$

Given the above transactions:  
$R = 3^5 - 2^{5+1} +1$  
$= 243 - 64 + 1$  
There could be 180 association rules.
Proof: 

\[ R = \sum_{k=1}^{d} \binom{d}{k} \sum_{i=1}^{d-k} \binom{d-k}{i} \]

\[ = \sum_{k=1}^{d} \binom{d}{k} (2^{d-k} - 1) \]

\[ = \sum_{k=1}^{d} \binom{d}{k} 2^d - k - \sum_{k=1}^{d} \binom{d}{k} \]

\[ = \sum_{k=1}^{d} \binom{d}{k} 2^d - k - [2^d + 1], \]

where

\[ = \sum_{i=1}^{n} \binom{n}{i} = 2^n - 1 \]

Since

\[(1 + x)^d = \sum_{i=1}^{d} \binom{d}{i} x^i + x^d \]

Substituting \(x = 2\) leads to:

\[ 3^d = \sum_{i=1}^{d} \binom{d}{i} 2^i \]

Therefore, the total number of rules is:

\[ R = 3^d - 2^d - [2^d + 1] = 3^d - 2^{d+1} + 1 \]

** The above derivation is from (Tan, Introduction to Data Mining, Instructor’s manual, 2005).

**Methods for finding association rules:**

**Problem definition:**
Given a large dataset with a large number of transactions find all of the rules which satisfy minimum support (minsup) and minimum confidence (minconf) thresholds.
Example: Repeating our transaction set from before, we have:

<table>
<thead>
<tr>
<th>TID</th>
<th>Items purchased</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>1, 3, 5</td>
</tr>
<tr>
<td>002</td>
<td>2, 3, 5</td>
</tr>
<tr>
<td>003</td>
<td>1, 4, 5</td>
</tr>
<tr>
<td>004</td>
<td>1, 3</td>
</tr>
<tr>
<td>005</td>
<td>2, 3, 4, 5</td>
</tr>
</tbody>
</table>

Table 5: Example market basket transactions and example Association rules

In table 5, there are 5 transactions with the largest size of transaction is 4. There are 5 items. One of the rules in the examples \{2, 3\} -> \{5\} states that any time a customer buys the itemset \{2, 3\}, they have high probability that they will also buy itemset \{5\}.

**Brute force approach:**

In this approach, we find all of the rules and then look at each rule's support. This can be represented by a lattice, which consisting of all the possible combinations of itemsets. We then find the support for each of itemset and then compare the support of itemsets to minsup, eliminating those itemsets below the minsup threshold. This process is exponential, and thus not practical nor possible for large sets with large numbers of items in the dataset. For example, for item size 6, we'll have $2^{6-1}$ candidate itemsets which is 64 itemsets. From the complexity perspective, we need to find the support for every candidate itemset in a lattice structure, i.e. we need to check every transaction for every itemset. That has a complexity of $O(NMw)$. For the above example, that is $O(5 \times 64 \times 5) = 1600$ comparisons (Tan, Introduction to Data Mining, 2005). In a brute force approach, we need to calculate support and confidence for every rule. Given that, there could be potentially $3^d - 2^{d+1} + 1$ rules, this is close to impossible for
large data sets. For example, for a data set with 100 items there will be $5.15 \times 10^{48}$ many rules. (This is a 49 digit number!). The same formula gives us approximately a number on the order of magnitude of $10^{1847}$ for a typical supermarket carrying an average of 38,718 items in.

Most of the rules get discarded if we use support and confidence thresholds. The higher the support threshold the more rules will filter out. Association rule algorithms in general, use a two-phase approach for efficiency. First, they find frequent itemsets (Frequent Itemset Generation), secondly they generate association rules (rule generation). In describing various algorithms we will break these into the two phases. The first phase is computationally hard, while the second phase is linear. Therefore, most of the algorithms are written to deal with the first phase, frequent itemset generation.

**Frequent Pattern Mining (FPM) Algorithms**

There are three major algorithms in FPM. In addition there are various extensions to these. We list them here:

**Major Algorithms**
1. Apriori
2. Eclat
3. FP-Growth

1. **Apriori algorithm**: (Agrawal and Srikant 1994 and Mannila et al. 1994)

Key points: Levelwise, iterative, Breadth-first (BFS) algorithm with candidate generation and join steps. It requires one full scan of the database for each level. Given that the set of $k$-itemset can be grouped into $k$ levels, we would need $k$ database scans.

Basic idea:

First find the frequent 1-itemsets. From the frequent 1-itemsets create candidate 2-itemsets and prune the ones which are infrequent 2-itemsets. From frequent 2-itemsets create
candidate 3-itemsets and so on. Apriori uses BFS --breadth-first search, a level wise, iterative, bottom up algorithmic approach (from 1-itemset to largest sets), which prunes the infrequent items immediately after it discovers them, before it moves to the next step. It then creates new level of itemsets only from those which are frequent. The search space of Apriori algorithm is $2^N$, where $N$ is the number of items. (C.Borgelt). Apriori is an efficient algorithm relying on the Downward Closure property\(^1\) or the Apriori property\(^2\). In other words, if a subset is not frequent, i.e. the support $< \minsup$ threshold, then, all of its supersets have also support $< \minsup$ threshold. This property significantly reduces the search space and reduces the candidate itemsets generated substantially. For example, in the example below 5 transactions generates 13 candidates. Using brute force this would have been 31 candidate itemsets.

**Assumptions:** Assumes that items in itemsets are in lexicographic order.

**Data Structures:** hash-tree, trie (prefix-tree),

**Apriori principle:**

It is based on the following observation:

If $A$ is a subset of $B$ then, support of $B$ cannot exceed that of $A$.

More formally:

$\forall A, B \mid A \subseteq B \Rightarrow s(A) \geq s(B)$

In other words, support count of a superset of an itemset cannot be greater than its subsets. Based on this rule, we have the following:

1. If an itemset is frequent, all of its subsets must be frequent. i.e. If an itemset has infrequent subsets, it cannot be frequent.

2. If an itemset is infrequent all of its supersets are also infrequent. (Tan, Introduction to Data Mining, 2005) (J. Han, 2005)

---

1 A superset of an itemset can not occur more than any of its subsets.

2 All subsets of a frequent itemset must also be frequent.
Figure 4: Example shows that subsets of a frequent itemset are all frequent.

As shown above once it is known that itemset \{1,2,3,4\} is frequent then, all of its subsets –i.e. \{1,2,3\}, \{1,2,4\}, \{1,3,4\},\{2,3,4\},\{1,2\}, \{1,3\},\{1,4\},\{2,3\},\{2,4\},\{3,4\},\{1\},\{2\},\{3\},\{4\}, are also frequent.
Figure 5: Example shows that supersets of an infrequent itemset are all infrequent too.

Once we know that itemset \{5\} is infrequent all of its supersets –i.e. \{1,5\}, \{2,5\}, \{3,5\},\{4,5\}, \{1,2,5\}, \{1,3,5\},\{1,4,5\}, \{2,3,5\}, \{2,4,5\}, \{3,4,5\}, \{1,2,3,5\},\{1,2,4,5\},\{1,3,4,5\},\{2,3,4,5\},\{1,2,3,4,5\} are also infrequent and there is no need to scan the database and check their count.

**Candidate generation**

**Join step:** In order to find frequent (k+1)-itemsets, all of the frequent k-itemsets, which have the first (k-1)-elements common, are joined with each other (called self join), to create k+1 itemsets (for k > 1 itemsets). The join steps ensure no duplicate candidate itemsets are created. It is assumed that the itemsets are in numerical order (for countable data types) or lexiographic order (for strings).
Example:

<table>
<thead>
<tr>
<th>Trans. ID</th>
<th>Items Bought</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>1,2,3</td>
</tr>
<tr>
<td>20</td>
<td>2,4</td>
</tr>
<tr>
<td>30</td>
<td>3,4</td>
</tr>
<tr>
<td>40</td>
<td>1,2,5</td>
</tr>
<tr>
<td>50</td>
<td>1,2,3,4</td>
</tr>
</tbody>
</table>

Table 6: Example transactions.

For the above transactions assume $k = 3$ and the frequent transactions are $\{1,2,3\}$, $\{1,2,4\}$, $\{1,3,4\}$ and $\{2,3,4\}$. In the join step, we have only two itemsets which have $k$-1 items in common (from the beginning in sequence) which are $\{1,2,3\}$ and $\{1,2,4\}$ and from these we create a $k$+1 itemset. $\{1,2,3,4\}$. Itemsets $\{1,3,4\}$ and $\{2,3,4\}$ have $k$-1 itemsets are in common but not from the beginning of the sets so, they don’t meet the join requirement.

Prune step: After the join step of subsets of each candidate itemset is checked in the frequent itemset list for membership. If all of the subsets are not in the frequent itemset list for a given candidate $(k+1)$-itemset, then that itemset is removed from the candidate $(k+1)$-itemset list, otherwise it is inserted in the frequent itemset list.

Joining algorithm:

"insert into $C_k$
select $p.item_1$, $p.item_2$, ..., $p.item_{k-1}$, $q.item_{k-1}$
from $L_{k-1}$ $p$, $L_{k-1}$ $q$
where $p.item_1$=$q.item_1$, ..., $p.item_{k-2}$=$q.item_{k-2}$, $p.item_{k-1}$ < $q.item_{k-1}$"

(J. Han, 2005)

Pruning algorithm

"forall itemsets $c$ in $C_k$ do
forall $(k-1)$-subsets $s$ of $c$ do
if ($s$ is not in $L_{k-1}$) then delete $c$ from $C_k$" (J. Han, 2005)
Apriori Pseudo-code

- **Apriori Pseudo-code:**

  “\( C_k \): Candidate itemset of size \( k \)
  
  \( L_k \): frequent itemset of size \( k \)

  \( L_1 = \{ \text{frequent items} \} \);

  for \( (k = 1; L_k \neq \emptyset; k++) \) do begin

  \( C_{k+1} = \text{candidates generated from } L_k \);

  for each transaction \( t \) in database do

  increment the count of all candidates in \( C_{k+1} \) that are contained in \( t \)

  \( L_{k+1} = \text{candidates in } C_{k+1} \text{ with } \text{min}_\text{support} \)

  end

return \( \cup_k L_k \)”

(J. Han, 2005)

**Apriori algorithm example:**

In the following database there are 5 items and 5 transactions. After scanning the database candidate 1-itemsets are found and their support is counted. Items 1,2,3,4 had support count \( \geq \text{minsup} \) therefore, they are put in the frequent 1-itemset list. Using frequent 1-itemset list candidate 2-itemset list is created as shown below and their support is counted. All candidate 2-itemsets except \( \{1,4\} \) has support count \( \geq \text{minsup} \) hence, they are inserted in frequent 2-itemset list. In this list only itemsets \( \{1,2\}, \{1,3\} \) generates \( \{1,2,3\} \) and \( \{2,3\}, \{2,4\} \) generates \( \{2,3,4\} \) 3-itemsets. All other possible candidates are not generated because they don’t satisfy the join rule. (i.e. \( k-1 \) itemsets need to be in common in two \( k \)-itemsets to be joinable.)
(minsup = 2)

### Table 7: A working example of Apriori algorithm.

<table>
<thead>
<tr>
<th>Trans. ID</th>
<th>Items Bought</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>1,2,3</td>
</tr>
<tr>
<td>20</td>
<td>2,4</td>
</tr>
<tr>
<td>30</td>
<td>3,4</td>
</tr>
<tr>
<td>40</td>
<td>1,2,5</td>
</tr>
<tr>
<td>50</td>
<td>1,2,3,4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Trans. ID</th>
<th>Items Bought</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>1,2,3</td>
</tr>
<tr>
<td>20</td>
<td>2,4</td>
</tr>
<tr>
<td>30</td>
<td>3,4</td>
</tr>
<tr>
<td>40</td>
<td>1,2,5</td>
</tr>
<tr>
<td>50</td>
<td>1,2,3,4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Candidate 1-Itemsets</th>
<th>Items</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Frequent 1-Itemsets</th>
<th>Items</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Candidate 2-Itemsets</th>
<th>Items</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>1,2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>1,3</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>2,3</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>2,4</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3,4</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Frequent 2-Itemsets</th>
<th>Items</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>1,3</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1,4</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2,3</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>2,4</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3,4</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Candidate 3-Itemsets</th>
<th>Items</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,2,3</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>2,3,4</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Frequent 3-Itemsets</th>
<th>Items</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,2,3</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>
Once it is known that an itemset is not frequent then its supersets are not generated and their occurrence are not counted and tested (because only the frequent k-itemsets are used to generate candidate (k+1)-itemsets. Hence, if an itemset is not frequent, it won't be in the frequent itemset list and any of its supersets will not be created or be used to create other supersets). In Figure 6, when we learned that 1-itemset, 5, is not frequent all of its supersets (see in red color) are not generated or tested. The same happens for itemset 14 (in green color), its supersets 124, 134 and 1234 are not created.

---

3 To simplify notation, itemsets in the following form such as \{1,2\} is written as 12.
Figure 7: Infrequent itemset discovery determines the supersets which are infrequent also.

*Notes on Apriori:*
Apriori is an easy to implement, effective candidate pruning algorithm. To work efficiently, the dataset needs to fit in memory.

*What are the issues with Apriori?*
The database needs to be scanned for every level. If there is m-frequent-itemsets, then it needs to scan database m times.
There are too many candidates that are generated which need to be tested.

*Extensions of Apriori* (each algorithm is briefly explained)
*Hashing technique* (An Effective Hash-Based Algorithm for Mining Association Rules): This algorithm successfully reduces overall number of candidates generated by specifically applying a hash-based algorithm on 2-itemsets and reduced the whole candidates in an earlier stage. (J. S. Park M.-S. C., 1995)
Dynamic Itemset counting or DIC dynamically counts the candidate itemsets as the algorithm continues and reduces the candidates generated. In other words DIC algorithm doesn’t wait for database scan to be completed to create the candidates, instead it looks at an itemset’s subsets and determines whether they all be frequent. When DIC determines that all of the subsets of an itemset is frequent or estimated to be frequent, it adds the itemset to the candidate itemset list and starts counting support for the item. This process reduces the number of scans. ***How? A little more detail please***: “Add new candidate itemsets only when all of their subsets are estimated to be frequent.” Why only introduce candidate itemsets at the end of a scan? Start counting them whenever there is enough support from smaller itemsets. Thus you have fewer passes over data”. (S. Brin, 1997)

Partition [Savasere et al., 1995]: Partitioning algorithm scans the database only twice hence reduces the I/O substantially. In the first scan it partitions the whole database into smaller vertical database pieces which are small enough to fit in memory. Then it finds the frequent itemsets found in each partition. In order for an itemset to be frequent, it has to be frequent at least in one of the partitions, otherwise, it will be discarded. Local frequent itemsets then are merged with other partitions by intersecting these itemsets and in the second database scan. Support for the locally frequent itemsets are determined using the entire database. In order for partition to be efficient, each partition needs to fit in memory, otherwise additional database scans will be required. Also, in partitions there could be too many frequent itemsets which are not frequent in the entire dataset. (Zaki., 2000) (Goethals, 2003)

Sampling: The Sampling algorithm finds the frequent itemsets from randomly selected samples. It verifies these results in the whole database and creates the complete association rules for these verified frequent itemsets. But there is a probability that some of the frequent itemsets might not appear on the selected samples, – hence they are never found to be frequent and no association rules would be created for those itemsets. In order to minimize this problem, a very low threshold is used but this creates too many candidates. The maximum number of database scans in the Sampling algorithm is 2. One to get the samples and verify the results, then if there
are missing frequent itemsets, these are found and verified in the second pass. (Toivonen, 1996)

**Eclat (Zaki M. J., 1997)**

Basic idea:

Eclat is a vertical database or dataset layout. It scans the database and builds Transaction ID list (TIDlist) for each item. From these, for each frequent single item, (k-itemset where $k = 1$), create a Transaction ID set for $(k+1)$ itemsets by taking the intersection between two itemsets. Eclat partitions the frequent itemset list into equivalence classes and obtains the support count from the number of intersections.

ECLAT doesn’t need another database scan for support count because each TIDlist has complete information about the database. (Zaki., 2000)

Major features of ECLAT: recursive, depth-first search (DFS) algorithm, uses tidlists, intersections, a divide-and-conquer algorithm.
Example:
Transaction Database:

<table>
<thead>
<tr>
<th>Trans. ID</th>
<th>Items Bought</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>1,2,3</td>
</tr>
<tr>
<td>20</td>
<td>2,4</td>
</tr>
<tr>
<td>30</td>
<td>3,4</td>
</tr>
<tr>
<td>40</td>
<td>1,2,5</td>
</tr>
<tr>
<td>50</td>
<td>1,2,3,4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Items</th>
<th>TIDs</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10,40,50</td>
</tr>
<tr>
<td>2</td>
<td>10,20,40,50</td>
</tr>
<tr>
<td>3</td>
<td>10,30,50</td>
</tr>
<tr>
<td>4</td>
<td>20,30,50</td>
</tr>
<tr>
<td>5</td>
<td>40</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>1,2</th>
<th>1,3</th>
<th>1,4</th>
<th>1,5</th>
<th>2,3</th>
<th>2,4</th>
<th>2,5</th>
<th>3,4</th>
</tr>
</thead>
<tbody>
<tr>
<td>10,40,50</td>
<td>10,50</td>
<td>50</td>
<td>40</td>
<td>10,50</td>
<td>20,50</td>
<td>40</td>
<td>30,50</td>
</tr>
</tbody>
</table>

Minsup = 2

1,2 \cap 1,3 \text{ i.e. } \{1,2,3\}
2,3 \cap 2,4 \text{ i.e. } \{2,3,4\}
10,50
50

Table 8: ECLAT process.
Maximal Itemset Mining (Max-patterns): [Bayordo, 1998]: When association rules involve many items, generating rules for all of the subsets becomes a combinatoric problem and not very practical. Hence, some frequent itemset algorithms focus instead on finding maximal frequent itemsets.

A frequent itemset is maximal if it doesn’t have a superset which is frequent. In other words, a maximal frequent itemset is not a subset of any frequent itemset. (Zaki., 2000)

MAFIA (D. Burdick, 2001) is one of the maximal frequent itemset mining algorithms. It works on transactional databases. It is a dept first search algorithm as well as using efficient pruning techniques. It is particularly more efficient if the itemsets are very long in the dataset.
In the example in table 8, itemset \{1, 2, 3\} is a maximal frequent itemset.

**Closed Itemsets:**
An itemset is closed if there is no superset of this itemset which has the same support as the support of this itemset.

**Close:** Closed Itemset mining (Closed patterns): Close tries to find the closed itemset by using a bottom-up search strategy. In order to find the closed itemsets it uses two steps. In the first step it finds the smallest frequent itemsets. After finding a k-itemset, it looks back and compares the support of the k-itemset with its subsets. If an itemset’s support count matches its subset then that itemset is pruned and discarded. In the second step the algorithm determines closure for all itemsets generated previously creating much fewer rules as a result. (N. Pasquier, 1999) (Goethals, 2003)

**A-CLOSE:** A-CLOSE is a variation of CLOSE which also finds the closed itemsets and the rules only those itemsets which are closed. A-Close first finds all of the frequent closed itemsets which generates all of the subset of the maximal frequent itemsets. From the closed itemsets then it finds all of the frequent itemsets. (N. Pasquier, 1999) (Goethals, 2003)

**COBBLER:** COBBLER is a frequent closed pattern mining algorithm which dynamically decides how to proceed according to whether the dataset to be mined has very large number of attributes and relatively small number of rows or vice versa. It is a recursive algorithm, uses both original table and a transposed table without the infrequent itemsets and employs DFS algorithm to traverse the tree. (Pan F, 2004)

**CLOSET:** Closet is another frequent closed pattern mining algorithm. It uses a compressed FP tree structure, a recursive divide and conquer strategy, prefix-based compression technique and partition-based database projection approach. (J. Pei J. M., 2000)
**CHARM**: CHARM is a frequent closed pattern mining algorithm which finds all of the frequent closed itemsets. It uses depth-first search technique to enumerate dual itemset-tidset (i.e. itemset and transaction space) search tree. It skips many levels through an efficient hybrid search method. It is a good algorithm when there are small number of features (columns) and large number of rows. CHARM uses a memory efficient technique diffset to minimize memory usage during intermediate operations. In addition, CHARM uses a hash based technique to find the non closed itemsets and when it finds them, removes them immediately. (M. J. Zaki, 2002)

**CLOSET, CLOSET+** adopts and uses previously successful techniques such as sub-itemset checking and pruning, itemset merging and item skipping techniques. It also uses an FP tree, divide and conquer strategy and depth-first search algorithm to find frequent closed patterns in a dataset. In addition, CLOSET+ also uses hybrid tree projection method which is used to create bottom-up physical tree-projection for dense databases and top-down pseudo tree-projection for sparse databases. CLOSET+ is scalable algorithm as well as an improvement for many earlier algorithms which includes CLOSET and CHARM in terms of running time, memory usage. (J. Wang J. H., 2003)

**FP-growth** [Han et al., SIGMOD 2000]

*Basic Idea:*
FP-growth algorithm finds all frequent itemsets without candidate generation. It first scans the database, finding all frequent 1-itemsets. It then sorts them in descending order. Using this order, it sorts transactions only with frequent items, in descending order.

FP-growth then reads the ordered database a second time, and builds an FP-tree. The FP-tree has a null root on the top, and at this stage, only frequent 1-itemsets as nodes nodes. Any subsequent transactions are stored on the tree with complete number of occurrence and association information. Each transaction is a path from the root thus preserving the descending order in the tree.

FP-growth algorithm works as a divide and conquer method, recursively without candidate generation. FP-Tree efficiency:
Whole dataset gets compressed into a tree-format (fp-tree) which contains complete information about all of the transactions.

No candidate generation. Instead, a pattern fragment growth method is used. This method starts from frequent 1-itemset.

Mining task is divided into smaller task (hence, recursive DFS algorithm) which uses partition-based divide and conquer method to mine association rules.

Example:

<table>
<thead>
<tr>
<th>Trans. ID</th>
<th>Items Bought</th>
<th>Frequent Items(Ordered)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1,2,3,5</td>
<td>2, 1,3</td>
</tr>
<tr>
<td>2</td>
<td>2,4</td>
<td>2,4</td>
</tr>
<tr>
<td>3</td>
<td>4,3</td>
<td>3,4</td>
</tr>
<tr>
<td>4</td>
<td>1,2,5</td>
<td>2,1</td>
</tr>
<tr>
<td>5</td>
<td>1,2,3,4</td>
<td>2,1,3,4</td>
</tr>
</tbody>
</table>

*minimum support =3

Table 9: A table from the transaction database with 5 transactions and 5 items

Frequent Items (in descending order): 2:4, 1:3, 3:3, 4:3

Header Table

<table>
<thead>
<tr>
<th>Item</th>
<th>Head of Node-links</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

Figure 8: Header table and FP-Tree.
Performance study shows have shown that FP-growth is an order of magnitude faster than Apriori, and is also faster than tree-projection. This is because there is no candidate generation and no candidate test. It uses:
- Use compact data structure
- Eliminate repeated database scan
- Basic operation is counting and FP-tree building”.

*Extension of FP-Growth*

Depth-first generation of frequent itemsets (Agarwal et al. 2001) H-Mine explores a hyperstructure mining of frequent patterns (Pei et al 2001)

**AFOPT** - top-down and bottom up traversal of sub-trees in pattern growth mining) (Liu et al 2002, 2003)

**FPClose** - Array based implementation of prefix-tree structure for efficient pattern growth mining (Grahne and Zhu 2003)

**CARPENTER**: CARPENTER is a frequent closed pattern mining algorithm specifically designed to deal with a dataset (such as data in bioinformatics) with very large number of columns but much fewer number of rows. It uses row enumeration to handle the very large number of columns that typically biological datasets have. CARPENTER uses DFS algorithm on the row enumeration tree and uses recursive algorithms to generate conditional transposed tables. (F. Pan, 2003)

*Other algorithms and improvements*

**Tree-Projection**: This algorithm uses much less memory through the use of a lexicographical, top down tree instead of using a hash tree. Itemsets are counted through projecting the transactions onto the nodes of the tree which increases the performance of counting the itemsets in the transactions. (R. C. Agarwal, 2000)

**Relim**: The algorithm uses an idea similar to FP-growth via a single recursive function. It is similar to the H-mine algorithm. It has complicated data structures. (Borgelt, 2005)
**DHP:** DHP is a hash-based, Apriori-based algorithm which utilizes pruning techniques as well as approximate counts to generate much fewer candidates, thus shrinking the database or transaction data size. Even though it generates candidates very efficiently the number of database scans are equal to the number of levels. (J. S. Park M. C., 1995)

Some other recent pattern mining algorithms include: Colossal pattern mining (very long pattern mining), Mining association rules with multiple minimum support, Mining multidimensional space, Top-down pattern mining and others.

---

**Chapter 3**

**Sequential Pattern Mining**

The association rule problem looks at customer behavior within the same transaction (i.e. the same visit to the supermarket) to find whether there are any associations or correlations between items, i.e. when a customer buys item A what is the likelihood that he/she also buys item B. The sequential pattern mining problem, on the other hand, looks at patterns in different transactions which occur at different times. i.e. if a person buys item A what is the implication that he goes and buys item B within a week or month. In sequential pattern mining there is a sequence of events. These events are recorded with a time dimension in the database. These databases or datasets are called time-series databases, where each record has a time dimension attribute showing the occurrence time of that transaction. Sequential pattern mining is not only interested in the specific time of events, it may also be interested in the order of events without being concerned about the exact timestamp.

**Problem:** Find all frequent subsequences in a dataset with an order of events (i.e. sequential dataset or database), given a user determined minimum support value.

**Terms:**
Transaction database: Traditional retail database where there is TID (Transaction ID) and itemsets.

Example:

<table>
<thead>
<tr>
<th>TID</th>
<th>Itemset</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>apple, orange, banana</td>
</tr>
<tr>
<td>002</td>
<td>milk, diaper, apple</td>
</tr>
<tr>
<td>003</td>
<td>diaper, orange, cereal</td>
</tr>
</tbody>
</table>

Table 12: A basic transaction database

Sequential database: SID (Sequence ID) and sequences.

<table>
<thead>
<tr>
<th>Customer ID</th>
<th>Data Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>&lt;{apple} {orange, banana}&gt;</td>
</tr>
<tr>
<td>002</td>
<td>&lt;{milk, diaper} { apple}&gt;</td>
</tr>
<tr>
<td>003</td>
<td>&lt;{diaper} {orange} {apple} {cereal}&gt;</td>
</tr>
</tbody>
</table>

Table 13: An example of sequential database

Transactions: Itemsets bought within the single visit of a supermarket.

Sequence: A list of items that are ordered according to time of their occurrence. In other words, a sequence is an ordered list of transactions where each transaction has occurrence time and might have multiple items in it.
For example: sequence \( s = \langle t_1, t_2, ..., t_k \rangle \) such that \( t_i = \{i_1, i_2, ..., i_n\} \)

**Frequent sequence**: percentage of customers who have the same sequence greater than minsup.

**Maximal Sequence**: a sequence which is not included in another sequence.

Sequential pattern: a pattern occurring between transactions. For example, 40% of investors who buy a stock and sell a covered call on that stock to hedge within 2 days. 25% of the people who taste sample food at Trader Joes buy those products within a week.

**Itemset**: a set of item(s) from the whole item set \( I \). These itemsets are mapped in the transformation phase to an integer for ease of calculation and efficiency.

Original transactions according to transaction date:

Example (Adapted from (R. Agrawal, Mining Sequential Patterns, 1995))

Original transactions by date:

<table>
<thead>
<tr>
<th>TransNumber</th>
<th>CustID</th>
<th>TransTime</th>
<th>ItemsBought</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>12-Jul-12</td>
<td>10,20</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>14-Jul-12</td>
<td>90</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>15-Jul-12</td>
<td>30</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>19-Jul-12</td>
<td>40,60,70</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>23-Jul-12</td>
<td>30</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>23-Jul-12</td>
<td>30,50,70</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
<td>23-Jul-12</td>
<td>30</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>29-Jul-12</td>
<td>90</td>
</tr>
<tr>
<td>9</td>
<td>4</td>
<td>29-Jul-12</td>
<td>40,70</td>
</tr>
<tr>
<td>10</td>
<td>4</td>
<td>31-Jul-12</td>
<td>90</td>
</tr>
</tbody>
</table>

Table 14: Original transactions list

We kept only the required attributes (columns) and sorted the transactions by CustID then TransTime which grouped the customer transactions together as shown below:
Table 15: Sorted transactions (by custid then date)

Minsup = 2

Large sequences are mapped to integer numbers in order to simplify the process. For the above example, large sequences are mapped as follows (sorted by itemset):

<table>
<thead>
<tr>
<th>Itemset</th>
<th>Count</th>
<th>Mapped to</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>40</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>70</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>40,70</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>90</td>
<td>3</td>
<td>5</td>
</tr>
</tbody>
</table>

Table 16: Large sequences and their mapping to an integer number

Customer sequences:

<table>
<thead>
<tr>
<th>custID</th>
<th>Original Sequence</th>
<th>Transformed (Large) Sequence</th>
<th>Mapped Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>{30} {90}</td>
<td>{30} {90}</td>
<td>{1} {5}</td>
</tr>
<tr>
<td>2</td>
<td>{10,20} {30} {40,60,70}</td>
<td>{30} {40} {70} {40,70}</td>
<td>{1} {2,3,4}</td>
</tr>
<tr>
<td>3</td>
<td>{30} {50,70}</td>
<td>{30}</td>
<td>{1} {3}</td>
</tr>
<tr>
<td>4</td>
<td>{30} {40,70} {90}</td>
<td>{30} {40} {70} {40,70} {90}</td>
<td>{1} {2,3,4} {5}</td>
</tr>
<tr>
<td>5</td>
<td>{90}</td>
<td>{90}</td>
<td>{5}</td>
</tr>
</tbody>
</table>

Table 17: Sequences are transformed (mapped) into integers according to their occurrence above.
Where sequential pattern mining is used:

Sequential pattern mining is used to find purchase patterns or customer behavior, stock market participants wrt how they buy into positions and hedge their investments, in other scientific areas such as biology, telecommunications, weather forecasts.

**Algorithms:**

**Apriori based**

(AprioriAll, AprioriSome, DynamicSome): All of these algorithms are introduced by Agarwal and Srikant in 1995. They consider only three attributes: customer id, timestamp of the transaction and items purchased (itemset). Accordingly, quantity of the items bought is immaterial. Customerid and timestamp together form a unique identifier.

These algorithms, before looking for sequential patterns, first sort the transactions by customer id and then by date so that all the transactions of a particular customer groups together into a sequence. The second phase is finding large 1-itemsets. These are found by finding the occurrence of a particular itemset greater than the minimum support threshold.

**GSP (Apriori based):** GSP is a Apriori based, level-wise, sequential pattern mining algorithm. At each level it counts support for the sequences and discards the ones which have a count smaller than the minsup threshold. From the frequent sequences, the algorithm creates the next level candidate sequences and counts their support and discard the sequences which are infrequent. This process continues until there is no level left. At the end of GSP all non-maximal frequent sequences are discarded. (Srikant, 1996)

**Vertical based approach**

**SPADE:** SPADE divides the problem into smaller problems by using combinatorial properties and solves each sub-problem in memory independently from eachother. SPADE solves these sub problems using efficient, join operations and lattice-based search techniques. It finds all of the sequences in the dataset in exactly three database scans. SPADE is a scalable, sequential
pattern discovery algorithm which outperforms the best known algorithms by a factor of 2 according to the experiments. (Zaki M. J., 2001)

**Pattern Growth based**

*FreeSpan*: FreeSpan combines the techniques of finding frequent itemset with that of frequent sequences and narrow down the search as well as the growth of the subsequences using projected sequence databases. FreSpan finds the complete set of frequent sequences and searches much smaller number of subsequences than the Apriori based GSP algorithm. (J. Pei J. H.-A.-C., 2001)

*PrefixSpan*: Prefix span uses pattern growth method for finding sequential patterns. It projects only the frequent prefixes through which it substantially reduces the dataset to be searched for frequent sequential patterns. Algorithm finds complete set of frequent sequential patterns. (J. Pei J. H.-A.-C., 2001)

*CloSpan*: This paper proposes mining only closed subsequences which is good for mining long sequences. It produces much fewer number sequences than the traditional algorithms. (X. Yan, 2003)

**Others**

*MEMISP*: “a memory indexing approach for fast sequential pattern mining, named MEMISP. During the whole process, MEMISP scans the sequence database only once to read data sequences into memory. The find-then-index technique is recursively used to find the items that constitute a frequent sequence and constructs a compact index set which indicates the set of data sequences for further exploration. As a result of effective index advancing, fewer and shorter data sequences need to be processed as the discovered patterns get longer. Moreover, we can estimate the maximum size of total memory required, which is independent of the minimum support threshold, in MEMISP. Experimental results indicate that MEMISP
outperforms both GSP and PrefixSpan (general version) without the need for either candidate generation or database projection. When the database is too large to fit into memory in a batch, we partition the database, mine patterns in each partition, and validate the true patterns in the second pass of database scanning. Experiments performed on extra-large databases demonstrate the good performance and scalability of MEMISP, even with very low minimum support. Therefore, MEMISP can efficiently mine sequence databases of any size, for any minimum support values.” (Lin, 2002)

**SPIRIT:** SPIRIT algorithm finds sequential patterns with Regular Expression (RE) constraints. RE constraints give user a lot of flexibility to control the constraints comparing to minsup. (M. N. Grofalakis, 1999)

**BIDE:** Efficient Mining of Frequent Closed Sequences without candidate maintenance. It uses a closure checking logic called BI-Directional Extension as well as Backscan method and ScanSkip techniques to more efficiently prune and optimize the process. (J. Wang J. H., 2004)

Chapter 4

**Incremental Algorithms**

Businesses analyze relationships among the products. These relationships are created from large amount of data to make correct business decisions. Association rule mining looks for finding these interesting relationships among data within the large datasets which is a computationally expensive operation. Even though these rules are found, the new data still
continues to come and gets added to the existing databases. or existing data gets updated or even deleted – hence changing the already found association rules.

New data is flowing into the existing storages constantly. Extracted information, such as association rules, gets updated and sometimes becomes more significant, other times it might become outdated and less important. Because of this, existing association rules might disappear or new rules might emerge. That is, if the datasets get updated the association rules extracted from it may change: some old rules may disappear and meanwhile new rules might come to existence. In other words, frequent Itemsets may become infrequent and infrequent ones may become frequent.

One of the ways to find the rules in the updated database is to run an AR algorithm again on this combined or updated dataset but this is not efficient as the all the previous efforts and computation will be wasted. Hence, it is undesirable and inefficient to find all of the rules from scratch. Instead, using the previously found rules and only scan the new data or updated data to find and update the existing relationships is a much more efficient solution.

Incremental association rule finding algorithms exist for this reason. They typically use previously discovered information to integrate new data and find new rules or update the previous ones. (D.Cheung, 1996) (M. Zaki, 1997). In other words, Incremental Assoication rule mining algorithms look for efficient and effective way to update the previously found association rules as the new (incremental) data comes in periodicaly without scanning the previous large dataset which is already scanned and processed. (Shah, 2012)

When new data is added to the existing data we have two options: either re-run the FIM algorithms on the updated data or use an incremental mining technique. But running the algorithm for the the entire, updated dataset will not utilize the existing rules and hence wasting the computation made previously to find them. Therefore, there are many tree or hashing based incremental algorithms proposed to efficiently update the existing rules when new data gets added. (Dubey, 2015)
There are two groups of incremental algorithms: Apriori based and tree based. (Shah, 2012)

**Apriori-based incremental algorithms**

Furthermore incremental algorithms are further divided into insertion and/or deletion algorithms.

**FUP (Fast Update):**

FUP uses the information about the old rules created prior to the update of the database. FUP is found to be 2-16 times faster than creating the whole set of rules using DHP or Apriori. One of the strengths of FUP is that it reduces the number of generated candidate itemsets.

Comparing to DHP, the size of candidates in FUP is 2-5 % of DHP.

FUP algorithm is similar to Apriori and DHP algorithms. It is based on iterative steps and in each step the new step is created from the previously selected large itemsets. The major differences of FUP from Apriori and DHP is that after each level FUP scans the db (i.e. the update) to get the new count for existing large itemsets to determine whether they are still large itemsets after the update or not. Since the count for large itemsets are already known from the previous calculations, only the increment, db, gets scanned, candidates, $C_k$ itemsets are created for db and these are counted for the support information of the itemsets in DB. Before any updates are done for the itemsets in DB, the candidates are pruned from $C_k$ and at each iteration, this pruning makes the database smaller. (D.Cheung, 1996)

FUP is an insertion algorithm as it only considers for new incremental data being added to the existing data. Deletions of transactions are not considered. Main points of the algorithm is: if a large itemset $X$ in D’s support in $DB+db < minSup – X$ become infrequent. Infrequent $X$ in D can be frequent only if $X$'s support in $db$ is greater than $minSup$ and if a subset of itemset $X$ becomes infrequent, itemset $X$ becomes infrequent too. This is one of the ways FUP reduces the number of candidate Itemsets get generated. FUP is an iterative algorithm which scans the increment only once on the kth iteration. If an itemset is in the original LIS, it is only checked in the increment. That is, FUP is based on Apriori only scans the db to start. Only in one of the 4
cases it also scans DB – the original database, when the item is frequent in the increment but infrequent in the original DB. In other 3 cases, when the itemset is frequent, infrequent in both or when it is frequent in the original DB, then, it only scans db – the increment. (Dubey, 2015)

**FUP2:**

FUP2 is similar to FUP in terms of insertion but adds features for transaction deletion as well. In addition to FUP algorithm, the basic idea is if a candidate itemset $Z_k$ is frequent in the deleted transactions, then it must be infrequent in the remaining original transactions. If candidate itemset $Z_k$ is also infrequent in the added transactions then it must be infrequent in the updated database as well.

FUP2 also considers deletion of transactions in addition to insertion as in the case in FUP. FUP2 separates candidate itemsets ($C_k$) into two groups in each iteration. For example in kth iteration, it will have $C_{k1}$ and $C_{k2}$ where $C_{k1}$ is the group which intersects with $L_k$ and $C_{k2}$ is the remaining candidate itemsets which doesn’t exist in $L_k$. Both FUP and FUP2 has poor performance. (Dubey, 2015)

**UWEP:**

UWEP is an insertion based algorithm which prunes the itemset and its supersets immediately after it knows that a frequent itemset in the original database is infrequent in the updated database. It also creates candidates only if an itemset is frequent in the increment and in the updated database—which reduces the number of candidates significantly. (N. F. Ayan, 1999)

UWEP is based on FUP and FUP2 but it is more efficient. It scans the original DB at most once, and the increment, db exactly once. It creates and counts minimal number of candidate itemsets. By look ahead pruning strategy it not only removes the candidate Itemsets which are infrequent but also its supersets also get removed too immediately once an itemset is known to be infrequent. UWEP keeps all transactions on a vertical representation, Tidlist format in memory.
This can potentially be a problem if the dataset is too large. UWEP uses hash table and queues as its data structures to fast count the occurrence of Itemsets. (N. F. Ayan, 1999) (Dubey, 2015)

UWEP algorithm is proposed to update large itemsets efficiently by considering previously discovered itemsets. This is accomplished by:
1) pruning an itemset immediately after it is understood to be small in the updated db (i.e. the increment).
2) keeping the candidate set as small as possible.

This algorithm outperforms other incremental update algorithms in terms of the number of times the db is scanned and the number of candidates are generated and counted. (N. F. Ayan, 1999)

UWEP Internals:
*Support count or minsup for DB and db are assumed to be the same.
*Can be useful in deciding emerging itemsets.

Either:

<table>
<thead>
<tr>
<th></th>
<th>Apriori (DB)</th>
<th>Large itemsets</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB</td>
<td></td>
<td>Li_DB={...}</td>
</tr>
<tr>
<td>db</td>
<td>Apriori (DB+db)</td>
<td>Li_DB+db={...}</td>
</tr>
</tbody>
</table>
but additional data will be needed to merge the results.

\[ \text{Apriori (DB)} \quad \text{Large itemsets} \]
\[ L_{\text{DB}} = \{ \ldots \} \]

\[ \text{Apriori (db)} \quad L_{\text{db}} = \{ \ldots \} \]

- What is needed?
- We need to know whether DB and db are Large (L) or small (S)

\[
\begin{array}{ccc}
L_{\text{DB}} & L_{\text{db}} & L_{\text{DB}+\text{db}} \\
L & L & L \\
L & S & ? \\
S & L & ? \\
S & S & S \\
\end{array}
\]

//e.g. if \( 50/n > \text{minsup} \) AND \( 20/n' > \text{minsup} \) then, we can conclude that \((50+20)/(n+n')\)

\[
\text{i.e. } (50/n + 20/n') > \text{minsup}
\]

Example: let’s say \( n = 1000 \) and \( n' = 100 \), \( \text{minsup} = 5\% \).

Then, we know that for DB we need support count \( \geq 50 \) and for db it is \( \geq 5 \), i.e. \( 55 \geq 1100 \)

Assume \( L_{\text{DB}} = 50 \) \( L_{\text{db}} = 4 \), i.e. \( L_{\text{DB}+\text{db}} = 54 \) //small (S)

Assume \( L_{\text{DB}} = 58 \) \( L_{\text{db}} = 4 \), i.e. \( L_{\text{DB}+\text{db}} = 62 \) //large (L)

What is needed?

- \( L_{\text{DB}} \): keep support count of the items in \( L_{\text{DB}} \)
- \( L_{\text{DB}} \): keep \( L_{\text{db}} \)
- \( s(x) \): \( L^{(10)} \) keep TID list for each item. \( X \) is calculated from TID list in DB for the items in \( L_{\text{db}} \).
- \( s \): \( s \)

How to implement UWEP:

Given DB, \( L_{\text{DB}} \) with support count,
Step 1:

Apply apriori on db, get \( L_{db} \) with support count and TID list

Step 2:

Combine Given info + Step 1

Finding the support count from TIDlist for itemset \( X \)
Suppose \( X \) is ABC
Then, support (ABC) = |TID (A) \( \cap \) TID (B) \( \cap \) TID(C)|

For example:
If: A: \{1, 2, 3\}
B: \{2, 3, 5, 6\}
C: \{1, 3, 4, 5, 6, 7\}
Support(ABC) = 1 as the minimum intersection of this set is 1 – i.e. they only occur in transaction 3 all together, therefore the support count for this set is 1.

\( C^1_{db} \) = all 1-itemsets in db with support > 0

Prune set = \( L_{DB} \) - \( C^1_{db} \)

e.g. \{A,B,C,D,E,F\} - \{D,E,F\} = \{A,B,C\} is in L in DB

Is S in db (with support =0)

Decide without using TID in db.
if an itemset is large in DB and has support count enough for DB+db, then, prune db for all the subsets of db for that itemset.

Time

1/1/11  1/8/11  1/15/11

--------------------------------------------------------------------------------------------------------------------------→

DB    db    (DB +db)+db    ((DB+db)+db) +db    ...

Figure 99: Timeline for periodic increments and their addition to the dataset.

Algorithm UWEP has the following inputs: DB, db, LDB (along with their supports in DB), |DB|, |db| and minsup. The output that we are interested in is LDB+db.

<table>
<thead>
<tr>
<th>Notation</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB</td>
<td>Set of old transactions</td>
</tr>
<tr>
<td>db</td>
<td>Set of new transactions</td>
</tr>
<tr>
<td>DB + db</td>
<td>Set of transactions after update</td>
</tr>
<tr>
<td></td>
<td>A</td>
</tr>
<tr>
<td>Minsup</td>
<td>Minimum support threshold</td>
</tr>
<tr>
<td>SupportA(X)</td>
<td>Support of X in A</td>
</tr>
<tr>
<td>TidlistsA(X)</td>
<td>Transaction list of X in A</td>
</tr>
<tr>
<td>C_A^k</td>
<td>Candidate k-itemsets in A</td>
</tr>
<tr>
<td>L_A^k</td>
<td>Large k-itemsets in A</td>
</tr>
<tr>
<td>PruneSet</td>
<td>Large itemsets in DB with 0 support in db</td>
</tr>
<tr>
<td>Unchecked</td>
<td>Large k-itemsets in DB that are not counted in db</td>
</tr>
</tbody>
</table>

Table 10: UWEP algorithm notations and their short definitions

Algorithm UWEP has the following 5 steps:
1) Counting 1-itemsets in db and creating a tidlist for each item in db
2) Checking the large itemsets in DB whose items are absent in db and their superset for largeness in DB+db
3) Checking the large itemsets in db for largeness in DB+db
4) Checking the large itemsets in DB that are not counted over db for largeness in DB+db
5) Generating the candidate set from the set of large itemsets obtained in the previous step.
**NUWEP (Negative Border Update With Early Pruning):**

NUWEP is a modification to the incremental pattern mining algorithm UWEP to find the emergent frequent itemsets. In addition to keeping track of the large itemsets it also counts the itemset in Negative border\(^4\). (S. Imberman, 2004)

**NFUP (Algorithm Utilizing Negative Borders):**

Improvement to FUP algorithms. It considers all Itemsets whose subset are all frequent but themselves right at the border and not frequent. The algorithm scans only the incremental database to generate frequent Itemsets. It doesn’t scan the the original database.

NFUP considers the database and its increments are like publication like databases. That is the items have a period of appearance.

Traditional algorithms don’t handle well publication like databases where Applying a traditional support count to all items in a standard way result in disqualifying newly appearing Itemsets – hence it doesn’t reflect the frequency of Itemsets correctly, in a fair manner.

When data gets updated periodically, old rules may become infrequent and previously infrequent Itemsets can become frequent.

The previous incremental algorithms focused on minimizing scanning the old database because the original database is typically very big part of the newly combined dataset.

In FUP algorithm when the increment is frequent but the original DB is infrequent, DB needs to get rescanned. NFUP thinks that in many cases the newly added data contains more valuable information than the old data. Hence, if an itemset is frequent in the increment, this is still valuable even if it is not frequent in the entire database. So, in order to extract the interesting frequent Itemsets NFUP divides the data into its logical segments based on the time period they were added to dataset. Last added go to last partition and so on. Then NFUP scans these

---

\(^4\) An itemset is in negative border if all of its subsets are large Itemsets but itself is not large itemset.
partitions starting from the last one. There are \( n \) partitions based on \( n \) arrival times of the increments. In NFUP each increment is denoted with continuous interval arrival time from time \( m \) to time \( n \). Hence the frequent Itemsets also categorized accordingly as follows:

In alpha set: there are frequent Itemsets in \( DB^+ \) which denotes the updated database.

In beta set: the Itemsets are frequent in the increment between \( m \) and \( n \) but not in \( m-1 \) and \( n \).

In r set: Itemsets are frequent in \( m \) and \( m \) but not \( m+1 \) and \( n \).

The process starts from the last partition, that is \( P_n \) with the count of 1-itemsets.

Three attributes get captured about each candidate and frequent Itemsets:
1. Itemset count in the current partition
2. the first partition when an itemset becomes frequent
3. whether the itemset is in alpha, beta or in r group. These groups start empty and gradually get filled with Itemsets.

NFUP’s process is based on Apriori algorithm. Hence once the frequent 1-itemsets are found in a partition they get placed in alpha set and 2-itemset candidates are generated from these Itemsets by joining them with eachother. Once all the candidates are generated \( P_{n-1} \) gets scanned for occurrence in \( P_{n-1} \) partition. Each partition is scanned and counted like this and the frequent Itemsets are moved to alpha set and the other ones are moved to beta or r sets. NFUP keeps track of which partition the frequent Itemsets are coming from.

Input to NFUP is \( DB \) and all the increments –each in a partition from 1-n.

Output is:

Alpha set which contains all frequent Itemsets from updated database \( DB^+ \),

Beta set: frequent Itemsets in \( db_{m,n} \) but infrequent in \( db_{m-1,n} \) or \( DB^+ \) if \( n=1 \) or

R set: frequent Itemsets in \( db_{m,m} \) but infrequent in \( db_{m+1,m+1} \).
NFUP doesn’t need to scan the original database because the alpha set is frequent in the updated database. The union of alpha, beta and r sets are superset of all the frequent Itemsets of Apriori and FUP algorithms. Therefore, scanning of the original database is not necessary.

**MAAP and Pelican:**

MAAP finds the longest frequent Itemsets using the previously discovered frequent Itemsets. Once it finds these, then adds all of their subsets in to the new large itemset list without counting their support. It finds the other large Itemsets through level wise approach.

Both MAAP and Pelican keeps only the maximal large itemset in the large itemset list, hence saving space as well as eliminating the count of the subsets. MAAP is based on Apriori algorithm while Pelican is based on vertical database and “lattice decomposition”. (Shah, 2012)

In other words, MAAP algorithm is just like look ahead early pruning technique in UWEP, it starts from the large Itemsets which comes from the original database and infers whether these large itemsets are still frequent in the updated database. If it finds out that they are frequent, then, based on Apriori all of the subsets of these large Itemsets also be frequent and therefore support count for these subsets can be then eliminated. It goes down and prunes the subsets of an infrequent large itemset in the increment using Apriori property also. (Dubey, 2015)

**Tree based algorithms:**

**DT-tree and PotFP-tree algorithms:**

DB tree construct an FP tree and stores all of the Itemsets in FP tree. Transactions in the increment gets added to tree as well without any filtering. The algorithm allows deletion of transactions, hence, the tree gets adjusted when deletion occurs. Since all the dataset and increment is stored in the tree, memory can be an issue.

**PotFp tree:**
PotFp tree stores only 1-itemsets and potentially frequent Itemsets—hence alleviating the heavy memory need. It uses a parameter to decided for an itemset to be potentially frequent. So, FP tree gets updated selectively reducing the memory needs.

**FELINE**: uses CATS tree—a compressed and arranged transaction sequences tree, similar to FP tree but not the same. It stores all the items in the tree, hence eliminating to go back to database for rescans. It scans the database only once.

**CAN Tree** (Canonical order tree): Orders all the transactions in a canonical order in the tree in a single database scan. Incremental updates would change the frequency of Itemsets but this doesn’t affect the order of items in the tree. Mining of frequent Itemsets are similar to FP growth algorithm once the CAN Tree is constructed. No merging, splitting or swapping required in CAN tree.

**Compressed and Arranged Transaction sequence (CATs) Tree and Canonical-Order (CAN) tree**: All items get inserted in the FP tree in order and in compressed way using only one database scan. No candidate itemset generation required. All items in the tree arranged accordingly with their descending frequency order. CAN tree traverses only upwards to find mergeable paths.

**Association rule mining by Modified Approach of Promising Frequent Itemset Algorithm Based on Bucket Sort Approach**: It only scans updated database. This algorithm works on streamed and dense data. It also use the term promising itemset which are not frequent but can be frequent if the minimum support of an itemset is equal to the minsup found in the following formula:

\[
\text{MinPI}_{\text{DBUdb}} = \text{minsup}_{\text{DBUdb}} - (\text{maxSupp/totalsize} \times \text{inc_size})
\]

The algorithm creates two lists: frequent itemset list and promising itemset (PI) list. Once an increment comes, it finds whether a promising itemset becomes a frequent itemset using the bucket lists and if so, it removes the old PI and adds it to new frequent list.
Minimal Perfect Hashing and Pruning Based Incremental Mining Algorithm: DHP (Direct Hashing and Pruning) and MPIP (Multi-Phase Indexing and Pruning) algorithms both employ hash tables which reduces database access times. This algorithm focuses on improving C2 level, typically the most time consuming level and hence employs Hashing and Hash tables to reduce access time to databases. All candidates gets hashed into hash table with no collisions. Their support count can be calculated later using hash function. The hash table only gets updated by scanning only the updated parts and whole hash table is not re-created. The new items get added to the end of the hash table and candidate Itemsets get generated from hash functions efficiently in DHP algorithm. In MPIP each itemset has a unique address. Information in the hash table is enough to be able to find the frequency of each itemset.

In this algorithm IMPHP the following minimum perfect hashing function is used for hashing which avoids collisions.

\[ F_n (J_1, J_2) = \begin{cases} 1, & \text{for } J_1 = 1 \text{ and } J_2 = 2 \\ C_2^{j-1} + j_1, & \text{Otherwise} \end{cases} \]

(Dubey, 2015)

Summary
FUP & FUP2 scans the DB twice, and generates candidates which is not the most efficient one.

Feline & CAN Tree scans the DB exactly once. Feline uses swapping, merging and splitting of tree nodes which is a drawback. CAN tree is an improvement over this.
Some incremental algorithms are summarized. There are two groups: Apriori based and tree based approaches.

Apriori based algorithms: FUP requires more than 2 DB scan in the worst case and FUP2 requires two complete inefficient database scan.

Tree based algorithms: Feline and CAN tree require single scan of the DB. CATS tree tries to compact the DB as much as possible and uses frequency in ordering and hence, it swaps, merges, splits the tree nodes in order to make the tree more compact. It uses downward traversal to find mergeble paths. CAN tree uses user defined ordering instead of frequency dependent ordering. DHP and MPIP uses hash tables. This approach minimizes DB access time. (Dubey, 2015)

Overall, the paper compares different approaches to incremental AR mining.

The findings are summarized in the table below:

<table>
<thead>
<tr>
<th></th>
<th>Algorithm Name</th>
<th>Based on</th>
<th>Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>FUP</td>
<td>Apriori Based</td>
<td>Poor</td>
</tr>
<tr>
<td>2</td>
<td>FUP2</td>
<td>Candidate Separation</td>
<td>Poor</td>
</tr>
<tr>
<td>3</td>
<td>UWEP</td>
<td>Look Ahead pruning</td>
<td>Average</td>
</tr>
<tr>
<td>4</td>
<td>MAAP</td>
<td>Efficient low level item generation via starting from Large itemsets</td>
<td>Average</td>
</tr>
<tr>
<td>5</td>
<td>CATS TREE</td>
<td>Compact tree</td>
<td>Good</td>
</tr>
<tr>
<td>6</td>
<td>CAN TREE</td>
<td>Canonical tree</td>
<td>Good</td>
</tr>
<tr>
<td>7</td>
<td>Bucket Sort Approach</td>
<td>Storage Buckets</td>
<td>Good</td>
</tr>
<tr>
<td>8</td>
<td>DHP and MPIP</td>
<td>Perfect Hashing</td>
<td>Better</td>
</tr>
</tbody>
</table>

(Dubey, 2015)
Chapter 5

Parallel Techniques, Parallel Algorithms and Issues

Parallel Techniques

**MPI** (Message Passing Interface) is a parallel programming technique, tools and libraries developed and maintained by MPI-Forum which is comprise of researchers from academic community and industry. MPI has its standard library, portable system, which allows paralled programs to be written in several programming languages—such as, C, C++, Java and Fortran. There are many different implementation of MPI some of which is in public domain. For example, Open MPI is open source implementation of MPI geared towards High Performance Computing community. (Gropp & Lusk, 1996)

**CUDA** (Compute Unified Device Architecture) is a parallel programming software API and platform developed by NVIDIA corporation which allows programmers to write parallel programs to utilize GPU processors for general purpose computing. This way of using GPUs is also known as GPGPU processing.

CUDA allows multiple programming languages to be used to parallelize the programs, such as Fortran, C and C++. CUDA also supports OpenACC and OpenCL framework. (Nvidia, 2011)

**OpenMP** (Open Multi-Processing) is a parallel programming platform and shared memory architecture with API and set of compiler directives and a library (Dagum Leonardo, 1998) which supports C, C++ and fortran languages as well as many different operating systems and architectures including linux, aix, windows, , is one of the frequently used parallel programming technique to parallelize programs. OpenMP is also commonly used in FPM algorithms which
requires parallelization. It is maintained by a board—i.e. OpenMP Architecture Review Board (OpenMP ARB) which has members from top hardware and software companies such as Intel, AMD, HP, IBM and Cray. (Dagum Leonardo, 1998)

**C++ AMP:** Microsoft introduced C++ Accelerated Massive Parallelism—i.e. C++ AMP, to utilize GPU processor for general computing. It is available in Visual studio 2012 version and includes the environment to write parallel programs once the developer includes amp.h header file. C++ AMP allows programmers to write parallel_for and parallel_for_each function with ease—providing that the tasks are independent so that they can do their calculations in parallel. C++ AMP available for C++ language and it is started being implemented started from DirectX 11—Microsoft’s windows gaming API. Current API version is DirectX 12 which arrived with DirectX 12. Code written using C++ AMP is still portable as it is open specification and if the code is unable to be run on GPU the processing falls back to the available CPUs and continue running seamlessly.

C++ AMP code compiles and outputs to different standards as well—such as OpenCL which is maintained by Khronos Group, is a standard for heterogeneous computation systems such as CPU, GPUs and DSPs (digital signal processors) and FPGAs (field programming gate arrays) when they co-exists in a system for computational purposes. (Stone E. John, 2010)

**OpenACC** (Open Accelerators): OpenACC is a parallel programming standard created by CRAY, NVIDIA and a few other vendors to simplify writing parallel programmes which utilizes both CPUs and GPUs in a heterogeneous computer system. It has compiler directives like OpenMP and used with C, C++ and Fortran languages to write higher level programs using directives to utilize accelerators. Using compiler directives, the independently executable part of the code can be accelerated. The code can be started both on CPUs and GPUs. (Wienke Sandra, 2012)
Hadoop: Data in the universe is not only large but also unstructured. Thus, Hadoop is one of the solutions for processing unstructured data. (Bhokare, 2014).

Hadoop is an Apache project, open source implementation for distributed data storage and computational framework. It has two major components: a file system, HDFS, based on Google File System (GFS) and MapReduce for parallel computation. Hadoop is a master-slave architecture --master node (aka namenode) and slave nodes (aka datanodes). The master node is responsible for distributing data and computational work to the slave nodes. Master node partitions the data to datanodes and keeps track of the data nodes and the partitions they contain, and schedules their computation. Hadoop scales well with increasing data and processing needs. It uses clusters of commodity computers, which are computers with the same standards manufactured by many vendors hence they are ubiquitous hence cheaper, low end computers. Therefore, additional computers can be added to the clusters of hundreds or even thousands of computers with ease. (Bhokare, 2014)

Hadoop is originally written in Java but individual functions or tasks can be written in any language—such as C, C++, Pig, Python, Java, etc.

Hadoop is written for very large data processing situations containing terabytes to petabytes of data. It is optimized for these large data processing tasks. It is not efficient for processing small amounts of data. Hadoop is not meant for OLTP – Online transaction processing, where data access and retrieval is expected to be a lot faster —e.g. a computer in front of a bank teller expected to be respond very quickly to customer requests. Hence, Hadoop can not be compared to the relational database systems, such as SQL Server, specifically created for faster data access and processing.

Hadoop is optimized for big tasks, not from small tasks. It is also optimized for failure and recovery.

HDFS: HDFS —i.e. Hadoop File System, is based on Google’s GFS —Google File System paper for distributed file system. HDFS is optimized for large datasets and high throughput, it stores data
in big blocks, typically 64MB to 128 MB or even 256 MB in block size. Data is locally processed to minimize network traffic and I/O costs. Hadoop is very scalable which is achieved via data replication and fault tolerance. Failure of nodes due to both software or hardware is not an exception in a Hadoop system, instead considered a norm. Hence data gets replicated so that in the event of failures, there are other copies to be used and recover from the failures. The number of data replications is configured by the user but 3 replicas of the data is quite common. Hadoop automatically replicates and re-replicates the data blocks and immediately assigns data and computation job to a new node if a node fails. This fault tolerance is achieved through constant monitoring of datanodes by the master, which pings slave nodes within user-defined intervals to see whether they still have heart-beat – i.e. whether they are failed or not. If nothing is heard back from a node, master node will assume that it failed and will give that node’s task to another idle node.

**MapReduce:** Automatic, batch, distributed and parallel processing programming model which works on a large cluster of computers. It is based on Google’s Mapreduce paper. Mapreduce translates the work submitted by a client into maps and then reduce tasks, and distributes and schedules these tasks among the slave nodes. Input data first goes through map functions and then the reduce functions, producing the desired outcome (an output file) at the end of processing.

While distributed and parallel computation is a complex-job, Hadoop hides this complexity from the users and allows them to focus on the map and reduce functions which can be written in several different languages, such as Java, Python or Pig.

Mapreduce framework has map and reduce functions. In Mapreduce Apriori implementation map function takes input and produces a list of <key, value> pairs as its initial step and then this output is summarized by the reduce function to aggregate results. Mapreduce partitions the input and distributes it to the available nodes. Then each node finds the candidate 1-itemset as <key, value> pair – i.e. in the context of Apriori <1-itemset, count>. These output gets generated in every node as temporary file and then a global <1-itemset, count> gets generated by Data aggregation process in mapreduce model. Once all this data gets aggregated, shuffled and sorted,
the data in the temporary files gets partitioned again and passed to reduce function. Reduce function then selects the Itemsets which meet the minimum support count and prunes the itemsets which don’t qualify. (Bhokare, 2014)

**Mapreduce has several benefits:** Mapreduce can handle both structured and unstructured data as well as it handles parallelization, communication, task assignment, resource management, concurrency control on the distributed environment, with multiple computers, cores or processors seamlessly.

Mapreduce and HDFS can decide many things automatically based on the default values but these settings can also be configurable by the programmers. The algorithm will stop when there is no intermediate temporary file to process.

In a Hadoop framework, the data nodes, that is previously called Task tracker, are not aware of or concerned about the other data nodes. They simply get an input, process it, and write the output to a temp file which is picked up and managed by resource manager –i.e. previously called Job tracker. (Bhokare, 2014)

Hadoop and Mapreduce framework on Apriori in an improved Mapreduce implementation can efficiently reduces the number of candidate Itemsets by checking the Apriori property in the the previous iterations temporary files. If a subset is not present, then its supersets gets pruned. (Bhokare, 2014)
A Mapreduce example:

![MapReduce - Word Count Example Flow](image)

Taken from (https://www.mssqltips.com/sqlservertip/3222/big-data-basics--part-5--introduction-to-mapreduce/) (Sindol, 2014)

Parallel Algorithms

Earlier Approaches:
The following three serial algorithms are compared based on computation, synchronization, communication, memory usage and data for a particular domain. These are: Count distribution, data distribution and candidate distribution algorithms.
The tests on the paper “Parallel mining of association rules”, (Agrawal, 1996) , are done using IBM SP2 Share nothing architecture –i.e. each processor has its own memory and hard drive. The
communication occurs via MPI protocol by passing messages from one processor to another. Data is evenly divided between hard disks and attached to each processor. (Agrawal, 1996)

**Count distribution algorithm**: This algorithm prefers redundant computations by processors to communication costs. Data doesn't get moved or exchanged. Instead, only the local counts are communicated and exchanged.

Algorithm steps: each processor creates their own $C_1$ from the data partition they have. In this step they may have different $C_k$ as they all have a subset of the whole dataset. They create $C_1$ only for the data in their partition. They then synchronize their results and start level 2, the second iteration level, with the same $L_1$ to create $C_2$. After each iteration each processor creates its own complete set of candidate $C_k$ itemsets from $L_{k-1}$. Each processor processes a portion of the data and thus work in parallel asynchronously. At the end of each level, they need to consolidate and synchronize their results after getting the partial support count for each candidate itemset in their partition. Next, each processor shares its local support count for candidate itemsets with other processors and aggregates the results to find the $L_k$ itemsets until no longer they can continue, which each processor decides individually. Yet, their decision will be the same for each node because they have the same $C_k$ and after consolidating their counts, they will have the same $L_k$. The way the local counts are aggregated and the global counts determined for each candidate is as follows: each processor stores its local count for a candidate in a count array. The values for each candidate at the end of each level is aggregated to find the global support count. That is each node or processor submit their local counts for each candidate, these counts update the global support count for each $C_k$ in the array and the results are compared to the support threshold to find the $L_k$ Itemsets. (Agrawal, 1996)

**Data distribution algorithm** focuses on effective use of the total memory and scales up well as more and more processors are added for the computation. In data distribution algorithm if there are N processors, each processor will get and work on $C_k/N$ candidate sets – i.e. disjoint set of $C_k$. But since the candidates are disjoint set in each processors, once each pass is completed for
each $C_k$ in every processor, in one pass the count for all $C_k$ will be completed. While scaling is an advantage of this algorithm, for each pass the processors need to broadcast their actual data to other processors. This can be a big bottleneck if the data pipeline is not very fast. (Agrawal, 1996)

**Algorithm internals:** Processor $P_i$ creates the $C_k$ from $L_{k-1}$ but keeps only $1/N$th of the candidates and calculates support only for those candidates. In this implementation candidates assigned to different processors use a round-robin order. All other processors will do the same. Each processor broadcasts their local (actual) data so that each processor can complete the support count for those candidates. At the end of each level processors combine their $L_k$ and synchronize with each other. Ultimately, they all get the same $L_k$ since the processors have identical $L_k$, they terminate at the same level without communicating with each other any further.

One of the bottlenecks in this process occurs when processors send their local data as well as when they receive other processors' data. Notwithstanding, this can be done asynchronously to avoid network congestion. In addition, candidates are counted locally asynchronously with the existing data as well as the received data from the other processors. (Agrawal, 1996)

**Candidate distribution algorithm:** One of the limitations of count algorithms is that all candidates need to be generated in every processors and at the end of each level the counts get consolidated after which the frequent Itemsets are synched. In the data distribution algorithm's case, local data needs to be broadcasted for all transactions to every node. Frequent Itemsets also get synchronized at the end of each level. If processors have different amounts of workload, some may have to wait for that job to be completed by other processors and wait in idle state. Hence, candidate algorithms focus on minimizing synchronization overhead among the processors and handles load balancing as well. (Agrawal, 1996)
Algorithm internals: Candidate algorithms distributes the $L_{k-1}$ itemsets in a well balanced manner. While each processor has access to all $L_{k-1}$ frequent Itemsets, their portion of frequent Itemsets are recorded for that processor and each processor is responsible for processing their frequent itemsets. The algorithm partitions the data based on the frequent itemsets produced by each processor using domain knowledge during partitioning of candidate-itemsets and there is only one time data redistribution process. Thus by using both a portion of the frequent Itemsets, and the relevant data partition, each processor can process these independently to create its own mutually exclusive candidate Itemsets and proceed with counts independently and asynchronously.

During this process, each processor, depending on their speed, sends the all the $L_k$ that that they have to all of the other processors. This may make it possible that while a processor is processing $C_k$ Itemsets, it can receive $L_{k-1}$, $L_k$ or $L_{k+1}$ frequent itemset from the other nodes. This information is used for pruning during the candidate generation step.

Among these 3 algorithms, count distribution algorithm works the fastest and scales very well. (Agrawal, 1996)

New approaches– A Spark Approach:

*Yafim:*

Apriori is one of the most popular FIM algorithms which works well when the data is small. Even single nodes have enough computational power and memory to mine data efficiently. In "big data", however, data is terabytes or even petabytes large. While Apriori works well on the smaller data, we need much more resources to work on big data. Apriori is an iterative algorithm which requires multiple database access and scans to determine whether a given candidate itemset is frequent or not.
One of the ways to handle big data is by using parallel programming techniques. Many parallel programming techniques were proposed. One implements mapreduce on a Hadoop cluster. Hence, YAFIM, Yet Another FIM, proposes using SPARK, which is a memory only framework, to speed up this process. YAFIM moves data from the Hadoop file system (HDFS) to memory and then into RDD (Resilient Distributed dataset) objects—which are a distributed, read only, immutable collection of records. (Qiu, 2012)

**YAFIM algorithm internals:** YAFIM is a parallel algorithm using Spark has two basic steps: A flatMap() function, which loads data from hdfs into cluster memory and RDD objects and then finds the frequent 1-itemsets first. It then finds the k+1 frequent Itemsets iteratively using the k-itemsets. (Qiu, 2012)

The YAFIM algorithm is a two phase processing algorithm. In Phase 1, worker processes run the flatMap function to extract all items from the transactions. It then map function maps them to key value pair. Afterwards, the reduceByKey function counts (which is the aggregation) the frequency of each item and prunes items which are below the given threshold. At the end of phase 1 the algorithm will find frequent 1-itemsets. In Phase 2, k+1 frequent Itemsets are found. That is, L1 is read from RDD for (frequent 1-itemsets) and from frequent 1-itemsets candidate 2-itemsets are created. These candidates are stored using a collect function using a hash-tree to enable the efficient search. The Flatmap function uses a findItemsetsinHashTree function, to find Itemsets in the hash tree and the map function will maps these after which teh reduceByKey function aggregates the results and finds the frequent itemsets. Lk is found and stored at each level using an RDD as a itemset - count pair. Transaction RDDs are scanned to find the count of the candidate Itemsets, but the actual candidate list is stored and iterated using the hash tree. (Qiu, 2012)

During phase 1, transactions are stored as RDDs in memory in worker nodes which are distributed through the cluster's memory. The input dataset is loaded only once at the beginning. The same RDDs will be used for finding all frequent Itemsets in each iteration. After each iteration some data needs to get shared among worker nodes. In YAFIM this is done using a “broadcast variable” abstraction where a piece of shared data is sent to the slaves nodes once instead of with every
task. The broadcast variable concept is used to send static data which is used for the duration of the program. (Qiu, 2012)

YAFIM is an Apriori implementation on Spark. It uses cluster memory, and hence in-memory calculations using distributed storage and distributed computing that its big data framework and environment provides. Using Spark offers an 18 times speed up over the Apriori algorithms when compared to the algorithms implemented using mapreduce based Apriori. (Qiu, 2012)

A MapReduce -Based Approach:
Parallelization might improve performance in many cases but also introduces other costs: communication of local information with other nodes, data location or partition and how data is distributed, load balancing, job assignment and tracking. A mapreduce framework, using the file system HDFS such as the Hadoop file system, and Mapreduce programing models to parallelize jobs, is introduced to deal with these problems. Cloud computing provides cheap commodity computers which offers unlimited memory and computational power, and it is very suitable to mine bigdata. MapReduce parallelizes jobs, distributes data, tracks jobs, handles failed node issues, implements load balancing, creates multiple copies of data seamlessly without user inpur. It allows the user to focus on the actual problem to be solved. MapReduce re-assigns slow running jobs to idle but faster nodes. MapReduce uses master worker relationship –i.e. master node may assign and then may even re-assign a task if it considers it will be more efficient. (Farzanyar, 2013)

Both map and reduce functions can run in parallel. While this is true for MapReduce, Apriori is a level-wise algorithm, hence unless all the reduce jobs are complete cannot advance to a new level, thus slowing the process. For Apriori with n-levels, we may need n phase map reduce jobs where further levels will require that all the reduce jobs are completed. One way to improve speed is to reduce the number of MapReduce phases. The above material seems repetitious.

In Farzanyar, et. al. 2013, the implementation of MapReduce finds all frequent Itemsets in two MapReduce phases.
**Mahout** uses a parallel implementation of FP-tree. It can create group-dependent transactions in a single MapReduce phase. Each group is assigned to a particular reducer and the relevant transactions are also sent to the same reducer, which in turn creates local and conditional FP-trees based on these transactions. This implementation has two problems: data replication and load balancing. Data replication can cause slow communication which slows the mining process significantly. Load balancing is also a major problem particularly when the data size is very big. It may result in some nodes being idle while others are working harder and harder if it is not managed effectively. (Farzanyar, 2013)

Some algorithms in the literature require \( n \) number of MapReduce phases if the maximum length of a frequent itemset is equal to \( n \), the length of the maximum frequent itemset. (Farzanyar, 2013) For example if the maximum frequent itemset is of length 1, a 1-itemset, it will have one phase, 2-itemset maximal length requires two phases 2 and so on. One of the algorithms in the literature finds all frequent Itemsets using a single MapReduce pass but the amount of data it generates in the map phase grows exponentially with the size of the transactions. (Farzanyar, 2013)

MRApriori requires two MapReduce phases to find all frequent Itemsets. In phase one classic Apriori is run on the partitioned data which outputs partial frequent Itemsets and their counts. The reduce function is run after all maps are completed and it outputs a key and value. In phase two, this output file which includes all the partial large Itemsets, i.e. \( L_{\text{partial}} \) is added to the input source data of phase I and all partial key/value pairs get generated from these partitions where the key is the itemset and the value is the count of this itemset. Finally global frequent Itemsets are generated from the whole dataset.

This algorithm creates too many partial frequent Itemsets which results in a lot of communication between nodes and takes too long to execute, hence, it is not that efficient.

An improvement on this algorithm is made through pruning which is done by reducing the number of candidate itemset to be generated and hence removing some partial frequent itemsets. (Farzanyar, 2013) but still this algorithm is not much better due to the large amount of
communication needed and the large execution time in phase 2. Additionally there are many partially created frequent Itemsets which also affects its efficiency.

This paper extends IMRApriori algorithm which uses a pruning technique to reduce the number of partially created frequent Itemsets. The main additional feature of this algorithm compared to MRApriori is that the reducer keeps track of the number of mappers which output a particular partially frequent itemset and then it estimates the global support count of a particular partial frequent itemset using the following formula:

\[ X_{\text{maxGlobalSupportCount}} = (X_{\text{supportCount}} + (((s \times D_i) - 1) \times (M - N_x))) \]

Where M is the total of mappers, Nx is the number of mappers which output X as a frequent itemset, and Di is the number of transactions. (Farzanyar, 2013)

SX’s global support count needs to be greater than the support – i.e. s \times D (total number of transactions.) and which needs to be in the Large partially frequent itemset list \( L_{\text{partial}} \) to be able to participate in phase 2. Other Itemsets are globally infrequent and therefore are pruned in phase 1.

IMRApriori is still not efficient due to the high level of communication needed and the high execution time of the Mappers.

**Accelerating the Frequent Itemset Mining Process Algorithm**

This algorithm, like IMRApriori has two phases. In the first phase it runs Apriori on a split dataset and finds the partial frequent Itemsets in each of of M splits. Each mapper function in each split has an identifier associated with that split. Thus the output of the mapper functions are key/value pairs i.e. <itemset, supportCount>, of a particular mapper i of a particular split and is then kept and passed to reducer functions. When all maps are finished the reduce tasks start. Reducers count the total support of a particular partial itemset and remove them from the partial frequent itemset list if their global support count is less than the support threshold.
Since reducers receive the output of phase 1 from the mappers with mapper and split information, in phase 2, those Itemsets which are already counted in splits don’t need to be counted again.
Therefore the preprocessing in phase 1 significantly reduces the the work needed to be done in phase two, reducing communication overhead and execution time. (Farzanyar, 2013)

_A Parallel Approach for Frequent Itemset Mining of Incremental Data_

In (Bhadane, 2012) the IMBT data structure is proposed to parallelize the FIM framework by using multiple machines to deal with the problems processing power and memory. The focus of this paper is on efficient processing and minimal communication. It first divides the input file into partitions. Each partition is processed in parallel. It uses an IMBT data structure and there is no support threshold during IMBT tree creation. IMBT gets created on local nodes and Itemset frequency is count in these local nodes. The results are merged and the superset of the partition frequent Itemsets are found for the entire database by calculating minimum support to the supersets of the partitioned itemsets to find the frequent Itemsets. Mapreduce works well for this overall process. (Bhadane, 2012)

IMBT data structure example with original transaction and increment:

Original DB:
4, 5, 7
Increment:
0, 6, 7, 8
0, 5, 7

The IMBT after all transactions are processed:
A Parallel Implementation of Apriori Algorithm:

The paper is about implementing Apriori in a parallel manner using OpenMP, a shared memory language. OpenMP is a relatively easier parallel programming model that was implemented on a quad core computer running Ubuntu Linux. This paper showed that a parallel implementation is faster than a serial one. The algorithm was tested on four standard datasets, run with support varying from 30%-70% The parallel implementation was observed to perform consistently better than the serial implementation. (Korde N. S., 2014)
Issues in Parallel programming Environment:
When writing parallel programs many issues can arise – depending of the system architecture, whether it is shared nothing or shared memory system, whether the data is stored locally or in a distributed environment, what data or information is communicated, shared and how, whether the data is replicated, how tasks are given, managed, how failures are handled and managed, how load or amount of work for different processors or nodes are balanced.

A robust system, such as Hadoop, tolerates some redundancy, so that it can recover from failures quickly. Likewise, some parallel programs may do redundant job in the local nodes, so that they minimize communication overhead.

Hence, many issues may arise due to reasons mentioned above. An efficient and scalable parallel program in a distributed environments most likely would need to consider and handle most of these issues efficiently.
Chapter 6

Proposal:
There are still many issues to address in frequent, incremental frequent and parallel and incremental frequent itemset mining. Efficiency of algorithms are significantly affected by each of the following:

- the length and density of data
- which data structures are used
- how and whether the infrequent itemsets are pruned
- how or whether candidates are generated
- how data is read, searched, and compressed
- how itemsets are counted
- whether there are multiple support levels
- the threshold value for the support level
- whether the data fits in memory
- whether it is distributed, distributed and computed in local nodes
- whether the system is a shared nothing architecture or a shared memory architecture
- how the parallelization is implemented
- how the implementation is done
- which if any optimization techniques are used.

Parallel programming methods and architecture have been developed over the last 20 years. Particularly GPU usage for computation can have dramatic improvements in the performance. Another development occurred in the recent years is MapReduce. MapReduce is also another powerfull programming technique which can also use GPUs effectively. (Upadhyayya, 2013)

The proliferation of petabytes of data has made finding efficient methods for analyzing these large sets of data important. Among others, Frequent Pattern Mining (FPM) is a common
technique used in discovering interesting patterns in data acquired in various application domains. Many algorithms have been proposed for searching for frequent patterns. The search space combinatorically explodes as the size of source data increases. Incremental algorithms further improve the efficiency of frequent pattern mining. Simply using more powerful computers, or even super-computers to handle the large data sets is not sufficient. However, commodity hardware and a distributed data storage environment allows parallel computations using Map Reduce framework in discovering frequent patterns in large datasets.

The proposed research will explore, investigate and devise incremental approaches with parallel computation to gain efficiency in the UWEP and NUWEP algorithms. We also plan to use the devised algorithms in exploring emerging patterns in data accumulated over periods of time in increments.

Bibliography


C.Borgelt. (n.d.). *Frequent Pattern Mining (FPM)*.


Goethals, B. (2003). *Survey on Frequent Pattern Mining*.  


Ma, W. &. (n.d.). A translation system for enabling data mining applications on GPUs. *Proceedings of the 23rd international conference on Supercomputing* (pp. 400 - 409). ACM.


