

BACHELOR OF COMPUTER SCIENCE (HONS)

BCS3222 – DATA MINING AND DATA WAREHOUSING

**Individual Assignment 2**

**Instructions:**

* Answer **ALL** questions
* Marks will be awarded for good presentation and thoroughness in your approach
* **NO** marks will be awarded for the entire assignment if any part of it is found to be copied directly from printed materials or from another student.
* Submit your assignment in **hardcopy** (printed) and **softcopy** (*upload it on e-QIU*)
* Complete this cover sheet (page 1 and 2) and attach it to your assignment.

**Student’s Particulars:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Matric No** | **:** | 202007004078 | **Intake / Semester** | **:** | 202004 |
| **Student’s Name** | **:** | SAVITHA SAN MARKAN | |  | |
| **Lecturer’s Name** | **:** | DR ANBUSELVAN | |  | |
| **Due Date** | **:** | 29/9/2022 | **Submission Date** | **:** | 29/9/2022 |
| **Total number of pages including this cover page** | | | | **:** | 26 |

**Official Use:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | **Examiner’s Comment** | | |
|  |  |  | | |
| **Marks Awarded** | **:** | **/ 100** | **Overall Marks (20%):** |  |

**Student declaration:**

*I declare that:*

* *I understand what is meant by plagiarism*
* *The implication of plagiarism has been explained to me by my institution*
* *This assignment is all my own work and I have acknowledged any use of the published and unpublished works of other people.*

Student’s signature: ………………………. Date: 30/9/2022

**Marks Sheet:**

|  |  |  |
| --- | --- | --- |
|  | **Marks Table** |  |
| Questions | Marks Allocated | Marks Awarded |
| Program Development | 80 |  |
| Report | 20 |  |
| **Total** | **100** |  |

|  |
| --- |
| **/100** |

**Overall Total:**

Chapter 1: Introduction

* Introduction
* Problem Statement
* Objectives
* Scope

Chapter 2: Literature Review

* Literature on the transactional data.
* Literature on the frequent pattern mining techniques.
* Literature on the development platform and tools.

Chapter 3: Methodology

* Structure and organization of the dataset with description.
* Design and development of the Apriori module.
* Design and development of the Frequent Pattern Growth module.

Chapter 4: Discussion and Analysis

* The detailed description and explanation for each of the stages of the frequent pattern mining operations, and the resulting frequent patterns with analysis and justification.
* The detailed description and explanation for the association rules generating process with analysis and justification.

Chapter 5: Conclusion

* Conclusion
* Strengths and weaknesses of the developed frequent pattern mining program.

# Chapter 1: Introduction

The size of databases has drastically grown in recent years. This has generated an increase in interest in the creation of technologies that can automatically extract knowledge from data. Data mining, also known as Knowledge Discovery in Databases, is the name given to a branch of study that deals with the automatic discovery of implicit knowledge or information inside databases. The implicit information within databases, and mainly the interesting association relationships among sets of objects, that lead to association rules, may disclose useful patterns for decision support, financial forecast, marketing policies, even medical diagnosis and many other applications and so the recent data mining studies paid a lot of attention to this fact.

This chapter introduces a data mining-based approach known as association analysis, which is used for extracting interesting relationships hidden in large transaction data sets. In order to forecast the availability of specific items in a transaction based on the presence of other items, the extracted relationships are expressed as association rules. For instance, the rule that follows shows that many customers who purchase cereal also frequently purchase milk.

*Cereal —> Milk*

The association rules may be helpful in a variety of applications, including market transaction analysis, store layout and product promotions, university course enrolment analysis, customer behaviour analysis in retailing, catalogue design, word occurrence in text documents, user visits to websites, stock transactions, tumour detection in digital mammography, building statistical thesauri from text databases, and finding related images from massive image databases.

A very influential association rule mining algorithm, Apriori, has been developed for rule mining in large transaction databases. Many other algorithms developed are derivative and/or extensions of this algorithm. A major step forward in improving the performances of these algorithms was made by the introduction of a novel, compact data structure, referred to as frequent pattern tree, or FP-tree, and the associated mining algorithm, FPgrowth.

THE APRIORI ALGORITHM

*“If an itemset is frequent, then all of its subsets must also be frequent. Conversely if an itemset is infrequent then all of its supersets must be infrequent too. “*

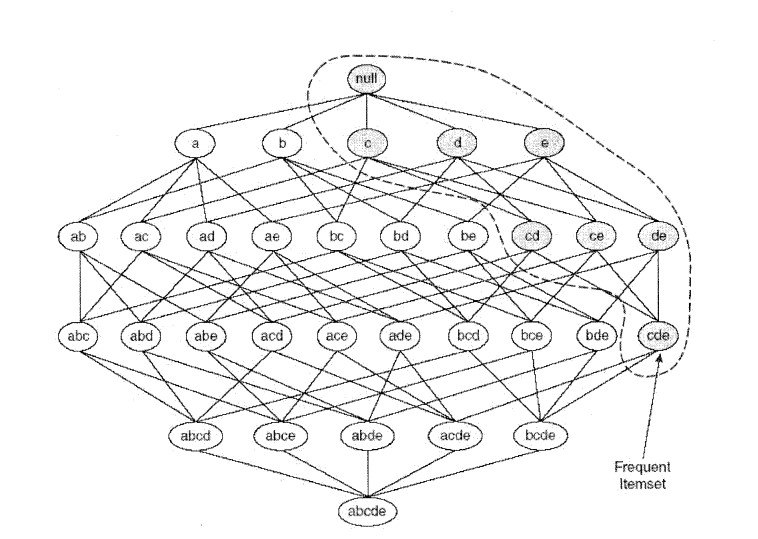


Figure 1 An Illustration of the Apriori principle. If {c, d, e} is frequent, then all subsets of this itemset are frequent.

To illustrate the idea behind the Apriori principle, consider the itemset lattice shown in Figure 1. Suppose {c, d, e} is a frequent itemset. Clearly, any transaction that contains {c, d, e} must also contain its subsets, {c, d}, {c, e}, {d, e}, {c}, {d} and {e}. As a result, if {c, d, e} is frequent, then all subsets of {c, d, e} (i.e. the shaded itemsets in this figure) must also be frequent.

Conversely, if an itemset such as {a, b} is infrequent, then all of its supersets must be infrequent too. As illustrated in Figure 1, the entire sub graph containing the supersets of {a, b} can be pruned immediately once {a. b} is found to be infrequent. This strategy of trimming the exponential search space based on the support measure is known as support-based pruning. Such a pruning strategy is made possible by a key property of the support measure, namely, that the support for an itemset never exceeds the support for its subsets. This property is also known as the anti-monotone property of the support measure.

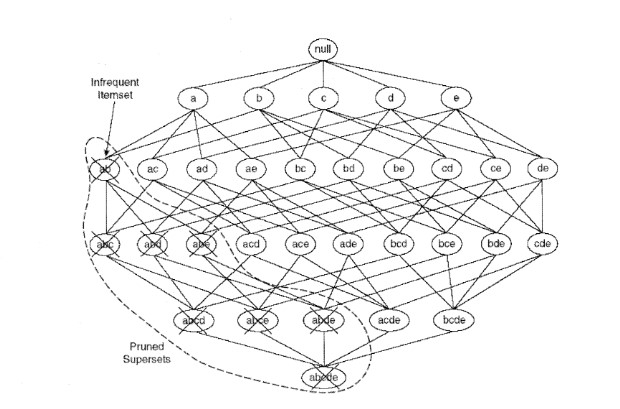


Figure 2 An illustration of support-based pruning. If {a, b} is infrequent, then all supersets of {a, b} are infrequent.

Let I be a set of items, and J = 2’ be the power set of I. A measure f is monotone if

v z ,y e y : (^ Ç y ) -4^ /(% )< /(y ),

which means that if X is a subset of V, then f(X) must not exceed f(Y). On the other hand, f is anti-monotone if

v x , y e y then X Ç y -> / ( x ) > /(y )

which means that if X is a subset of Y, then f(Y) must not exceed f(X).

FP-GROWTH

FP-tree(Frequent Pattern tree) is the data structure of the FP-growth algorithm for mining frequent itemsets from a database by using association rules. It’s a perfect alternative to the apriori algorithm.

All created itemsets' supports are counted by FP-growth using a combined vertical and horizontal database configuration for main memory database storage. It maintains the database's real transactions in a trie structure rather than saving the covers for each item, and each item has a linked list that traverses all transactions that contain it. This novel data structure, known as the FP-tree (Frequent Pattern tree), is made in the manner described follows. For the same reasons as before, we once more arrange the database's contents in increasing order of support. Create the "null"-labeled root node of the tree first. The items are processed for each transaction in the database in reverse order (thus, support descending), and a branch is made for each transaction. A counter that counts the number of transactions that share each node in the FP-tree is also stored at each node. The count of each node along the common prefix is increased by 1 specifically when taking into account the branch that will be added for a transaction, and nodes for the transaction's items that follow the prefix are formed and linked appropriately. Furthermore, an item header table is constructed with each item pointing to its instances in the tree by a series of node-links. This header table's items each store their support as well. The more often occurring items are grouped closer to the root of the FP-tree and are therefore more likely to be shared, so it is believed that doing things this way will result in the FP-tree representation of the database being kept as small as feasible.

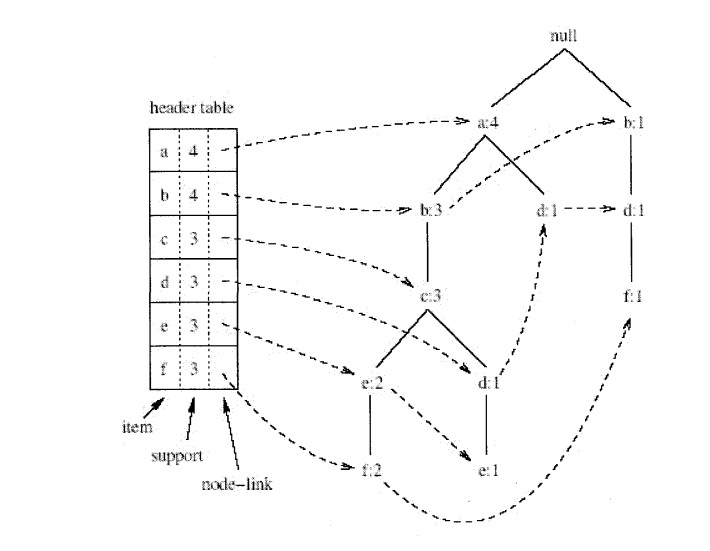


Figure 3 An example of an FP-tree

FP-growth algorithm uses some additional steps to maintain the FP-tree structure during the recursion steps. More specifically, in order to generate for every i g I all frequent itemsets in F[{i}](D, a), FP-growth creates the so called iprojected database of D. The FP-growth algorithm is given Figure 3

# Problem Statement

The problem of mining association rules is to generate all association rules that have support and confidence greater than the user-specified minsup and minconf, respectively. Formally, the problem is generating all association rules X ->F, where



The problem of finding association rules can be decomposed into two subproblems

* Generate all combinations of items with fractional transactions support (i e g certain threshold, called minsup.

* Use the frequent itemsets to generate association rules. For every frequent itemset I, find all non empty subsets of I. For every such subset a, output a rule of the form a-^(l-a) if the ratio of the support(l) to Support(a) is at least minconf. If an itemset is found to be frequent in the first step, the support of that itemset should be maintained in order to compute the confidence of the rule in the second step.

Finding all frequent itemsets is the more computationally intensive of the two subproblems, and research efforts have focused on finding more efficient algorithms. Some methods search for only a subset of all frequent itemsets that has the property of summarizing or of allowing to infer the information on all frequent itemsets.

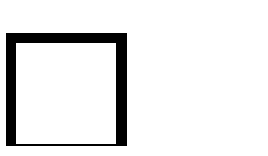
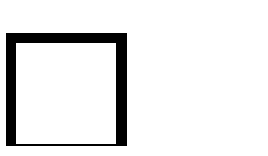
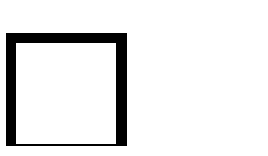
The rule generation stage is usually much more efficient than the first stage, but has the drawback of possibly producing a very large number of rules, more than a human analyst could handle. It is not unusual to obtain tens of thousands of rules. To address this problem, researchers have focused either on defining measure for the interesting ness or degree of surprise of the rule, or on determining subsets of rules from which all other rules can be inferred using a set of interference rules.

Note that both these stages have exponential complexity in their worst case scenarios. In the case of finding the frequent itemsets, the complexity is in terms of the number of items. Indeed, given n items, we have 2" itemsets, all of which could be frequent. In the case of generating the association rules, the complexity is determined by the number of frequent itemsets and by their size, because for a k-itemset we can generate 2\*-2 association rules. This value was obtained by considering the number of possible antecedents of the rule and by excluding the case of the empty set and that of the maximal set, which would imply an empty antecedent and, respectively, an empty consequent. Although both subproblems have exponential complexity, in practice the first one is more costly because for the computation of the support of an itemset we need to access the transactions of T and verify the inclusion of the itemset in each transaction.

# Objectives

Association rule mining (ARM) is used for finding the frequent patterns between the item-sets. Its goal is extracting interesting association rules, frequent patterns among item-sets in the database. For Example, In a Laptop store in Delhi, 90 % customer who purchased a laptop, they also buy a mouse. The statement of ARM problem was firstly specified by R. Agrawal. Let E = E1, E2... En be a set or group of n different attributes, B be the transaction such that B E, G be a database with different transactions Bs.

An association rule is defined as M N, where M, N E are sets of items, and M N = . M is called antecedent. N is named consequent. The rule means M implies N. Support and Confidence are the two basic measures for association rules. These measures are used for the evolution of association rule interestingness. There are two thresholds that is minimum support and minimum confidence which can define by the users. Support: The support is defined as the probability of task with relevant data transactions for which the pattern is true [2].

Support (A B) = P(A B) Confidence: The confidence is defined in terms of measure of certainty associated with each discovered pattern [2]. Confidence (A B) = P(B|A) If rules that satisfy both a minimal support threshold (and minsup) and a minimal confidence threshold (or minconf) are called strong association rules [1]

# Scope

In association rule mining, a given data collection is examined for intriguing connections between its constituent parts. The development of a systematic approach that uses the provided data set and identifies associations between the various variables is one goal of association rule mining. Finding associations between things from a set of transactions that contain a set of items is the aim of an association rule.

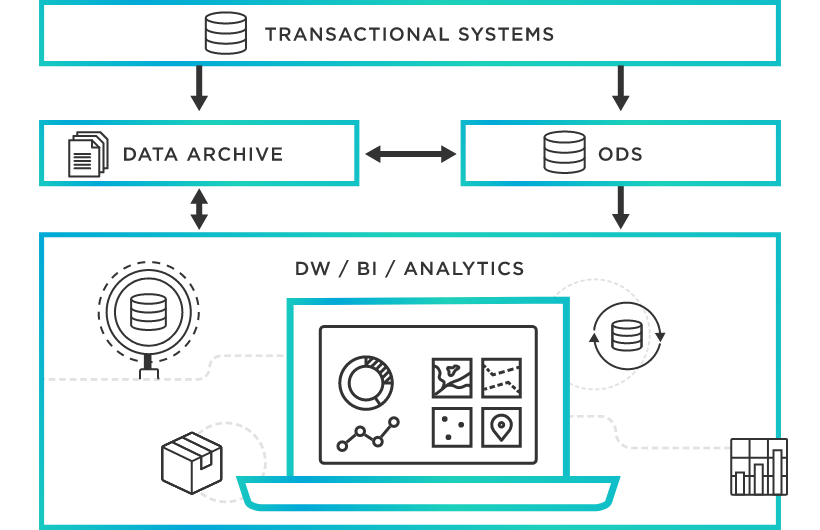
In association mining, several innovative and effective algorithms have been created. Additionally, new suggestions for algorithms were made to reduce the amount of time needed to generate association rules or frequent itemsets. A crucial factor in data mining is the mining algorithm's efficiency.

In the work that is the subject of this thesis, we have analysed and explained the implementations of the well-known association rule algorithms. Study focuses on Apriori and FPGrowth algorithms. The similarities and differences among the algorithms have been demonstrated. The benefits and drawbacks of these algorithms are carefully examined. Almost identical frequent item sets and association rules were generated by all algorithms, however they took different amounts of time to run. The time it took for the

algorithms to run on the datasets with various minimum support values served as the performance indicator.

# Chapter 2: Literature Review Literature on the transactional data

Information that is gathered from transactions is referred to as transactional data. It keeps track of the date and location of the transaction, the time it took place, the price ranges of the goods purchased, the mode of payment used, any discounts applied, and other quantities and characteristics related to the transaction. Typically, transactional data is recorded at the point of sale. In other words, transactional data is the information produced by different business applications as they carry out or support routine purchasing and selling activities. Data from point-of-sale servers, security programmes, ATMs, and payment gateways, coming from any device imaginable used to execute a financial transaction, is woven together into a vast and complex network.



The resulting data is frequently difficult to understand or contains extraneous information, such as characters, symbols, or numbers, because to the sheer volume of touchpoints. In order to conduct downstream analytics, avoid costly customer support calls, or find the truth behind fraud accusations, transactional data must be captured accurately. Each transaction that takes

place is given a special identification number, or "trans ID," along with a list of the products that are a part of the transaction.

Transactional data differs from the other main data categories, which are:

* Analytical Data: Analytical data, as the name suggests, comes into being through calculations or analyses run on the transactional data.

* Master Data: Master data represents the actual, critical business objects upon which said transactions are performed, also taking into account the parameters on which data analysis is conducted.

Who Uses Transactional Data in an Organization?

The primary handlers of transactional data in a company are the data analytics team and the operational information technology team. There are two advantages:

1. Real-time transaction monitoring is a function of information technology operations. They track down, identify, and resolve any performance problems that could result in significant service interruptions using the data and streaming products. This helps you save time and money.

1. Real-time transaction data is used by business managers and data analysts to comprehend customer behaviour and determine how their goods and services are utilised. In this case, the transaction data produces insightful information that enhances the service provided. Transaction data helps businesses increase profitability, provide better customer experiences, and attract new customers.

Advantages of Transactional Data

* Optimized credit and debit card management
* Rapid detection of fraudulent transactions
* Improved threat detection
* More accessible insights

# Literature on the frequent pattern mining techniques

The goal of the data mining subject known as frequent pattern mining is to extract common item sets from a database. Many Data Mining activities depend on frequent itemsets, which are also connected to intriguing data patterns like Association Rules.

VARIOUS FREQUENT PATTERN MINING TECHNIQUE

1. Apriori Algorithm

Apriori was one of the earliest algorithms to develop for frequent itemset mining. It uses a database with a horizontal layout. Level wise search is an iterative strategy used by this algorithm. To limit the search, it makes use of a crucial attribute called the Apriori property.

Apriori employs a two-step process.

* + It connects two sets of things in the first phase that have k-1 common items in the kth pass. The candidate set Ck is the collection that results from the first run, which starts with a single item.
  + The method counts how often each candidate set occurs and prunes any uncommon itemsets in the second stage. When no more extension is identified, the algorithm finishes.

1. FP-growth algorithm

A tree-based technique called frequent pattern growth, commonly known as FP-growth, is used to mine common patterns in databases. Databases are kept in the FP-growth algorithm in the form of a small data structure called an FP-Tree. It employs the divide and conquer strategy. It uses the FP tree to mine frequent patterns rather than a candidate frequent itemset. In the first phase, a list of frequently used things is created and arranged in decreasing order of support. The node structure serves as the representation for this list. The item name, support count, and a pointer to a node in the tree with the same item name are all contained in each node in the FP tree, with the exception of the root node.

The FP tree is constructed using these nodes. During FP tree development, shared prefixes can be used. According to their support, the pathways from root to leaf nodes are ordered in a non-increasing manner. After the FP tree has been created, common patterns are then retrieved starting with the leaf nodes. Recursively processing each prefix path subtree identifies frequent itemsets. Due to its efficient storage and projected layout, FP Growth requires the least amount of RAM. If we take into account transactions including a specific itemset and then eliminate that itemset from all transactions, we can construct a conditional FP tree variation.

COMPARISON OF VARIOUS FREQUENT PATTERN MINING TECHNIQUES

The effectiveness of various FPM algorithms has been evaluated by comparing them against three criteria: the number of database scans necessary to generate a frequent itemset, the candidate generation method employed, and the algorithm's sensitivity to changes in user parameters, such as support.

The Apriori approach uses an effective technique for reducing the candidate item sets, but generating candidate item sets takes a lot of computation time and many dataset scans. By enabling on-the-fly addition and deletion of counted itemsets, DIC offers a great deal of flexibility, but it also consumes a lot of processing effort.

Only two database scans are enough for the FP-growth algorithm to produce frequent patterns. With this approach, the complete database is represented by a small tree structure. Because it doesn't need candidate generation, the computation time is cut down.

# Literature on the development platform and tools

Implementation of Apriori and FP-growth algorithm using Python & Jupiter Notebook

Python

* Python is commonly used for developing websites and software, task automation, data analysis, and data visualization. Since it's relatively easy to learn, Python has been adopted by many non-programmers such as accountants and scientists, for a variety of everyday tasks, like organizing finances.

Main things can be done using Python include:

* Data analysis and machine learning
* Web development
* Automation or scripting
* Software testing and prototyping
* Everyday tasks

Python (PyCharm)

* PyCharm is a dedicated Python Integrated Development Environment (IDE) providing a wide range of essential tools for Python developers, tightly integrated to create a convenient environment for productive Python, web, and data science development.

Jupyter Notebook

* You may create and share documents with live code, equations, visualisations, and text using the open source web tool known as the Jupyter Notebook.

# Chapter 3: Methodology

**NOTE\* DESIGN WILL BE PRESENTED DURING PRESENTATION**

**Structure and organization of the dataset with description.**

(1) 2015TT10917.sh:

Executing the command: sh 2015TT10917.sh <inputfile> X -apriori <filename> generates a file <filename>.txt containing the frequent itemsets at >=X% support threshold with the Apriori algorithm. Similarly, executing the command:sh 2015TT10917.sh

<inputfile> X -fptree <filename> generates a file<filename>.txt containing the frequent itemsets using FP-tree algorithm.

X is in percentage and not the absolute count. Our implementations ensure that the transactions are not loaded into main memory.

However, the frequent patterns and candidate sets are stored in memory.

<filename>.txt follows the following format:

1. Each frequent itemset is on a new line.
2. The items in each itemset are space separated.

**Design and development of the Frequent Pattern Growth module**

(2) FPTree.py

Node class is the node element used to form fp-tree. It contains: item: element name - integer in this case count: count of the item parent: its parent in fptree

next: pointer to next node in fptree containing same element chidren: list of its children

FPTree class contains the implementation of the fp-tree and fp-growth for mining frequent itemsets. It contains:

root: root of the tree - null in this case

HeaderTable: Hash map for accessing the nodes of a particular element in the tree

flist: List of all elements in decreasing order of their support minSup: minimum support for this tree in percentage

Functions:

makeHashmap: takes transactions and makes a hashmap of support counts corresponding to each item in DB sortHashmap: generates flist from hashmap

RemoveNonFrequent: removes non frequent items from every transaction and arranges the items in each transaction as per flist

insert: given a transaction, inserts a path in the fp tree and also links the node to header table

constructTree: constructs the initial fp-tree from datafile

buildCPB: builds the conditional pattern base given an item and fp tree generateFPT: generates conditional fp tree given the CPB

FPgrowth: implements the fp-growth algorithm and mines all the frequent item sets

getAllComb: returns all combinations of a list

**Design and development of the Apriori module**

(3) Apriori.py

Variable names and their purposes:

old\_item\_set : stores last iteration frequent itemsets for candidate generation. curr\_item\_set : current candidate item\_set. counter : for storing frequency of candidate item\_sets.

is\_mergable : checks if two item\_set are mergable for generating candidate item\_set of greater size.

is\_pruned : checks if candidate itemset was pruned.

transaction : Stores the currently accessed transaction set from dataset. support,support\_perc : Stores support threshold and its precentage. input\_file,output\_file : names of input and output files.

Functions

bin\_search : returns true if a item\_set exists in old\_item\_set generate\_F1 : generates single size frequent itemsets by itearting in dataset. generateFreqItemSets : generates iteratively all the frequent itemset.

Apriori.java implements apriori algorithm for mining frequent itemsets. Function generateFreqItemSets takes input file name,

output file name and support percentage, reads data set from input file and writes all the frequent item sets in output file with given support.

1. Test.py

Contains the main function. Executes fp-growth or apriori depending on the arguments given. Call to this is made by the bash file

1. plot.py

generates a plot using matplotlib where the x-axis varies the support threshold and y-axis shows the corresponding running times.

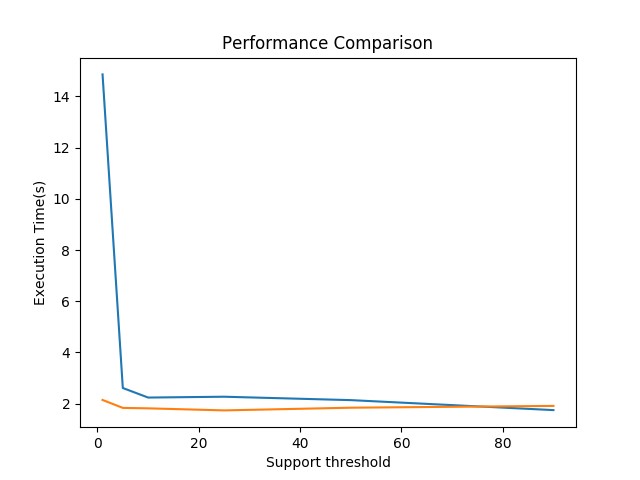
It plots the running times of apriori and fptree at support thresholds of 1%, 5%, 10%, 25%, 50%, and 90%

The performance of Apriori algorithm is compared with FP-tree in the report.

Executing the command: sh 2015TT10917.sh <inputfile> -plot generates a plot using plot.py

The results obtained are further explained in the report.

# Chapter 4: Discussion and Analysis The detailed description and explanation for each of the stages of the frequent pattern mining operations, and the resulting frequent patterns with analysis and justification



# Conclusion

We observe that the running time of FP-growth is lesser than Apriori algorithm. Also, with increase in support threshold, the running time of both Apriori and FP-growth decrease. The decrease for FP-growth is less as compared to Apriori.

In Apriori, the pattern matching operation of comparing candidate sets with transactions becomes expensive due to the increasing size of candidate sets. As large number of candidates are generated, they require more memory space. The breadth-first search approach can be quite costly in terms of memory as it requires at any moment to keep in the worst case all k and k − 1 itemsets in memory (for k > 1).

On the other hand, for fp-tree, there is no candidate generation and hence it requires less memory. It removes the need to calculate the pairs to be counted (which requires expensive computation). The FP-Growth algorithm stores in memory a compact version of the database. A major advantage of fpgrowth algorithm is that it only explores the frequent itemsets in the search space, thus avoiding considering many itemsets not appearing in the database, i.e., infrequent itemsets.

With the increase in number of items, more search space is needed and I/O will also increase for Apriori. Apriori requires more database scans as compared to fp-growth; whenever new candidates are generated, DB scan is required. However, FP- tree requires only 2 DB scans. Thus, fp-growth is much better computationally.

Another limitation of Apriori is that the time complexity is O(m^2 \* n), where m is the number of distinct items and n is the number of transactions. On the other hand, fp-growth algorithm is O(n) which is much faster than Apriori.

**Turnitin Report**

