

STATUS OF THESIS

Title of thesis An Efficient Frequent Itemset Mining Algorithm Using the FP-DB Approach

I CHEE CHIN HOONG

hereby allow my thesis to be placed at the Information Resource Center (IRC) of Universiti Teknologi PETRONAS (UTP) with the following conditions:

1. The thesis becomes the property of UTP
2. The IRC of UTP may make copies of the thesis for academic purposes only.
3. This thesis is classified as

Confidential

Non-confidential

If this thesis is confidential, please state the reason:

\_\_\_\_\_  
\_\_\_\_\_

The contents of the thesis will remain confidential for \_\_\_\_\_ years.

Remarks on disclosure:

\_\_\_\_\_  
\_\_\_\_\_

Endorsed by

\_\_\_\_\_  
Signature of Author

\_\_\_\_\_  
Signature of Supervisor

Permanent address:

1742, Jalan Emas, Taman Bandar Baru,  
31900 Kampar, Perak.

Name of Supervisor

AP Dr. Jafreezal bin Jaafar

Date : \_\_\_\_\_

Date : \_\_\_\_\_

UNIVERSITI TEKNOLOGI PETRONAS

AN EFFICIENT FREQUENT ITEMSET MINING ALGORITHM  
USING THE FP-DB APPROACH

by

CHEE CHIN HOONG

The undersigned certify that they have read, and recommend to the Postgraduate Studies Programme for acceptance of this thesis for the fulfillment of the requirements for the degree stated.

Signature:

\_\_\_\_\_

Main Supervisor:

AP Dr. Jafreezal bin Jaafar

\_\_\_\_\_

Signature:

\_\_\_\_\_

Co-Supervisor:

AP Dr. Izzatdin bin Abdul Aziz

\_\_\_\_\_

Signature:

\_\_\_\_\_

Head of Department:

Dr. Aliza binti Sarlan

\_\_\_\_\_

Date:

\_\_\_\_\_

AN EFFICIENT FREQUENT ITEMSET MINING ALGORITHM  
USING THE FP-DB APPROACH

by

CHEE CHIN HOONG

A Thesis

Submitted to the Postgraduate Studies Programme

as a Requirement for the Degree of

DOCTOR OF PHILOSOPHY  
INFORMATION TECHNOLOGY  
UNIVERSITI TEKNOLOGI PETRONAS  
BANDAR SERI ISKANDAR,  
PERAK

MARCH 2020

DECLARATION OF THESIS

Title of thesis

AN EFFICIENT FREQUENT ITEMSET MINING ALGORITHM  
USING THE FP-DB APPROACH

I CHEE CHIN HOONG

hereby declare that the thesis is based on my original work except for quotations and citations which have been duly acknowledged. I also declare that it has not been previously or concurrently submitted for any other degree at UTP or other institutions.

Witnessed by

\_\_\_\_\_  
Signature of Author

\_\_\_\_\_  
Signature of Supervisor

Permanent address:

1742, Jalan Emas, Taman Bandar Baru,  
31900 Kampar, Perak.

Name of Supervisor

AP Dr. Jafreezal bin Jaafar

Date : \_\_\_\_\_

Date : \_\_\_\_\_

## ACKNOWLEDGEMENTS

First and foremost, I would like to thank God for His blessings in completing my research. Then, I would like to thank my supervisor, AP Dr. Jafreezal bin Jaafar, for his guidance and advice throughout this research. In addition, I would also like to thank my co-supervisor, AP Dr. Izzatdin bin Abdul Aziz, who has also given me advice and guidance.

Apart from this, I would like to thank the Education Ministry of Malaysia and Universiti Teknologi PETRONAS (UTP) for providing me the financial sponsorship to support my studies throughout the past few years.

Last but not least, I would like to thank my wife, Joanne Yew, for all her love, support and prayer. I am also grateful to my family for their encouragement and support during the entire journey of this research.

## ABSTRACT

Data mining provides insights that offer vast benefits such as increased revenue, cost cutting, and improved competitive advantage. However, the hidden patterns of the frequent itemsets become more time consuming to be mined when the amount of data is big. Moreover, significant memory consumption is needed in mining the hidden patterns of the frequent itemsets due to its enormous combinations that are required to be processed. Most of the current algorithms are still facing these two problems because the frequent itemsets are mined into the main memory and the storage space is quite limited for mining the entire data set. Therefore, an efficient algorithm is necessary to be constructed for mining the hidden patterns of the frequent itemsets especially when the amount of data is big. Frequent Itemset Mining (FIM) and Association Rule Mining (ARM) are the two main steps in Frequent Pattern Mining (FPM), and the focus of this research is in FIM. The objectives of this research are as follows: (1) to design an algorithm that constructs a Frequent Pattern Collection (FP-Collection) in a Frequent Pattern Database (FP-DB) for storing the frequent patterns which need to be used for data analysis in FPM, (2) to develop an algorithm that efficiently mines the frequent patterns within a shorter run time and with less memory consumption even though the amount of data is big in the data warehouse, and (3) to evaluate the algorithm in order to ensure that it is capable to mine the frequent patterns within a shorter run time and with less memory consumption for both the sparse and dense data sets. In this research, FP-NoSQL is proposed and constructed as an algorithm for FIM using the Not Only Structured Query Language (NoSQL) because NoSQL is able to support the mining of big data set in a flexible manner. The experimental research method is used as the methodology to implement this research. Four sets of data that are in the sparse or dense structure have been utilized for experimental testing to evaluate the performance of the algorithm. In order to further confirm that the algorithm is robust enough for mining the frequent itemsets in an efficient manner, two sets of data have

been mined to compare against the Apriori and Extended Frequent Pattern (EFP) algorithms. The experiments conducted have proven that the FP-NoSQL algorithm is able to mine the hidden patterns of the frequent itemsets within a shorter run time and with less memory consumption even though the amount of data is big in the data warehouse. The FP-NoSQL algorithm is also evaluated to having a linear, logarithmic or log linear time complexity relationship through the Big-O notation. Apart from this, the FP-NoSQL algorithm is able to selectively retrieve the frequent patterns that matched the requirements of users from the FP-DB for generating the frequent itemsets. Thus, it is not required to mine the entire data warehouse again for identifying the frequent patterns even after a power failure.

## ABSTRAK

Perlombongan data menawarkan manfaat yang luas seperti peningkatan pendapatan, pemotongan kos, dan kelebihan daya saing yang lebih baik. Walau bagaimanapun, corak tersembunyi itemset yang kerap memerlukan lebih banyak masa untuk dilombong apabila jumlah data adalah besar. Selain itu, penggunaan memori yang banyak diperlukan dalam perlombongan corak tersembunyi itemset yang kerap disebabkan oleh kombinasi besar yang perlu diproseskan. Kebanyakan algoritma yang sedia ada masih menghadapi kedua-dua masalah ini kerana itemset yang kerap dilombong ke dalam memori utama dan ruang penyimpanan adalah agak terhad untuk melombong keseluruhan set data. Frequent Itemset Mining (FIM) dan Association Rule Mining (ARM) adalah dua langkah yang penting di dalam proses Frequent Pattern Mining (FPM), dan fokus penyelidikan ini adalah FIM. Oleh itu, suatu algoritma yang cekap perlu dibina bagi perlombongan corak tersembunyi itemset yang kerap terutamanya apabila jumlah data adalah besar. Objektif penyelidikan ini adalah seperti berikut: (1) merangka suatu algoritma yang membina satu Frequent Pattern Collection (FP-Collection) dalam satu Frequent Pattern Database (FP-DB) untuk menyimpan corak kerap yang perlu digunakan untuk analisis data dalam FPM, (2) membina suatu algoritma yang melombongkan corak kerap dalam jangka masa yang lebih singkat dengan penggunaan memori yang kurang walaupun jumlah data adalah besar, dan (3) menilai algoritma yang dibina untuk memastikan bahawa ia mampu melombongkan corak kerap dalam masa yang lebih singkat dengan penggunaan memori yang kurang untuk kedua-dua set data jarang dan padat. Dalam penyelidikan ini, FP-NoSQL dicadang dan dibina sebagai algoritma untuk FIM dengan menggunakan Not Only Structured Query Language (NoSQL) kerana NoSQL dapat menyokong perlombongan data besar secara fleksibel. Kaedah penyelidikan eksperimen digunakan sebagai metodologi untuk melaksanakan penyelidikan ini. Empat set data yang berada dalam struktur jarang atau padat telah digunakan dalam ujian eksperimen untuk menilai prestasi FP-NoSQL. Untuk mengesahkan bahawa FP-



NoSQL adalah sesuai untuk perlombongan itemset yang kerap, dua set data telah dilombong untuk dibandingkan dengan algoritma Apriori dan Extended Frequent Pattern (EFP). Eksperimen yang dijalankan telah membuktikan bahawa FP-NoSQL dapat melombong corak itemset yang kerap dalam masa yang lebih pendek dan dengan penggunaan memori yang kurang walaupun jumlah data adalah besar. Algoritma FP-NoSQL juga dipastikan mempunyai hubungan kompleksiti masa yang linear, logaritma atau log linear melalui notasi Big-O. Selain itu, FP-NoSQL dapat memilih corak kerap yang selaras dengan syarat pengguna dari FP-DB untuk menghasilkan itemset yang kerap. Oleh itu, gudang data tidak perlu dilombong lagi untuk menghasilkan corak yang kerap walaupun selepas kegagalan kuasa.

In compliance with the terms of the Copyright Act 1987 and the IP Policy of the university, the copyright of this thesis has been reassigned by the author to the legal entity of the university,

Institute of Technology PETRONAS Sdn Bhd.

Due acknowledgement shall always be made of the use of any material contained in, or derived from, this thesis.

© CHEE CHIN HOONG, 2020

Institute of Technology PETRONAS Sdn Bhd

All rights reserved.

## TABLE OF CONTENT

ABSTRACT.....	vi
ABSTRAK.....	viii
LIST OF FIGURES .....	xv
LIST OF TABLES .....	xviii
LIST OF ABBREVIATIONS.....	xix
LIST OF SYMBOLS .....	xxi
CHAPTER 1 INTRODUCTION .....	1
1.1 Introduction.....	1
1.2 Motivation.....	6
1.3 Problem Background .....	7
1.4 Problem Statement.....	9
1.5 Research Questions.....	9
1.6 Research Objectives.....	9
1.7 Scope of Research.....	10
1.8 Research Significance.....	11
1.9 Organization of Thesis.....	12
CHAPTER 2 LITERATURE REVIEW .....	15
2.1 Data Analytics .....	15
2.1.1 The Process of Knowledge Discovery in Databases.....	16
2.1.1.1 Data Selection .....	16
2.1.1.2 Data Preprocessing.....	16
2.1.1.3 Data Transformation .....	17
2.1.1.4 Data Mining .....	17
2.1.1.5 Data Interpretation / Evaluation.....	18
2.2 Data Mining .....	19
2.2.1 Data Mining Techniques .....	20
2.2.1.1 Classification.....	20
2.2.1.2 Clustering.....	21
2.2.1.3 Outlier Detection.....	23

2.2.1.4 Frequent Pattern Mining .....	23
2.2.2 Applications of Frequent Pattern Mining.....	26
2.2.2.1 Customer Analysis .....	26
2.2.2.2 Data Indexing and Retrieval .....	26
2.2.2.3 Web Data Mining.....	26
2.2.2.4 Software Bug Detection.....	27
2.2.2.5 Event Detection.....	27
2.2.2.6 Spatiotemporal Analysis .....	27
2.2.2.7 Image Processing .....	27
2.2.2.8 Chemical and Biological Analysis.....	27
2.2.2.9 Facilitator for Other Data Mining Solutions.....	28
2.2.3 Algorithms of Frequent Pattern Mining .....	28
2.2.3.1 Apriori Algorithm .....	28
2.2.3.2 FP-Growth Algorithm.....	31
2.2.3.3 EClat Algorithm .....	33
2.2.3.4 TreeProjection Algorithm.....	35
2.2.3.5 COFI Algorithm.....	36
2.2.3.6 TM Algorithm.....	38
2.2.3.7 P-Mine Algorithm.....	39
2.2.3.8 LP-Growth Algorithm.....	41
2.2.3.9 Can-Mining Algorithm .....	42
2.2.3.10 EXTRACT Algorithm .....	43
2.2.3.11 HYBRID Algorithm .....	45
2.2.3.12 FPNR-Growth Algorithm .....	45
2.2.3.13 SSFIM Algorithm .....	47
2.2.3.14 PFIM Algorithm.....	48
2.2.3.15 Comparison of Frequent Pattern Mining Techniques.....	48
2.3 Data Extraction .....	56
2.3.1 Data Extraction Techniques .....	56
2.3.1.1 Incremental ETL .....	56
2.3.1.2 Real-Time ETL .....	57
2.3.1.3 Parallel ETL .....	59

2.3.1.4 Script Automated ETL.....	60
2.3.1.5 Data Quality ETL.....	61
2.3.1.6 Scalable and High Performance ETL .....	63
2.3.1.7 Semantic ETL .....	64
2.3.1.8 Comparison of Data Extraction Techniques .....	65
2.4 Chapter Summary .....	68
CHAPTER 3 RESEARCH METHODOLOGY .....	69
3.1 Overview of Research and Types of Methodologies.....	69
3.2 Experimental Research Method.....	73
3.3 Research, Development and Evaluation .....	74
3.4 Chapter Summary .....	79
CHAPTER 4 FP-NOSQL: NOSQL-BASED FREQUENT PATTERN MINING WITH FP-DB APPROACH .....	81
4.1 Flow Chart of FP-NoSQL Algorithm.....	81
4.2 Algorithm of Data Loader .....	83
4.3 Algorithm of Frequent Item Generator.....	87
4.4 Algorithm of Frequent Pattern Processor .....	91
4.5 Algorithm of FP-Collection Constructor .....	94
4.6 Algorithm of Frequent Pattern Analyzer .....	97
4.6.1 Procedure to Search All Items.....	99
4.6.2 Procedure to Search Specific Item .....	102
4.6.3 Procedure to Search Specific Pattern .....	104
4.6.4 Procedure to Mine Frequent Itemsets.....	106
4.7 Chapter Summary .....	109
CHAPTER 5 EXPERIMENT RESULTS AND DISCUSSION .....	111
5.1 Experiment Platform.....	111
5.2 Experiment Data Set.....	112
5.3 Evaluating FP-NoSQL on Different Data Sets .....	115
5.3.1 Experiment on Retail Data Set .....	117
5.3.2 Experiment on Connect Data Set .....	120
5.3.3 Experiment on Pumsb Data Set.....	123
5.3.4 Experiment on Kosarak Data Set .....	126

5.3.5 Summary of Experiment Results on Different Data Sets.....	129
5.4 Evaluating Run Time of FP-NoSQL by Comparing to Apriori and EFP.....	130
5.5 Evaluating Memory Usage of FP-NoSQL without FP-DB and with FP-DB	136
5.6 Evaluating FP-NoSQL with Big-O Notation.....	141
5.6.1 Constant Complexity – $O(c)$ .....	142
5.6.2 Linear Complexity – $O(n)$ .....	144
5.6.3 Quadratic Complexity – $O(n^2)$ .....	145
5.6.4 Big-O Notation Analysis for FP-NoSQL .....	146
5.7 Chapter Summary .....	151
CHAPTER 6 CONCLUSION.....	153
6.1 Summary of Research.....	153
6.2 Research Contribution .....	155
6.3 Research Limitation.....	156
6.4 Direction for Future Work .....	156

## LIST OF FIGURES

Figure 1.1: Annual Size of the Global Datasphere (Reinsel et al., 2017).....	2
Figure 1.2: Capabilities that Made CEOs' Company Stand Out (Raskino, 2015).....	3
Figure 1.3: CEOs' Five-Year Investment Plan (Raskino, 2015).....	3
Figure 1.4: Demand Statistics for Data Analytics Expertise (Columbus, 2017) .....	5
Figure 2.1: The Process of Knowledge Discovery in Databases (KDD) (Gullo, 2015) .....	18
Figure 2.2: The Process of Extraction, Transformation and Loading for Data (Prema & Pethalakshmi, 2013).....	19
Figure 2.3: An Example of Classification (J. Han, Kamber, & Pei, 2012a).....	21
Figure 2.4: An Example of Clustering (J. Han, Kamber, & Pei, 2012b).....	22
Figure 2.5: An Example of Outlier Detection (J. Han, Kamber, & Pei, 2012d).....	23
Figure 2.6: Market Basket Analysis (J. Han, Kamber, & Pei, 2012c).....	24
Figure 2.7: Generation of Candidate Itemsets and Frequent Itemsets (J. Han et al., 2012c) .....	30
Figure 2.8: Frequent Pattern Tree (FP-Tree) (J. Han et al., 2012c).....	32
Figure 2.9: Conditional FP-Tree Associated with Node I3 (J. Han et al., 2012c).....	33
Figure 2.10: Lexicographic Tree (Agarwal et al., 2001) .....	36
Figure 2.11: COFI-Trees (El-Hajj & Zaïane, 2003).....	37
Figure 2.12: Example of Transaction Mapping (Song & Rajasekaran, 2006).....	39
Figure 2.13: Architecture of the P-Mine Algorithm (Baralis et al., 2013) .....	40
Figure 2.14: Structure of Linear Prefix Nodes (LPNs) (Pyun et al., 2014) .....	41
Figure 2.15: Architecture of the Can-Mining Algorithm (Hoseini et al., 2015).....	43
Figure 2.16: Architecture of the EXTRACT Algorithm (Feddaoui et al., 2016) .....	44
Figure 2.17: Structure of the FPNR-Tree (Jiang & He, 2017).....	46
Figure 2.18: Alternative Approaches for SSFIM (Djenouri et al., 2018).....	47
Figure 2.19: Classification of Frequent Pattern Mining Algorithms .....	52
Figure 2.20: Matrix Representation of an Incremental Join (Jörg & Deßloch, 2008) .	57
Figure 2.21: Sample Sales Data Warehouse Schema (Santos & Bernardino, 2008) ...	59

Figure 2.22: A Flow with Three Functions in Three Processes (Thomsen & Pedersen, 2011) .....	60
Figure 2.23: Architecture of Script Automated ETL (Radhakrishna et al., 2012).....	61
Figure 2.24: Architecture of Data Quality ETL (Endler, 2012) .....	62
Figure 2.25: Architecture of Scalable and High Performance ETL (K. Sun & Lan, 2012) .....	64
Figure 2.26: Architecture of Semantic ETL (Nath et al., 2015) .....	65
Figure 3.1: Generalized Process of Research (Prajapati et al., 2015).....	70
Figure 3.2: Research Problem Formulation (Prajapati et al., 2015) .....	71
Figure 3.3: Process of Literature Review (Prajapati et al., 2015).....	72
Figure 3.4: Research Work Flow .....	75
Figure 3.5: Integrated Development Environment of C++ in Microsoft Visual Studio .....	77
Figure 3.6: MySQL Shell – The CLI Software for MySQL Database .....	78
Figure 3.7: MySQL Workbench – The GUI Software for MySQL Database .....	78
Figure 4.1: Flow Chart of the FP-NoSQL Algorithm.....	82
Figure 4.2: Transactional Data from Retail Data Set (Brijs, 1999) .....	84
Figure 4.3: Main Menu of FP-NoSQL.....	99
Figure 4.4: Procedure to Search All Items .....	102
Figure 4.5: Procedure to Search Specific Item .....	104
Figure 4.6: Procedure to Search Specific Pattern .....	106
Figure 4.7: Frequent Itemsets Related to Item 32.....	108
Figure 5.1: Retail Data Set (Brijs, 1999) .....	113
Figure 5.2: Kosarak Data Set (Bodon, 2003).....	114
Figure 5.3: Connect Data Set (Bayardo, 2007a) .....	114
Figure 5.4: Pumsb Data Set (Bayardo, 2007b) .....	115
Figure 5.5: Performance Profiler of Microsoft Visual Studio .....	116
Figure 5.6: Run Time Evaluation for Retail Data Set.....	119
Figure 5.7: Memory Usage Evaluation for Retail Data Set .....	120
Figure 5.8: Run Time Evaluation for Connect Data Set.....	122
Figure 5.9: Memory Usage Evaluation for Connect Data Set .....	123
Figure 5.10: Run Time Evaluation for Pumsb Data Set .....	125



Figure 5.11: Memory Usage Evaluation for Pumsb Data Set.....	126
Figure 5.12: Run Time Evaluation for Kosarak Data Set.....	128
Figure 5.13: Memory Usage Evaluation for Kosarak Data Set .....	129
Figure 5.14: Run Time Evaluation for T25I10D10K Data Set by Apriori and EFP (Shang, 2005).....	132
Figure 5.15: Run Time Evaluation for T25I10D10K Data Set by FP-NoSQL .....	133
Figure 5.16: Run Time Evaluation for T10I4D100K Data Set by EFP Algorithm (Shang, 2005).....	135
Figure 5.17: Run Time Evaluation for T10I4D100K Data Set by FP-NoSQL .....	135
Figure 5.18: Time and Space for HIL Strategy of GMiner Algorithm (Chon, Hwang, & Kim, 2018).....	136
Figure 5.19: Communication Costs and Performance of FPM Algorithms (Apiletti et al., 2017) .....	137
Figure 5.20: Memory Usage of PRM and FP-Growth at Support = 10% (Abraham & Joseph, 2016) .....	137
Figure 5.21: Memory Usage Evaluation for T25I10D10K Data Set.....	140
Figure 5.22: Memory Usage Evaluation for T10I4D100K Data Set.....	141
Figure 5.23: Constant Complexity (Malik, 2019).....	143
Figure 5.24: Linear Complexity (Malik, 2019) .....	145
Figure 5.25: Quadratic Complexity (Malik, 2019) .....	146
Figure 5.26: Plot of Common Big-O Functions (Miller & Ranum, 2013) .....	151

## LIST OF TABLES

Table 2.1: Sample of Transactional Data (J. Han et al., 2012c) .....	29
Table 2.2: Conditional Pattern Base and Conditional FP-Tree (J. Han et al., 2012c) .....	33
Table 2.3: Sample of Transactional Data in Vertical Data Format (J. Han et al., 2012c) .....	34
Table 2.4: 2-Itemsets in Vertical Data Format (J. Han et al., 2012c) .....	34
Table 2.5: 3-Itemsets in Vertical Data Format (J. Han et al., 2012c) .....	35
Table 2.6: Comparison of Frequent Pattern Mining Algorithms .....	53
Table 2.7: Comparison of Data Extraction Techniques .....	66
Table 4.1: Transaction Table for Retail Data Set .....	86
Table 4.2: Item Frequency Table for Retail Data Set .....	89
Table 4.3: Frequent Item Table for Retail Data Set .....	90
Table 4.4: Frequent Pattern Table for Retail Data Set .....	93
Table 4.5: Frequent Pattern Collection for Retail Data Set .....	96
Table 5.1: C++ Code to Measure Run Time of Algorithm .....	116
Table 5.2: Experiment Results for Retail Data Set .....	118
Table 5.3: Experiment Results for Connect Data Set .....	121
Table 5.4: Experiment Results for Pumsb Data Set .....	124
Table 5.5: Experiment Results for Kosarak Data Set .....	127
Table 5.6: Details of Four Experiment Data Sets .....	130
Table 5.7: Details of Two Experiment Data Sets .....	131
Table 5.8: Commonly Used Big-O Notations (Malik, 2019) .....	142

## LIST OF ABBREVIATIONS

ARM	Association Rule Mining
BIA	Business Intelligence and Analytics
Can-Tree	Canonical-Order Tree
CDA	Change Data Application
CDC	Change Data Capture
CEO	Chief Executive Official
CLI	Command Line Interface
COFI	Co-Occurrence Frequent Itemset
CPU	Central Processing Unit
EClaT	Equivalence Class Transformation
EFP	Extended Frequent Pattern
ETL	Extract, Transform and Load
FIM	Frequent Itemset Mining
FIMI	Frequent Itemset Mining Implementations
FP-Collection	Frequent Pattern Collection
FP-DB	Frequent Pattern Database
FP-Growth	Frequent Pattern Growth
FPM	Frequent Pattern Mining
FP-NoSQL	Frequent Pattern NoSQL
FP-Tree	Frequent Pattern Tree
GB	Gigabytes
GUI	Graphical User Interface
HDD	Hard Disk Drive
HY-Tree	Hybrid-Tree
IDC	International Data Corporation
IDE	Integrated Development Environment
IoT	Internet of Things
IT	Information Technology
LP-Growth	Linear Prefix Growth

LPN	Linear Prefix Node
LP-Tree	Linear Prefix Tree
MB	Megabytes
No-SQL	Not-Only Structured Query Language
OLAP	Online Analytical Processing
PFIM	Precomputation-Based Frequent Itemset Mining
POS	Point-Of-Sale
RAM	Random Access Memory
RDF	Resource Description Framework
SETL	Scalable and High Performance ETL
SQL	Structured Query Language
SSFIM	Single Scan Approach for Frequent Itemset Mining
TDQM	Total Data Quality Management
TID	Transaction Identifier
TM	Transaction Mapping

## LIST OF SYMBOLS

$c$	Confidence
$C_n$	Candidate n-Itemsets
$D$	Data Warehouse
$I$	Item
$L_n$	Frequent n-Itemsets
$min\_sup$	Minimum Support
$O(c)$	Constant Complexity
$O(n)$	Linear Complexity
$O(n^2)$	Quadratic Complexity
$s$	Support
$T$	Transaction



## CHAPTER 1

### INTRODUCTION

This chapter provides an introduction to the field of data analytics, specifically in the area of Frequent Pattern Mining (FPM). First of all, an introduction and the background about data analytics are given in Section 1.1 . Then, Section 1.2 describes the motivation for conducting this research. Section 1.3 and 1.4 highlights the problems that exist in the current FPM algorithms which need to be resolved. Section 1.5 lists down the questions that are required to be addressed and the objectives of this research are presented in Section 1.6. Next, the scope of this research is defined in Section 1.7 and the significance of this research is discussed in Section 1.8. Finally, the organization of this thesis is presented in Section 1.9.

#### **1.1 Introduction**

In recent years, the amount of data throughout the entire world has been increased exponentially because of the advanced technologies in digital sensors and storage devices (Praveena & Bharathi, 2017). International Data Corporation (IDC) predicted that the amount of data in the whole world may reach 163 zettabytes by the year 2025 as shown in Figure 1.1 (Reinsel, Gantz, & Rydning, 2017). Apart from this, the invention of Cloud Computing and Internet of Things (IoT) have also further promoted the growth of data in many areas (Hashem et al., 2015). This is because the digital sensors in IoT enable data to be collected easily from various sources into the cloud storage, and the technology of Cloud Computing enables data to be accessed in a convenient manner through the internet. Therefore, many organizations in different industries are strongly depending on data in order to obtain valuable insights that are able to produce positive outcomes in the competitive business market.

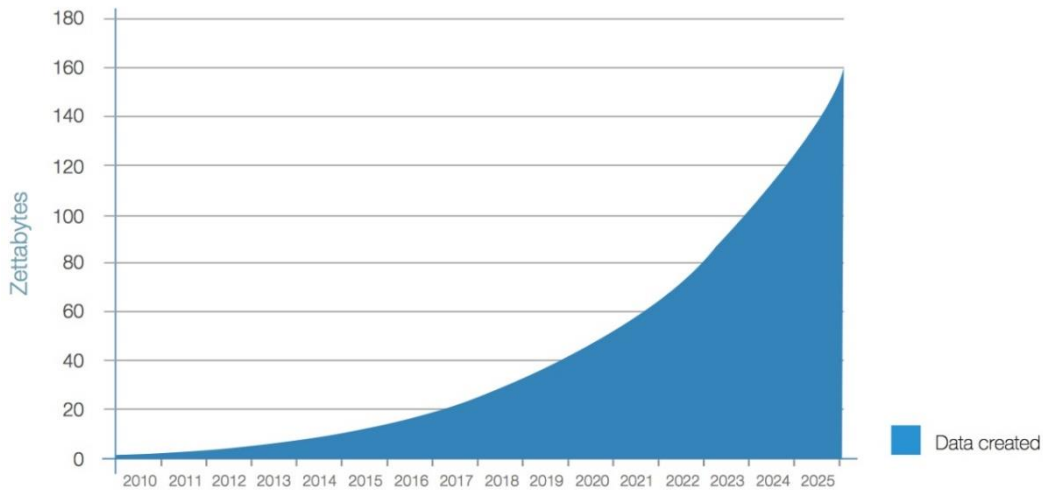


Figure 1.1: Annual Size of the Global Datasphere (Reinsel et al., 2017)

According to a survey conducted by Gartner, data analytics has appeared to be one of the top preferred capabilities of Information Technology (IT) for many Chief Executive Officials (CEOs) (Raskino, 2015). The number of CEOs that support the implementation of different IT capabilities in their companies is shown in Figure 1.2. The support for data analytics to be implemented in an organization is 12%, which is the highest rating among 23 IT capabilities being surveyed. This is because data analytics enables the stakeholders of a company to make informed decision for their business when knowledge or information is easily extracted from the data available in the entire organization. As the stakeholders are able to make fact-based decision using the capability of data analytics, the company will be capable of increasing revenue, cutting cost, and gaining competitive advantage in the challenging market.

Moreover, data analytics is also listed by the CEOs as one of the top preferred IT capabilities in the five-year investment plan of their companies (Raskino, 2015). The percentage of CEOs that support the investment towards various IT capabilities in the five-year plan of their companies is shown in Figure 1.3. The support to invest into business or data analytics is 28%, which is the third highest rating among 26 IT capabilities being surveyed. As a result, this survey has indicated that data analytics is a very important IT capability that needs to be acquired by the business analysts and stakeholders of any organization.



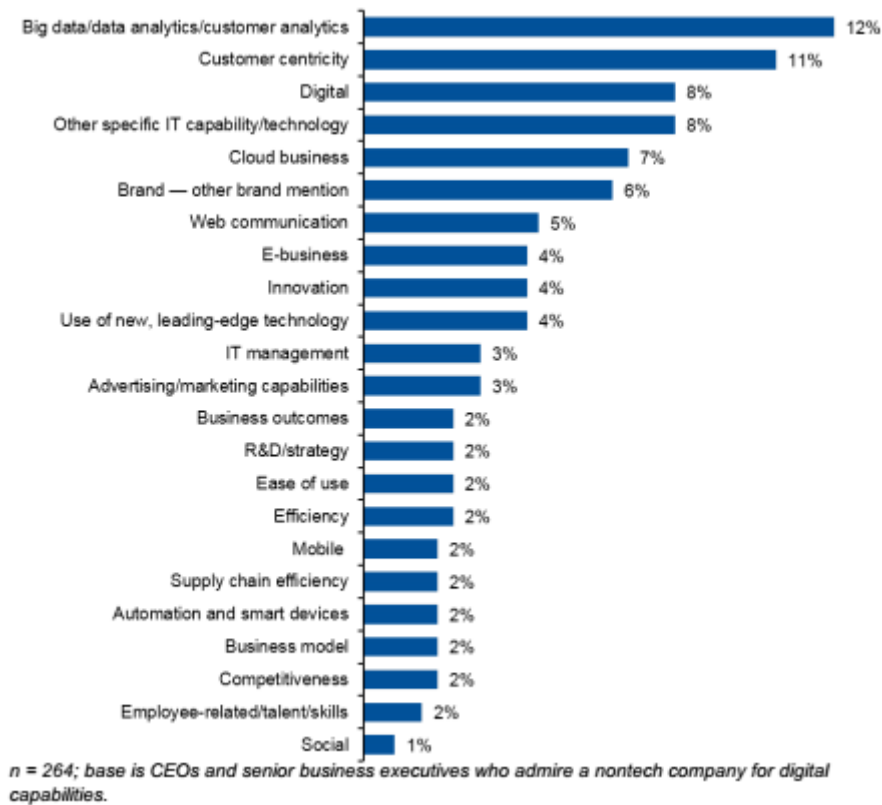


Figure 1.2: Capabilities that Made CEOs' Company Stand Out (Raskino, 2015)



Figure 1.3: CEOs' Five-Year Investment Plan (Raskino, 2015)

According to (Wixom & Watson, 2010), Business Intelligence and Analytics (BIA) is “a broad category of technologies, applications, and processes used for gathering, storing, accessing, and analyzing data to help its users make better decisions”. Due to its popular demand by the CEOs, data analytics has been adopted to support the operations in many businesses or industries like healthcare (McGlothlin & Khan, 2013), electrical power supply (Qiu et al., 2013), manufacturing (Jesus & Bernardino, 2014), railway safety management (Lira et al., 2014), financial service (Chang, 2014), tourism (Rebón, Ocariz, Gerrikagoitia, & Alzua-Sorzabal, 2015), education (Haupt, Scholtz, & Calitz, 2015) and even non-profit organizations (Oakley, Iyer, & A.F.Salam, 2015).

Although data analytics is considered as one of the significant technologies in this competitive business environment, the McKinsey Global Institute speculated that the United States will be encountering a shortage of 140,000 to 190,000 professionals with good analytical skills, and a lack of 1.5 million data-savvy managers who can really analyze business data in order to make decision for their organizations effectively (Manyika et al., 2011). It was predicted by International Business Machines Corporation (IBM) that the demand for Data Scientist will soar 28% by the year 2020 (Columbus, 2017). The demand of jobs related to data analytics are shown in Figure 1.4 which includes the Data-Driven Decision Makers, Functional Analysts, Data Systems Developers, Data Analysts, Data Scientists & Advanced Analysts, and Analytics Managers. A total of 2,352,681 jobs related to data analytics are posted in the year 2015, and it is estimated that this number will be raised to 2,716,425 in the year 2020, with a demand increase of 363,744 jobs within 5 years. Since data analytics is considered as a vital subject of many CEOs and there is a shortage of professional in this area, a lot of research has been conducted for the field of data analytics by both the academic and industrial researchers.

DSA Framework Category	Number of Postings in 2015	Projected 5-Year Growth	Estimated Postings for 2020	Average Time to Fill (Days)	Average Annual Salary
All	2,352,681	15%	2,716,425	45	\$80,265
Data-Driven Decision Makers	812,099	14%	922,428	48	\$91,467
Functional Analysts	770,441	17%	901,743	40	\$69,162
Data Systems Developers	558,326	15%	641,635	50	\$78,553
Data Analysts	124,325	16%	143,926	38	\$69,949
Data Scientists & Advanced Analysts	48,347	28%	61,799	46	\$94,576
Analytics Managers	39,143	15%	44,894	43	\$105,909

Figure 1.4: Demand Statistics for Data Analytics Expertise (Columbus, 2017)

Among various phases in the entire implementation process of data analytics, data mining plays an important role for discovering the significant patterns that may exist frequently in the data sets. This is because identifying the hidden patterns of a data set enables users to make the appropriate decision and action especially in a critical situation. In the midst of numerous data mining techniques, Frequent Pattern Mining (FPM) is one of the most important techniques due to its ability to locate the repeating relationships between different items in a data set and represent the hidden patterns in the form of association rules. FPM has been extensively studied by many researchers because of its abundant applications to a range of data mining tasks like classification, clustering, and outlier analysis (C. C. Aggarwal, 2014b).

In order to improve the methods for classifying or clustering a set of data, and detecting the outliers or anomalies set of data, FPM plays an important role in performing many tasks for data mining. It is the fundamental step to identify the hidden patterns that exist frequently in a data set for generating association rules to be used in data analysis. Most of the time, it is used for market basket analysis where deep insight into product associations can be identified for improving the merchandising, promotions, personalization, and store layouts setup of products (Lobel, 2014). Therefore, the Point-Of-Sale (POS) transaction data is considered as

the most valuable data for retailers of consumer products (Hage, 2017). Apart from this, FPM has various applications in different domains like spatiotemporal data analysis, biological data analysis, and software bug detection (C. C. Aggarwal, 2014b).

## **1.2 Motivation**

In the scenario of a transactional data environment, a lot of patterns can be hidden among the different sets of data in the organizations. The hidden patterns that exist frequently among the data sets are called the frequent patterns. It is also called the frequent itemsets because each transaction of data contains different number of itemsets. Market Basket Analysis is a common case of Frequent Pattern Mining (FPM) where the buying habits of customers are analyzed when the associations between different items purchased by them are identified. At a retail store, a transaction can contain multiple sets of items that are purchased by a customer at one time of visit to the supermarket. It is a competitive advantage for a retail company to be able to identify the purchasing habits of its customers because this enables them to promote the right products to the right customers at the right time through their online advertising campaign. In addition, this also allows the retail company to place the products that are always purchased together by the customers at the same location so that it will be more convenient to the customers who like to go for shopping in the supermarket to locate the products easily.

Apart from implementing FPM for Market Basket Analysis, FPM can also be used in other areas like Data Indexing and Retrieval (Nanopoulos & Manolopoulos, 2002), Web Data Mining (Kachhadiya & Patel, 2018), Software Bug Detection (C. Liu, Yan, Yu, Han, & Yu, 2005), Event Detection (L. Wang, Cao, Wan, & Wang, 2017) and Spatiotemporal Analysis (A. Aggarwal & Toshniwal, 2018). This is because identifying the hidden patterns that exist frequently among the data sets enables users to gain valuable insights to make proper decisions for the business of their organizations. Since a lot of benefits can be obtained when FPM is implemented

into the data analysis process of any organization, it is important to invest into the research and development of a more robust FPM algorithm for data analysis.

### 1.3 Problem Background

Even though many algorithms have been proposed by different researchers throughout the world to enhance the technique in Frequent Pattern Mining (FPM), improvements are still required to be done towards the performance of the existing FPM algorithms (Meenakshi, 2015). This is because most of the current algorithms are not that efficient for mining a data set with a large amount of data (Chen et al., 2015). In a dynamic business environment, it is vital for FPM algorithms to be efficient in mining the frequent patterns of data (Dave, Rathod, Sheth, & Sakhapara, 2015). The two major challenges faced by most of the algorithms are shortening the run time and reducing the memory consumption for executing the algorithm to mine the hidden frequent patterns (Jamsheela & G., 2015). Since these are the two major problems in FPM, many researchers in the area of data mining focus their attention on developing algorithms that can produce better performance in FPM with shorter execution run time and lower memory consumption (Meenakshi, 2015). Although many algorithms have been proposed, some of the algorithms still require a lot of computational time to mine the hidden frequent patterns in a data set especially when the amount of data is large (Mittal, Nagar, Gupta, & Nahar, 2015). Apart from this, some of the algorithms require more memory to mine the hidden frequent patterns in a data set, even though they have a shorter computational time (Dave et al., 2015). Therefore, there is a need to construct an algorithm that is able to mine the significant frequent patterns within a data set in an efficient manner even if the amount of data may be big in a data set.

The hidden patterns of the frequent itemsets become more time consuming to be mined when the amount of data increases in a data set. This is because mining the frequent itemsets from a huge data set will generate a great number of frequent itemsets that satisfied the threshold of minimum support (*min\_sup*), especially when the *min\_sup* is set to a very small value. The minimum support is a value set by the

users in order to determine the minimum occurrence that needs to be satisfied by any item throughout the data set so that the item will be included into the mining process. For example, if the *min\_sup* is set to 10, an item has to occur at least 10 times in the data set in order to be included into the mining process for identifying the hidden patterns of data. At the same time, when the amount of data is big in a data set, it also causes a large memory consumption for mining the hidden patterns of the frequent itemsets due to a heavy computation by the data mining algorithm.

Apart from these, the frequent patterns that have been mined from a data set previously, need to be mined again if there is a change for the value of *min\_sup* determined by the users. This is because the set of frequent patterns will not be the same again when the value of *min\_sup* is changed either to a lower or higher number as well as percentage. For example, if the figure of *min\_sup* is changed to a lower value, more frequent patterns will be generated, whereas if the figure of *min\_sup* is changed to a higher value, less frequent patterns will be considered into the mining process. Since a change of the *min\_sup* value will cause the entire set of frequent patterns to be totally different, the whole data set needs to be mined again in order to produce the correct result of mining.

Furthermore, the frequent patterns are only mined into the main memory or Random Access Memory (RAM) by most of the existing algorithms. In this case, the entire process of Frequent Pattern Mining (FPM) has to be repeated if the system is down or there is a power failure. It is required to be so because none of the frequent patterns being mined previously are retained in the RAM. Therefore, it is necessary to construct an algorithm that is capable of mining the hidden patterns of the frequent itemsets within a shorter run time and with less memory consumption although the amount of data is big in a data set. In addition, the algorithm should be performing well for mining the hidden patterns of the frequent itemsets even though the *min\_sup* value may be changed by the users, the system may be down, or there may be a power failure.

## 1.4 Problem Statement

First, the total number of frequent itemsets to be mined for  $n$  number of items can be estimated based on the formula  $2^n - 1$  (Meenakshi, 2015). For example, if there are 100 items,  $1.2676 * 10^{30}$  itemsets will be generated. Then, the total number of association rules to be generated for  $n$  number of items can be estimated based on the formula  $3^n - 2^{n+1} + 1$  (Meenakshi, 2015). For instance, if there are 100 items,  $5.1537752 * 10^{47}$  rules will be generated. Therefore, as the amount of data increases in a data set, the execution run time and memory consumption of the Frequent Pattern Mining (FPM) algorithm will be definitely increased.

## 1.5 Research Questions

In order to solve the problems stated above, this research has been performed to address the following questions:

- (1) What needs to be constructed in order to retain the frequent patterns that have been mined previously for data analysis in Frequent Pattern Mining?
- (2) How can the frequent patterns be mined efficiently within a shorter run time even though the amount of data is big in a data set?
- (3) How can the memory consumption for mining the frequent itemsets be reduced even though the amount of data is big in a data set?

## 1.6 Research Objectives

In this research, the following objectives have been achieved:

- (1) To design an algorithm that constructs a Frequent Pattern Collection (FP-Collection) in a Frequent Pattern Database (FP-DB) for storing the frequent patterns which need to be used for data analysis in Frequent Pattern Mining (FPM).

- (2) To develop an algorithm that efficiently mines the frequent patterns within a shorter run time and with less memory consumption even though the amount of data is big in the data warehouse.
- (3) To evaluate the algorithm in order to ensure that it is capable to mine the frequent patterns within a shorter run time and with less memory consumption for both the sparse and dense data sets.

## **1.7 Scope of Research**

The action of Frequent Pattern Mining (FPM) can be divided into two main sections as follows (Giacometti, Li, Marcel, & Soulet, 2014):

### **(1) Frequent Itemset Mining (FIM)**

- FIM is the first step of FPM where the frequent itemsets that satisfied the threshold of a minimum support value are generated by the FPM algorithm. For example, in a database that consists of many transactions, each transaction contains a set of different items and these items can be grouped into various combinations of itemsets. The FIM algorithm generates the frequency of occurrence for every combination of the itemsets by identifying the number of transactions that contain them in the entire database. Finally, all the itemsets that have a frequency of occurrence which fulfill the minimum support threshold are to be discovered by the FIM algorithm.

### **(2) Association Rule Mining (ARM)**

- ARM is the second step of FPM where the association rules that indicate interesting relationships among the frequent itemsets are generated by the FPM algorithm. Once the frequent itemsets that fulfill the minimum support threshold are discovered by the FIM algorithm, it can be used to identify the association rules that describe how those itemsets are related to one another. The association rules found by the ARM algorithm which fulfill the minimum confidence threshold are categorized as strong association rules.



The work of this research focus on the first step of FPM. An algorithm for FIM is proposed to construct the frequent patterns and their support counts into a collection in the database within a shorter run time and with lesser memory consumption even though the amount of data is big in the data warehouse.

Apart from this, the work of this research focus on mining the transactional data in a data warehouse where the IDs of transactions and records are processed by the algorithm. In this research, the focus is to mine data in the form of text and numbers because the data in these formats contain the most significant information that can be retrieved if it is processed thoroughly. Therefore, mining the pattern of data from image or video is out of the scope of this research.

Six sets of data that are specifically prepared for the purpose of FIM are downloaded from the internet in order to conduct the experiments for evaluating the algorithm. These data sets contain multiple transactions with multiple items in the dense or sparse format. This is to ensure that the algorithm is able to mine the data even though they are in different formats. The data is processed into a collection in the NoSQL database for the frequent itemsets to be mined using the appropriate NoSQL queries.

## **1.8 Research Significance**

With the implementation of the proposed algorithm, the following benefits can be gained for Frequent Pattern Mining (FPM):

### **(1) Retain the frequent patterns for further analysis**

- All frequent patterns that have been mined can be retained for further analysis using a Frequent Pattern Database (FP-DB). The FP-DB enables each unique pattern that can be found from a data warehouse to be consolidated into a Frequent Pattern Collection (FP-Collection) along with its frequency of occurrence. As every pattern from a data warehouse is stored into the FP-DB, the frequent patterns that have been mined can be retrieved anytime for further analysis even after a power failure or the system is down.

## **(2) Mine the frequent patterns within a shorter run time**

- The frequent patterns can be mined within a shorter run time because every pattern that exists frequently in the data warehouse is inserted into a frequent pattern table at the same time after they are being concatenated together. Concatenating the frequent patterns together and inserting them into a frequent pattern table at the same time helps to reduce the number of times for scanning the data warehouse. When the number of times for scanning the data warehouse is reduced in the FPM process, the total run time for constructing all the frequent patterns can be reduced significantly.

## **(3) Mine the frequent itemsets using lesser memory consumption**

- The frequent itemsets can be mined using lesser memory consumption since every unique pattern that exists frequently in the data warehouse is consolidated into the FP-DB together with its frequency of occurrence. In order to discover the frequent itemsets that are related to a specific item, it is not necessary to mine the entire data warehouse because only the frequent patterns that matched the requirements of users are required to be retrieved for generating the relevant frequent itemsets.

## **1.9 Organization of Thesis**

The structure of this thesis is organized into six chapters and the remaining chapters are briefly described as follows:

**Chapter 2** presents the fundamental concept of Frequent Pattern Mining (FPM) and reviews some state-of-the-art FPM algorithms by describing how each of them works. The FPM algorithms are also classified into different categories and compared to identify their advantages and disadvantages respectively. Apart from this, the significant data extraction techniques are also reviewed in this chapter because it is a necessary step to be implemented prior to data mining. Similarly, the data extraction techniques are also compared to identify their advantages and disadvantages respectively.

**Chapter 3** discusses about the methodology used for conducting this research. All the processes of the research methodology being selected are described in detail throughout every stage of research, development and evaluation for the proposed FPM algorithm. Apart from this, the tools used for conducting the research and implementing the development for the proposed FPM algorithm are also described in this chapter.

**Chapter 4** describes how the proposed FPM algorithm works. The entire architecture of the algorithm is presented in a flow chart and the pseudocode of the algorithm is explained part by part in this chapter.

**Chapter 5** presents the results of the experiment conducted to evaluate the performance of the proposed FPM algorithm. The experiment results are discussed in this chapter in order to verify that the algorithm is suitable to be used for Frequent Pattern Mining (FPM) in a big data set.

**Chapter 6** concludes the entire research into a brief summary and highlights the significant contributions of this research. The potential future work that can be conducted by other researchers is also discussed in this chapter.



## CHAPTER 2

### LITERATURE REVIEW

This chapter reviews several algorithms for Frequent Pattern Mining (FPM). First, an overview of the entire implementation process of data analytics is given in Section 2.1. Then, the significant data mining techniques and the fundamental knowledge of FPM are provided in Section 2.2. The previous research conducted on the fundamental and significant FPM algorithms are presented in Section 2.2.3.1 to Section 2.2.3.14. Next, Section 2.2.3.15 presents an organization chart that categorizes the FPM algorithms into different groups. A table which provides a comparison for the advantages and disadvantages of the FPM algorithms is given in the same section. Last but not least, since extracting data is part of the tasks for data mining to be conducted, some significant data extraction techniques are given in Section 2.3. A table which provides a comparison for the advantages and disadvantages of the data extraction techniques is given in Section 2.3.1.8.

#### **2.1 Data Analytics**

In the field of Business Intelligence and Analytics (BIA), the two terms that are often used in an interchangeable manner are “data analysis” and “data analytics” (Inteliment, 2016). But what is the major difference between them? The main distinction between these two terms is this: data analysis presents data by only looking at the past while data analytics tries to predict the future by using every data that is available (Park, 2017). In other words, data analysis attempts to answer the “What has happened?” question, whereas data analytics attempts to answer the “Why did it happen and what will happen later?” question. Therefore, data analysis can also be viewed as a detailed study of the data for any kind of decision-making situation,

while data analytics provides actionable insights for users to make the appropriate decisions in the future apart from answering questions that have happened in the past.

### **2.1.1 The Process of Knowledge Discovery in Databases**

Analyzing all the data that is collected in the data warehouse is definitely a necessity for every enterprise because the proper decisions can be made for taking the appropriate actions especially in a critical situation. In general, the entire implementation process of data analytics that is in common practice is shown in Figure 2.1 (Gullo, 2015) and it contains five important phases to select, pre-process, transform, mine and evaluate or interpret the data as follows:

#### *2.1.1.1 Data Selection*

First, the data is required to be extracted from its original sources into various target data sets in the data warehouse. This is a very important step because data can be existing in multiple departments throughout the whole organization. Apart from this, data can also be appearing in different formats like text files, spreadsheets, databases, xml files and others (Kherdekar & Metkewar, 2016). In order to completely analyze all data that is available in the entire organization, it is necessary to consolidate the data from numerous sources into the data warehouse for pre-processing, transformation, and mining before it can be interpreted or evaluated properly as useful knowledge for business analysis and decision-making in the organization.

#### *2.1.1.2 Data Preprocessing*

As the data is consolidated from various sources that are in different forms, it is essential to preprocess the data so that it can be stored appropriately in the data warehouse. This step is also necessary due to the problems of data quality like missing data, noisy data, inconsistent data or even wrongly sampled data

(JayaramHariharakrishnan, Mohanavalli, Srividya, & Kumar, 2017). If the quality of data is low, it may lead to wrong conclusions when the data is processed by any kind of data mining techniques.

#### *2.1.1.3 Data Transformation*

After preprocessing the data, the processes of transformation are required to be implemented towards the data so that it is formatted in the appropriate forms for data mining to be conducted accordingly. This is usually the most time-consuming step because constructing a data set that is suitable for analysis by consolidating data from numerous tables and views in a database requires complicated queries to be written properly (Chaudhari & Khanuja, 2015). The processes of extraction, transformation and loading for data are shown in a summarized manner in Figure 2.2 (Prema & Pethalakshmi, 2013).

#### *2.1.1.4 Data Mining*

When the data is transformed into the appropriate formats, it is ready for mining in order to discover the hidden patterns that exist in the data set. In general, data mining techniques can be categorized into four main categories, namely, Classification, Clustering, Outlier Detection, and Frequent Pattern Mining (J. Han, Kamber, & Pei, 2012e). Classification groups data into different classes using a classifier that is constructed from a set of training data with some predefined class labels, while Clustering groups data objects with high similarity into the same cluster without using a classifier. Outlier Detection locates data with characteristics that are very unusual compared to the common ones, and Frequent Pattern Mining locates the hidden patterns that exist frequently in the data set.

### 2.1.1.5 Data Interpretation / Evaluation

In order for the data to be interpreted or evaluated as useful knowledge, one of the methods to present data in an effective manner is to visualize the data in various forms like graph visualization, text visualization, map visualization, and multivariate data visualization (S. Liu, Cui, Wu, & Liu, 2014). This is because data and information can be analyzed and understood by the human mind in an easier way when it is represented in the appropriate visual form.

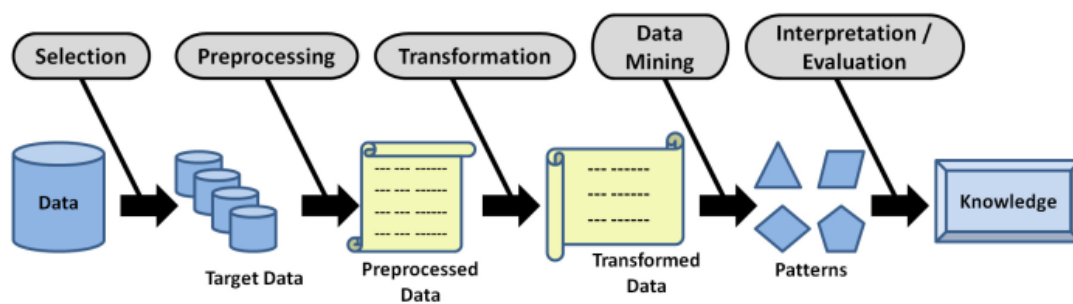


Figure 2.1: The Process of Knowledge Discovery in Databases (KDD) (Gullo, 2015)



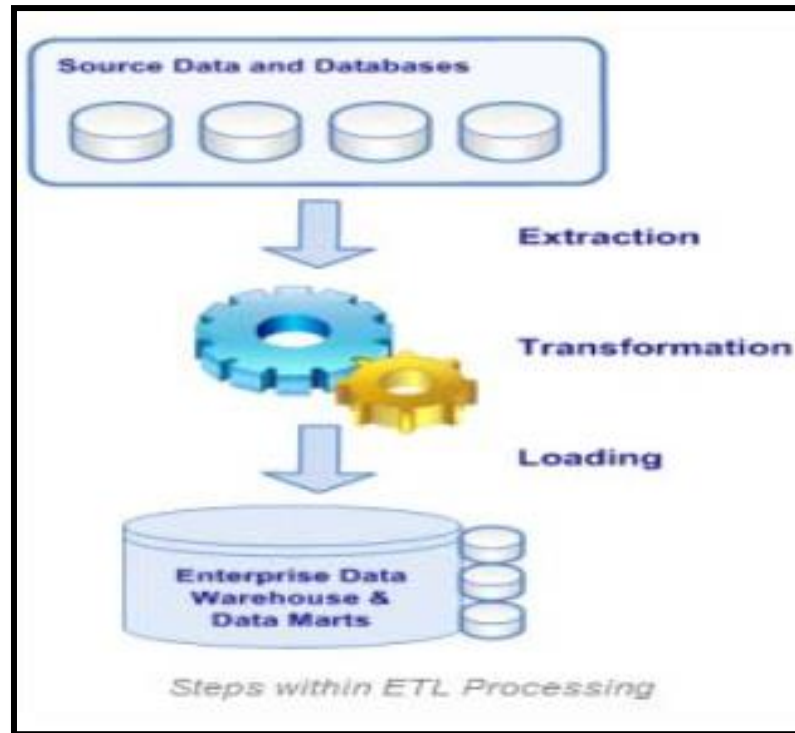


Figure 2.2: The Process of Extraction, Transformation and Loading for Data  
(Prema & Pethalakshmi, 2013)

## 2.2 Data Mining

Among the five steps in the entire process of data analytics, data mining is considered as the most important step. This is because a tremendous amount of data is captured everyday and there is a vital need to analyze it, so that valuable information can be discovered from the data and be transformed into useful knowledge (J. Han, Kamber, & Pei, 2012f). The dynamic growth of such a huge data volume is an outcome of the computerization of the world and the fast invention of powerful tools for data collection and storage. However, the ability to identify the significant patterns in a set of data depends on the capability of the algorithm that is being implemented for mining. Therefore, appropriate data mining algorithms and techniques should be proposed in order to provide a suitable solution for extracting valuable information from the vast amount of data available.

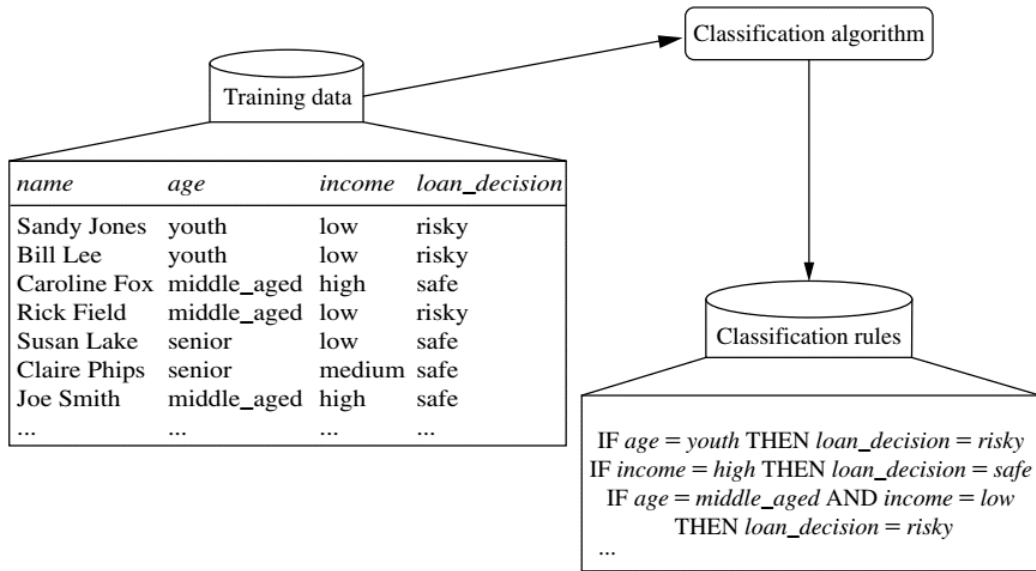
### **2.2.1 Data Mining Techniques**

The massive volume of data that has grown exponentially has caused a lot of data mining techniques to be developed (Henke et al., 2016). In general, data mining techniques can be categorized into four main categories, namely, Classification, Clustering, Outlier Detection, and Frequent Pattern Mining (J. Han et al., 2012e).

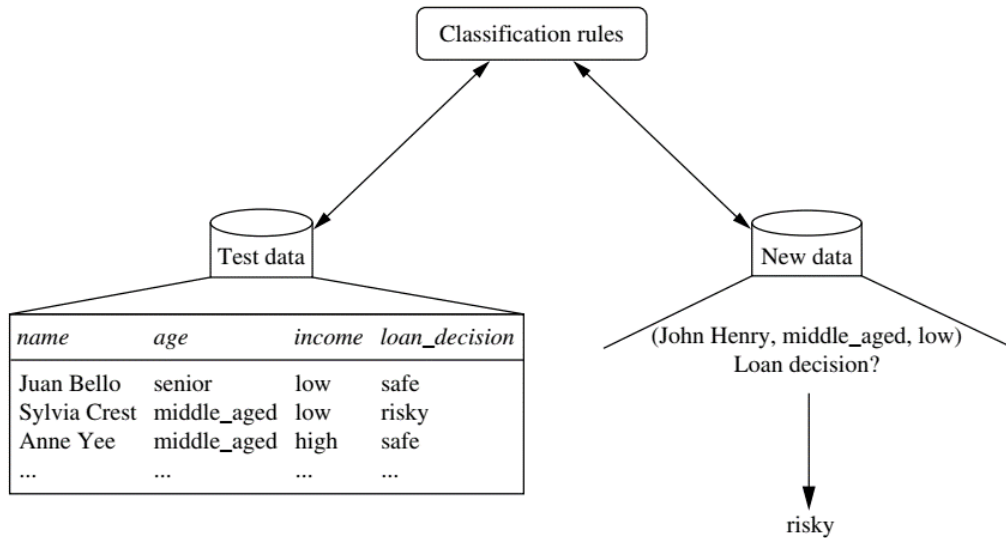
#### *2.2.1.1 Classification*

Classification is a data mining technique that groups data into different classes using a classifier that is constructed from a set of training data with some predefined class labels (Midha & Singh, 2015). It is useful in situations where a bank loan officer may be interested to know which loan applicants are considered as “safe” or “risky” for the bank, a marketing manager may like to know whether a customer with a certain profile will buy a particular product, and a medical researcher may be interested to identify which specific treatment a patient of breast cancer should receive.

The process of classification consists of two steps for processing the data, namely the learning step and the classification step as shown in Figure 2.3. In the learning step, various sets of training data are analyzed by the classification algorithm in order to construct a classifier in the form of classification rules, mathematical formulae or decision trees. The training data is made up of some database records that are attached with the appropriate class labels. Since a class label is available in every record of the training data, classification is considered as a kind of supervised learning technique. In the classification step, test data is used to estimate the accuracy of the classifier. If the accuracy rate of the classifier is considered as acceptable, the classification algorithm can be used to classify new data into the relevant classes.



(a)



(b)

Figure 2.3: An Example of Classification (J. Han, Kamber, & Pei, 2012a)

### 2.2.1.2 Clustering

Clustering is a data mining technique that groups data objects with high similarity into the same cluster without using a classifier that is constructed from a set of training data with some predefined class labels (Hruschka, Campello, Freitas, & Carvalho, 2009). It is useful in a situation where there are a lot of customers that purchase products from a company and it is quite impossible to categorize them into a

few strategic groups manually. Due to its ability to automatically identify the groupings from a set of data, clustering is also called automatic classification.

The process of clustering groups the objects in a set of data into different categories, having the objects with high similarity to be grouped into the same category as shown in Figure 2.4. Clustering is considered as unsupervised learning since the information of class labels is not available. Hence, it is a data mining technique that is learning by observation instead of learning by examples. However, different methods of clustering may group objects in the same data set into different categories. Even though it may be so, clustering is still considered as a very useful technique of data mining because it helps to discover groups that are unknown previously within the data set.

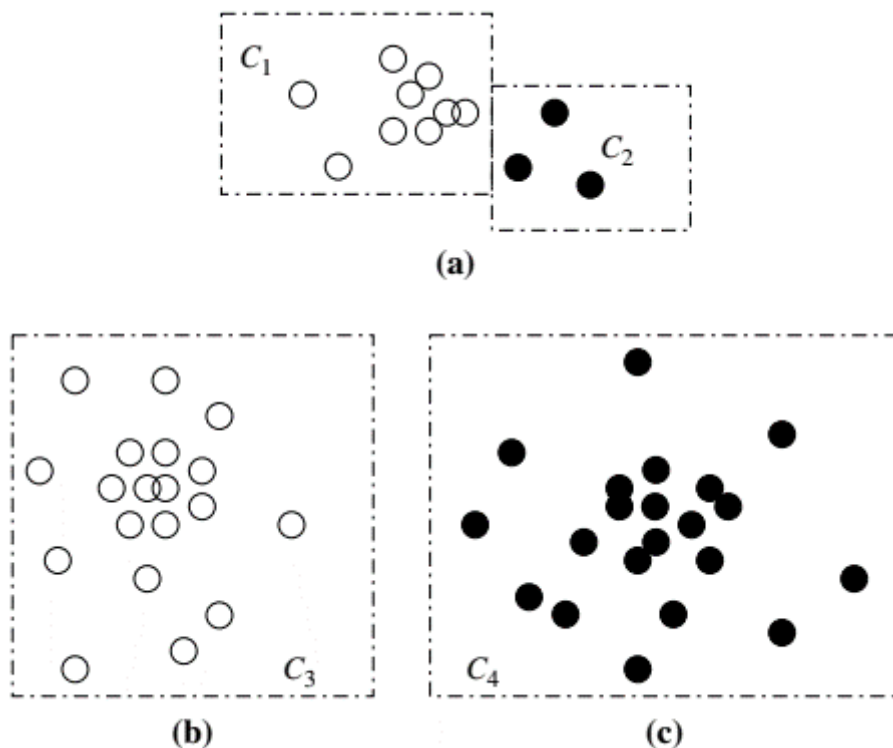


Figure 2.4: An Example of Clustering (J. Han, Kamber, & Pei, 2012b)

### 2.2.1.3 Outlier Detection

Outlier Detection locates data with characteristics that are very unusual compared to the common ones as shown in Figure 2.5. While the technique of Clustering is to identify data with common patterns and group them into different categories, Outlier Detection tries to identify those extraordinary data that deviates significantly from the data with common patterns (Bansal, Gaur, & Narayan, 2016). The extraordinary data can be categorized into three different types of outliers, namely Global Outlier, Contextual Outlier, and Collective Outlier. Global Outlier deviates substantially from all the other data sets, while Contextual Outlier deviates substantially depending on a particular situation for the data. For Collective Outlier, the data deviates substantially together as a whole from the entire data set.

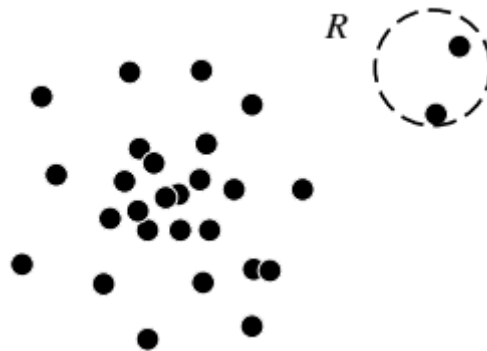


Figure 2.5: An Example of Outlier Detection (J. Han, Kamber, & Pei, 2012d)

### 2.2.1.4 Frequent Pattern Mining

Among various techniques of data mining, Frequent Pattern Mining (FPM) is one of the most important techniques because it has plentiful applications to a range of data mining tasks in classification, clustering, and outlier analysis. In order to provide users with information that is more useful for data analysis and decision-making, it is important to mine and identify all the significant hidden patterns that exist frequently in a data set. Therefore, the main focus of this research is to analyze a number of FPM algorithms and propose an FPM algorithm that is able to mine a big data set within a shorter run time and with less memory consumption.

Frequent Pattern Mining (FPM) locates repeating relationships in a data set (Garg & Sharma, 2011). The patterns that exist frequently in a data set are called frequent patterns. It can be appearing in the form of itemsets, subsequences or substructures. A frequent itemset is a set of items that exist together frequently among the transactions of a data set. For example, in the context of a supermarket, the items that are always being purchased at the same time by the customers are bread and butter. In this case, FPM can be conducted to perform market basket analysis so that the consumers buying habits can be analyzed by locating the associations between different items that have been purchased. The application of FPM for the scenario of market basket analysis is illustrated in Figure 2.6.

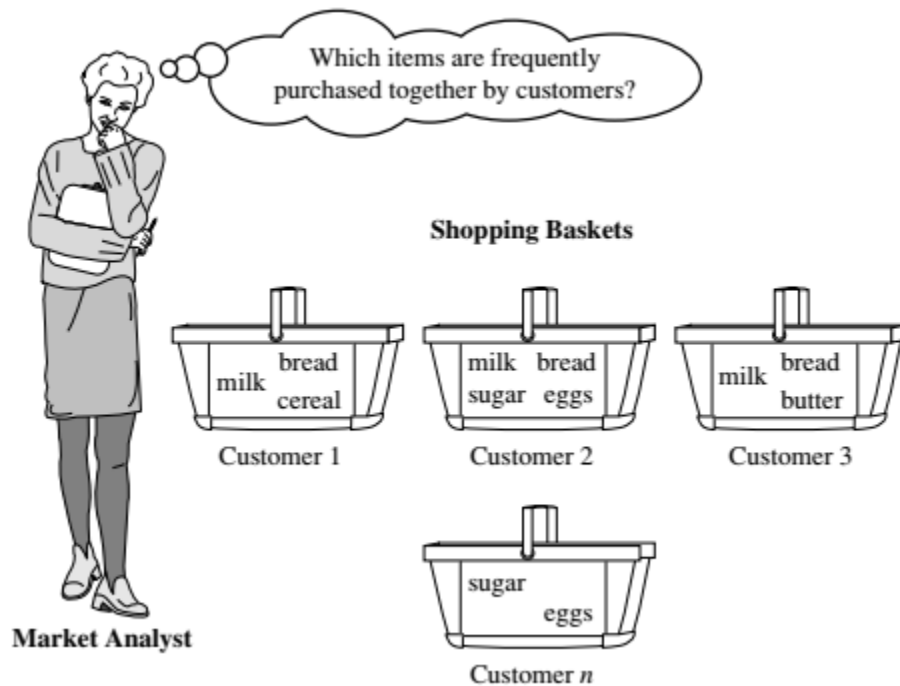


Figure 2.6: Market Basket Analysis (J. Han, Kamber, & Pei, 2012c)

Let  $D$  be a data warehouse with a set of different items  $I = (I_1, I_2, \dots, I_m)$ . Then, every transaction  $T$  is an itemset that is not empty in which  $T \subseteq I$ . An itemset is a set of items. If it contains  $k$  items, then it is a  $k$ -itemset. For instance, the set (bread,

butter) is a 2-itemset. Every transaction contains one or more items and each transaction is represented by a Transaction Identifier, *TID*.

The hidden patterns of a data set can be represented in the form of association rules. An association rule is commonly denoted as  $A \Rightarrow B$ , having  $A \neq \emptyset$ ,  $B \neq \emptyset$ ,  $A \subset I$ ,  $B \subset I$ , and  $A \cap B = \emptyset$ . The interestingness of a rule is determined by two measures, support  $s$  and confidence  $c$ , as stated in the following formulas:

$$\text{support}(A \Rightarrow B) = P(A \cup B) \dots\dots\dots [2.1]$$

$$\text{confidence}(A \Rightarrow B) = P(B | A) = \frac{\text{support}(A \cup B)}{\text{support}(A)} \dots\dots\dots [2.2]$$

In this scenario, support  $s$ , indicates the percentage of  $T$  in  $D$  that consist of  $A \cup B$ , whereas confidence  $c$ , indicates the percentage of  $T$  in  $D$  that consist of  $A$  also consist of  $B$ . The values of  $s$  and  $c$  will occur between 0% and 100%. Normally, the association rules are considered remarkable if they fulfill a minimum threshold for both the support and confidence measures. Apart from support and confidence, a lot of measures like coverage, prevalence, recall, specificity, accuracy and many others can be used to quantify the interestingness of an association rule (Le & Lo, 2015). However, it is out of the scope of this research for other types of measures to be discussed in this thesis.

For the measures of support and confidence, in the situation of a computer retailer, the data that shows consumers who buy laptop computers also tend to purchase office software simultaneously can be represented as follows:

$$\text{laptop computer} \Rightarrow \text{office software} [\text{support} = 10\%, \text{confidence} = 70\%].$$

A value of 10% for support indicates that 10% of all the records under analysis involve the purchase of laptop computer and office software at the same time, whereas a value of 70% for confidence indicates that 70% of the consumers who bought a laptop computer also purchased the office software.

## **2.2.2 Applications of Frequent Pattern Mining**

Frequent Pattern Mining (FPM) is an important data mining technique that can be applied into several areas as follows (C. C. Aggarwal, 2014a):

### *2.2.2.1 Customer Analysis*

FPM is commonly used by companies that sell products to analyze the purchasing behaviour of their customers (Wenzhe et al., 2017). By implementing FPM, companies can identify which products are always being purchased at the same time or by the same customers. With this capability, retailers are able to arrange their products for display at the appropriate locations so that it can be easily found by the customers. Apart from this, the relevant products can also be promoted to the right customers at the right time. As a result, this helps to increase the sales of their products.

### *2.2.2.2 Data Indexing and Retrieval*

FPM is also used for data indexing and retrieval in which a concise representation of the data is constructed (Nanopoulos & Manolopoulos, 2002). In this process, the data are categorized into different groups based on its patterns in order to enable the branch-and-bound search to be performed when the similarities-based queries are processed.

### *2.2.2.3 Web Data Mining*

One of the applications of FPM is in the area of web data mining where frequent patterns are discovered in order to monitor the navigational behaviour of the users for creating a more suitable advertising strategy (Kachhadiya & Patel, 2018).



#### *2.2.2.4 Software Bug Detection*

Another application of FPM is for identifying the bugs of software in which the program executions are classified with some software behaviour graphs for discovering the program sections that may cause the faulty executions (C. Liu et al., 2005).

#### *2.2.2.5 Event Detection*

One of the most common applications of FPM in event detection is the use for intrusion detection within a secure network where web logs are analyzed in order to predict possible web attacks that are currently unknown (L. Wang et al., 2017).

#### *2.2.2.6 Spatiotemporal Analysis*

FPM is used for spatiotemporal analysis in which the frequent patterns of data are mainly affected by the time and location of occurrence for the transactions (A. Aggarwal & Toshiwal, 2018). In this scenario, the data is usually captured continuously from the mobile phones of users.

#### *2.2.2.7 Image Processing*

Another application of FPM is for image classification where the features of images are considered as attributes in the transactions and frequent patterns can be identified from it in order to determine their important characteristics (Fernando, Fromont, & Tuytelaars, 2012).

#### *2.2.2.8 Chemical and Biological Analysis*

FPM can also be used for chemical and biological analysis because most of the chemical and biological data for chemical compounds, complex biological molecules,

microarrays or protein interaction networks are often represented as graphs. The technique that can be used for identifying frequent patterns from chemical and biological data is called frequent subgraph discovery (Kuramochi & Karypis, 2001).

#### *2.2.2.9 Facilitator for Other Data Mining Solutions*

The technique of FPM is closely related to other data mining techniques like classification, clustering and outlier detection because it is the most fundamental method that is required to be implemented before other data mining solutions can be accomplished (Tiwari, Gupta, & Agrawal, 2010).

### **2.2.3 Algorithms of Frequent Pattern Mining**

A lot of algorithms have been proposed to solve the problem of Frequent Pattern Mining (FPM). Among all the FPM algorithms, the fundamental ones are the algorithms of Apriori (Agrawal & Srikant, 1994), FP-Growth (J. Han, Pei, & Yin, 2000) and EClat (Zaki, 2000). This section describes how the fundamental and significant algorithms in FPM work and compares their advantages and disadvantages respectively.

#### *2.2.3.1 Apriori Algorithm*

Apriori (Agrawal & Srikant, 1994) is an algorithm proposed by R. Agrawal and R. Srikant to mine frequent itemsets for generating Boolean association rules. It uses an iterative level-wise search technique to discover  $(k+1)$ -itemsets from  $k$ -itemsets. First, the database is scanned to identify all the frequent 1-itemsets by counting each of them and capturing those that satisfy the minimum support threshold. The result of frequent 1-itemsets is represented as  $L_1$ . Then,  $L_1$  is used to locate  $L_2$ , the set of frequent 2-itemsets, which is used to locate  $L_3$ , and the rest, until no more frequent  $k$ -itemsets is possible to be identified. The identification of each  $L_k$  requires of scanning the entire database.

A sample of transactional data that consists of product items being purchased at different transactions is shown in Table 2.1. In order to locate all the possible frequent itemsets, the entire database is scanned multiple times for identifying the count of each frequent itemset as described in Figure 2.7. The minimum support threshold used in this example is ‘2’. Therefore, only the records that fulfill a minimum support count of ‘2’ will be included into the next cycle of algorithm processing.

Table 2.1: Sample of Transactional Data (J. Han et al., 2012c)

<i>TID</i>	<i>List of item_IDs</i>
T100	I1, I2, I5
T200	I2, I4
T300	I2, I3
T400	I1, I2, I4
T500	I1, I3
T600	I2, I3
T700	I1, I3
T800	I1, I2, I3, I5
T900	I1, I2, I3

In the first cycle, the algorithm scans the database to count the number of occurrences of each item to produce the candidate 1-itemsets,  $C_1$ . All the items in  $C_1$  are counted as the members of frequent 1-itemsets,  $L_1$  because every item satisfies the minimum support count of ‘2’. Then, the algorithm joins  $L_1$  to itself in order to generate the candidate 2-itemsets,  $C_2$  for further discovering the frequent 2-itemsets,  $L_2$ . In this step, no candidates are removed from  $C_2$  because every subset of the candidates is also frequent in the database.

After generating  $C_2$ , the database is scanned again to count the number of occurrences of each item in  $C_2$ . The items in  $C_2$  that fulfill a minimum support count of ‘2’ are counted as the members of frequent 2-itemsets,  $L_2$ . Then, the algorithm

continues to join  $L_2$  to itself in order to generate the candidate 3-itemsets,  $C_3$  for further discovering the frequent 3-itemsets,  $L_3$ . In this step, the actual result of having  $L_2$  to be joined to itself is (I1, I2, I3), (I1, I2, I5), (I1, I3, I5), (I2, I3, I4), (I2, I3, I5), and (I2, I4, I5). But the four latter candidates have been removed from  $C_3$  as they are not frequent in the database. With the implementation of this level-wise search technique, it saves the effort of calculating their support counts unnecessarily in the next scanning of the database to determine  $L_3$ .

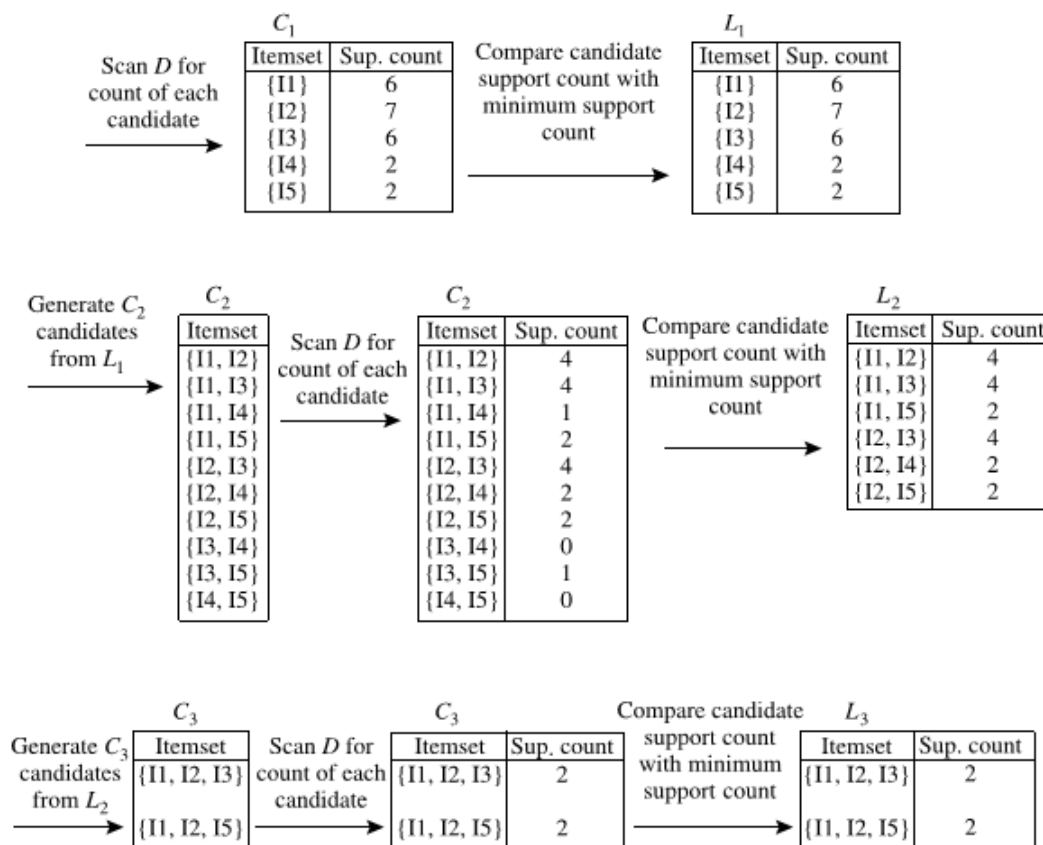


Figure 2.7: Generation of Candidate Itemsets and Frequent Itemsets  
(J. Han et al., 2012c)

After generating  $C_3$ , the database is scanned again to count the number of occurrences of each item in  $C_3$ . The items in  $C_3$  that fulfill a minimum support count of '2' are counted as the members of frequent 3-itemsets,  $L_3$ . Then the algorithm

continues to join  $L_3$  to itself in order to generate the candidate 4-itemsets,  $C_4$ . In this step, even though the join produces itemset  $(I_1, I_2, I_3, I_5)$ , it is removed from  $C_4$  because one of its subsets  $(I_2, I_3, I_5)$  is not frequent in the database. Therefore,  $C_4 = \emptyset$ , and the algorithm is terminated at this point, having all of the frequent itemsets to be discovered.

In many cases, the Apriori algorithm reduces the size of candidate item sets significantly and provides a good performance gain. However, it is still suffering from two critical limitations (J. Han et al., 2012c). First, a large number of candidate item sets may still need to be generated if the total count of a frequent  $k$ -itemsets increases (D. Sun, Teng, Zhang, & Zhu, 2007). Then, the entire database is required to be scanned repeatedly and a huge set of candidate items are required to be verified using the technique of pattern matching (F. Wang & Li, 2008).

#### 2.2.3.2 FP-Growth Algorithm

Frequent Pattern Growth (FP-Growth) (J. Han et al., 2000) is an algorithm proposed by Jiawei Han to mine frequent itemsets without a costly candidate generation process. It implements a divide-and-conquer technique to compress the frequent items into a Frequent Pattern Tree (FP-Tree) that retains the association information of the frequent items. It is built by accessing the data set to retrieve one transaction at a time and plotting each item of the transaction onto a path in the FP-Tree (Kim, Lee, Kim, & Son, 2010). The FP-Tree is further divided into a set of Conditional FP-Trees for each frequent item so that they can be mined separately.

An example of the FP-Tree that represents all the frequent items found from the transactional data listed in Table 2.1 is shown in Figure 2.8. Similar to the Apriori algorithm, the FP-Growth algorithm generates the frequent 1-itemsets and their support counts at the first scan of the database. Then, the set of frequent items is sorted in the descending order according to their support counts, having the frequent itemsets  $L = ((I_2: 7), (I_1: 6), (I_3: 6), (I_4: 2), (I_5: 2))$ . Next, the FP-Tree is built by creating the root of the tree which is represented as “null”. The database will be scanned a second time and the items in every transaction are processed into the FP-

Tree according to the descending order of support counts identified in L. For example, the first transaction record, “T100: I1, I2, I5” will be reordered as “T100: I2, I1, I5” and placed under the FP-Tree, with I2 to be linked to the root, I1 to be linked to I2, and I5 to be linked to I1. To simplify the tree traversal, an item header table is constructed so that every item can be linked to its positions in the FP-Tree through a series of node-links. In this manner, the problem in mining frequent patterns from a database is simplified to mining from the FP-Tree.

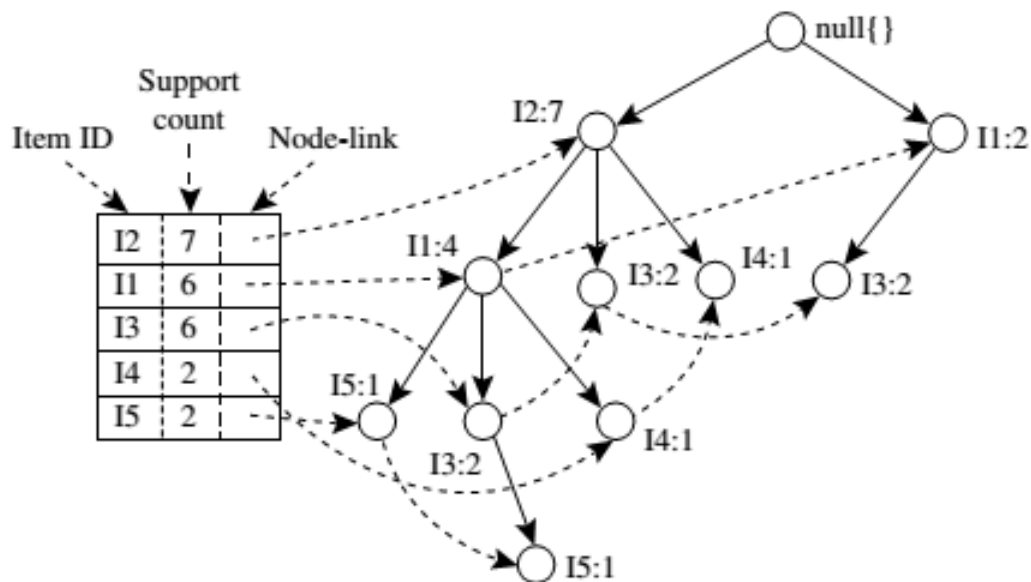


Figure 2.8: Frequent Pattern Tree (FP-Tree) (J. Han et al., 2012c)

The FP-Growth algorithm solves the problem of identifying long frequent patterns by searching through smaller Conditional FP-Trees repeatedly. An example of the Conditional FP-Tree associated with node I3 is shown in Figure 2.9, and the details of all the Conditional FP-Trees found in Figure 2.8 are shown in Table 2.2. The Conditional Pattern Base is a “sub-database” which consists of every prefix path in the FP-Tree that co-occurs with every frequent length-1 item. It is used to construct the Conditional FP-Tree and generate all the frequent patterns related to every frequent length-1 item. In this way, the cost of searching for the frequent patterns is

substantially reduced. However, constructing the FP-Tree is time consuming if the data set is very large (Meenakshi, 2015).

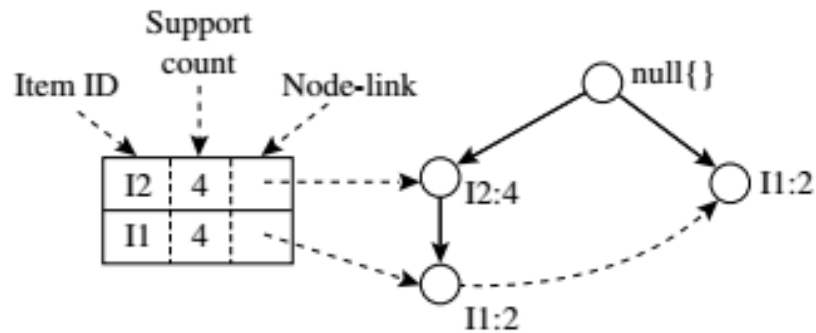


Figure 2.9: Conditional FP-Tree Associated with Node I3 (J. Han et al., 2012c)

Table 2.2: Conditional Pattern Base and Conditional FP-Tree (J. Han et al., 2012c)

<i>Item</i>	<i>Conditional Pattern Base</i>	<i>Conditional FP-tree</i>	<i>Frequent Patterns Generated</i>
I5	{{I2, I1: 1}, {I2, I1, I3: 1}}	{I2: 2, I1: 2}	{I2, I5: 2}, {I1, I5: 2}, {I2, I1, I5: 2}
I4	{{I2, I1: 1}, {I2: 1}}	{I2: 2}	{I2, I4: 2}
I3	{{I2, I1: 2}, {I2: 2}, {I1: 2}}	{I2: 4, I1: 2}, {I1: 2}	{I2, I3: 4}, {I1, I3: 4}, {I2, I1, I3: 2}
I1	{{I2: 4}}	{I2: 4}	{I2, I1: 4}

### 2.2.3.3 EClat Algorithm

Equivalence Class Transformation (EClat) (Zaki, 2000) is an algorithm proposed by Zaki to mine frequent itemsets efficiently using the vertical data format. The vertical format of the data found in Table 2.1 is shown in Table 2.3. In this method of data representation, all the transactions that contain a particular itemset are grouped into the same record. For example, the transactions of T100, T400, T500, T700, T800, and T900 are all the records that contain the I1 itemset in the database.

First, the EClat algorithm transforms data from the horizontal format into the vertical format by scanning the database once. The frequent (k+1)-itemsets are

generated by intersecting the transactions of the frequent k-itemsets. For instance, when the itemsets of I1 and I2 are intersected with one another, the transactions that are common in both itemsets will be included into the 2-itemset of (I1, I2) as T100, T400, T800, and T900. This process repeats until all the frequent itemsets are intersected with one another and no frequent itemsets can be found as shown in Table 2.4 and Table 2.5.

Table 2.3: Sample of Transactional Data in Vertical Data Format (J. Han et al., 2012c)

<i>itemset</i>	<i>TID_set</i>
I1	{T100, T400, T500, T700, T800, T900}
I2	{T100, T200, T300, T400, T600, T800, T900}
I3	{T300, T500, T600, T700, T800, T900}
I4	{T200, T400}
I5	{T100, T800}

Table 2.4: 2-Itemsets in Vertical Data Format (J. Han et al., 2012c)

<i>itemset</i>	<i>TID_set</i>
{I1, I2}	{T100, T400, T800, T900}
{I1, I3}	{T500, T700, T800, T900}
{I1, I4}	{T400}
{I1, I5}	{T100, T800}
{I2, I3}	{T300, T600, T800, T900}
{I2, I4}	{T200, T400}
{I2, I5}	{T100, T800}
{I3, I5}	{T800}



Table 2.5: 3-Itemsets in Vertical Data Format (J. Han et al., 2012c)

<i>itemset</i>	<i>TID_set</i>
{I1, I2, I3}	{T800, T900}
{I1, I2, I5}	{T100, T800}

For the EClat algorithm, the database is not required to be scanned multiple times in order to identify the  $(k+1)$ -itemsets. The database is only scanned once to transform data from the horizontal format into the vertical format. After scanning the database once, the  $(k+1)$ -itemsets are discovered by just intersecting the  $k$ -itemsets with one another. Apart from this, the database is also not required to be scanned multiple times in order to identify the support count of every frequent itemset because the support count of every itemset is simply the total count of transactions that contain the particular itemset. However, the transactions involved in an itemset can be quite a lot, making it to take extensive memory space and processing time for intersecting the itemsets (Z. Zhang, Ji, & Tang, 2013).

#### 2.2.3.4 *TreeProjection Algorithm*

TreeProjection is an algorithm that mines frequent itemsets through a few different searching techniques for constructing a lexicographic tree, such as depth-first (Agarwal, Aggarwal, & Prasad, 2000), breadth-first (Agarwal, Aggarwal, & Prasad, 2001), or a mixture of the two. In this algorithm, the support of each frequent itemset in every transaction is counted and projected onto the lexicographic tree as a node. This greatly improves the performance of calculating the total transactions that contain a particular frequent itemset. An example of the lexicographic tree that represents the frequent items is shown in Figure 2.10.

In the hierarchical structure of a lexicographic tree, only the subset of transactions that can probably hold the frequent itemsets will be searched by the algorithm. The search is performed by traversing the lexicographic tree with a top-down approach.

Apart from the lexicographic tree, a matrix structure is used to provide a more efficient method for calculating the frequent itemsets that have very low level of support count. In this way, cache implementations can be made available efficiently for the execution of the algorithm. However, the main problem faced by this algorithm is that different representations of the lexicographic tree present different limitations in terms of efficiency at memory consumption (C. C. Aggarwal, Bhuiyan, & Hasan, 2014).

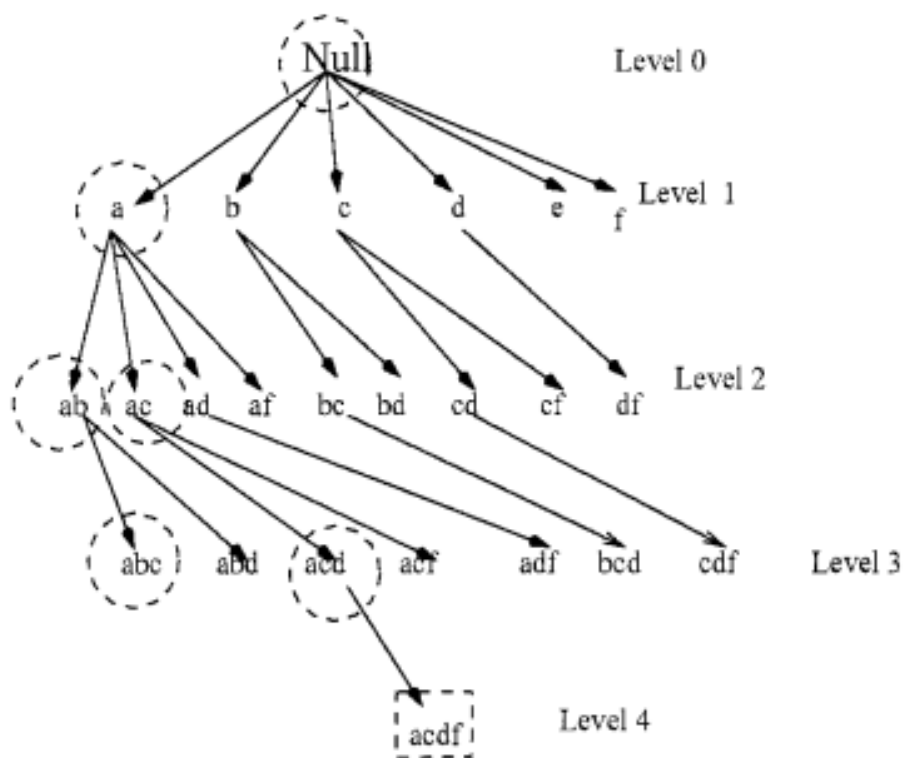


Figure 2.10: Lexicographic Tree (Agarwal et al., 2001)

### 2.2.3.5 COFI Algorithm

Co-Occurrence Frequent Itemset (COFI) (El-Hajj & Zaïane, 2003) is an algorithm that mines frequent itemsets using a pruning method that reduces the use of memory space significantly. Its intelligent pruning method constructs relatively small trees from the FP-Tree on the fly, and it is based on a special property that is derived from the top-down approach mining technique of the algorithm (Hemalatha, Krishnan,

Senthamarai, & Hemamilini, 2005). Some examples of the COFI-Trees are shown in Figure 2.11.

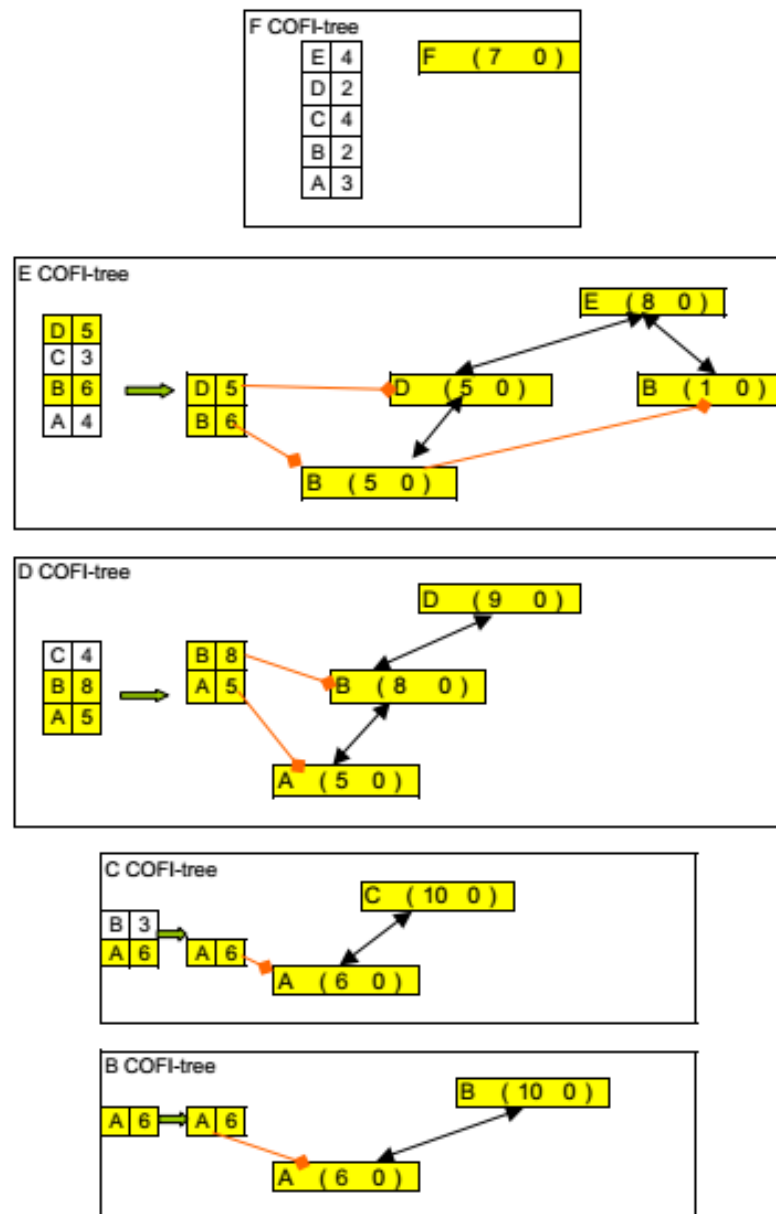


Figure 2.11: COFI-Trees (El-Hajj & Za'iane, 2003)

Comparing to the FP-Growth algorithm, the COFI algorithm is better mainly in terms of memory consumption and occasionally in terms of execution run time. This

is because of the following two implementations: (1) A non-recursive method is used during the process of mining to traverse through the COFI-Trees in order to generate the entire set of frequent patterns. (2) The pruning method implemented in the algorithm has removed all the non-frequent patterns, so only frequent patterns are left in the COFI-Trees. However, if the threshold value of the minimum support is low, the performance of the algorithm degrades in a sparse database (Gupta & Garg, 2011).

#### *2.2.3.6 TM Algorithm*

Transaction Mapping (TM) (Song & Rajasekaran, 2006) is an algorithm that mines frequent itemsets using the vertical data representation like the EClat algorithm. In this algorithm, the transaction IDs of every itemset are transformed and mapped into a list of transaction intervals at another location. Then, intersection will be performed between the transaction intervals in a depth-first search order throughout the lexicographic tree to count the itemsets. An example of the transaction mapping technique is shown in Figure 2.12.

When the value of minimum support is high, the transaction mapping technique is able to compress the transaction IDs into the continuous transaction intervals significantly. As the itemsets are compressed into a list of transaction intervals, the intersection time is greatly saved. The TM algorithm is proven to be able to gain better performance over the FP-Growth and dEClat algorithms on data sets that contain short frequent patterns. Apart from this, it is suitable to be used for mining the specifications of software from the traces of program execution (Jeevarathinam & Thanamani, 2009). Even though it is so, the TM algorithm is still slower in terms of processing speed compared to the FP-Growth\* algorithm.

Item	Mapped transaction interval list
1	[1,500]
2	[1,200], [501,800]
3	[1,300], [501,600]
4	[601,800]

Figure 2.12: Example of Transaction Mapping (Song & Rajasekaran, 2006)

### 2.2.3.7 P-Mine Algorithm

P-Mine (Baralis, Cerquitelli, Chiusano, & Grand, 2013) is an algorithm proposed by Elena Baralis to mine frequent itemsets using a parallel disk-based approach on a multi-core processor. It decreases the time required to produce a dense version of the data set on disk using the VLDBMine data structure. A Hybrid-Tree (HY-Tree) is used in the VLDBMine data structure to store the entire data set and other information required to support the data retrieval process. To enhance the efficiency for disk access, a pre-fetching technique has been implemented to load multiple projections of the data set into different processor cores for mining the frequent itemsets. Finally, the results are gathered from each processor core and merged in order to construct the entire frequent itemsets. The architecture of the P-Mine algorithm is shown in Figure 2.13.



Figure 2.13: Architecture of the P-Mine Algorithm (Baralis et al., 2013)

As the data set is represented in the VLDBMine data structure, the performance and scalability of Frequent Itemset Mining (FIM) are further improved. This is because the HY-Tree of the VLDBMine data structure enables the data to be selectively accessed in order to effectively support the data-intensive loading process with a minimized cost. Apart from this, when the process of FIM is executed across different processor cores in parallel at the same time, the performance is optimized locally on every node. However, the algorithm can only be optimized to the maximum level when multiple cores are available in the processor.

### 2.2.3.8 LP-Growth Algorithm

Linear Prefix Growth (LP-Growth) (Pyun, Yun, & Ryu, 2014) is an algorithm proposed by Gwangbum Pyun to mine frequent itemsets using arrays in a linear structure. It minimizes the information required in the data mining process by constructing a Linear Prefix Tree (LP-Tree) that is composed of arrays instead of pointers. With this implementation, the efficiency in memory usage is increased since the information of connection between different nodes is reduced significantly.

A structure of the Linear Prefix Nodes (LPNs) in the LP-Tree is shown in Figure 2.14. One LP-Tree is composed of multiple LPNs in a linear structure. Every set of frequent items is stored into different nodes that are composed of multiple arrays. In order to link all the arrays together, every array consists of a header in its first location to indicate its parent array. If the LPN is the first node to be inserted in the LP-Tree, the header of that LPN indicates the root of the LP-Tree.

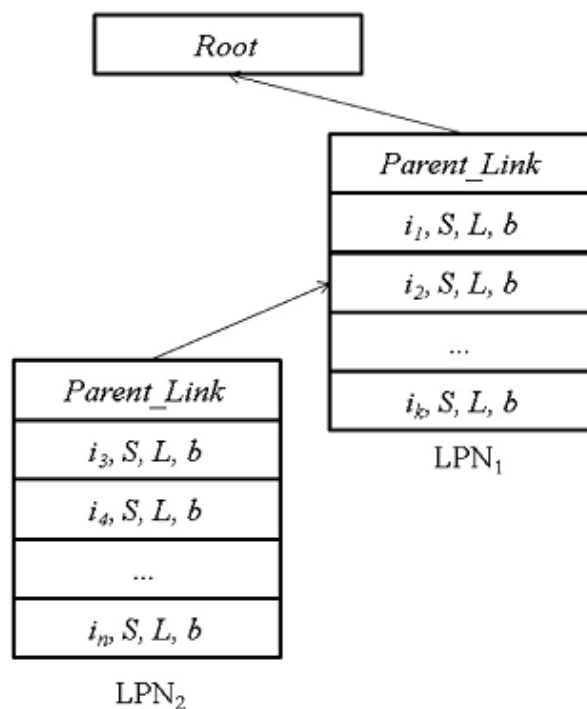


Figure 2.14: Structure of Linear Prefix Nodes (LPNs) (Pyun et al., 2014)

The LP-Growth algorithm is able to generate the LP-Tree in a faster manner compared to the FP-Growth algorithm. This is because a series of array operations are used in the LP-Growth algorithm to create multiple nodes at the same time, while the FP-Growth algorithm creates the nodes one at a time. As the nodes are saved in the form of arrays, any parent or child nodes are accessible without using any pointers while searching through the LP-Tree. In addition, it is also possible to traverse through the LP-Tree in a faster manner because the corresponding memory locations can be directly accessed when all the nodes are stored using the array structure. Apart from this, when pointers are not utilized to link up all the nodes, the memory usage for every node becomes comparatively less as well. However, the LP-Growth algorithm has a limitation in the insertion process of nodes because the items from a transaction may be saved in various LPNs (Jamsheela & G., 2015). Therefore, to insert a transaction into the LP-Tree successfully, the memory needs to be freed continuously.

#### *2.2.3.9 Can-Mining Algorithm*

Can-Mining (Hoseini, Shahraki, & Neysiani, 2015) is an algorithm that mines frequent itemsets from a Canonical-Order Tree (Can-Tree) in an incremental manner. Similar to the FP-Growth algorithm, a header table that contains information of all the database items is used in the algorithm. The header table consists of the frequency of each item and its pointers to the first and last nodes that contain the item in the Can-Tree. In order to extract frequent patterns from the Can-Tree, a list of frequent items is required for the algorithm to perform the mining operation. The Can-Mining algorithm is able to reduce the time of mining in nested Can-Trees because only frequent items are appended into the trees in a predefined order. When the minimum support has a high threshold value, the Can-Mining algorithm is able to outperform the FP-Growth algorithm. However, if the threshold value of the minimum support is much lower, the FP-Growth algorithm is more efficient. The architecture of the Can-Mining algorithm is shown in Figure 2.15.



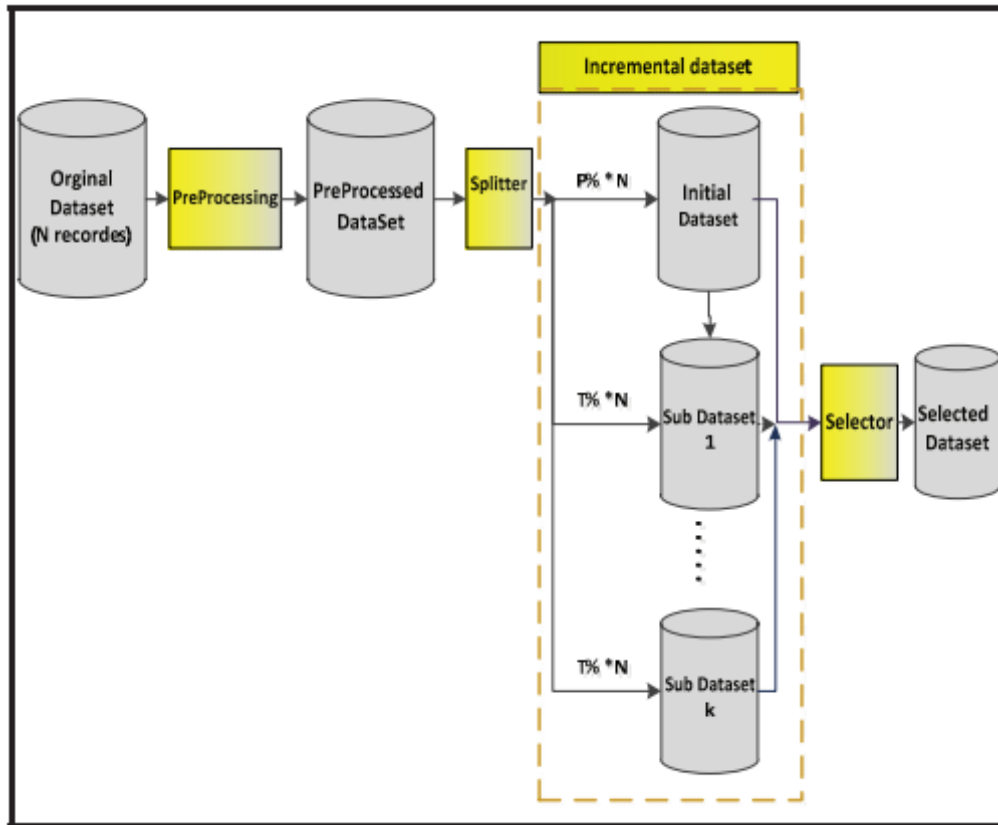


Figure 2.15: Architecture of the Can-Mining Algorithm (Hoseini et al., 2015)

### 2.2.3.10 EXTRACT Algorithm

EXTRACT (Feddaoui, Felhi, & Akaichi, 2016) is an algorithm proposed by Ilhem Feddaoui to mine frequent itemsets using the mathematical concept of Galois lattice. The architecture of the EXTRACT algorithm is shown in Figure 2.16. It is partitioned into four functions for calculating the support count, combining the itemsets, eliminating the itemsets that are repeated, and extracting association rules from the frequent itemsets.

First, EXTRACT calculates the support count of each frequent 1-itemset that satisfied the minimum support determined by the user. All frequent 1-itemset that did not satisfy the minimum support determined by the user will be removed from the calculation. Then, EXTRACT will combine the itemsets to discover all the possible combinations of frequent itemsets. After identifying all the frequent itemsets, the combinations of frequent itemsets that are redundant will be eliminated. Once all the

unique frequent itemsets are mined, the association rules that satisfied the minimum confidence determined by the user will be generated. All association rules that did not satisfy the minimum confidence determined by the user will be removed from the rule discovery process.

EXTRACT outperforms the Apriori algorithm for mining more than 300 objects and 10 attributes with an execution time that does not exceed 1200 milliseconds. However, since the frequent itemsets that have been mined are not stored in any disk or database, the algorithm is required to be executed again in order to mine the new set of frequent itemsets if there is a change in the data set.

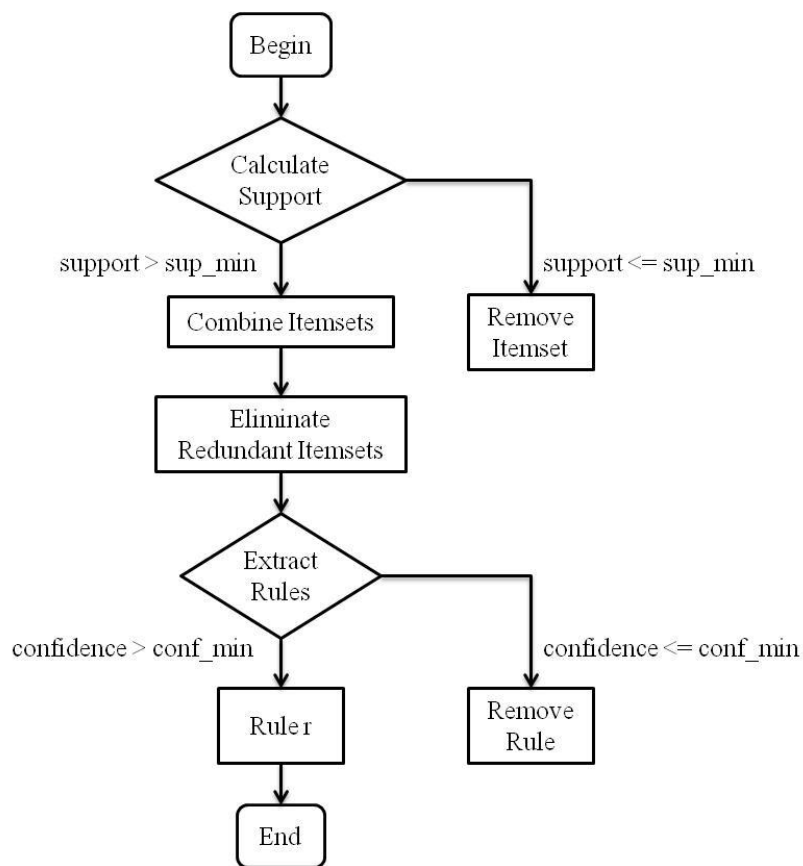


Figure 2.16: Architecture of the EXTRACT Algorithm (Feddaoui et al., 2016)

#### 2.2.3.11 HYBRID Algorithm

HYBRID (Zulkurnain & Shah, 2017) is an algorithm that combines the Improved Apriori (Wei, Yang, & Liu, 2009) and FP-Growth (J. Han et al., 2000) algorithms for mining the frequent itemsets of data. The unifying process has concatenated the significant features of the two algorithms in order to produce a faster execution time and lesser memory consumption in mining the frequent itemsets of data.

The first portion of the algorithm utilizes the Improved Apriori property to identify all the maximal frequent itemsets that have a support value that is equivalent to or more than the minimum support specified by the users. In this way, the data set is pruned to become smaller and easier for traversing. The data set which has been pruned serves as an input to the second portion of the algorithm for discovering all the frequent-1 itemsets and removing all the infrequent-1 itemsets. Finally, the transactions which have been pruned are used to construct an FP-Tree using the FP-Growth algorithm.

HYBRID produced better results in execution time and memory consumption compared to both the Improved Apriori and FP-Growth algorithms. This is because the candidate itemsets are not required to be generated and the FP-Tree is constructed for a pruned data set only. Therefore, the FP-Tree can be easily fit into the main memory for mining the frequent itemsets of data. However, the execution time of HYBRID is almost the same as FP-Growth for discovering the frequent itemsets with a higher support count.

#### 2.2.3.12 FPNR-Growth Algorithm

FPNR-Growth (Jiang & He, 2017) is an algorithm that is evolved from the FP-Growth (J. Han et al., 2000) algorithm for mining the frequent itemsets of data. In this algorithm, a structure like the FP-Tree is used to store the frequent patterns, which is called the FPNR-Tree. The structure of an FPNR-Tree is shown in Figure 2.17. The information about how a node is associated with another node is stored into an array, which is called the FPNR-Array.

First, FPNR-Growth compress the data set into the FPNR-Tree in a slightly different manner compared to FP-Growth. Every node in the FPNR-Tree has a pointer that connects itself to the parent node, and every leaf node has a pointer that connects itself to the next leaf node. Then, the FPNR-Array is constructed to store the information about how a node is associated with another node in the FPNR-Tree. This is accomplished by having every element in the FPNR-Array to hold the index of its parent node in the array. Next, a HashTable is used to keep track of the location information of every element in the FPNR-Array. Finally, the frequent itemsets are mined from the FPNR-Tree and FPNR-Array with the referencing information of the HashTable.

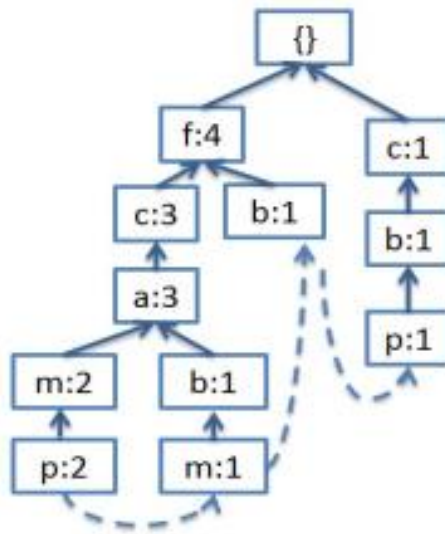


Figure 2.17: Structure of the FPNR-Tree (Jiang & He, 2017)

FPNR-Growth outperforms FP-Growth in terms of execution time and memory consumption because a non-recursive method is used in mining the frequent itemsets of data. However, the implementation of FPNR-Growth is more complicated because different data structure like tree, array and table are used at the same time for the mining of the frequent itemsets to be carried out.

### 2.2.3.13 SSFIM Algorithm

SSFIM (Djenouri, Djenouri, Lin, & Belhadi, 2018) is an algorithm that implements a single scan approach for Frequent Itemset Mining (SSFIM). Alternative approaches like heuristic (EA-SSFIM) and parallel implementation on the Hadoop clusters (MR-SSFIM) are also implemented for the SSFIM algorithm. The architecture of these alternative approaches for SSFIM is shown in Figure 2.18.

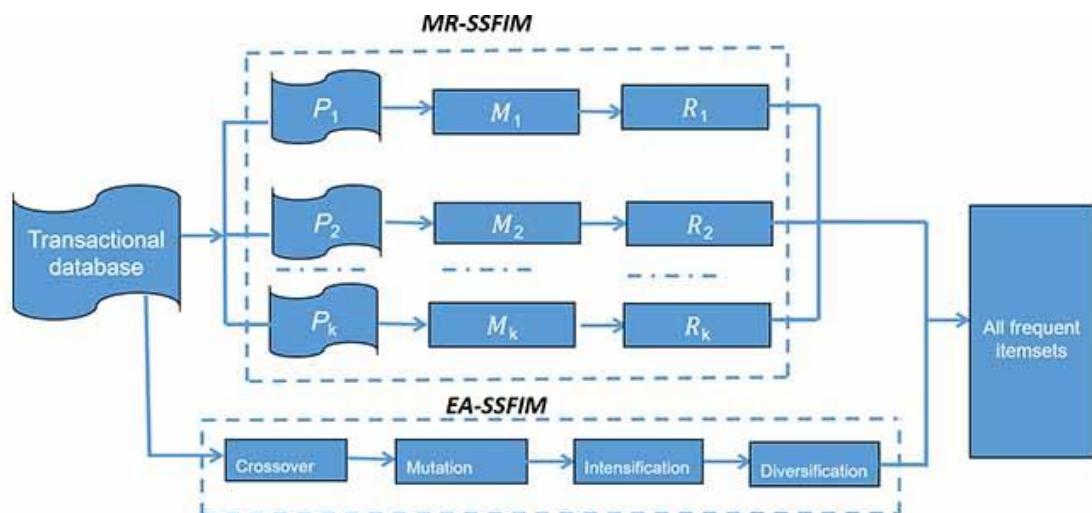


Figure 2.18: Alternative Approaches for SSFIM (Djenouri et al., 2018)

The transactions in a data set are processed only once by the SSFIM algorithm. After the data set is scanned once by SSFIM, all the itemsets of every transaction and their support counts are generated into a hash table by the algorithm. If an itemset has already been generated at the processing of a previous transaction, its support count is just incremented by one. Otherwise, the itemset will be inserted into the hash table and its support count will be set to one. This process is repeated until every transaction in the data set is processed.

In this way, SSFIM discovers all the frequent itemsets with just one scan of the entire data set. SSFIM is able to outperform other algorithms because it does not need to scan the data set multiple times in order to generate all the frequent itemsets.

However, a heavy computation is still required to be performed by SSFIM since the candidate itemsets are still necessary to be generated for every transaction in the whole data set.

#### *2.2.3.14 PFIM Algorithm*

Precomputation-Based Frequent Itemset Mining (PFIM) (X. Han et al., 2019) is an algorithm that quickly computes the frequent itemsets on massive amount of data. PFIM partitions the transaction table into two main parts, having a large table that stores the historical data, and a relatively small table that stores the newly generated data. The quasi-frequent itemsets on the large table that contains the historical data are pre-constructed first by the algorithm. In this manner, PFIM managed to generate the frequent itemsets on massive amount of data efficiently. In order to maintain such efficiency, an incremental update technique is designed to merge the old table and the new table to re-construct the quasi-frequent itemsets.

A number of experiments conducted on some synthetic and real-life data sets have indicated that PFIM has a significant advantage and able to run two orders of magnitude faster compared to other algorithms. This is because the frequent itemsets are only required to be generated from the new table since the quasi-frequent itemsets have already been pre-computed in the old table. However, if there are changes in the historical data due to some updates or corrections, the PFIM is not able to generate the frequent itemsets in an accurate manner.

#### *2.2.3.15 Comparison of Frequent Pattern Mining Techniques*

In general, the algorithms for Frequent Pattern Mining (FPM) can be classified into three main categories (C. C. Aggarwal et al., 2014), namely Join-Based, Tree-Based, and Pattern Growth as shown in Figure 2.19. First, the Join-Based algorithms apply a bottom-up approach to identify the frequent items in a data set and expand them into larger itemsets as long as those itemsets appear more than a minimum threshold value defined by the user in the database. Then, the Tree-Based algorithms

use set-enumeration concepts to solve the problem of frequent itemset generation by constructing a lexicographic tree that enables the items to be mined through a variety of ways like the breadth-first or depth-first order. Finally, the Pattern Growth algorithms implement a divide-and-conquer method to partition and project databases depending on the presently identified frequent patterns and expand them into longer ones in the projected databases.

The advantages and disadvantages of various significant FPM algorithms are summarized in Table 2.6. The initial three most popular algorithms in FPM are Apriori, FP-Growth and EClat. Each of these algorithms has its strengths in mining the frequent itemsets of data. For example, Apriori applies an iterative level-wise search technique to identify  $(k+1)$ -itemsets from  $k$ -itemsets, and FP-Growth preserves the information of how all itemsets are associated by utilizing an FP-Tree that compresses the amount of data to be searched. Then, EClat does not need to scan through the entire database in order to determine the support count of  $(k+1)$ -itemsets. However, each of these algorithms has its limitations in mining the frequent itemsets of data too. For instance, Apriori needs to generate a lot of candidate sets if the  $k$ -itemsets are large in numbers, and needs to scan the database repeatedly for identifying the support count of the itemsets. For FP-Growth, building the FP-Tree is time consuming if the data set is very large. Then, EClat requires more memory space and processing time for intersecting the long TID sets.

From these three most common algorithms in FPM, many algorithms have been proposed or improved from them. Some of the significant ones are TreeProjection, Co-Occurrence Frequent Itemset (COFI), Transaction Mapping (TM), P-Mine, Linear Prefix Growth (LP-Growth), Can-Mining and EXTRACT. Similarly, each of these algorithms has its strengths in mining the frequent itemsets of data. Among these FPM algorithms, most of them are categorized as the Pattern Growth algorithms, which include COFI, P-Mine, LP-Growth, Can-Mining and EXTRACT. COFI utilizes a pruning technique to decrease the use of memory space significantly, and P-Mine improves its performance and scalability by mining the frequent itemsets in parallel using multiple processor cores. LP-Growth produces the LP-Tree in a faster way because a series of array operations are implemented to construct multiple nodes

together, and Can-Mining is able to mine the frequent itemsets in a very fast manner when the minimum support has a high threshold value. Then, EXTRACT is able to mine more than 300 objects and 10 attributes with a run time that does not exceed 1200 milliseconds.

Nevertheless, these Pattern Growth algorithms have their limitations in mining the frequent itemsets of data also. The mining performance of COFI is reduced in a sparse database when the threshold value of the minimum support is low, while the mining performance of P-Mine can only be optimized to the highest level when multiple cores are available in the processor. LP-Growth needs to free the memory continuously because the items from the same transaction can be stored in different Linear Prefix Nodes (LPNs), and the mining run time of Can-Mining will become longer if the threshold value of the minimum support becomes very low. Finally, EXTRACT needs to be executed again in order to mine the new set of frequent itemsets if there is a change in the data set.

Amongst the existing Pattern Growth algorithms, most of them are evolved from the FP-Growth algorithm. This is because FP-Growth generates all the frequent patterns using only two scans for the data set, representing the entire data set with a compressed tree structure, and decreases the execution time by removing the need to generate the candidate itemsets (Mittal et al., 2015). Although the existing FPM algorithms are able to mine the frequent patterns in a data set by identifying the association between different data items, a lengthy processing run time and a large consumption of memory space are still the two major problems faced by FPM especially when the amount of data is big in a data set.

In a situation where the threshold of the minimum support is set to a lower value, the patterns of items that exist less frequently in the data set are also required to be included into the FPM process. Consequently, more patterns need to be mined from the data set and this causes the mining run time to be drastically increased. In order to reduce the run time of mining, many existing algorithms prune the data to be mined by ignoring the patterns that exist less frequently in the data set. However, this is not a suitable technique to implement FPM because patterns that exist less frequently in the



data set do not mean that they are not important. Sometimes problems can happen as a result of the pattern that exist less frequently in the data set.

Apart from this, most of the existing algorithms mine the frequent patterns into the Random Access Memory (RAM). As the frequent patterns are mined into the RAM, a problem of power failure or system down will cause all the frequent patterns that have been mined previously to be lost. This is because whatever that is stored into the RAM will not be retained after a power failure or system down situation. As a result, all the frequent patterns that have been mined previously need to be mined again whenever there is a problem of power failure or system down.

Therefore, a more robust and efficient FPM algorithm needs to be developed for identifying the important frequent patterns in a big data set. The algorithm should be able to mine the frequent patterns from a data set in a shorter run time using less memory consumption, even though the threshold of minimum support is set to a lower value. In addition, the algorithm should be able to retrieve the frequent patterns even after a power failure or system down situation without the need to mine the entire data set again.

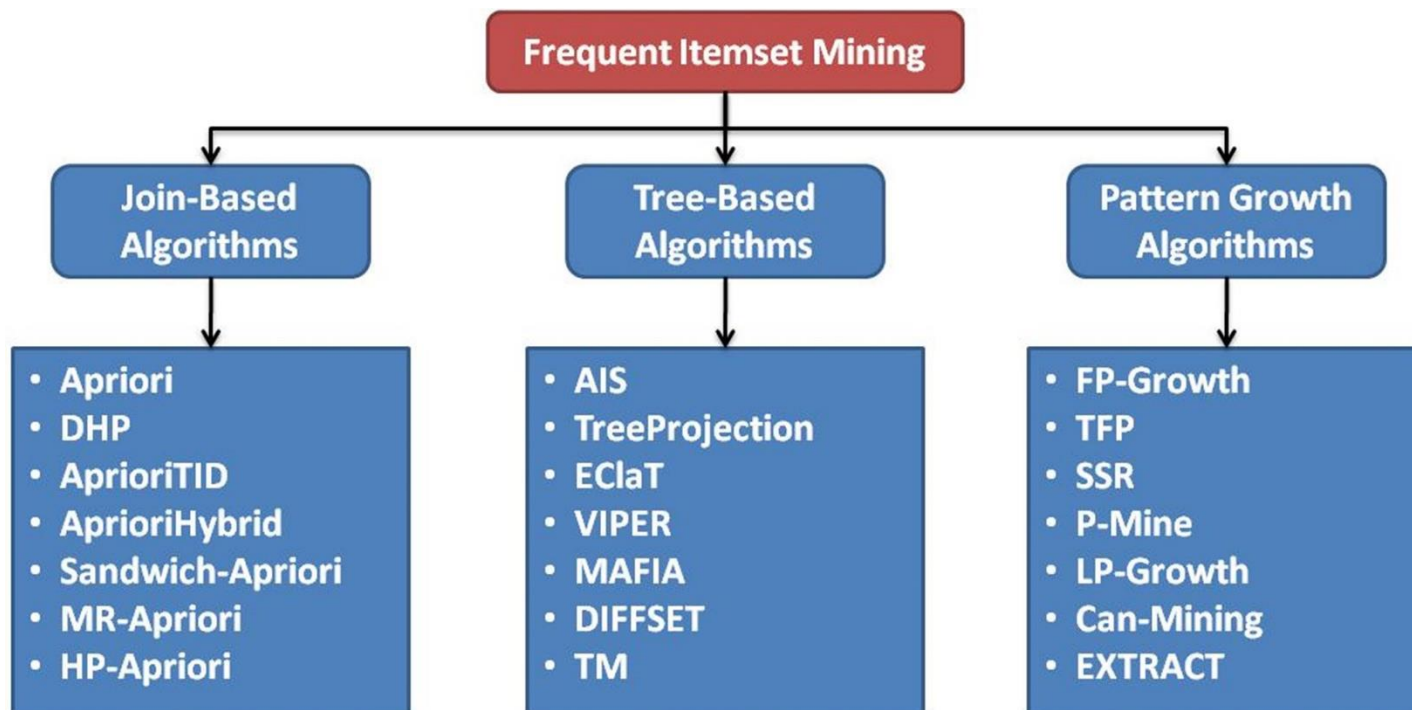


Figure 2.19: Classification of Frequent Pattern Mining Algorithms

Table 2.6: Comparison of Frequent Pattern Mining Algorithms

FPM Algorithm	Advantages	Disadvantages
<b>Apriori</b> (Agrawal & Srikant, 1994)	Uses an iterative level-wise search technique to discover (k+1)-itemsets from k-itemsets.	Has to produce a lot of candidate sets if k-itemsets are more in numbers.  Has to scan the database repeatedly to determine the support count of the itemsets.
<b>FP-Growth</b> (J. Han & Pei, 2000)	Preserves the association information of all itemsets.  Shrinks the amount of data to be searched.	Constructing the FP-Tree is time consuming if the data set is very large.
<b>EClat</b> (Zaki, 2000)	Scanning the database to find the support count of (k+1)-itemsets is not required.	More memory space and processing time are required for intersecting long TID sets.
<b>TreeProjection</b> (Agarwal et al., 2001)	Identifies the frequent itemsets in a fast manner because only the subset of transactions that can probably hold the frequent itemsets is searched by the algorithm.	Different representations of the lexicographic tree present different limitations in terms of efficiency for memory consumption.
<b>COFI</b> (El-Hajj & Zaïane, 2003)	Uses a pruning method to reduce the use of memory space significantly by constructing smaller COFI-Trees while mining for the frequent itemsets.	The performance of the algorithm degrades in a sparse database if the threshold value of the minimum support is low.

Table 2.6 continued: Comparison of Frequent Pattern Mining Algorithms

FPM Algorithm	Advantages	Disadvantages
<b>TM</b> (Song & Rajasekaran, 2006)	Compresses the itemsets into a list of transaction intervals in order to greatly save the intersection time for mining the frequent itemsets.	Still slower in terms of processing speed compared to the FP-Growth* algorithm.
<b>P-Mine</b> (Baralis et al., 2013)	Optimizes performance and scalability by executing the mining of frequent itemsets in parallel with multiple processor cores.	The algorithm can only be optimized to the maximum level when multiple cores are available in the processor.
<b>LP-Growth</b> (Pyun et al., 2014)	Generates the LP-Tree in a faster manner as a series of array operations are used to create multiple nodes together.	Memory needs to be freed continuously as the items from a transaction may be saved in various LPNs.
<b>Can-Mining</b> (Hoseini et al., 2015)	Outperforms the FP-Growth algorithm when the minimum support has a high threshold value.	Mining time is longer if the threshold value of the minimum support is much lower.
<b>EXTRACT</b> (Feddaoui et al., 2016)	Mines more than 300 objects and 10 attributes with an execution time that does not exceed 1200 milliseconds.	The algorithm needs to be executed again in order to mine the new set of frequent itemsets if there is a change in the data set.

Table 2.6 continued: Comparison of Frequent Pattern Mining Algorithms

<b>FPM Algorithm</b>	<b>Advantages</b>	<b>Disadvantages</b>
<b>HYBRID</b> (Zulkurnain & Shah, 2017)	The candidate itemsets are not required to be generated and the FP-Tree is constructed for a pruned data set only.	The execution time of HYBRID is almost the same as FP-Growth for discovering the frequent itemsets with a higher support count.
<b>FPNR-Growth</b> (Jiang & He, 2017)	Outperforms FP-Growth in terms of execution time and memory consumption because a non-recursive method is used in mining the frequent itemsets of data.	Its implementation is more complicated because different data structure like tree, array and table are used at the same time for the mining of the frequent itemsets to be carried out.
<b>SSFIM</b> (Djenouri et al., 2018)	Outperforms other algorithms because it does not need to scan the data set multiple times in order to generate all the frequent itemsets.	A heavy computation is still required to be performed by SSFIM since the candidate itemsets are still necessary to be generated for every transaction in the whole data set.
<b>PFIM</b> (X. Han et al., 2019)	The frequent itemsets are only required to be generated from the new table since the quasi-frequent itemsets have already been pre-computed in the old table.	If there are changes in the historical data due to some updates or corrections, the PFIM is not able to generate the frequent itemsets in an accurate manner.

## **2.3 Data Extraction**

It is a normal situation in any company for data to be distributed at various parts of the organization especially the corporation is composed of multiple branches in different locations. In order to mine the data for identifying hidden trends that may provide insights to business analysis and decision making, it is essential to extract, transform and load (ETL) all the necessary data from a variety of sources into a data warehouse.

### **2.3.1 Data Extraction Techniques**





A number of ETL techniques are analyzed in this research for constructing a useful data mining algorithm for Frequent Itemset Mining (FIM). This section describes how the fundamental and significant techniques in ETL work and compares their advantages and disadvantages respectively.

#### *2.3.1.1 Incremental ETL*

Incremental ETL (Jörg & Deßloch, 2008) is a technique for extracting, transforming, and loading only the changed data from heterogeneous sources and propagating it into the data warehouse. This technique is designed to be so due to the increased data volumes and shortened data loading intervals of the organization. It is surely an exhaustive way to extract all the data from different sources and reconstruct the entire data warehouse in each ETL cycle. Thus, implementing incremental update to any kind of corporate data is definitely more efficient than applying full update because the volume of updated data is usually smaller compared to the entire data set.

The approaches used in this technique to gather the updates from the data sources and refresh the data warehouse are the Change Data Capture (CDC) and Change Data Application (CDA) methods. A matrix that represents the status of change data that

need to be reflected into the data warehouse with the following indications is shown in Figure 2.20:

-  represents records that have been inserted,
-  represents records that have been deleted,
-  represents the current state of records that have been updated,
-  represents the initial state of records that have been updated.













































					join predicate unaffected		join predicate affected	
								
join predicate unaffected								
								
								
								
								
								
join predicate affected								

Figure 2.20: Matrix Representation of an Incremental Join (Jörg & Deßloch, 2008)

### 2.3.1.2 Real-Time ETL

Real-Time ETL (Santos & Bernardino, 2008) is a technique that enables users to make use of the data available in a data warehouse for business analysis and decision making in a real time mode. The ETL tasks need to be implemented in real time because most of the enterprises require the ability of decision support for their business in a real time manner. As the size of a data warehouse increases, it becomes very difficult to update the data warehouse efficiently in real-time. The implementation of ETL for the transactional data in real-time will overload the server due to its update frequency and data volume. As a result, the immense and complex

operation on the data warehouse will significantly degrade the performance of Online Analytical Processing (OLAP).

In order to enable new information to be disseminated across an organization in real-time while maintaining the capability of continuous data integration, Real-Time ETL creates an exact structural replica of all the tables of the data warehouse with a unique sequential identifier for the latest data to be stored as shown in Figure 2.21. The temporary replicated tables are to be constructed without any data or settings like index, primary key, foreign key, or constraints of any kind so that the insertion of data can be performed much faster compared to the original big size tables. Moreover, this enables users who wish to query only the most recent information to only query the temporary replicated tables. The records in the temporary tables will be updated to the data warehouse and the temporary tables will be recreated when the OLAP performance becomes not that acceptable.

In a nutshell, the main contributions of Real-Time ETL are as follows:

1. Ensures the data to be up to date by integrating the most recent transactional data into the data warehouse rapidly and efficiently;
2. Optimizes the performance of OLAP while performing continuous data integration at the same time;
3. Maximizes the availability of the data warehouse by reducing its data synchronization time, in which the access from OLAP application is not in use.



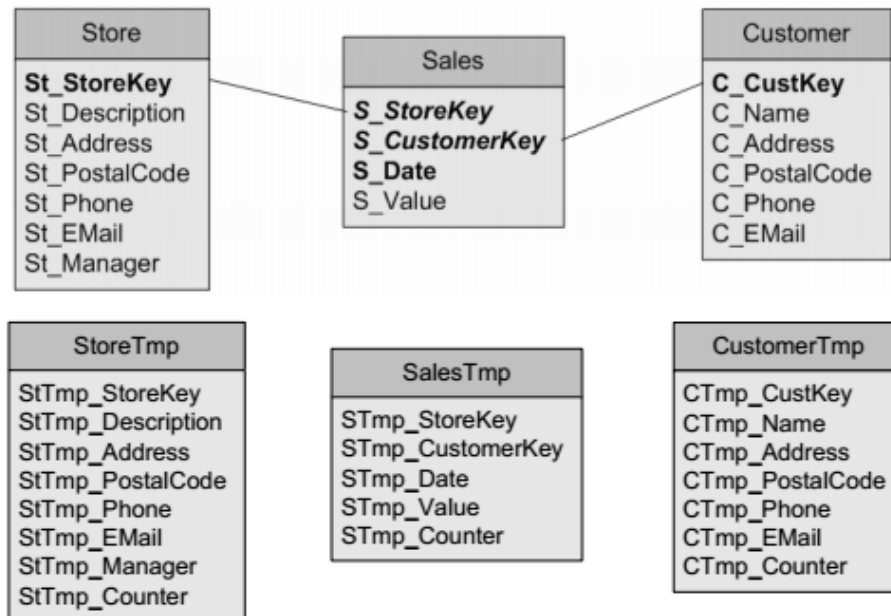


Figure 2.21: Sample Sales Data Warehouse Schema (Santos & Bernardino, 2008)

### 2.3.1.3 Parallel ETL

Parallel ETL (Thomsen & Pedersen, 2011) is a technique for extracting, transforming, and loading data by parallelizing the typical tasks that need to be performed in an ETL operation. The ETL tasks need to be implemented in parallel because it is time consuming to construct and to execute an ETL program. Parallelization of tasks is not an impossible implementation with the use of multi-core Central Processing Units (CPUs) in a computer as they are designed with such capability to enable true parallelism. However, not many programmers have exploited the power of parallel processing in multi-core CPUs for the implementation of ETL operations.

Parallel ETL pushes the ETL operations into separate processes and execute them in parallel as shown in Figure 2.22. It enables a single database connection to be used in parallel by sharing the ConnectionWrapper object to multiple processes using the SharedConnectionWrapperClient object. This allows several tasks to be executed together but only one operation is performed on the database at the same time. Hence,

by utilizing slightly more CPU time, the overall total time required to execute an ETL program is decreased significantly.

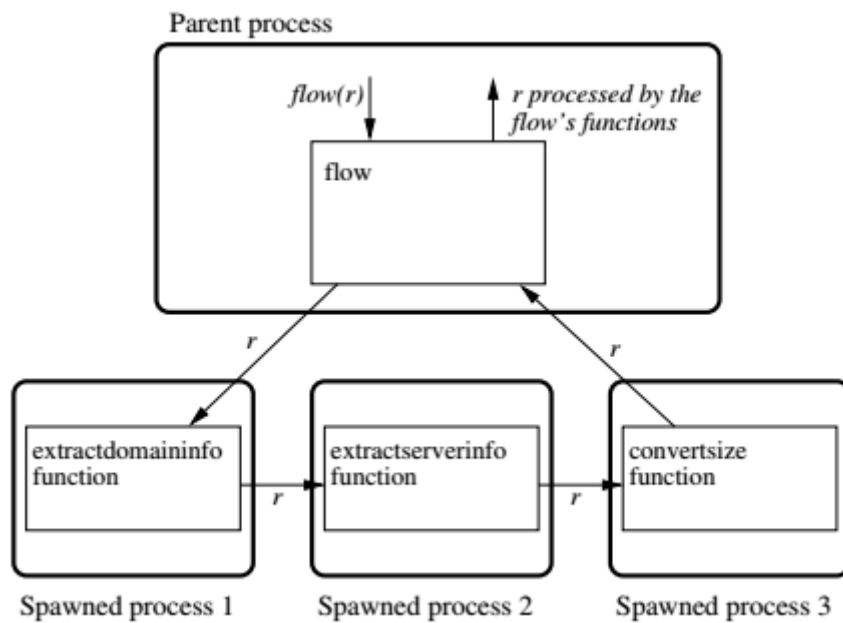


Figure 2.22: A Flow with Three Functions in Three Processes  
(Thomsen & Pedersen, 2011)

#### 2.3.1.4 Script Automated ETL

Script Automated ETL (Radhakrishna, SravanKiran, & Ravikiran, 2012) is a technique that utilizes the scripting method to automatically extract, transform, and load data from heterogeneous sources into a data warehouse. This technique is designed to be so due to the heavy manual tasks involved for executing the ETL process throughout the entire organization. It is definitely an exhaustive job to manually extract all the data from different sources, transform it according to some requirements and load it into the data warehouse from time to time. Thus, automating the ETL process with the relevant scripting technology is surely a more efficient solution for consolidating all the data of an enterprise.

The architecture of the Script Automated ETL framework is shown in Figure 2.23. In this technique, three different types of maps are generated for data extraction, transformation and loading. Apart from this, information about errors and other statistics are logged along the execution of the ETL process in order to ensure a proper debugging to be conducted when any problem is encountered in the midst of ETL execution. By automating the ETL process, the tasks of data processing and error handling are simplified, human effort involvement is reduced, execution is made faster, and performance is also improved for the entire system.

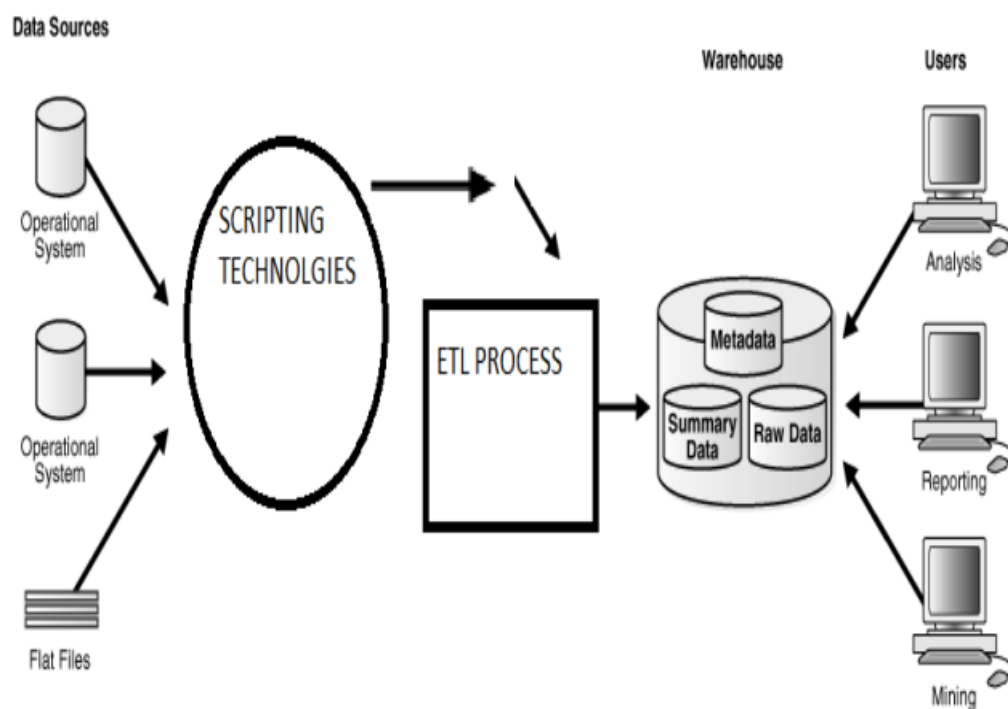


Figure 2.23: Architecture of Script Automated ETL (Radhakrishna et al., 2012)

### 2.3.1.5 Data Quality ETL

Data Quality ETL (Endler, 2012) is a technique for extracting, transforming, and loading data into a data warehouse using the data quality approach so that the data collected will be more useful for business analysis and decision making. It is important to use a data quality approach to implement the ETL process because the existence of invalid data will definitely affect the results of analysis for data (Cao,

Diao, Jiang, & Du, 2010). The data quality approach implemented in this technique is the Total Data Quality Management (TDQM) practice that delivers high quality information to users by treating data much like the products found in a manufacturing environment of any industry (Fisher, Lauria, Chengalur-Smith, & Wang, 2012).

The architecture of the Data Quality ETL framework implemented in the environment of a medical supply center is shown in Figure 2.24. The main features available in this technique includes metrics measurement, rules verification, quality requirements definition, cost estimation for resolving quality problems, and alarm warning when data of insufficient quality is identified. To accomplish a more robust data quality approach for implementing the ETL process, the system enables users to define and verify the appropriate characteristics for data quality monitoring apart from accessing the preset data quality features.

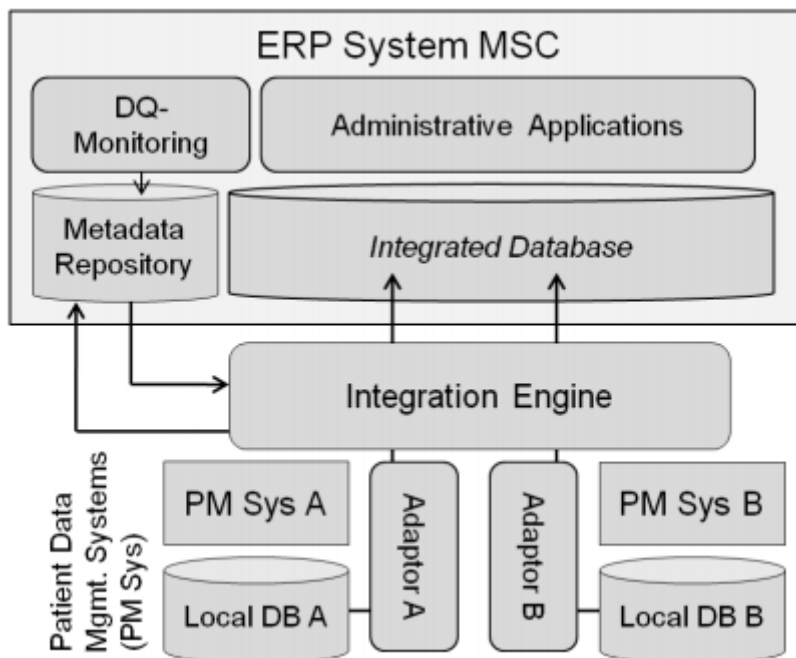


Figure 2.24: Architecture of Data Quality ETL (Endler, 2012)

### *2.3.1.6 Scalable and High Performance ETL*

Scalable and High Performance ETL (SETL) (K. Sun & Lan, 2012) is a technique that utilizes the subroutine attribute and data partition of the PERL programming language to develop the ETL process for extracting, transforming and loading large scale data from heterogeneous data sources into a data warehouse. It is a technique that is developed to execute the ETL process in a distributed environment easily and efficiently with high scalability as the removal or addition of an ETL job will not affect the current active ETL jobs due to the flexible plug-in design. As a result, there is no hindrance between different tasks that are involved in the entire ETL process.

The architecture of the SETL framework is shown in Figure 2.25. The SETL system is primarily formed by three components, namely the job collector, job dispatcher and ETL pipeline. The job collector is responsible to gather the ETL jobs specified by the users, analyze its syntax, and verify its semantics. Then, the job dispatcher is responsible to create the ETL pipelines and dispatch each ETL job to a pipeline according to the configuration specified by the users. After each ETL job is accomplished successfully, the data is loaded into the appropriate target database accordingly.

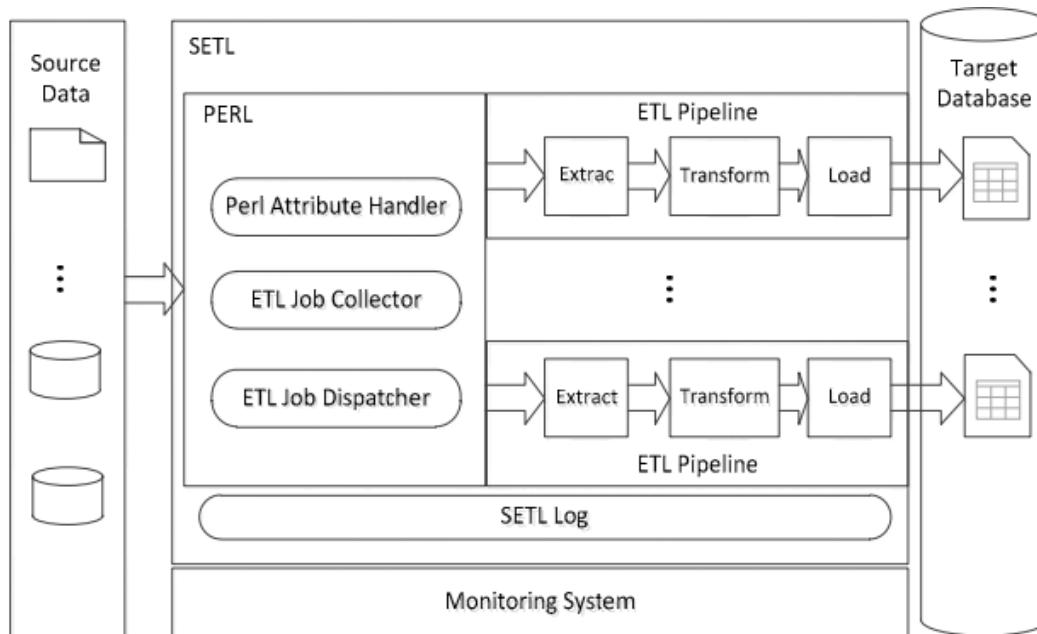


Figure 2.25: Architecture of Scalable and High Performance ETL  
(K. Sun & Lan, 2012)

### 2.3.1.7 Semantic ETL

Semantic ETL (Nath, Hose, & Pedersen, 2015) is a technique for extracting, transforming, and loading data into a data warehouse semantically so that the data is integrated in the right order. It is important to apply the semantic technology into the ETL process because it is often desirable for enterprises to include external data from different sources into their data warehouse in order to generate the required business knowledge.

The architecture of the Semantic ETL framework is shown in Figure 2.26. First, it integrates and processes data from different sources semantically with an ontology defined by the users. Then, the data will be transformed into triples of the Resource Description Framework (RDF) according to the ontology, and the created RDF dataset will be loaded into a triple store to be queried by the internal users. Last but not least, the created RDF dataset can also be connected to the external users for information sharing purpose.

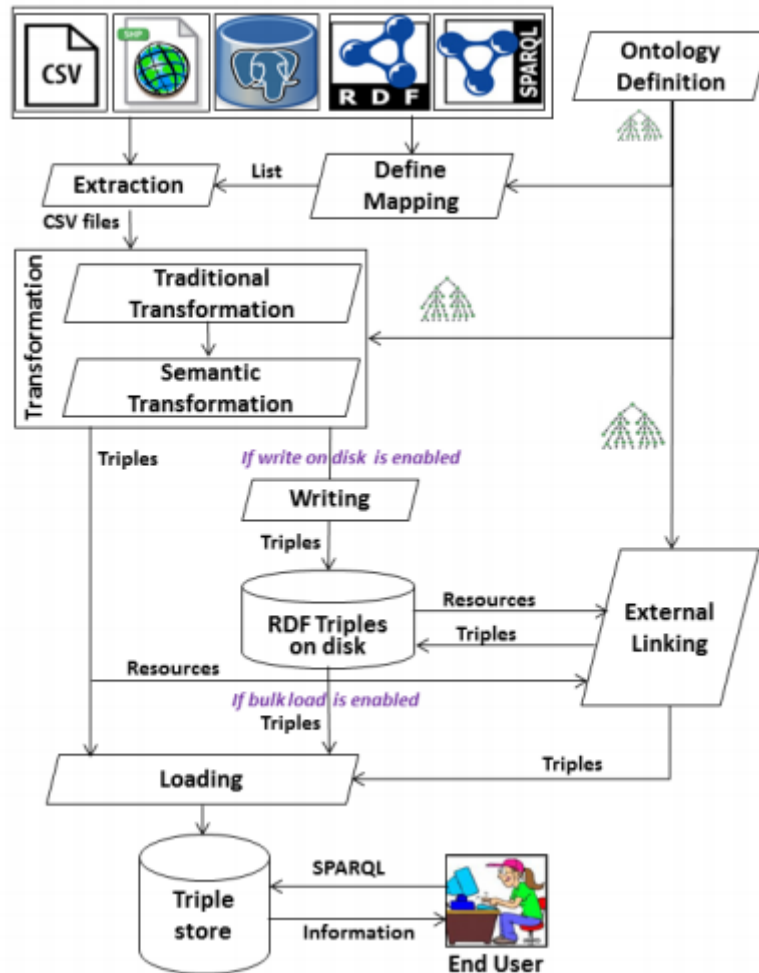


Figure 2.26: Architecture of Semantic ETL (Nath et al., 2015)

### 2.3.1.8 Comparison of Data Extraction Techniques

The advantages and disadvantages of various significant data extraction techniques are summarized in Table 2.7. Each of these methods for data extraction, transformation and loading consists of different capabilities that are useful for consolidating data to be in the appropriate forms so that it can be easily processed by the data mining algorithm.

Table 2.7: Comparison of Data Extraction Techniques

<b>ETL Technique</b>	<b>Advantages</b>	<b>Disadvantages</b>
<b>Incremental ETL</b> (Jörg & Deßloch, 2008)	Being more efficient than applying full update because the volume of updated data is usually smaller.	Lacks of the functionality for implementing real-time ETL in order to provide real-time decision support.
<b>Real-Time ETL</b> (Santos & Bernardino, 2008)	Ensures the data to be up to date by integrating the most recent transactional data into the data warehouse rapidly and efficiently.	Lacks of the capability to implement data quality even though real-time data can be retrieved for analysis from the data warehouse.
<b>Parallel ETL</b> (Thomsen & Pedersen, 2011)	Decreases the overall total time required to execute an ETL program by pushing the ETL operations into separate processes and execute them in parallel.	Lacks of the ability to scale the ETL process in a distributed environment although the ETL jobs can be separated into different processes for execution.
<b>Script Automated ETL</b> (Radhakrishna et al., 2012)	Being a more efficient solution for consolidating all the data of an enterprise when the ETL process is automated using the relevant scripting technology.	Lacks of the functionality for implementing the ETL process in an incremental manner so that the volume of data to be updated each time is reduced.



Table 2.7 continued: Comparison of Data Extraction Techniques

<b>ETL Technique</b>	<b>Advantages</b>	<b>Disadvantages</b>
<b>Data Quality ETL</b> (Endler, 2012)	Ensures the data collected to be more useful for business analysis using a data quality approach.	Lacks of the capability to implement real-time ETL even though quality data can be collected for analysis.
<b>Scalable and High Performance ETL</b> (K. Sun & Lan, 2012)	Executes the ETL process in a distributed environment easily and efficiently with high scalability using a flexible plug-in design to manage the ETL jobs.	Lacks of the ability to implement data quality although the ETL process can be scaled in a distributed environment easily and efficiently.
<b>Semantic ETL</b> (Nath et al., 2015)	Integrates data in the right order when data is extracted, transformed, and loaded into the data warehouse semantically.	Lacks of the functionality to implement parallel execution for the ETL jobs in different processes while applying the semantic technology in the ETL process.

## 2.4 Chapter Summary

Several techniques for data mining and data extraction have been reviewed in this chapter. The data mining techniques being discussed are Classification, Clustering, Outlier Detection and Frequent Pattern Mining (FPM). FPM has been chosen as the main topic of study in this research because it is the fundamental and significant step that is required to be further enhanced in many areas of data mining. The significant and recent algorithms in FPM that have been reviewed are Apriori, FP-Growth, EClat, TreeProjection, COFI, TM, P-Mine, LP-Growth, Can-Mining, EXTRACT, HYBRID, FPNR-Growth, SSFIM, and PFIM. Since extracting, transforming and loading (ETL) data from various sources are necessary before implementing data mining, different types of ETL techniques are analyzed in this chapter. The significant and recent ETL techniques that have been reviewed are Incremental ETL, Real-Time ETL, Parallel ETL, Script Automated ETL, Data Quality ETL, Scalable and High Performance ETL, and Semantic ETL. When the amount of data is big in a data set, the two major problems faced by the existing FPM algorithms are a lengthy processing time and a large consumption of memory space. Therefore, a more robust FPM algorithm needs to be designed and implemented.

## CHAPTER 3

### RESEARCH METHODOLOGY

This chapter discusses about the methodology that has been selected to perform the research of this study. First, an overview of research and the various types of methodologies are presented in Section 3.1 for a suitable methodology to be selected to conduct this research. Then, the selected research method for this study is described in Section 3.2. Finally, the entire flow of research, development and evaluation procedures of the selected research method are discussed in Section 3.3.

#### **3.1 Overview of Research and Types of Methodologies**

Adopting an appropriate methodology is vital for producing valuable results in any research. Even though the ultimate aim of research is the same for all fields of study in science and humanities, every field involves a specific application of the methods which is useful for the research (Hassani, 2017). In general, the important steps of research methodology in the field of computer science and engineering includes the following steps as shown in Figure 3.1 (Prajapati, Dabhi, & Bhensdadia, 2015):

- (a) Research Problem Formulation,
- (b) Literature Review,
- (c) Research in Action, and
- (d) Research Communication.

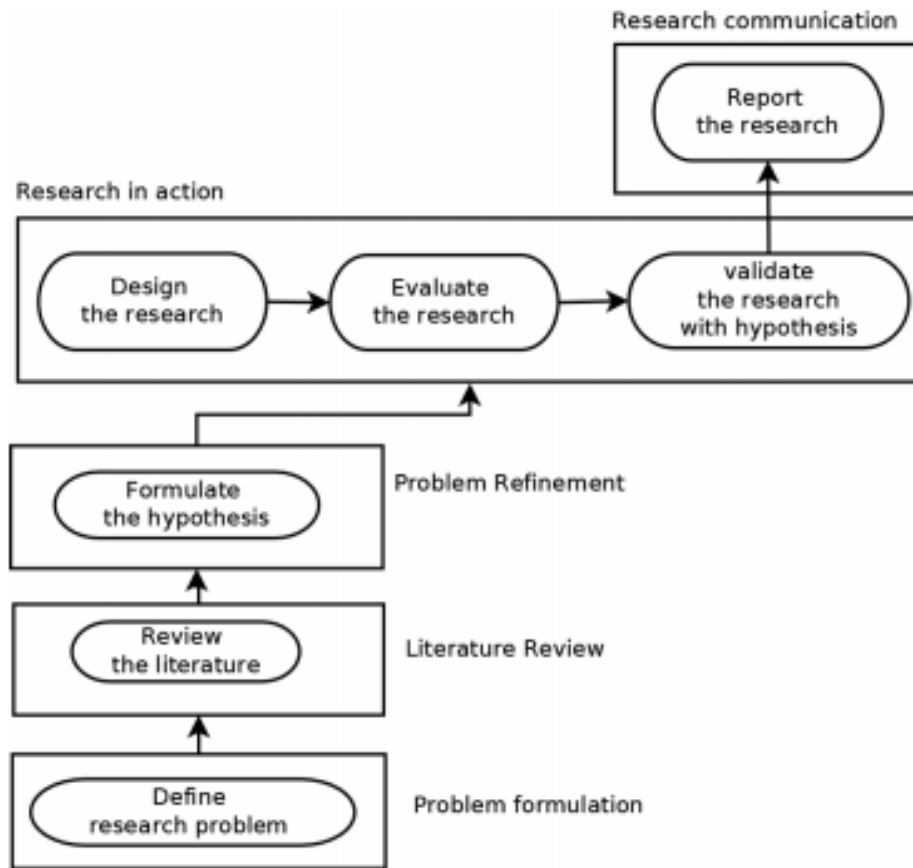


Figure 3.1: Generalized Process of Research (Prajapati et al., 2015)

Research Problem Formulation is the most important step in research because it identifies the problem to be resolved throughout the research. The problem that has been identified should be a valid one which has a great impact on the field that is being studied (Ellis & Levy, 2008). It can be a new problem that has not been published in the literature or an existing problem that has been indicated in the future work section of the relevant research papers. The step of Research Problem Formulation is shown in Figure 3.2 where a particular subject of interest is scoped down to a specific topic for a significant problem to be identified with the appropriate research questions. The research problem can be further refined after conducting the literature review in an exhaustive manner.

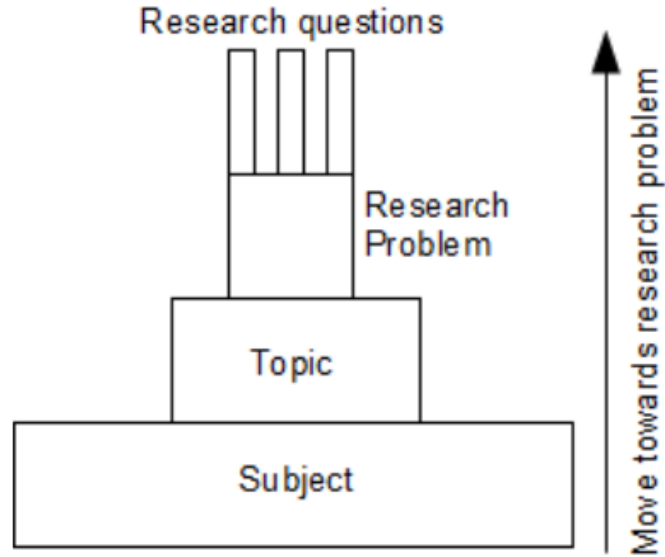


Figure 3.2: Research Problem Formulation (Prajapati et al., 2015)

The step of literature review conducts a thorough study and performs a critical analysis on the relevant literature of the research topic. This is because the outcome of a research can only be considered as valuable if it is a novel one. Therefore, all researchers are required to ensure that the researches performed by them are producing new contributions to the body of knowledge. In order to achieve this, the researchers have to conduct a thorough study on the existing knowledge at the time of research. The entire process of literature review is shown in Figure 3.3. After formulating the problem, the researcher should collect the relevant sources from the digital libraries or printed materials like journal papers, conference papers, books, technical reports, and websites for research. Then, the sources collected needs to be evaluated so that the materials for studying can be reduced and much attention can be given to those that are more important.

When the sources are sorted out according to their relevance to the research topic, a thorough study and critical analysis is to be conducted by the researcher on the selected ones. Then, as the selected literatures are interpreted and synthesized by the researcher, the appropriate results and conclusions can be drawn to highlight the major issues that have been addressed and approaches of solution that have been

implemented by other researchers. Finally, the literature review should be presented in a summarized manner to show the strengths and weaknesses of different related work in order to identify the existing gaps that need to be filled.

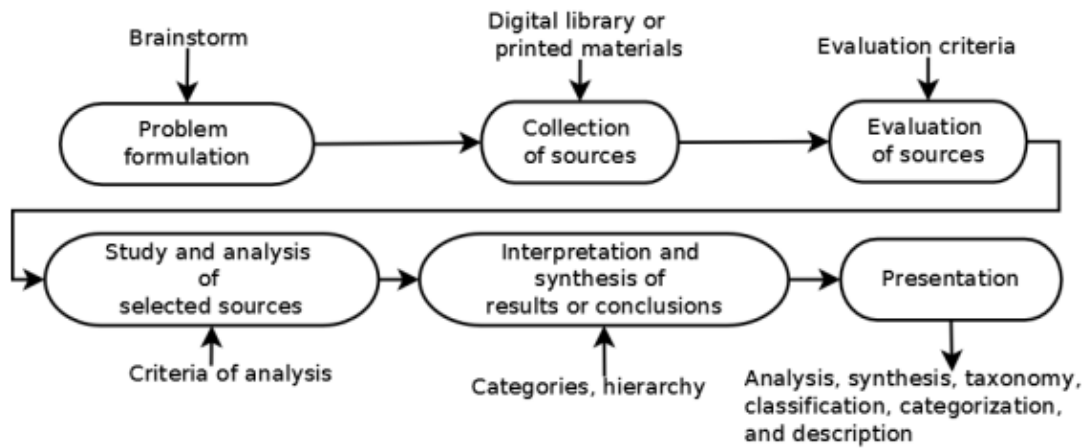


Figure 3.3: Process of Literature Review (Prajapati et al., 2015)

Research in Action is the step where the research is designed, evaluated and validated. As the research is designed, it comprises the steps of proposing a new solution and putting it into implementation. After the proposed solution is accomplished, it needs to be evaluated by comparing it with other recent works using the appropriate metrics or measures. At this stage, the researcher should identify the situations in which the proposed solution will produce the best or negative results, and try to analyze the reasons for such scenarios to happen. Finally, the researcher needs to validate the research by providing enough proofs to persuade other researchers that the research conducted is valid. The validation can be done in an experimental manner where the results of the research are compared with the best results that are available in the existing literature, or in a theoretical manner where mathematical and analytical evidences are provided for justification.

Last but not least, the final step of the general process of research is Research Communication. It is a stage where the researcher should write some research papers

about the work that has been done and submit it to some conferences or journals for publications if the research is being accepted. Publishing the work of research to a conference or journal with a good reputation is also very important because it helps to establish the validation of the research.

Computer Science is an area of research that has always been struggling with its identity. This is because its foundation is drawn from various disciplines and it needs to implement the concepts from many different areas of studies (Crnkovic, 2002). On one hand, it is a field that is intensely rooted in Mathematics with complicated theories, but on the other hand, it is a field that is intensely rooted in Engineering with the approaches of quantification, measurement and comparison (Demeyer, 2011). Therefore, in order to determine a more suitable research method to be used, various kinds of research methods are reviewed. The different types of research methodologies available are Field Study, Group Feedback Analysis, Opinion Research, Participative Research, Case Study, Archival Research, Philosophical Research, Math Modeling, Experimental Simulation, Laboratory Experiment, Free Simulation, Field Experiment, and Adaptive Experiment (Jenkins, 1985). Among all these research methodologies, the one being chosen to conduct this research is the Experimental Research Method.

### **3.2 Experimental Research Method**

Experimental Computer Science is a kind of study that implements the best practices, methods, procedures, and techniques which help practitioners of computing to move from the theoretical base towards an applied one in the field of Computer Science (Hassani, 2017). Typically, it can be grouped into five categories, namely, Feasibility Experiment, Trial Experiment, Field Experiment, Comparison Experiment, and Controlled Experiment (Tedre & Moisseinen, 2014).

Feasibility Experiment is conducted when it is necessary to know how efficient, reliable and feasible a research has been performed. Usually, a demonstration of the technology being proposed is conducted to show that it can be implemented successfully. For the Trial Experiment, it verifies many areas of a proposed

technology with different sets of variables in order to determine its qualities. The experiments are conducted in a laboratory most of the time. But it can also be performed in the real environment with some limitations. On the other hand, Field Experiment conducts the tests for a technology out of the laboratory where it is evaluated in a real environment to verify its robustness, usability or performance. Comparison Experiment is performed to compare multiple solutions that are available to implement a certain technology in order to identify the best solution for solving a particular problem. Last but not least, the Controlled Experiment is a test that is conducted under controlled conditions, in which one or more factors are modified at a time, while all other factors remain the same.

In order to make the research in Computer Science to be more applicable to the society, the Experimental research method is used to implement a number of experiments which analyze and generate results from multiple real world data sets. Furthermore, it is necessary to ensure that all the experiments conducted and results presented should be reproducible if every step of the research is executed again accordingly (Ayash, 2014).

### **3.3 Research, Development and Evaluation**

The entire work flow of this research is presented in Figure 3.4. It includes the five processes of literature review, data collection and analysis, design of the FPM algorithm, implementation of the FPM algorithm, and evaluation of the FPM algorithm. As shown in the diagram, the three objectives of this research are achieved through the design, development and evaluation phases of the entire FPM algorithm.

Just like any other research, this research is started with the literature review process. In the literature review process, the significant and recent algorithms for Frequent Pattern Mining (FPM) are studied so that a more robust FPM algorithm can be designed and developed. As the algorithms are being reviewed, the advantages and disadvantages of each algorithm are analyzed in detail. The purpose of conducting such a detailed analysis is to identify the strengths and weaknesses of every algorithm.



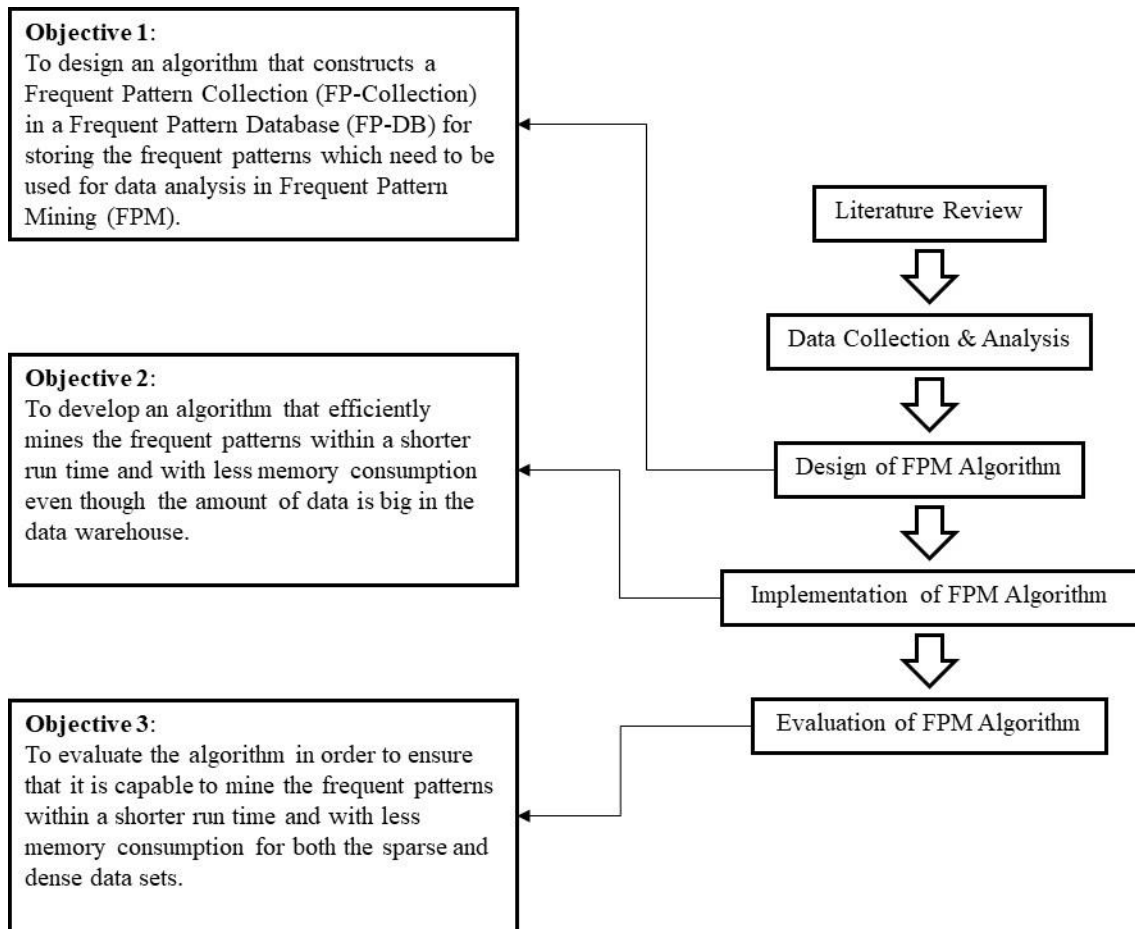


Figure 3.4: Research Work Flow

After discovering the strengths and limitations of all the significant and recent FPM algorithms, a few sets of data are downloaded online from the Frequent Itemset Mining Implementations (FIMI) Dataset Repository for testing the proposed FPM algorithm (Goethals, 2004). These data sets contain data from a few different sources as follows:

- (a) Sales Data of Products from a Retail Store
- (b) Network Connection Data from a Computer Server
- (c) Census Data from the United States of America
- (d) Click-Stream Data from a Hungarian Online News Portal

All the data are filtered accordingly before being processed by every algorithm that is designed and developed in this research.

The first objective of this research is achieved at the design stage. When the data is ready to be processed, the whole architecture of the FPM algorithm is designed from the beginning to the end. The appropriate diagramming tools available in the Microsoft Office 365 software (Microsoft, 2018c) are used to construct the flow chart for the algorithm as shown in Figure 4.1. The flow chart describes how the data is processed by all the algorithms throughout the entire architecture. In the flow chart, the input, process, storage, and output of the algorithm are clearly identified in the appropriate order.

The second objective of this research is achieved at the development stage. Once the architecture of the algorithm is designed completely, all the algorithms are implemented one after another using the C++ programming language (CPlusPlus.com, 2018). C++ is utilized as the programming language for development because it is a general-purpose programming language that has the imperative and object-oriented functionalities, while providing the features for low-level memory manipulation. Apart from this, the C++ programming language ensures that variable declaration is compulsory to be implemented in any program to prevent any variable from being used with the inappropriate data type. This is an important feature because using the inappropriate data type for any variable in the program, may cause invalid data to be captured for data mining, and having inaccurate results to be produced. To make it more convenient for development, Microsoft Visual Studio (Microsoft, 2018b) is used as the Integrated Development Environment (IDE) for constructing the algorithms as shown in Figure 3.5.

Since this research focus on constructing an algorithm for Frequent Pattern Mining (FPM), the algorithm is developed as a Windows Console Application (Technopedia, 2018) to be executed under the Windows Command Prompt. This is because a console application is suitable to be used as a proof-of-concept demonstration of the functionalities that are potential to be implemented into a desktop application in any platform of operating system (Microsoft, 2018a).

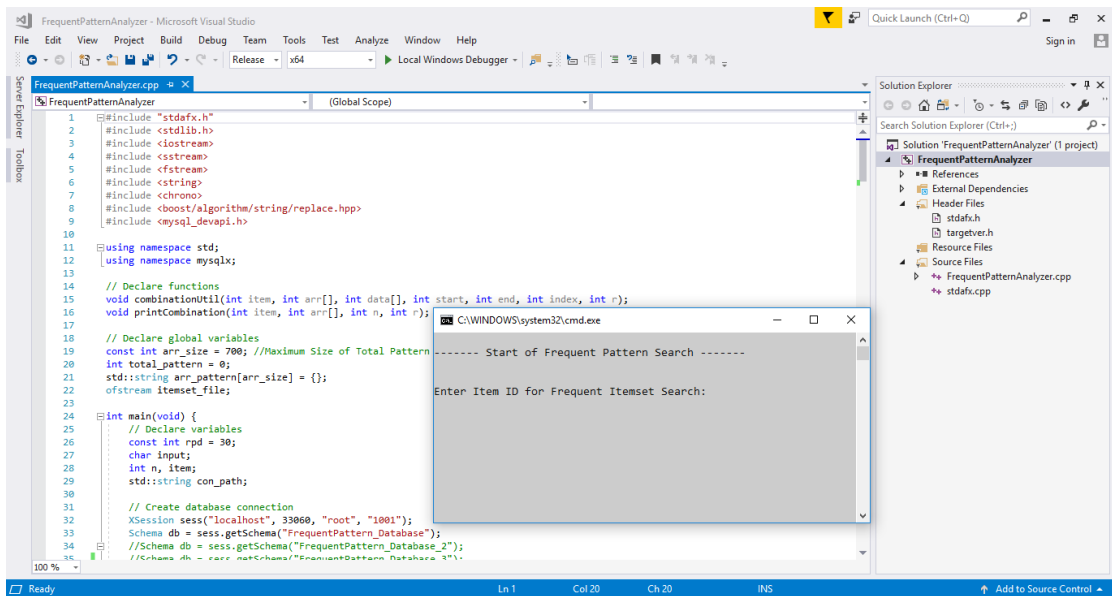


Figure 3.5: Integrated Development Environment of C++ in Microsoft Visual Studio

The database used to store the data sets is the MySQL Database (MySQL, 2018a). Both the Structured Query Language (SQL) (Heller, 2017) and Not-Only Structured Query Language (No-SQL) (Yegulalp, 2017) databases are utilized to support the process of Frequent Itemset Mining (FIM) in this research. For data manipulation, MySQL Shell (MySQL, 2018b) is the Command Line Interface (CLI) software used to execute the SQL or NoSQL queries as shown in Figure 3.6. To make it more convenient in manipulating the data, MySQL Workbench (MySQL, 2018c) is used as the Graphical User Interface (GUI) software to access the databases as shown in Figure 3.7.

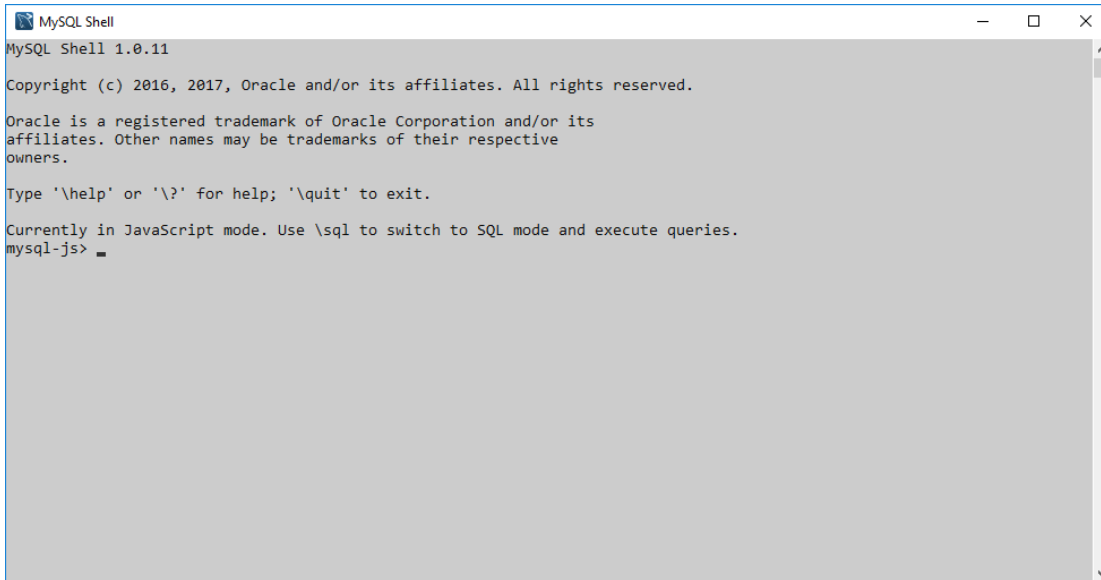


Figure 3.6: MySQL Shell – The CLI Software for MySQL Database

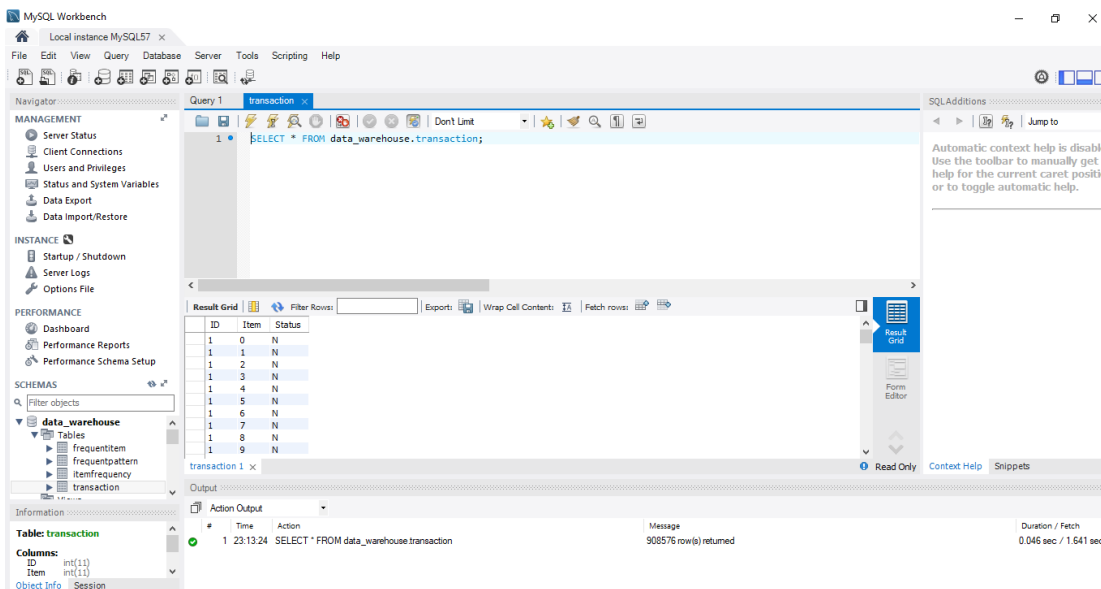


Figure 3.7: MySQL Workbench – The GUI Software for MySQL Database

The third objective of this research is achieved at the evaluation stage. As all the algorithms are completed successfully, each algorithm is executed on one of the data sets in order to test its performance in mining the frequent patterns from the data. It is done by measuring the run-time execution and memory consumption of the algorithm.

The purpose of conducting this evaluation is to verify that all the algorithms being constructed are suitable to be used for FPM. In addition, to confirm that the algorithms are really fit to be used for FPM, the algorithms are executed on multiple data sets with a dense or sparse structure. Moreover, the proposed FPM algorithm is also compared with the Apriori (Agrawal & Srikant, 1994) and EFP (Shang, 2005) algorithms in order to ensure that it is capable to mine the frequent itemsets in a shorter run time and with less memory consumption. The algorithm is compared against Apriori and EFP because Apriori is the fundamental algorithm in FPM and EFP utilized a similar approach of FPM using the database. Finally, if any part of the algorithm is found to be not working properly, the design and implementation processes are repeated until a robust FPM algorithm is constructed.

### **3.4 Chapter Summary**

Among the various types of research methodologies, the Experimental Research Method has been chosen to implement the research of this study. First, the step of literature review is conducted to study the significant and recent algorithms for Frequent Pattern Mining (FPM). Then, a few sets of data are downloaded online to be used for testing the proposed FPM algorithm. Next, the architecture of the algorithm is designed using the appropriate diagramming tools available in the Microsoft Office 365 software. After that, the algorithm is implemented with the C++ programming language through the Integrated Development Environment (IDE) of Microsoft Visual Studio. Then, the MySQL Database is utilized to store the data to be used for Frequent Itemset Mining (FIM). Last but not least, the algorithm is evaluated on different kinds of sparse and dense data sets.



## CHAPTER 4

### FP-NOSQL: NOSQL-BASED FREQUENT PATTERN MINING WITH FP-DB APPROACH

This chapter discusses about the proposed algorithm in this research to solve the problem of Frequent Pattern Mining (FPM) so that the hidden patterns of the frequent itemsets can be mined within a shorter run time and with less memory consumption. First, the entire architecture of the algorithm is illustrated in the form of a flowchart in Section 4.1. Then, the different parts of the algorithm are described in detail from Section 4.2 to Section 4.6 respectively.

#### 4.1 Flow Chart of FP-NoSQL Algorithm

As discussed in Chapter 2, many algorithms have been proposed by different researchers throughout the world to improve the method of Frequent Pattern Mining (FPM). But most of the algorithms for FPM are designed to mine all the frequent itemsets from a text file into the main memory or Random Access Memory (RAM). Therefore, the data needs to be mined again if the system is down or there is a power failure. This is because none of the frequent patterns are stored in such a manner so that it can be retrieved later for further data analysis. To solve this problem, FP-NoSQL is proposed in this research as an algorithm that mines the frequent itemsets using the Frequent Pattern Database (FP-DB) approach. The flow chart of the FP-NoSQL algorithm is shown in Figure 4.1 and it is constructed from several algorithms as follows:

- (a) Data Loader
- (b) Frequent Item Generator
- (c) Frequent Pattern Processor
- (d) FP-Collection Constructor
- (e) Frequent Pattern Analyzer

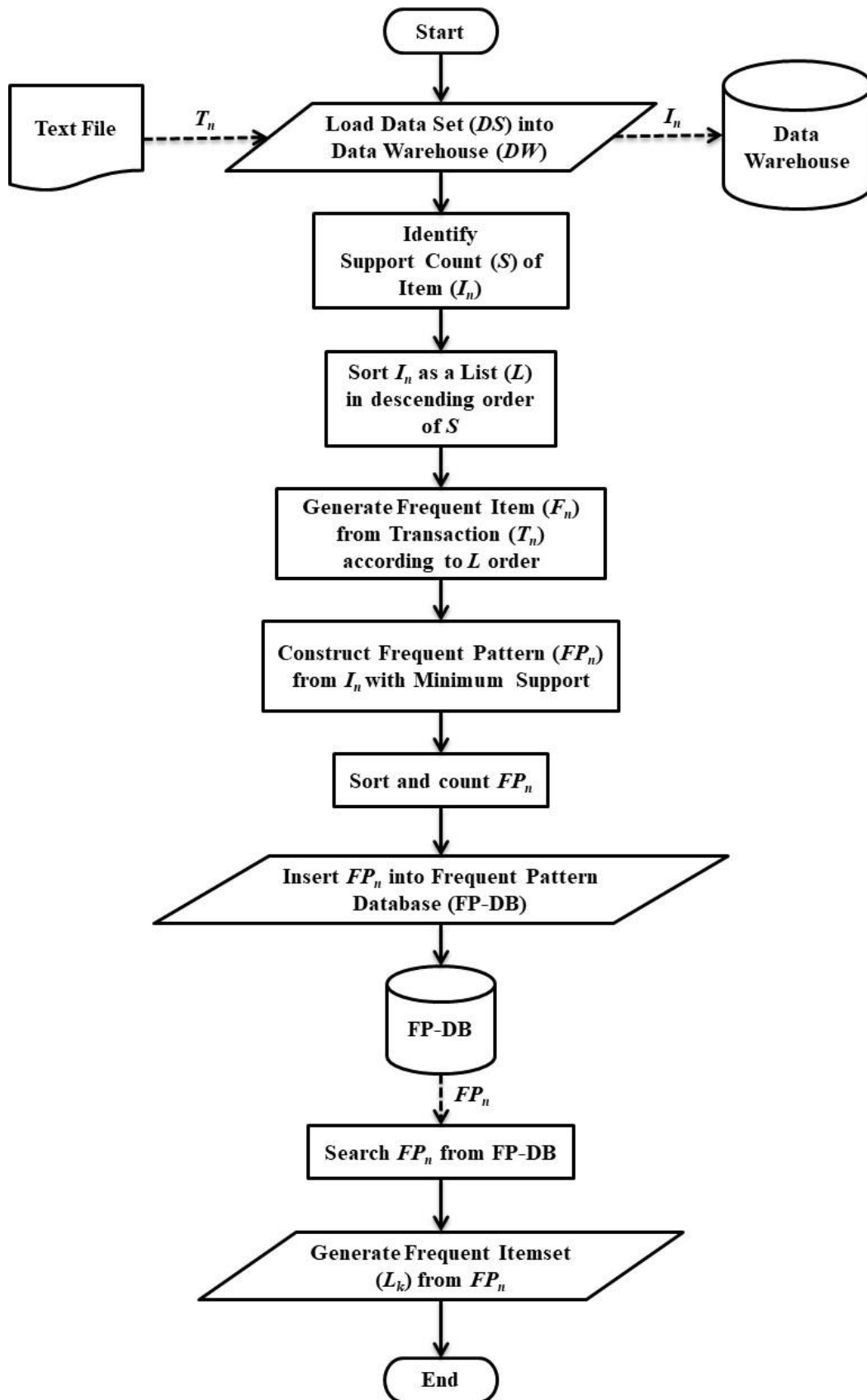


Figure 4.1: Flow Chart of the FP-NoSQL Algorithm



## 4.2 Algorithm of Data Loader

Most of the existing FPM algorithms mine the frequent patterns of data from a text file rather than a database or data warehouse. However, almost every system of an organization stores their data in a database or data warehouse for further analysis (Gull & Pervaiz, 2018). Therefore, it is necessary to construct an FPM algorithm that is able to mine the frequent patterns of data directly from the database or data warehouse, so that the data does not need to be exported to text files for mining by the algorithm. In order to construct such an FPM algorithm, the Data Loader algorithm is designed to load the data from a text file into a data warehouse since most of the data available for mining are stored in text files. The pseudocode of the Data Loader algorithm is presented as Algorithm 1.

---

### Algorithm 1: Data Loader

**Function:** Construction of *Transaction* Table

**Input:** Transactional Data in Text File

**Output:** Transactional Data in Data Warehouse

---

```
1   Begin
2   Connect to Data Warehouse
3   Initialize TID to the first transaction to be processed
4   Open data file
5   While NOT End of File Do
6       Read line from file
7       Split line into items
8       While NOT End of Line Do
9           records += "(" + TID + "," + item + "),"
10      End While
11      TID++
12  End While
13  records = records.substr(0, records.length() - 1)
14  Insert records into Transaction table
15  Close data file
16  Disconnect from Data Warehouse
17  End
```

---

The Data Loader algorithm will construct a Transaction table when the data is loaded from the text file into the data warehouse. Figure 4.2 shows a portion of the transactional data from a retail store in the form of a text file. Each line represents a transaction of purchase from a customer in the actual database of the retail store. In every line, the numbers are separated with a space and they represent the item codes of all products purchased by the customer in that transaction.

```

Retail.dat
1 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29
2 30 31 32
3 33 34 35
4 36 37 38 39 40 41 42 43 44 45 46
5 38 39 47 48
6 38 39 48 49 50 51 52 53 54 55 56 57 58
7 32 41 59 60 61 62
8 3 39 48
9 63 64 65 66 67 68
10 32 69
11 48 70 71 72
12 39 73 74 75 76 77 78 79
13 36 38 39 41 48 79 80 81
14 82 83 84
15 41 85 86 87 88
16 39 48 89 90 91 92 93 94 95 96 97 98 99 100 101
17 36 38 39 48 89
18 39 41 102 103 104 105 106 107 108
19 38 39 41 109 110
20 39 111 112 113 114 115 116 117 118
21 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133
22 48 134 135 136
23 39 48 137 138 139 140 141 142 143 144 145 146 147 148 149
24 39 150 151 152
25 38 39 56 153 154 155
26 48 156 157 158 159 160
27 39 41 48
28 161 162 163 164 165 166 167
29 38 39 48 168 169 170 171 172 173
30 32 39 41 48 174 175 176 177 178

```

Figure 4.2: Transactional Data from Retail Data Set (Brijs, 1999)

When the algorithm is executed, it will connect to the data warehouse where transactional data is stored permanently for analysis. Then, the Transaction ID (TID) is set to the value of '1' if the data is loaded for the first time from the text file into the data warehouse, else it can be set to the next value of the current maximum TID. For example, if the last TID in the data warehouse is '1000', then the current TID can be set to '1001' to start the processes of Extract, Transform and Load (ETL) for the data.

The ETL processes establish the integration layer that consolidates data from various sources into the appropriate targets through a set of processing steps (Kabiri & Chiadmi, 2013). After initializing the value of TID, the relevant text file is accessed in order to locate the data as input to the data warehouse.

As the algorithm accesses the text file, each line of transaction is read and split into individual items. Next, all the items are concatenated into the *records* string with the intention of inserting them together into the Transaction table in the data warehouse. Then, the value of TID is incremented by 1 in order to proceed for processing the next line of transaction. These three steps are repeated until the last line of transaction in the text file is being processed. Before inserting the records, the last comma (,) of the *records* string is removed, so that the query of the Structured Query Language (SQL) can be executed successfully. Finally, the text file is closed and the connection to the data warehouse is disconnected.

Table 4.1 shows a portion of the Transaction table created for the Retail data set. The SQL query used to retrieve the records is as follows:

```
SELECT * FROM data_warehouse.transaction;
```

The table contains of three columns having the name ID, Item and Status. The ID column stores the TIDs while the Item column stores the item codes of the products. For the Status column, it will hold the value of 'N' if the data is newly inserted into the table, and it will hold the value of 'O' if the data has been processed by the Frequent Item Generator algorithm described in Section 4.3. Hence, the Status column is used to keep track whether the data has been processed by the Frequent Item Generator algorithm or not. In the next execution, only the items that contain the value of 'N' for the Status column will be processed by the algorithm.

Table 4.1: Transaction Table for Retail Data Set

<b>ID</b>	<b>Item</b>	<b>Status</b>
1	0	N
1	1	N
1	2	N
1	3	N
1	4	N
1	5	N
1	6	N
1	7	N
1	8	N
1	9	N
1	10	N
1	11	N
1	12	N
1	13	N
1	14	N
1	15	N
1	16	N
1	17	N
1	18	N
1	19	N
1	20	N
1	21	N
1	22	N
1	23	N
1	24	N
1	25	N
1	26	N
1	27	N
1	28	N
1	29	N
2	30	N
2	31	N
2	32	N
3	33	N
3	34	N
3	35	N

### 4.3 Algorithm of Frequent Item Generator

Analyzing the frequent patterns of data is important because organizations need to know what their customers would like to purchase by identifying which products are relevant to one another (Tripathi, Vartak, Chaudhari, & Naik, 2018). To analyze the patterns that exist frequently in a data warehouse, the frequency of occurrence for each item at every transaction in the data warehouse needs to be determined. Therefore, the Frequent Item Generator algorithm is designed to produce the list of frequent items by arranging the items in every transaction according to the sequence of the Item Frequency table. The Item Frequency table contains the frequency of occurrence for each item that exists in every transaction throughout the entire data warehouse. The pseudocode of the Frequent Item Generator algorithm is presented as Algorithm 2 and an example of the Item Frequency table is given in Table 4.2.

---

**Algorithm 2:** Frequent Item Generator

**Function:** Construction of *Frequent Item* Table

**Input:** Transactions

**Output:** Frequent Items

---

```
1   Begin
2   Connect to Data Warehouse
3   Select maximum TID of old transactions into max_old_id
4   Select maximum TID of new transactions into max_new_id
5   start_id = max_old_id + 1
6   Insert items into FrequentItem table based on descending order
      of ItemFrequency table
7   Update transactions status between start_id and max_new_id
8   Disconnect from Data Warehouse
9   End
```

---

Before constructing the Frequent Item table, the Item Frequency table is required to be built with the following SQL query:

```
INSERT INTO ItemFrequency  
SELECT item, COUNT(*) FROM Transaction  
GROUP BY item ORDER BY COUNT(*) DESC;
```

This SQL query calculates the frequency of every item in all the transactions by grouping the records according to their item codes and arranging them in the descending order of their frequencies. After calculating the frequencies of all the items, the SQL query will store them into the Item Frequency table.

Table 4.2 shows a portion of the Item Frequency table generated for the Retail data set, having Item 39 to be the one with the highest frequency among all the transactions. The SQL query used to retrieve the records is as follows:

```
SELECT * FROM data_warehouse.itemfrequency;
```

The Total column indicates the frequency for each item in the entire data warehouse. In this data set, 16470 unique items are found in it. The top three items that are purchased the most by the customers are Item 39, Item 48 and Item 38, with the frequencies of 50675, 42135 and 15596 respectively. The last few items in the table are those items that exist the least in the entire data set, having each item to appear only 1 time. The Item Frequency table can be updated from time to time in order to ensure that the patterns of new data that have been added into the data warehouse at a later time will also be mined by the algorithm.

Table 4.2: Item Frequency Table for Retail Data Set

Item	Total
39	50675
48	42135
38	15596
32	15167
41	14945
65	4472
89	3837
225	3257
170	3099
237	3032
...	...
12774	1
8285	1
12790	1
12806	1
6771	1
11160	1
12838	1
6819	1
6835	1
8413	1

The Frequent Item Generator algorithm will construct a Frequent Item table when the data in the Transaction table is processed to generate the list of frequent items. Once the algorithm is executed, it will connect to the data warehouse and select the maximum TIDs of transactions that have the ‘O’ status and ‘N’ status respectively. The records with an ‘O’ status are the data that has been processed by the algorithm previously. Thus, the initial TID for producing the list of frequent items will be equivalent to adding a value of ‘1’ to the maximum TID of transactions that have the ‘O’ status in the Transaction table. After setting the initial TID and maximum TID of transactions that have the ‘N’ status, the algorithm will insert all the items into the Frequent Item table according to the descending order of frequencies in the Item Frequency table. Last but not least, the status of transactions that have been processed currently are changed from ‘N’ to ‘O’ before the connection to the data warehouse is

disconnected. This is to prevent the transactions that have been processed from being processed again in the next data pre-processing stage.

Table 4.3 shows a portion of the Frequent Item table created for the Retail data set. The SQL query used to retrieve the records is as follows:

```
SELECT * FROM data_warehouse.frequentitem;
```

The records are arranged according to the sequence of the Item Frequency table. In this case, all the items in every transaction are organized by having the item with the highest frequency to be ranked first followed by others that have a lower frequency. For example, in Transaction 2, Item 32 is having a higher frequency compared to Item 31 and Item 30. The purpose for arranging the items in the descending order of their frequencies is to enable the items that exist frequently to be mined with a higher priority during the process of Frequent Itemset Mining (FIM).

Table 4.3: Frequent Item Table for Retail Data Set

<b>TID</b>	<b>Item</b>	<b>Status</b>
1	9	N
1	19	N
1	18	N
1	23	N
1	10	N
1	11	N
1	2	N
1	12	N
1	15	N
1	5	N
1	1	N
1	22	N
1	0	N
1	27	N
1	6	N
1	26	N
1	16	N
1	7	N
1	24	N



Table 4.3 continued: Frequent Item Table for Retail Data Set

<b>TID</b>	<b>Item</b>	<b>Status</b>
1	25	N
1	17	N
1	4	N
1	8	N
1	14	N
1	13	N
1	3	N
1	29	N
1	28	N
1	20	N
1	21	N
2	32	N
2	31	N
2	30	N
3	35	N
3	33	N
3	34	N

#### **4.4 Algorithm of Frequent Pattern Processor**

In a data warehouse, many patterns can exist in any data set because of the improvement of technology for Internet of Things (IoT) that enables data to be generated and collected easily (Chen et al., 2015). The pool of data can be retrieved from different sources like websites, mobile applications, machine logs and sensor data (Radhika, Kumar, Sailaja, & Gayatri, 2017). However, the huge amount of data that has been captured is difficult to be processed and analyzed in reasonable amount of time (Sharma, Sawai, & Surve, 2017). Furthermore, the patterns that exist may be duplicated throughout the entire data set. Therefore, the Frequent Pattern Processor algorithm is designed to discover all the patterns of data that exist in the data warehouse by processing the Frequent Item table and consolidate them into a Frequent Pattern table. The pseudocode of the Frequent Pattern Processor algorithm is presented as Algorithm 3.

---

**Algorithm 3:** Frequent Pattern Processor**Function:** Construction of *Frequent Pattern* Table**Input:** Frequent Items**Output:** Frequent Pattern Table

---

```
1   Begin
2   Connect to Data Warehouse
3   Select TID and Item from FrequentItem and ItemFrequency
      tables that satisfied threshold of minimum support
4   While NOT End of Resultset Do
5       If cur_id != pre_id Then
6           con_path = "root"
7       Else
8           con_path += "~" + pre_item
9       End If
10      records += "(" + cur_item + "," + con_path + "),"
11      pre_id = cur_id
12      pre_item = cur_item
13  End While
14  records = records.substr(0, records.length() - 1)
15  Insert records into FrequentPattern table
16  Disconnect from Data Warehouse
17  End
```

---

The Frequent Pattern Processor algorithm will construct a Frequent Pattern table when the data in the Frequent Item table is processed to generate the list of frequent patterns. When the algorithm is executed, it will connect to the data warehouse to select the TIDs and items from the Frequent Item and Item Frequency tables. The TIDs and items to be selected are those that satisfied the threshold of minimum support. Minimum support is a value fixed by the user as the minimum frequency which needs to be fulfilled for every item that is to be included into the Frequent Pattern Mining (FPM) (Chaure & Singh, 2016).

After selecting the relevant TIDs and items, the TIDs and items are processed in order to construct all the patterns that exist frequently in the data warehouse. If the

current TID (*cur\_id*) is not equivalent to the previous TID (*pre\_id*), then the connection path (*con\_path*) of an item will be set as “root”, else the previous item (*pre\_item*) is concatenated to the connection path of the current item (*cur\_item*). Then, the current item and its connection path are concatenated to the *records* string for inserting into the Frequent Pattern table later. Next, the values of *cur\_id* and *cur\_item* are stored into *pre\_id* and *pre\_item* so that they can be used in the following loop for comparison purpose. This process is repeated until all the frequent patterns are constructed from the Frequent Item table. Before inserting the records into the Frequent Pattern table, the last comma (,) of the *records* string is removed so that the SQL query can be executed successfully. Finally, the connection to the data warehouse is disconnected.

Table 4.4 shows a portion of the Frequent Pattern table created for the Retail data set. The SQL query used to retrieve the records is as follows:

```
SELECT * FROM data_warehouse.frequentpattern;
```

All the items in every transaction that satisfied the threshold of minimum support fixed by the user are stored as the frequent patterns in this table. The minimum support value set for this case is 0.01% of the entire data set which is equivalent to about 90 times of occurrence. A total of 661856 frequent patterns are found from 908576 items in 88162 transactions.

Table 4.4: Frequent Pattern Table for Retail Data Set

Item	ConnectionPath
9	root
19	root~9
18	root~9~19
23	root~9~19~18
10	root~9~19~18~23
11	root~9~19~18~23~10
2	root~9~19~18~23~10~11
12	root~9~19~18~23~10~11~2
15	root~9~19~18~23~10~11~2~12
5	root~9~19~18~23~10~11~2~12~15

Table 4.4 continued: Frequent Pattern Table for Retail Data Set

1	root~9~19~18~23~10~11~2~12~15~5
22	root~9~19~18~23~10~11~2~12~15~5~1
32	root
31	root~32
30	root~32~31
39	root
38	root~39
41	root~39~38
36	root~39~38~41
37	root~39~38~41~36
45	root~39~38~41~36~37
43	root~39~38~41~36~37~45
40	root~39~38~41~36~37~45~43
44	root~39~38~41~36~37~45~43~40

#### 4.5 Algorithm of FP-Collection Constructor

Since many patterns may be duplicated in a data set within a data warehouse, it is necessary to construct a collection of patterns and frequencies of occurrence for the data so that a thorough analysis can be performed easily. The FP-Collection Constructor algorithm is designed to construct a Not-Only Structured Query Language (NoSQL) collection that consists of all the patterns and frequencies of data by processing the Frequent Pattern table. The pseudocode of the FP-Collection Constructor algorithm is presented as Algorithm 4. The NoSQL collection is utilized to store all the unique patterns and frequencies of data because the schema less data model is a better solution for managing the huge volume of data being captured and processed at every moment in the organizations (Bhogal & Choksi, 2015).

The FP-Collection Constructor algorithm will construct a Frequent Pattern Collection (FP-Collection) in the NoSQL database when the data in the Frequent Pattern table is processed to calculate the frequencies of the patterns. Once the algorithm is executed, it will connect to the data warehouse and the Frequent Pattern Database (FP-DB) in which the FP-Collection will be created.

---

**Algorithm 4: FP-Collection Constructor****Function:** Construction of *Frequent Pattern Collection***Input:** Frequent Pattern Table**Output:** Frequent Pattern Collection

---

```
1   Begin
2   Connect to Data Warehouse
3   Connect to Frequent Pattern Database
4   pat_count = 0
5   Select pattern from FrequentPattern table
6   Sort pattern in ascending order
7   While NOT End of Resultset Do
8       pat_count++
9       If cur_pat != next_pat Then
10          documents += "({'\"Item\":" + item
11              + "\",\"Frequency\":" + pat_count
12              + "\",\"Connection\":" + con_path + "\"})\","
13          pat_count = 0
14      End If
15  End While
16  documents = documents.substr(0, documents.length() - 1)
17  Insert documents into FP-Collection
18  Disconnect from Data Warehouse
19  Disconnect from Frequent Pattern Database
20  End
```

---

After connecting to the data warehouse and FP-DB, all the frequent patterns will be selected from the Frequent Pattern table into a list and sorted in the ascending order. Then, the unique frequent patterns will be counted from the list by comparing whether the current frequent pattern (*cur\_pat*) is the same as the next frequent pattern (*next\_pat*). As the algorithm loops through the list, a counter (*pat\_count*) is used to count the unique frequent patterns. If *cur\_pat* is not equivalent to *next\_pat*, then the value of *pat\_count* is considered as the frequency of the frequent pattern. Next, the current item (*item*) is concatenated with its connection path (*con\_path*) and frequency (*pat\_count*) to the *documents* string for inserting into the FP-Collection later.

Subsequently, the value of *pat\_count* is reset to zero for counting the following unique frequent pattern in the list. This process is repeated until all the unique frequent patterns are counted respectively from the list. Before inserting the documents into the FP-Collection, the last comma (,) of the *documents* string is removed so that the SQL query can be executed successfully. Finally, the connections to the data warehouse and FP-DB are disconnected.

Table 4.5 shows a portion of the FP-Collection created for the Retail data set. The SQL query used to retrieve the records is as follows:

```
SELECT * FROM frequentpattern_database.fp_collection;
```

Each document in the collection represents a unique frequent pattern. The document with a *Frequency* value that is more than one represents a repeated pattern in the data set. How every *Item* is connected to the root of the entire data set is represented by the *Connection* string of alphanumeric characters.

Table 4.5: Frequent Pattern Collection for Retail Data Set

<b>Document</b>
{"Item": 1, "Frequency": 4, "Connection": "root"}
{"Item": 1, "Frequency": 1, "Connection": "root~101~179~23~622"}
{"Item": 1, "Frequency": 1, "Connection": "root~101~338~4336~643~652"}
{"Item": 1, "Frequency": 1, "Connection": "root~1020~3510"}
{"Item": 1, "Frequency": 1, "Connection": "root~1113~2673~1808"}
{"Item": 1, "Frequency": 1, "Connection": "root~1291"}
{"Item": 1, "Frequency": 1, "Connection": "root~161"}
{"Item": 1, "Frequency": 1, "Connection": "root~1714~1704~1214"}
{"Item": 1, "Frequency": 1, "Connection": "root~1714~2080~10444~12981"}
{"Item": 1, "Frequency": 1, "Connection": "root~185~365~423"}
{"Item": 1, "Frequency": 1, "Connection": "root~201~258~910~1444"}
{"Item": 1, "Frequency": 1, "Connection": "root~209"}
{"Item": 1, "Frequency": 1, "Connection": "root~2238~4994"}
{"Item": 1, "Frequency": 1, "Connection": "root~2353"}
{"Item": 1, "Frequency": 2, "Connection": "root~237"}
{"Item": 1, "Frequency": 1, "Connection": "root~237~249~10515~405~10~1144~12~168~4685"}
{"Item": 1, "Frequency": 1, "Connection": "root~237~249~4685"}

Table 4.5 continued: Frequent Pattern Collection for Retail Data Set

{"Item": 1, "Frequency": 1, "Connection": "root~237~31~30~856~1987~490~1842~805"}
{"Item": 1, "Frequency": 1, "Connection": "root~310~161~3616~441~1239~345~2325~2215~8867"}
{"Item": 1, "Frequency": 2, "Connection": "root~32"}
{"Item": 1, "Frequency": 1, "Connection": "root~3270~2051"}
{"Item": 1, "Frequency": 1, "Connection": "root~32~1393~201~1020~979~12943~1003~1282"}
{"Item": 1, "Frequency": 1, "Connection": "root~32~237~310~249~783~10515~1144~856~694~1046~2633~168~718~234 ~4685"}
{"Item": 1, "Frequency": 1, "Connection": "root~32~338"}
{"Item": 1, "Frequency": 1, "Connection": "root~32~41~1393~78~2958~53~1677~4698"}

#### 4.6 Algorithm of Frequent Pattern Analyzer

Once the Frequent Pattern Collection (FP-Collection) is constructed successfully in the Frequent Pattern Database (FP-DB), Frequent Itemset Mining (FIM) can be performed on the data using the appropriate Not-Only Structured Query Language (NoSQL) queries (Marinov, Georgiev, & Popova, 2018). The Frequent Pattern Analyzer algorithm is designed to analyze the patterns and frequencies of data by processing the FP-Collection to generate the frequent itemsets. The pseudocodes of the algorithm are presented from Section 4.6.1 to Section 4.6.4, and the main program of the Frequent Pattern Analyzer algorithm is presented as Algorithm 5. First, the algorithm will connect to the FP-DB and display a menu that consists of a few options for the user to select in order to perform Frequent Pattern Mining (FPM).

The options available to perform FPM are shown in Figure 4.3 as follows:

(1) Search\_All\_Items()

- To search for all the patterns of every item that exists in the database.

(2) Search\_Specific\_Item()

- To search for all the patterns of a specific item that exists in the database.

(3) Search\_Specific\_Pattern()

- To search for a specific pattern that may exist in the database.

#### (4) Mine\_Frequent\_Itemsets()

- To mine all the frequent itemsets of a specific item that exists in the database.

---

**Algorithm 5:** Frequent Pattern Analyzer

**Function:** Generation of *Frequent Itemsets*

**Input:** Frequent Pattern Collection

**Output:** Frequent Itemsets

---

```
1   Begin
2   Connect to Frequent Pattern Database
3   Do
4       Display menu
5       Get option
6       Case option Of
7           Case 1: Search_All_Items()
8           Case 2: Search_Specific_Item()
9           Case 3: Search_Specific_Pattern()
10          Case 4: Mine_Frequent_Itemsets()
11          Case 5: Exit
12          Default: Prompt user to enter a valid option
13      End Case
14  While (option != 5)
15  Disconnect from Frequent Pattern Database
16  End
```

---



```
C:\WINDOWS\system32\cmd.exe

----- FP-NoSQL : Searching for Frequent Patterns with NoSQL -----

Option 1 : Search for the patterns of all items
Option 2 : Search for the patterns of a specific item
Option 3 : Search for the items that contains a specific pattern
Option 4 : Mine the frequent itemsets of a specific item
Option 5 : Exit

-----

Please enter your option:
```

Figure 4.3: Main Menu of FP-NoSQL

When an invalid value is entered, which is not within ‘1’ to ‘5’, the user will be prompted to enter another valid value again. As a valid value is entered, the algorithm will execute the appropriate procedure and return to the main menu if the user decided to exit from the procedure. If the value of ‘5’ is entered by the user, the algorithm will be terminated and disconnected from the FP-DB.

#### 4.6.1 Procedure to Search All Items

When option ‘1’ is entered by the user, the algorithm will execute the procedure to retrieve all the patterns of every item that exist in the database. The pseudocode of this procedure is presented as Procedure 1. Initially, a few variables are set so that the display of patterns can be controlled accordingly. The *input* variable is set as ‘c’ to ensure that the *do ... while* loop can be repeated until the user entered the character ‘e’ to terminate. Then, the *rpd* variable stands for “row per display”. It limits the number of patterns to be displayed on screen at one time. This enables users to analyze a smaller amount of patterns, instead of being overloaded with too much information. For example, when the value ‘30’ is set to *rpd*, only 30 documents of patterns will be displayed on screen. After displaying 30 documents of patterns on screen, the algorithm will prompt the user to continue or exit. If the user enters the character ‘e’, the algorithm will terminate, else it will continue to display the next 30 documents of

patterns from the database. Last but not least, the  $n$  variable is used to set the starting position of the patterns to be displayed. For instance, when the value '0' is set to *rpd*, 30 documents of patterns will be displayed starting from the first one being retrieved into the *dres* document result set.

All the documents of patterns are sorted according to their frequencies of occurrence in the descending order. This is because the ultimate aim of Frequent Pattern Mining (FPM) is to identify the patterns that exist frequently in a data set and analyze how they are related to one another. As the patterns are sorted in the descending order of their frequencies, the ones that exist the most in the database will be displayed on screen first. When the patterns of data are able to be discovered according to their frequencies of appearance, many problems can be resolved by studying the trend of the patterns in the database.

---

**Procedure 1: Search\_All\_Items()**

---

```
1   Begin
2   input = 'c'
3   rpd = 30
4   n = 0
5   Do
6       dres = col.find().sort("Frequency DESC").limit(rpd)
           .offset(n).execute()
7   If (dres.count() > 0) Then
8       While (doc = dres.fetchOne())
9           Display doc
10      End While
11      n += rpd
12      Display "Press 'e' to exit or other keys to
           continue: "
13      Get input
14  Else
15      Display "No Records Found."
16  End If
17  While (input != 'e')
18  End
```

---

A screenshot of the procedure to search all items from the FP-DB is shown in Figure 4.4. In the Retail data set, item 39 is the item that exists the most in the entire data set with a frequency of '50675'. The same item may consist of more than one frequency value if it contains multiple patterns by relating to different number of other items. For example, in the screenshot at Figure 4.4, item 38 contains 4 different patterns as follows:

- {"Item": 38, "Frequency": 6102, "Connection": "root~39~48"}
- {"Item": 38, "Frequency": 4243, "Connection": "root~39"}
- {"Item": 38, "Frequency": 3409, "Connection": "root"}
- {"Item": 38, "Frequency": 1842, "Connection": "root~48"}

Among all the patterns of item 38, the first pattern (root~39~48) has the highest frequency with a value of '6102' and it is linked to item 39 and item 48. This pattern indicated that item 38 appears the most at the same time with item 39 and item 48. The second pattern (root~39) with a frequency of '4243' indicated that item 38 appears such number of time with item 39 only, whereas the third pattern (root) with a frequency of '3409' indicated that item 38 appears on its own without the influence of other items for such a number of occurrence.

```

C:\WINDOWS\system32\cmd.exe

----- FP-NoSQL : Searching for Frequent Patterns with NoSQL -----

Option 1 : Search for the patterns of all items
Option 2 : Search for the patterns of a specific item
Option 3 : Search for the items that contains a specific pattern
Option 4 : Mine the frequent itemsets of a specific item
Option 5 : Exit

-----

Please enter your option: 1
{"Item": 39, "Frequency": 50675, "Connection": "root"}
{"Item": 48, "Frequency": 29142, "Connection": "root~39"}
{"Item": 48, "Frequency": 12993, "Connection": "root"}
{"Item": 38, "Frequency": 6102, "Connection": "root~39~48"}
{"Item": 38, "Frequency": 4243, "Connection": "root~39"}
{"Item": 41, "Frequency": 4177, "Connection": "root~39~48"}
{"Item": 32, "Frequency": 4166, "Connection": "root~39~48"}
{"Item": 32, "Frequency": 3497, "Connection": "root"}
{"Item": 38, "Frequency": 3409, "Connection": "root"}
{"Item": 32, "Frequency": 2449, "Connection": "root~39"}
{"Item": 41, "Frequency": 2449, "Connection": "root~39"}
{"Item": 32, "Frequency": 2222, "Connection": "root~48"}
{"Item": 38, "Frequency": 1842, "Connection": "root~48"}
{"Item": 41, "Frequency": 1543, "Connection": "root~39~48~38"}
{"Item": 32, "Frequency": 1236, "Connection": "root~39~48~38"}
{"Item": 41, "Frequency": 1198, "Connection": "root~39~48~32"}
{"Item": 41, "Frequency": 1087, "Connection": "root"}
{"Item": 89, "Frequency": 956, "Connection": "root~39~48"}
{"Item": 41, "Frequency": 944, "Connection": "root~48"}
{"Item": 41, "Frequency": 886, "Connection": "root~39~38"}
{"Item": 65, "Frequency": 866, "Connection": "root~39~48"}
{"Item": 65, "Frequency": 711, "Connection": "root"}
{"Item": 32, "Frequency": 604, "Connection": "root~39~38"}
{"Item": 32, "Frequency": 583, "Connection": "root~38"}
{"Item": 65, "Frequency": 548, "Connection": "root~39"}
{"Item": 170, "Frequency": 544, "Connection": "root~39~48~38"}
{"Item": 41, "Frequency": 539, "Connection": "root~39~32"}
{"Item": 225, "Frequency": 501, "Connection": "root~39~48"}
{"Item": 170, "Frequency": 487, "Connection": "root~39~38"}
{"Item": 65, "Frequency": 463, "Connection": "root~48"}

Press 'e' to exit or other keys to continue:

```

Figure 4.4: Procedure to Search All Items

#### 4.6.2 Procedure to Search Specific Item

As option '2' is entered by the user, the algorithm will execute the procedure to retrieve all the patterns of a specific item that exist in the database. The pseudocode of this procedure is presented as Procedure 2. Similarly, a few variables are set so that

the display of patterns can be controlled accordingly. Before retrieving the patterns from the database, the user will be prompted to enter the item that needs to be analyzed. After capturing an Item ID from the user, the algorithm will retrieve all the patterns in which the item is linked with. If there are no patterns that relate to the specific item, the message of “No Records Found” will be displayed on screen. As the patterns related to an item are able to be identified easily, it enables the user to solve a problem that involves a particular item by locating all the other relevant items. A screenshot of the procedure to retrieve all the patterns of a specific item in the Retail data set from the FP-DB is shown in Figure 4.5. In this case, the patterns being retrieved are linked with item 30.

---

**Procedure 2: Search\_Specific\_Item()**

---

```
1   Begin
2   input = 'c'
3   rpd = 30
4   n = 0
5   Display "Enter Item ID for Specific Pattern Search: "
6   Get item
7   Do
8       dres = col.find("Item=:param1").sort("Frequency DESC")
           .limit(rpd).offset(n).bind("param1", item)
           .execute()
9       If (dres.count() > 0) Then
10          While (doc = dres.fetchOne())
11             Display doc
12          End While
13          n += rpd
14          Display "Press 'e' to exit or other keys to
                continue: "
15          Get input
16       Else
17          Display "No Records Found."
18       End If
19   While (input != 'e')
20   End
```

---

```

C:\WINDOWS\system32\cmd.exe
----- FP-NoSQL : Searching for Frequent Patterns with NoSQL -----

Option 1 : Search for the patterns of all items
Option 2 : Search for the patterns of a specific item
Option 3 : Search for the items that contains a specific pattern
Option 4 : Mine the frequent itemsets of a specific item
Option 5 : Exit

-----

Please enter your option: 2

Enter Item ID for Specific Pattern Search: 30
{"Item": 30, "Frequency": 13, "Connection": "root"}
{"Item": 30, "Frequency": 13, "Connection": "root~39~48"}
{"Item": 30, "Frequency": 12, "Connection": "root~39"}
{"Item": 30, "Frequency": 8, "Connection": "root~48"}
{"Item": 30, "Frequency": 7, "Connection": "root~39~48~41"}
{"Item": 30, "Frequency": 6, "Connection": "root~31"}
{"Item": 30, "Frequency": 4, "Connection": "root~14098"}
{"Item": 30, "Frequency": 3, "Connection": "root~32"}
{"Item": 30, "Frequency": 3, "Connection": "root~39~41"}
{"Item": 30, "Frequency": 3, "Connection": "root~39~48~31"}
{"Item": 30, "Frequency": 2, "Connection": "root~161"}
{"Item": 30, "Frequency": 2, "Connection": "root~32~31"}
{"Item": 30, "Frequency": 2, "Connection": "root~39~14098"}
{"Item": 30, "Frequency": 2, "Connection": "root~39~32"}
{"Item": 30, "Frequency": 2, "Connection": "root~39~438"}
{"Item": 30, "Frequency": 2, "Connection": "root~39~48~237"}
{"Item": 30, "Frequency": 2, "Connection": "root~39~48~237~592~449"}
{"Item": 30, "Frequency": 2, "Connection": "root~39~48~41~89"}
{"Item": 30, "Frequency": 2, "Connection": "root~39~48~65"}
{"Item": 30, "Frequency": 2, "Connection": "root~39~48~89"}
{"Item": 30, "Frequency": 2, "Connection": "root~39~604"}
{"Item": 30, "Frequency": 2, "Connection": "root~48~41"}
{"Item": 30, "Frequency": 2, "Connection": "root~62"}
{"Item": 30, "Frequency": 1, "Connection": "root~101~31~389~396~267"}
{"Item": 30, "Frequency": 1, "Connection": "root~101~548~31~389~267"}
{"Item": 30, "Frequency": 1, "Connection": "root~10515~703~11"}
{"Item": 30, "Frequency": 1, "Connection": "root~123"}
{"Item": 30, "Frequency": 1, "Connection": "root~12925"}
{"Item": 30, "Frequency": 1, "Connection": "root~1393"}
{"Item": 30, "Frequency": 1, "Connection": "root~1393~1986"}

```

Figure 4.5: Procedure to Search Specific Item

### 4.6.3 Procedure to Search Specific Pattern

Once option '3' is entered by the user, the algorithm will execute the procedure to retrieve all the patterns that exist in the database which contain a particular pattern. The pseudocode of this procedure is presented as Procedure 3. Likewise, a few variables are set so that the display of patterns can be controlled accordingly. Before retrieving the patterns from the database, the user will be prompted to enter a specific pattern that is of interest to be analyzed. After capturing a specific pattern from the

user, the algorithm will retrieve all the patterns that contain the particular pattern. Similarly, if there are no patterns that relate to the specific pattern, the message of “No Records Found” will be displayed on screen. When the patterns related to a specific pattern are able to be located without much trouble, the problem that arises due to a certain trend of data can be resolved easily. A screenshot of the procedure to retrieve all the patterns which contain a particular pattern in the Retail data set from the FP-DB is shown in Figure 4.6. In this case, the patterns being retrieved contain the pattern of ‘root~39~48’.

---

**Procedure 3: Search\_Specific\_Pattern()**

---

```
1   Begin
2   input = 'c'
3   rpd = 30
4   n = 0
5   Display "Enter Connection Path for Specific Pattern Search: "
6   Get con_path
7   con_path = "%" + con_path + "%"
8   Do
9       dres = col.find("Connection LIKE :param1")
           .sort("Frequency DESC").limit(rpd).offset(n)
           .bind("param1", con_path.c_str()).execute()
10      If (dres.count() > 0) Then
11          While (doc = dres.fetchOne())
12              Display doc
13          End While
14          n += rpd
15          Display "Press 'e' to exit or other keys to
                continue: "
16          Get input
17      Else
18          Display "No Records Found."
19      End If
20  While (input != 'e')
21  End
```

---

```

C:\WINDOWS\system32\cmd.exe
----- FP-NoSQL : Searching for Frequent Patterns with NoSQL -----

Option 1 : Search for the patterns of all items
Option 2 : Search for the patterns of a specific item
Option 3 : Search for the items that contains a specific pattern
Option 4 : Mine the frequent itemsets of a specific item
Option 5 : Exit

-----

Please enter your option: 3

Enter Connection Path for Specific Pattern Search: root~39~48
{"Item": 38, "Frequency": 6102, "Connection": "root~39~48"}
{"Item": 41, "Frequency": 4177, "Connection": "root~39~48"}
{"Item": 32, "Frequency": 4166, "Connection": "root~39~48"}
{"Item": 41, "Frequency": 1543, "Connection": "root~39~48~38"}
{"Item": 32, "Frequency": 1236, "Connection": "root~39~48~38"}
{"Item": 41, "Frequency": 1198, "Connection": "root~39~48~32"}
{"Item": 89, "Frequency": 956, "Connection": "root~39~48"}
{"Item": 65, "Frequency": 866, "Connection": "root~39~48"}
{"Item": 170, "Frequency": 544, "Connection": "root~39~48~38"}
{"Item": 225, "Frequency": 501, "Connection": "root~39~48"}
{"Item": 41, "Frequency": 448, "Connection": "root~39~48~38~32"}
{"Item": 36, "Frequency": 440, "Connection": "root~39~48~38"}
{"Item": 237, "Frequency": 428, "Connection": "root~39~48"}
{"Item": 310, "Frequency": 418, "Connection": "root~39~48"}
{"Item": 110, "Frequency": 383, "Connection": "root~39~48~38"}
{"Item": 101, "Frequency": 325, "Connection": "root~39~48"}
{"Item": 475, "Frequency": 322, "Connection": "root~39~48"}
{"Item": 65, "Frequency": 322, "Connection": "root~39~48~41"}
{"Item": 170, "Frequency": 281, "Connection": "root~39~48~38~41"}
{"Item": 438, "Frequency": 253, "Connection": "root~39~48"}
{"Item": 89, "Frequency": 242, "Connection": "root~39~48~41"}
{"Item": 89, "Frequency": 234, "Connection": "root~39~48~38"}
{"Item": 1327, "Frequency": 221, "Connection": "root~39~48"}
{"Item": 89, "Frequency": 220, "Connection": "root~39~48~32"}
{"Item": 225, "Frequency": 217, "Connection": "root~39~48~41"}
{"Item": 255, "Frequency": 212, "Connection": "root~39~48"}
{"Item": 413, "Frequency": 202, "Connection": "root~39~48"}
{"Item": 12925, "Frequency": 201, "Connection": "root~39~48"}
{"Item": 65, "Frequency": 189, "Connection": "root~39~48~32"}
{"Item": 110, "Frequency": 187, "Connection": "root~39~48~38~41"}

```

Figure 4.6: Procedure to Search Specific Pattern

#### 4.6.4 Procedure to Mine Frequent Itemsets

When option ‘4’ is entered by the user, the algorithm will execute the procedure to retrieve all the patterns of a specific item that exist in the database, and generate all the combination of frequent itemsets for the specific item together with their



frequencies. The pseudocode of this procedure is presented as Procedure 4. Similarly, a few variables are set so that the display of patterns can be controlled accordingly. Before retrieving the patterns from the database, the user will be prompted to enter the item in which its frequent itemsets are required to be generated. After capturing an Item ID from the user, the algorithm will retrieve all the patterns in which the item is linked with, and export all the combination of frequent itemsets that are related to it into a log file. Being able to generate all the relevant frequent itemsets helps the user to identify which items are closely related to one another so that a problem in the data set can be rectified as soon as possible. An example of the frequent itemsets generated for Item 32 of the Retail data set is shown in Figure 4.7.

---

**Procedure 4: Mine\_Frequent\_Itemsets()**

---

```

1   Begin
2   input = 'c'
3   rpd = 30
4   n = 0
5   Display "Enter Item ID for Frequent Itemset Search: "
6   Get item
7   Open log file for storing frequent itemsets
8   Do
9       dres = col.find("Item = :param1").bind("param1", item)
           .execute()
10      While (doc = dres.fetchOne())
11          str_p = doc["Connection"]
12          str = str_p
13          boost::replace_all(str, "root~", " ")
14          boost::replace_all(str, "~", " ")
15          arr_pattern[total_pattern] = str + " "
16          arr_frequency[total_pattern] = doc["Frequency"]
17          total_pattern++
18      End While
19      dres = col.find("Item = :param1").limit(rpd).offset(n)
           .bind("param1", item).execute()
20      If (dres.count() > 0) Then
21          While (doc = dres.fetchOne())

```

---

---

```

22         str_i = doc["Connection"]
23         str = str_i
24         boost::replace_all(str, "root~", "")
25         boost::replace_all(str, "~", " ")
26         stringstream ss(str)
27         While (ss >> temp)
28             arr_item[total_item] = temp
29             total_item++
30         End While
31         For (total_combination = 1; total_combination <=
32             total_item; total_combination++)
33             Generate_Frequent_Itemsets()
34         End For
35         total_item = 0
36     End While
37     n += rpd
38     Display "Press 'e' to exit or other keys to continue:"
39     Get input
40 Else
41     Display "No Records Found."
42 End If
43 While (input != 'e')
44 Close log file for storing frequent itemsets
45 End

```

---

```

C:\Users\Timothy\source\repos\FrequentPatternAnalyzer\FrequentPatternAnalyzer\Itemset.txt - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
Itemset.txt x
1 Itemsets : 32, 38, Frequency : 2833
2 Itemsets : 32, 39, Frequency : 8455
3 Itemsets : 32, 39, 38, Frequency : 1840
4 Itemsets : 32, 48, Frequency : 8034
5 Itemsets : 32, 39, 48, Frequency : 5402
6 Itemsets : 32, 48, 38, Frequency : 1646
7 Itemsets : 32, 39, 48, 38, Frequency : 1236
8

```

Figure 4.7: Frequent Itemsets Related to Item 32

## 4.7 Chapter Summary

FP-NoSQL is proposed as an algorithm that mines the frequent itemsets using the Frequent Pattern Database (FP-DB) approach. The algorithm is constructed from a few modules to implement the respective task of Frequent Itemset Mining (FIM), namely, Data Loader, Frequent Item Generator, Frequent Pattern Processor, FP-Collection Constructor, and Frequent Pattern Analyzer. First, the Data Loader loads the data from a text file into a data warehouse since most of the data available for mining are stored in text files. Then, the Frequent Item Generator produces the list of frequent items by arranging the items in every transaction according to the sequence of the Item Frequency table. Frequent Pattern Processor discovers all the patterns of data that exist in the data warehouse by processing the Frequent Item table and consolidate them into a Frequent Pattern table. Next, the FP-Collection Constructor constructs a Not-Only Structured Query Language (NoSQL) collection that consists of all the patterns and frequencies of data by processing the Frequent Pattern table. Finally, the Frequent Pattern Analyzer analyzes the patterns and frequencies of data by processing the FP-Collection to generate the frequent itemsets.



## CHAPTER 5

### EXPERIMENT RESULTS AND DISCUSSION

This chapter presents the results of the experiments being conducted on six different data sets in order to evaluate the performance of the FP-NoSQL algorithm. Apart from presenting the experiment results, the computer platforms and data sets used for conducting the experiments are also described in Section 5.1 and Section 5.2 respectively. The experiment results for mining four different data sets are presented and discussed in Section 5.3. In Section 5.4, the experiment results for mining two additional data sets are presented and discussed in order to compare the run time performance of the FP-NoSQL algorithm with the Apriori and EFP algorithms. Then, the FP-NoSQL algorithm is also compared with other algorithms in terms of memory usage in Section 5.5. Last but not least, the performance of the FP-NoSQL algorithm is evaluated with the Big-O Notation in Section 5.6.

#### **5.1 Experiment Platform**

The FP-NoSQL algorithm is executed for performance testing on the following two platforms of computer hardware:

##### **Computer Platform 1**

Computer	: DELL Laptop
Central Processing Unit (CPU)	: Intel Core (TM) i7-7700HQ 2.8 GHz
Random Access Memory (RAM)	: 16 GB
Hard Disk Drive (HDD)	: 915 GB

## **Computer Platform 2**

Computer	: DELL Laptop
Central Processing Unit (CPU)	: Intel (R) Core (TM) i5-2520M 2.5 GHz
Random Access Memory (RAM)	: 4 GB
Hard Disk Drive (HDD)	: 250 GB

The software that are used to develop and execute the FP-NoSQL algorithm on both of the computer platforms above are as follows:

Operating System	: Windows 10
Integrated Development Environment	: Microsoft Visual Studio 2017
Programming Language	: C++
Database Server	: MySQL Server 5.7
Database Command Line Interface (CLI)	: MySQL Shell 1.0
Database Graphical User Interface (GUI)	: MySQL Workbench 6.3 CE
C++ to MySQL Connector	: Connector C++ 1.1

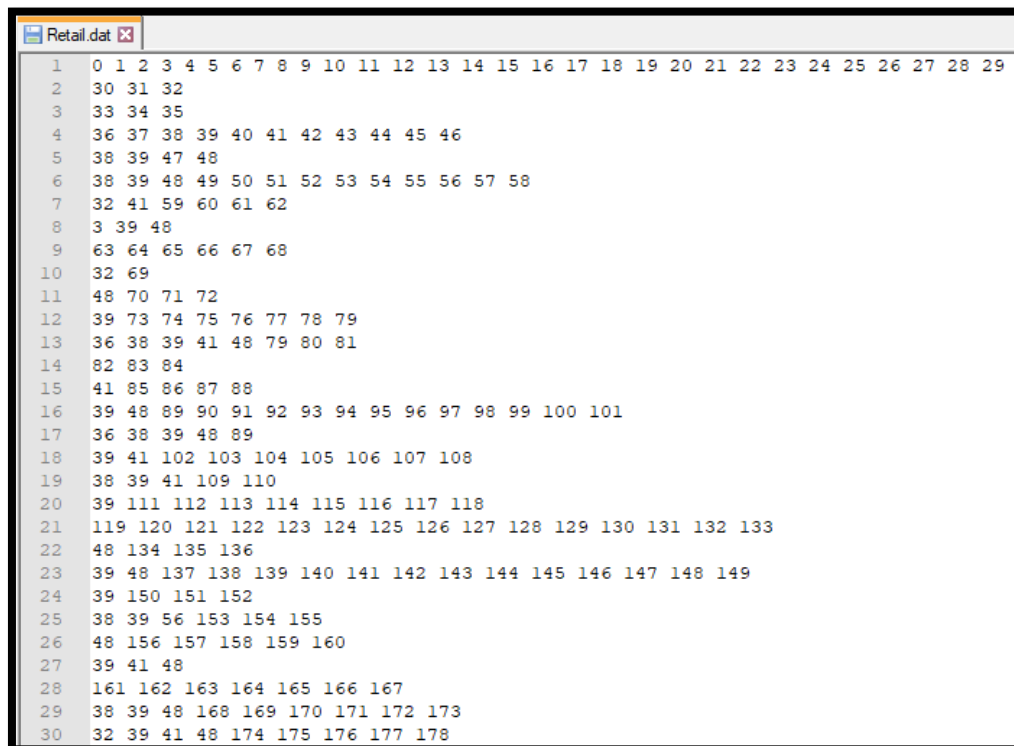
## **5.2 Experiment Data Set**

Four sets of data are downloaded from the internet in order to conduct the experiments for evaluating the FP-NoSQL algorithm. Frequent Itemset Mining Implementations (FIMI) Dataset Repository is the website where the data sets are downloaded online (Goethals, 2004). The data sets are specifically prepared for the purpose of Frequent Itemset Mining (FIM).

The four sets of data that have been downloaded online are the Retail, Connect, Pumsb, and Kosarak data sets. The Retail data set is a set of product sales data from a retail store, and the Connect data set is a set of network connection data from a computer server. The Pumsb data set is a set of census data from the United States of

America (USA), and the Kosarak data set is a set of click-stream data from a Hungarian online news portal.

The four sets of data can be categorized into two categories, namely as a sparse data set or a dense data set. A sparse data set contains transactions that have different number of items in each transaction, whereas a dense data set contains transactions that have similar number of items in each transaction (Pyun et al., 2014). Retail and Kosarak are considered as sparse data sets, while Connect and Pumsb are considered as dense data sets. The examples of sparse data sets are shown in Figure 5.1 and Figure 5.2 respectively for the Retail and Kosarak data sets, whereas the examples of dense data sets are shown in Figure 5.3 and Figure 5.4 respectively for the Connect and Pumsb data sets.



```
Retail.dat
1 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29
2 30 31 32
3 33 34 35
4 36 37 38 39 40 41 42 43 44 45 46
5 38 39 47 48
6 38 39 48 49 50 51 52 53 54 55 56 57 58
7 32 41 59 60 61 62
8 3 39 48
9 63 64 65 66 67 68
10 32 69
11 48 70 71 72
12 39 73 74 75 76 77 78 79
13 36 38 39 41 48 79 80 81
14 82 83 84
15 41 85 86 87 88
16 39 48 89 90 91 92 93 94 95 96 97 98 99 100 101
17 36 38 39 48 89
18 39 41 102 103 104 105 106 107 108
19 38 39 41 109 110
20 39 111 112 113 114 115 116 117 118
21 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133
22 48 134 135 136
23 39 48 137 138 139 140 141 142 143 144 145 146 147 148 149
24 39 150 151 152
25 38 39 56 153 154 155
26 48 156 157 158 159 160
27 39 41 48
28 161 162 163 164 165 166 167
29 38 39 48 168 169 170 171 172 173
30 32 39 41 48 174 175 176 177 178
```

Figure 5.1: Retail Data Set (Brijs, 1999)

```

Kosarak.dat
1 1 2 3
2 1
3 4 5 6 7
4 1 8
5 9 10
6 11 6 12 13 14 15 16
7 1 3 7
8 17 18
9 11 6 19 20 21 22 23 24
10 1 25 3
11 26 3
12 11 27 6 3 28 7 29 30 31 32 33 34 35 36 37
13 6 2 38
14 39 11 27 1 40 6 41 42 43 44 45 46 47 3 48 7 49 50 51
15 52 6 3 53
16 54 1 6 55
17 11 6 56 57 58 59 60 61 62 63 64
18 3
19 1 65 66 67 68 3
20 69 11 1 6
21 11 70 6
22 6 3 71
23 72 6 73
24 74
25 75 76
26 6 3 77
27 78 79 80 81
28 82 6 83 7 84 85 86 87 88
29 11 27 1 6 89 90 91 92 93 14 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110
30 6 138

```

Figure 5.2: Kosarak Data Set (Bodon, 2003)

```

Connect.dat
1 1 4 7 10 13 16 19 22 25 28 31 34 37 40 43 46 49 52 55 58 61 64 67 70 73 76 79 82 85 88 91
2 1 4 7 10 13 16 19 22 25 28 31 34 37 40 44 46 49 52 55 58 61 64 67 70 73 77 79 82 85 88 91
3 1 4 7 10 13 16 19 23 25 28 31 34 37 40 44 46 49 52 55 58 61 64 67 70 73 76 79 82 85 88 91
4 1 4 7 10 13 16 19 22 25 28 31 34 37 40 44 46 49 52 55 58 61 64 67 70 73 76 79 82 85 88 91
5 1 5 7 10 13 16 19 22 25 28 31 34 37 40 44 46 49 52 55 58 61 64 67 70 73 76 79 82 85 88 91
6 1 4 7 10 13 16 19 22 25 28 31 34 37 40 44 46 49 52 55 58 61 64 67 70 73 76 79 82 85 88 91
7 2 4 7 10 13 16 19 24 25 28 31 34 37 41 44 46 49 52 55 58 61 64 67 70 73 76 79 82 85 88 91
8 1 4 7 10 13 16 19 24 25 28 31 34 37 42 44 46 49 52 55 58 61 64 67 70 73 77 79 82 85 88 91
9 1 4 7 10 13 16 19 24 26 28 31 34 37 42 44 46 49 52 55 58 61 64 67 70 73 76 79 82 85 88 91
10 1 4 7 10 13 16 19 24 25 28 31 34 37 42 44 46 49 52 55 58 61 64 67 70 73 76 79 82 85 88 91
11 1 5 7 10 13 16 19 22 25 28 31 34 37 42 44 46 49 52 55 58 61 64 67 70 73 76 79 82 85 88 91
12 1 4 7 10 13 16 19 24 25 28 31 34 37 42 44 46 49 52 55 58 61 64 67 70 73 76 79 82 85 88 91
13 1 6 7 10 13 16 19 22 25 28 31 34 37 41 44 46 49 52 55 58 61 64 67 70 73 76 79 82 85 88 91
14 1 6 7 10 13 16 19 22 25 28 31 34 37 42 44 46 49 52 55 58 61 64 67 70 73 77 79 82 85 88 91
15 3 6 7 10 13 16 19 23 25 28 31 34 37 42 44 46 49 52 55 58 61 64 67 70 73 76 79 82 85 88 91
16 1 6 7 10 13 16 19 22 25 28 31 34 37 42 44 46 49 52 55 58 61 64 67 70 73 76 79 82 85 88 91
17 1 6 8 10 13 16 19 22 25 28 31 34 37 42 44 46 49 52 55 58 61 64 67 70 73 76 79 82 85 88 91
18 1 6 7 10 13 16 19 22 25 28 31 34 37 42 44 46 49 52 55 58 61 64 67 70 73 76 79 82 85 88 91
19 3 4 7 10 13 16 19 22 25 28 31 34 37 41 43 46 49 52 55 58 61 64 67 70 74 76 79 82 85 88 91
20 1 4 7 10 13 16 19 22 25 28 31 34 37 41 44 46 49 52 55 58 61 64 67 70 74 77 79 82 85 88 91
21 1 4 7 10 13 16 19 23 25 28 31 34 37 41 44 46 49 52 55 58 61 64 67 70 74 76 79 82 85 88 91
22 1 4 7 10 13 16 19 22 25 28 31 34 37 41 44 46 49 52 55 58 61 64 67 70 74 76 79 82 85 88 91
23 1 5 7 10 13 16 19 22 25 28 31 34 37 41 44 46 49 52 55 58 61 64 67 70 74 76 79 82 85 88 91
24 2 4 7 10 13 16 19 22 25 28 31 34 37 41 44 46 49 52 55 58 61 64 67 70 74 76 79 82 85 88 91
25 1 4 7 10 13 16 19 22 25 28 31 34 37 41 45 46 49 52 55 58 61 64 67 70 73 76 79 82 85 88 91
26 1 4 7 10 13 16 19 22 25 28 31 34 37 41 45 47 49 52 55 58 61 64 67 70 75 76 79 82 85 88 91
27 1 4 7 10 13 16 19 22 25 28 31 34 37 41 45 46 49 52 55 58 61 64 67 70 75 77 79 82 85 88 91
28 1 4 7 10 13 16 19 23 25 28 31 34 37 41 45 46 49 52 55 58 61 64 67 70 75 76 79 82 85 88 91
29 1 4 7 10 13 16 19 22 25 28 31 34 37 41 45 46 49 52 55 58 61 64 67 70 75 76 79 82 85 88 91
30 1 5 7 10 13 16 19 22 25 28 31 34 37 41 45 46 49 52 55 58 61 64 67 70 75 76 79 82 85 88 91

```

Figure 5.3: Connect Data Set (Bayardo, 2007a)



Transaction ID	Item 1	Item 2	Item 3	Item 4	Item 5	Item 6	Item 7	Item 8	Item 9	Item 10	Item 11	Item 12	Item 13	Item 14	Item 15	Item 16	Item 17	Item 18	Item 19	Item 20	Item 21	Item 22	Item 23	Item 24	Item 25	Item 26	Item 27	Item 28	Item 29	Item 30
1	0	14	17	60	66	75	84	125	155	161	163	168	170	180	184	188	198	205	260	265	277	290	324	2297	2300					
2	1	15	54	60	66	74	84	111	155	161	167	168	170	180	184	188	198	252	260	265	277	291	1218	2297	2300					
3	0	14	17	60	66	73	84	124	155	161	163	168	170	180	184	188	197	210	260	265	277	291	349	1380	2299					
4	1	15	17	60	66	73	84	111	155	161	165	168	170	180	184	188	197	252	260	265	277	291	1293	2297	2299					
5	2	15	17	57	70	74	84	111	160	161	167	168	170	180	184	188	197	252	260	265	278	291	349	2297	2299					
6	2	15	17	57	70	74	85	111	160	162	167	169	170	180	184	188	197	252	260	265	278	287	349	2297	2299					
7	2	15	17	57	70	73	85	111	154	162	167	169	170	180	184	188	197	252	260	265	278	285	349	2297	2299					
8	0	14	17	62	66	77	84	123	155	161	163	168	170	180	184	188	197	245	260	265	277	289	320	1348	2300					
9	1	15	17	62	66	76	84	111	155	161	167	168	170	180	184	188	197	252	260	265	277	289	387	1320	2300					
10	0	14	17	59	66	73	84	125	155	161	163	168	170	180	184	188	197	205	260	265	277	290	348	1320	2300					
11	1	15	17	59	66	73	84	111	155	161	166	168	170	180	184	188	197	252	260	265	277	291	387	1446	2300					
12	2	15	17	57	70	73	86	125	154	162	167	169	170	180	184	188	197	229	260	265	278	284	493	2297	2300					
13	2	15	17	56	70	73	86	125	154	162	167	169	170	180	184	188	197	229	260	265	278	283	493	2297	2300					
14	2	15	17	56	70	73	86	125	154	162	167	169	170	180	184	188	197	229	260	265	279	281	493	2297	2298					
15	0	14	17	60	66	74	84	111	155	161	163	168	170	180	184	188	198	252	260	265	278	292	330	2297	2300					
16	1	15	17	60	66	74	84	111	155	161	167	168	170	180	184	188	198	252	260	265	277	288	320	2297	2300					
17	2	14	17	58	70	75	84	111	160	161	163	168	170	180	184	188	198	252	260	265	277	290	330	2297	2300					
18	0	15	17	60	66	74	84	125	155	161	167	168	170	180	184	188	198	205	260	265	277	290	348	2297	2299					
19	1	14	17	62	66	75	84	111	155	161	163	168	170	180	184	188	198	252	260	265	278	294	320	1387	2299					
20	0	15	17	59	68	73	84	111	158	161	167	168	170	180	184	188	197	252	260	265	277	290	440	1347	2299					
21	0	15	17	61	66	74	84	111	155	161	165	168	170	180	184	188	197	252	260	265	277	290	330	1348	2299					
22	1	14	17	61	66	74	84	111	155	161	163	168	170	180	184	188	197	252	260	265	277	285	385	1318	2299					
23	2	14	17	57	70	73	87	111	160	162	163	169	170	180	184	188	197	252	260	265	278	286	330	1385	2299					
24	0	14	17	59	66	74	84	123	155	161	163	168	170	180	184	188	197	237	260	265	277	290	330	1387	2300					
25	1	15	17	59	66	74	84	123	155	161	165	168	170	180	184	188	197	217	260	265	277	288	319	2297	2300					
26	2	14	17	57	70	73	89	111	160	162	163	169	170	180	184	188	197	252	260	265	277	286	330	1387	2300					
27	2	15	17	56	70	75	89	111	154	162	167	169	170	180	184	188	197	252	260	265	278	284	330	1387	2300					
28	0	14	17	59	66	75	84	111	155	161	163	168	170	180	184	188	197	252	260	265	277	290	319	1320	2300					
29	1	15	17	59	66	73	84	125	155	161	165	168	170	180	184	188	197	203	260	265	277	290	320	1348	2300					
30	2	15	17	57	70	75	89	111	160	162	167	169	170	180	184	188	197	252	260	265	278	286	319	1320	2300					

Figure 5.4: Pumsb Data Set (Bayardo, 2007b)

### 5.3 Evaluating FP-NoSQL on Different Data Sets

In order to verify that the FP-NoSQL algorithm is suitable to be used for Frequent Pattern Mining (FPM), the algorithm is tested on four sets of data to evaluate its performance based on two important criteria: run time and memory usage. The objective of this evaluation is to confirm that the FP-NoSQL algorithm is able to mine any kind of data set, whether it is a sparse one or a dense one. In a sparse data set, every transaction has different number of items, whereas in a dense data set, every transaction has similar number of items. As shown in Figure 5.5, the Performance Profiler of Microsoft Visual Studio is used to monitor the run time and memory usage of the algorithm during its execution. The run time is measured in seconds (s) at the horizontal axis and the memory usage is measured in megabytes (MB) or gigabytes (GB) at the vertical axis.

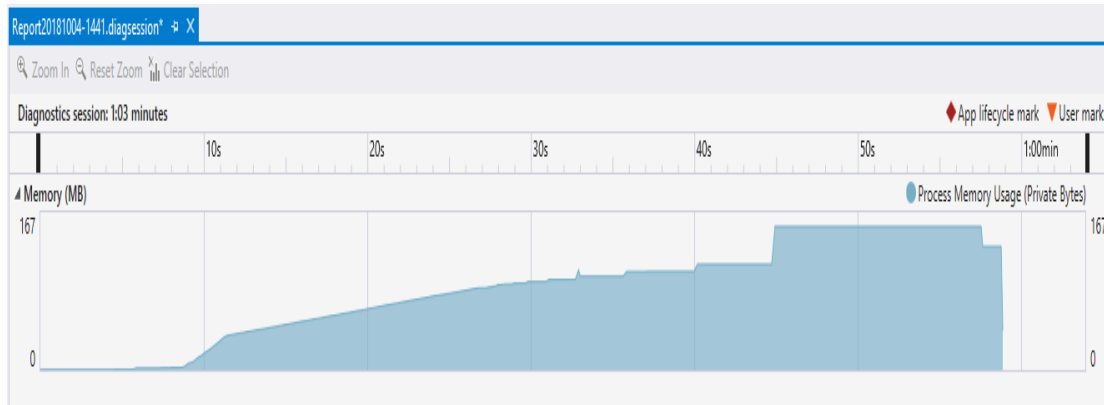


Figure 5.5: Performance Profiler of Microsoft Visual Studio

Since the execution of Performance Profiler will prolong the overall run time, a block of code implemented with the C++ programming language is included into the algorithm for measuring a more accurate run time as shown in Table 5.1. The code captures the start time and end time of execution for the algorithm in order to calculate its total run time. The run time is measured in microseconds so that a more accurate reading can be obtained. Then, it is converted from microseconds to seconds for easier analysis. After that, a log file is opened for the run time to be exported from the algorithm. Last but not least, the log file is closed once the entire algorithm is executed successfully.

Table 5.1: C++ Code to Measure Run Time of Algorithm

```

std::chrono::steady_clock::time_point begin = std::chrono::steady_clock::now();

std::chrono::steady_clock::time_point end = std::chrono::steady_clock::now();

ofstream log_file;

log_file.open("FrequentPatternProcessorLog.txt");

log_file << "Total Run Time : " << std::chrono::duration_cast

    <std::chrono::microseconds>(end - begin).count() << " microseconds";

log_file.close();

```

### 5.3.1 Experiment on Retail Data Set

The Retail data set is a set of product sales data from a retail store in the country of Belgium. It was donated to the FIMI Dataset Repository by Tom Brijs (Brijs, Swinnen, Vanhoof, & Wets, 1999). The data are gathered from three separated periods. The first period is between the second half month of December 1999 to the first half month of January 2000. Then, the second period is between January 2000 to the starting of June 2000. Last but not least, the third and final period is between the end of August 2000 to the end of November 2000. The data set contains 88162 transactions with a total amount of 908576 items. It is a sparse data set where every transaction has different number of items. There are 16740 unique items in the entire data set. Each transaction represents a purchase by a customer that visited the retail store. The items in every transaction are the products purchased by the customers.

To evaluate the performance of the FP-NoSQL algorithm in FPM, the algorithms of Frequent Pattern Processor and FP-Collection Constructor are executed on the whole data set multiple times using different values of minimum support. The entire experiment results for the Retail data set are recorded in Table 5.2. The experiment is performed on the data set based on different values of minimum support ranging from 0.5 % down to 0.02 %, which are equivalent to the minimum support count of 4543 down to 182. At each execution of the algorithms on the data set with a different value of minimum support, the Total Pattern, Unique Pattern, Run Time and Memory Usage are recorded accordingly. Total Pattern indicates the total number of patterns that can be located, whereas Unique Pattern represents the number of patterns that are unique throughout the data set when the algorithm is executed on it at a specific value of minimum support.

Table 5.2: Experiment Results for Retail Data Set

<b>Minimum Support (%)</b>	<b>Minimum Support Count</b>	<b>Total Pattern</b>	<b>Unique Pattern</b>	<b>Run Time (s)</b>	<b>Memory Usage (MB)</b>
0.5	4543	138518	30	9.019413	35.3
0.2	1817	174780	2613	10.8903	45.4
0.1	909	233882	29289	19.492268	66.6
0.08	727	258307	46766	22.025137	76.2
0.06	545	295450	77554	25.1766	94.1
0.04	363	364876	140834	42.671708	124.4
0.02	182	515347	286341	71.942734	209.3

As the algorithms are executed on the Retail data set using different values of minimum support, the run times recorded are between 9.02s to 71.94s as shown in Figure 5.6. When the minimum support is set to 0.5 % or 4543 counts, the total run time is only about 9.02s because the number of total patterns that can be located is less comparatively, which is equivalent to 138518 patterns. Furthermore, the total patterns are compressed into 30 unique patterns only. While the minimum support is set to a lower value at each execution of the algorithms, the number of total patterns and unique patterns that can be located from the data set are increasing gradually. When the minimum support is set to 0.02 % or 182 counts, the algorithms mine 515347 patterns from the data set and compress them into 286341 unique patterns around the time of 71.94s. Even though the number of unique patterns has been drastically increased as the minimum support is set to a lower value, the run time for each execution of the algorithms is increasing at a reasonable rate.

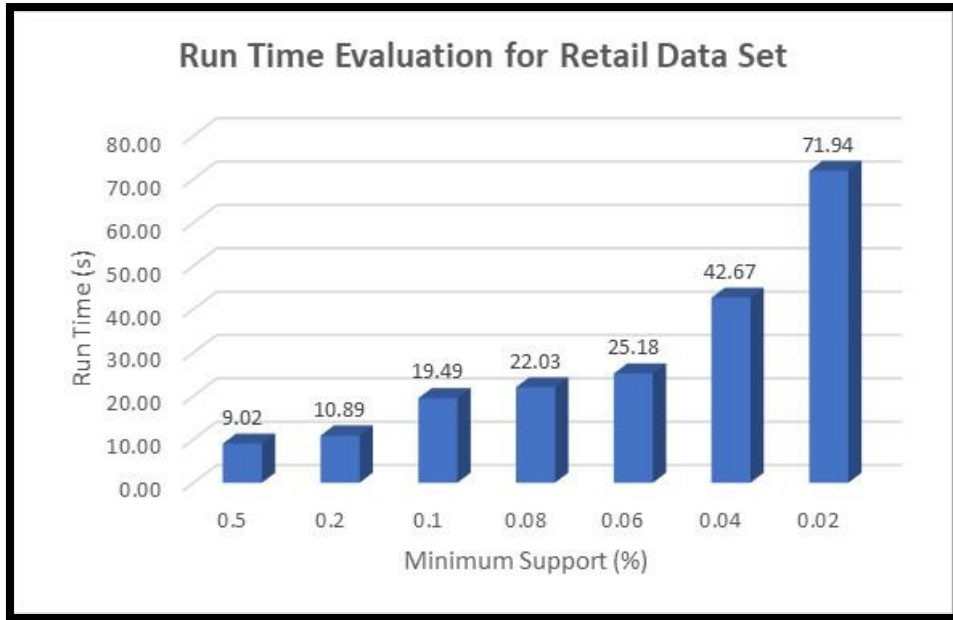


Figure 5.6: Run Time Evaluation for Retail Data Set

For memory usage, the consumptions are between 35.3 MB to 209.3 MB in mining the Retail data set as shown in Figure 5.7. Since the number of total patterns that can be located is less comparatively when the minimum support is set to 0.5 % or 4543 counts, the memory consumption for mining the data is also at a very less amount of 35.3 MB only. When the minimum support is set to 0.02 % or 182 counts, the algorithms mine 515347 patterns from the data set and compress them into 286341 unique patterns using about 209.3 MB of memory. Although the number of unique patterns has been drastically increased when the minimum support is set to a lower value, the memory consumption for each execution of the algorithms is increasing at a reasonable rate.

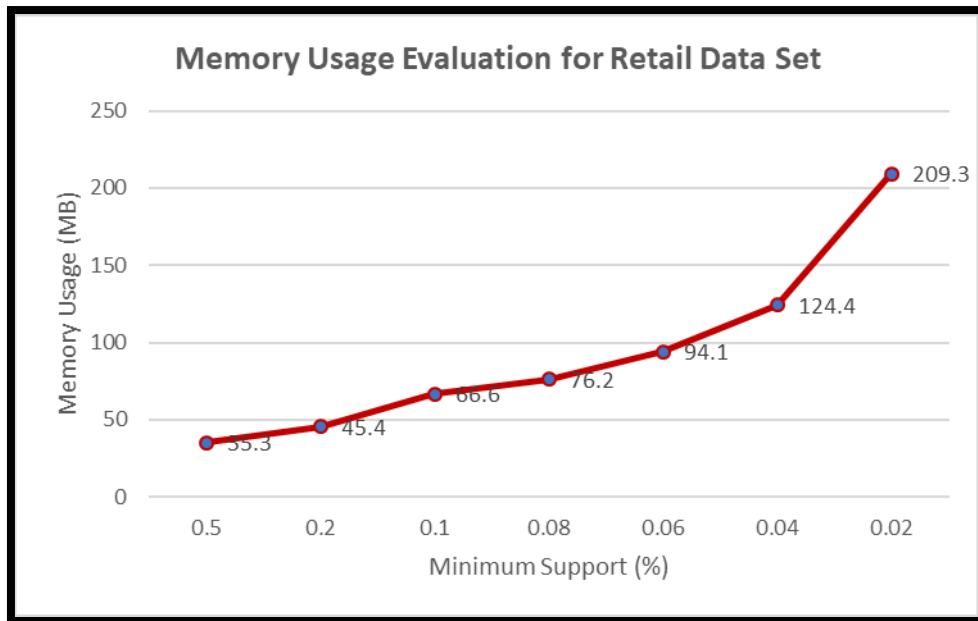


Figure 5.7: Memory Usage Evaluation for Retail Data Set

Both graphs in Figure 5.6 and Figure 5.7 present a reasonable increase for the run time and memory usage as the minimum support decreases to a lower value that allows more patterns to be discovered. According to the experiment results, the algorithms are able to mine 515347 patterns from the data set and compress them into 286341 unique patterns around the time of 71.94s using about 209.3 MB of memory when the minimum support is set to 0.02 % or 182 counts. In other words, it is equivalent to having the capability to mine around 7164 patterns within a second using about 2.91 MB only. Therefore, this indicated that the algorithms are capable enough to mine the data within a shorter run time using less memory consumption although the total number of records is increased to a much greater value.

### 5.3.2 Experiment on Connect Data Set

The Connect data set is a set of network connection data from a computer server. It was donated to the FIMI Dataset Repository by Roberto Bayardo who prepared it from one of the data sets in the UCI Machine Learning Repository (Asuncion & Newman, 2007). The data set contains 67557 transactions with a total amount of

2904951 items. It is a dense data set where every transaction has similar number of items. There are only 129 unique items in the entire data set.

Similarly, the algorithms of Frequent Pattern Processor and FP-Collection Constructor are executed on the whole data set multiple times using a different value of minimum support. The entire experiment results for the Connect data set are recorded in Table 5.3. The experiment is performed on the data set based on different values of minimum support from 0.5 % down to 0.02 %, which are equivalent to the minimum support count of 14525 down to 581.

Table 5.3: Experiment Results for Connect Data Set

<b>Minimum Support (%)</b>	<b>Minimum Support Count</b>	<b>Total Pattern</b>	<b>Unique Pattern</b>	<b>Run Time (s)</b>	<b>Memory Usage (GB)</b>
0.5	14525	2627410	118715	159.63	1.3854
0.2	5810	2824055	279695	182.00	1.6347
0.1	2905	2871443	326014	190.63	1.6902
0.08	2324	2881850	336240	190.14	1.7021
0.06	1743	2885745	340104	192.59	1.7068
0.04	1162	2892423	346770	194.29	1.7156
0.02	581	2900616	354955	193.51	1.7530

As shown in Figure 5.8, the run time of the algorithms are between 159.63s to 193.51s when the algorithms are executed on the Connect data set using different values of minimum support. Even though the minimum support is set to a lower value at each execution of the algorithms, the run time for each execution of the algorithms is quite consistent with one another. This is because the total amount of items available in each transaction is very similar to one another in a dense data set.

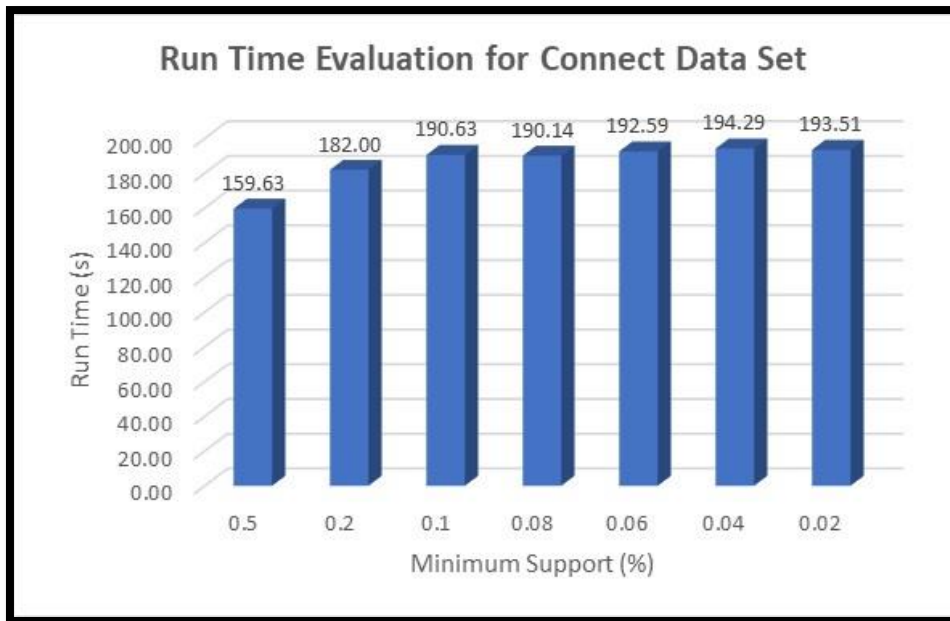


Figure 5.8: Run Time Evaluation for Connect Data Set

For memory usage, the consumptions are between 1.3854 GB to 1.7530 GB in mining the Connect data set as shown in Figure 5.9. When the minimum support is set to 0.5 % or 14525 counts, the memory consumption for mining the data is at the amount of 1.3854 GB. Comparing to the Retail data set, the memory consumption for mining the Connect data set is much higher because the number of total patterns and unique patterns that can be located from the data set are much more. However, the run time for each execution of the algorithms on the Connect data set is just slightly higher compared to mining the Retail data set.



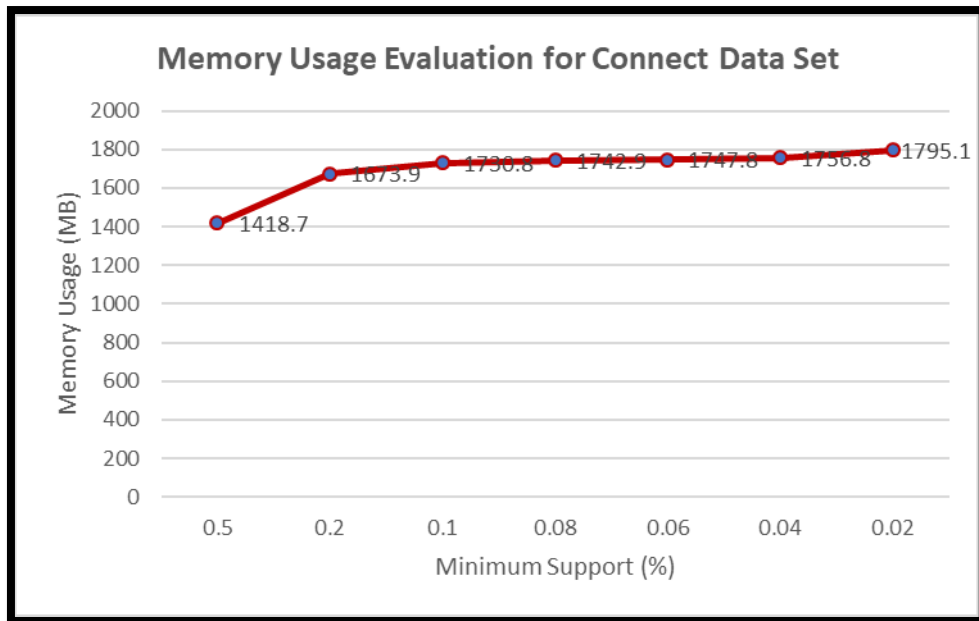


Figure 5.9: Memory Usage Evaluation for Connect Data Set

Both graphs in Figure 5.8 and Figure 5.9 present a more stable differences of values for the run time and memory usage as the minimum support decreases to a lower value that allows more patterns to be identified. According to the experiment results, the algorithms are able to mine 2900616 patterns from the data set and compress them into 354955 unique patterns around the time of 193.51s using about 1.753 GB of memory when the minimum support is set to 0.02 % or 581 count. In other words, it is equivalent to having the capability to process around 14990 patterns within a second using about 9.28 MB only. Thus, this showed that the algorithms are still able to mine the data within a short run time using reasonable memory consumption even though the total number of records is increased to a much greater value especially when the data set is in a dense structure.

### 5.3.3 Experiment on Pumsb Data Set

The Pumsb data set is a set of census data from the United States of America (USA). It was donated to the FIMI Dataset Repository by Roberto Bayardo who prepared it from one of the datasets in the UCI Machine Learning Repository

(Asuncion & Newman, 2007). The data set contains 49046 transactions with a total amount of 3629404 items. It is a dense data set where every transaction has similar number of items. There are 2113 unique items in the entire data set.

Likewise, the algorithms of Frequent Pattern Processor and FP-Collection Constructor are executed on the whole data set multiple times using a different value of minimum support. The entire experimental results for the Pumsb data set are recorded into Table 5.4. The experiment is performed on the data set based on different values of minimum support from 0.5 % down to 0.02 %, which are equivalent to the minimum support count of 18147 down to 726.

Table 5.4: Experiment Results for Pumsb Data Set

<b>Minimum Support (%)</b>	<b>Minimum Support Count</b>	<b>Total Pattern</b>	<b>Unique Pattern</b>	<b>Run Time (s)</b>	<b>Memory Usage (GB)</b>
0.5	18147	2557281	310484	225.28	2.1992
0.2	7259	3057326	586571	304.03	3.0000
0.1	3629	3205939	720525	327.07	3.5000
0.08	2904	3260587	772910	349.08	3.5000
0.06	2178	3302210	812542	369.88	3.7002
0.04	1452	3374315	884119	384.44	3.7998
0.02	726	3471736	981077	420.99	4.2002

As shown in Figure 5.10, the run time of the algorithms are between 225.28s to 420.99s when the algorithms are executed on the Pumsb data set using different values of minimum support. Just like the Connect data set, although the minimum support is set to a lower value at each execution of the algorithms, the run time for each execution of the algorithms is quite consistent with one another for the Pumsb data set. This is because the total amount of items available in every transaction in a

dense data set is quite similar with one another. When the minimum support count is set to 0.5 % or 18147 counts, the run time of the algorithms is much lower compared to the others because the number of unique pattern that can be identified is at a much lesser amount.

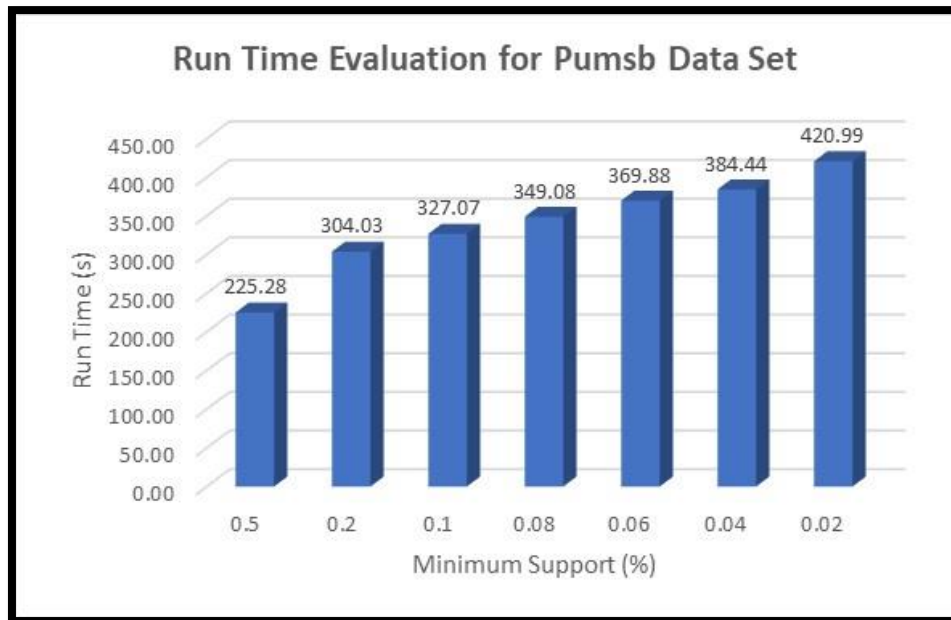


Figure 5.10: Run Time Evaluation for Pumsb Data Set

For memory usage, the consumptions are between 2.1992 GB to 4.2002 GB in mining the Pumsb data set as shown in Figure 5.11. When the minimum support is set to 0.5 % or 18147 counts, the memory consumption for mining the data is at the amount of 2.1992 GB. Similarly, comparing to a sparse data set, the memory consumption for mining the Pumsb data set is much higher because more total patterns and unique patterns can be located from a dense data set. Nevertheless, the run time for each execution of the algorithms on the Pumsb data set is only slightly higher compared to mining a sparse data set.

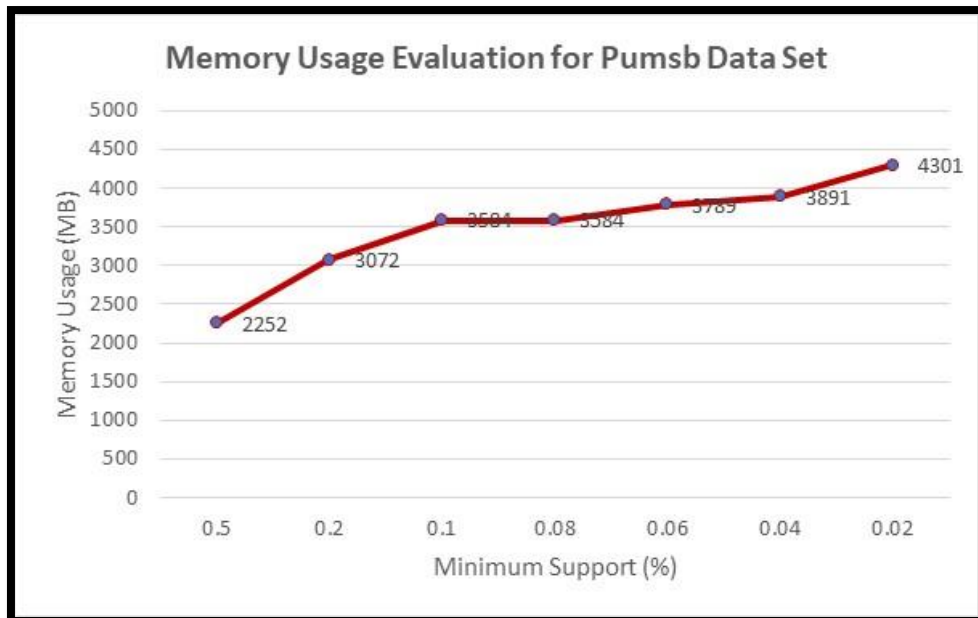


Figure 5.11: Memory Usage Evaluation for Pumsb Data Set

Both graphs in Figure 5.10 and Figure 5.11 present a more constant changes of values for the run time and memory usage as the minimum support decreases to a lower value that allows more patterns to be discovered. This is because the total amount of items available in each transaction is very alike to one another. According to the experiment results, the algorithms are able to mine 3471736 patterns from the data set and compress them into 981077 unique patterns around the time of 420.99s using about 4.2002 GB of memory when the minimum support is set to 0.02 % or 726 count. In other words, it is equivalent to having the capability to process around 8247 patterns within a second using about 10.22 MB only.

### 5.3.4 Experiment on Kosarak Data Set

The Kosarak data set is a set of click-stream data from a Hungarian online news portal. It was donated to the FIMI Dataset Repository by Ferenc Bodon. The data set contains 990002 transactions with a total amount of 8019015 items. It is a sparse data set where every transaction has different number of items. There are 41270 unique items in the entire data set. It is the biggest one among the four different data sets.

Similarly, the algorithms of Frequent Pattern Processor and FP-Collection Constructor are executed on the whole data set multiple times using a different value of minimum support. The entire experiment results for the Kosarak data set are recorded into Table 5.5. The experiment is performed on the data set based on different values of minimum support from 0.5 % down to 0.02 %, which are equivalent to the minimum support count of 40095 down to 1604.

Table 5.5: Experiment Results for Kosarak Data Set

<b>Minimum Support (%)</b>	<b>Minimum Support Count</b>	<b>Total Pattern</b>	<b>Unique Pattern</b>	<b>Run Time (s)</b>	<b>Memory Usage (MB)</b>
0.5	40095	2208696	3390	76.13	494.3
0.2	16038	2704833	140438	108.61	662.4
0.1	8019	3101903	422014	162.09	850.8
0.08	6415	3310181	598569	199.81	973.9
0.06	4811	3656279	894556	232.35	1208.4
0.04	3208	4107273	1316876	319.58	1608.3
0.02	1604	5227955	2327143	667.91	2867

As the algorithms are executed on the Kosarak data set using different values of minimum support, the run times recorded are between 76.13s to 667.91s as shown in Figure 5.12. When the minimum support is set to 0.5 % or 40095 counts, the total run time is only about 76.13s because the number of total patterns that can be located is less comparatively, which is equivalent to 2208696 patterns. Moreover, the total patterns are compressed into 3390 unique patterns only. While the minimum support is set to a lower value at each execution of the algorithms, the number of total patterns and unique patterns that can be located from the data set are increasing gradually. When the minimum support is set to 0.02 % or 1604 counts, the algorithms mine 5227955 patterns from the data set and compress them into 2327143 unique patterns

around the time of 667.91s. Although the number of unique patterns has been drastically increased when the minimum support is set to a lower value, the run time for each execution of the algorithms is increasing at a reasonable rate.

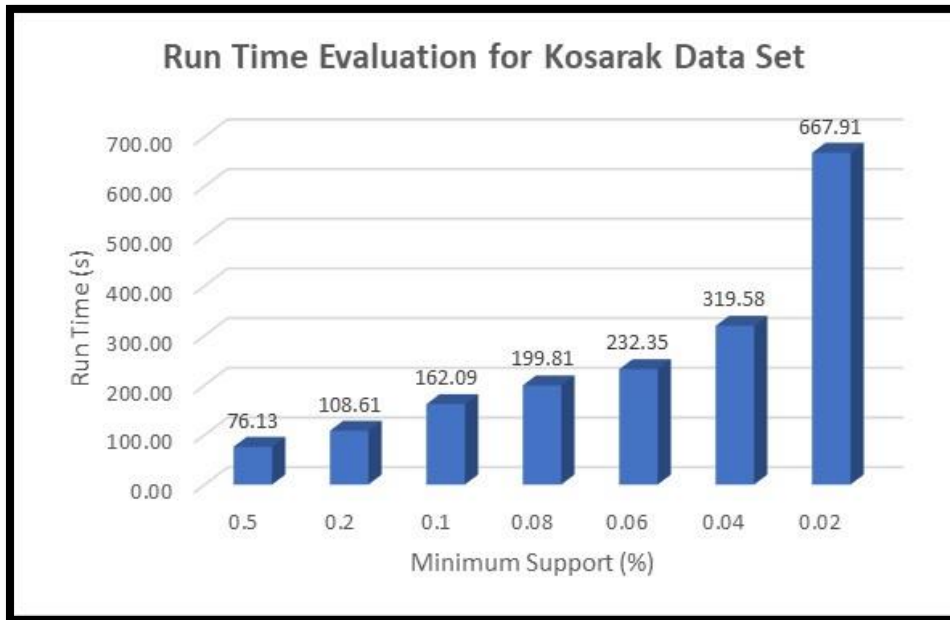


Figure 5.12: Run Time Evaluation for Kosarak Data Set

For memory usage, the consumptions are between 494.3 MB to about 2.8 GB in mining the Kosarak data set as shown in Figure 5.13. Since the number of total patterns that can be located is less comparatively when the minimum support is set to 0.5 % or 40095 counts, the memory consumption for mining the data is also at a very less amount of 494.3 MB only. When the minimum support is set to 0.02 % or 1604 counts, the algorithms mine 5227955 patterns from the data set and compress them into 2327143 unique patterns using about 2.8 GB of memory. Even though the number of unique patterns has been drastically increased when the minimum support is set to a lower value, the memory consumption for each execution of the algorithms is increasing at a reasonable rate.

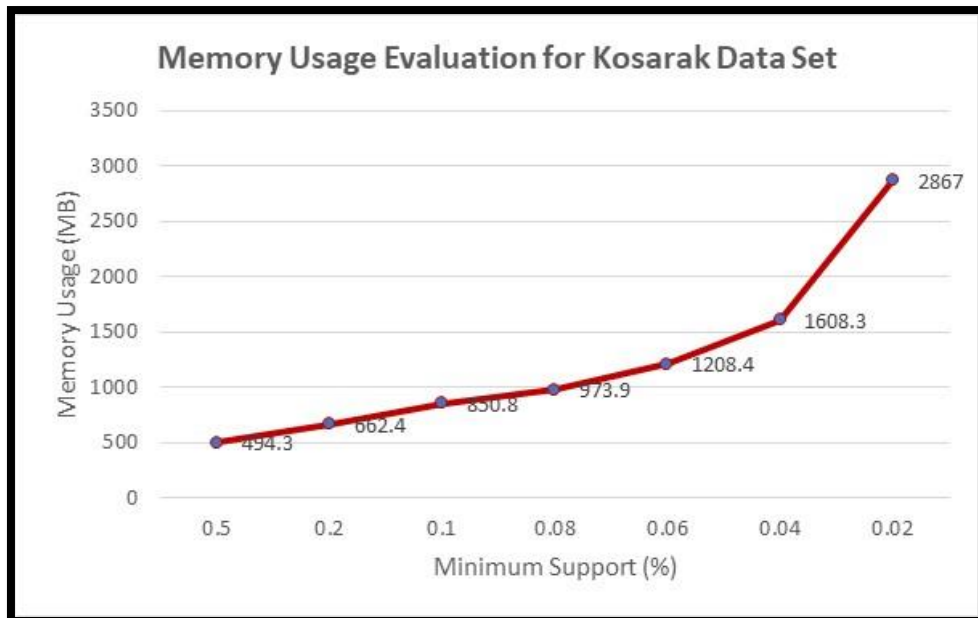


Figure 5.13: Memory Usage Evaluation for Kosarak Data Set

Both graphs in Figure 5.12 and Figure 5.13 indicate a reasonable rise for the run time and memory usage as the minimum support decreases to a lower value that allows more patterns to be identified. According to the experiment results, the algorithms are able to mine 5227955 patterns from the data set and compress them into 2327143 unique patterns around the time of 667.91s using about 2.8 GB of memory when the minimum support is set to 0.02 % or 1604 count. In other words, it is equivalent to having the capability to process around 7828 patterns within a second using about 4.29 MB only.

### 5.3.5 Summary of Experiment Results on Different Data Sets

The purpose of executing the FP-NoSQL algorithm on four different sets of data is to confirm that the algorithm is capable of mining the frequent itemsets from various kind of data sets. Both the sparse and dense data sets used to conduct the experiments consist of short and long patterns. Among the four data sets, the total number of transactions are between 49 thousands to 0.990002 million records, while

the total number of items are between 0.908576 million to 8.019015 millions records. The details of the four sets of experiment data are summarized in Table 5.6.

Table 5.6: Details of Four Experiment Data Sets

<b>Data Set</b>	<b>Type</b>	<b>Total Transaction</b>	<b>Total Item</b>	<b>Unique Item</b>	<b>File Size (MB)</b>
Retail	Sparse	88162	908576	16740	3.89
Connect	Dense	67557	2904951	129	8.76
Pumsb	Dense	49046	3629404	2113	15.8
Kosarak	Sparse	990002	8019015	41270	30.5

For the sparse data sets (Retail and Kosarak), the FP-NoSQL algorithm is able to mine the frequent patterns with a gradual increase of run time and memory usage even though the minimum support is decreased to a lower percentage value. On the other hand, for the dense data sets (Connect and Pumsb), the FP-NoSQL algorithm is able to mine the frequent patterns with a more consistent run time and memory usage although the minimum support is decreased to a lower percentage value. Therefore, the experiments conducted have proven that the FP-NoSQL algorithm is able to mine a big data set with a shorter run time and lesser memory usage since it is scalable although the minimum support is set to a very low percentage value.

#### **5.4 Evaluating Run Time of FP-NoSQL by Comparing to Apriori and EFP**

In order to prove that the FP-NoSQL algorithm is able to mine the frequent patterns using a shorter run time even though the size of a data set is increased, the algorithm is executed on two additional sets of data that have been mined with the Apriori and Extended Frequent Pattern (EFP) algorithms (Shang, 2005). The objective of this evaluation is to confirm that the FP-NoSQL algorithm is able to outperform the existing significant algorithms in Frequent Pattern Mining. The details of the two sets



of experiment data are summarized in Table 5.7. Figure 5.14 and Figure 5.16 show the experiment results of the run time evaluation performed by the Apriori and EFP algorithms. The experiment results of the run time evaluation performed by the FP-MySQL algorithm are presented in Figure 5.15 and Figure 5.17.

Table 5.7: Details of Two Experiment Data Sets

<b>Data Set</b>	<b>Type</b>	<b>Total Transaction</b>	<b>Total Item</b>	<b>Unique Item</b>	<b>File Size (MB)</b>
T25I10D10K	Sparse	9976	247148	929	0.92
T10I4D100K	Sparse	100000	1010228	870	3.74

For the T25I10D10K data set in Figure 5.14 and Figure 5.15, when the values of minimum support are set between 1 % (2471 counts) to 0.2 % (494 counts), the Apriori algorithm is able to mine the frequent patterns within the range of 500 seconds to 20 thousands seconds. The Apriori algorithm is able to mine the frequent patterns at a shorter run time of 500 seconds and 800 seconds, when the values of minimum support are set to higher values of 1 % (2471 counts) and 0.75 % (1854 counts). However, the run time of the Apriori algorithm is drastically increased to 16 thousands seconds and 20 thousands seconds, when the values of minimum support are set to lower values of 0.25 % (618 counts) and 0.2 % (494 counts). This is because the Apriori algorithm needs to generate a large number of candidate itemsets throughout the Frequent Pattern Mining (FPM) process by repeatedly scanning the entire database to perform pattern matching.

On the other hand, with the same values of minimum support, the EFP algorithm is able to mine the frequent patterns within the range of 1600 seconds to 8700 seconds. Although the values of minimum support are set to lower values of 0.25 % (618 counts) and 0.2 % (494 counts), the EFP algorithm is still able to mine the frequent patterns at 6900 seconds and 8700 seconds. This is because the EFP algorithm does not need to perform the step of candidate itemsets generation and test.

Even though the run time of the EFP algorithm is significantly lower at a few thousands seconds compared to the Apriori algorithm, the FP-NoSQL algorithm is able to mine the frequent patterns at much shorter run times between 2.87 seconds and 16.47 seconds, using the same values of minimum support. EFP still requires a few thousands seconds in mining the frequent patterns when the minimum support is low because EFP mines the frequent patterns transaction by transaction and connects to the database several times throughout the FPM process. But FP-NoSQL is able to have such a powerful performance in mining the frequent patterns because the frequent patterns are mined and stored into the Frequent Pattern Database (FP-DB) at the same time. The data warehouse is only scanned once for retrieving all the data that need to be mined. As the data is being processed, all the frequent patterns found are concatenated together in the memory, so that it can be inserted into the FP-DB concurrently. This method of mining the frequent patterns enables the run time to be reduced significantly since only two database connections are required.

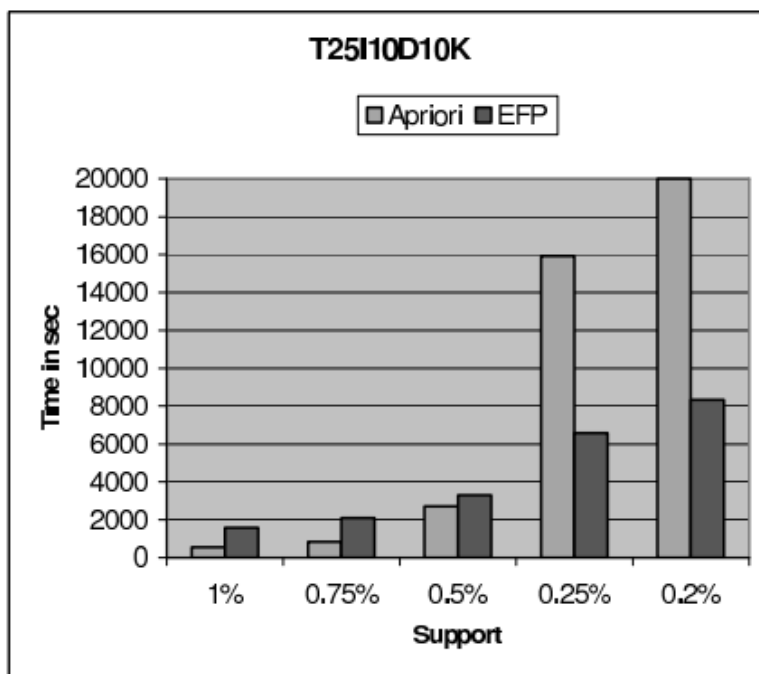


Figure 5.14: Run Time Evaluation for T25I10D10K Data Set by Apriori and EFP (Shang, 2005)

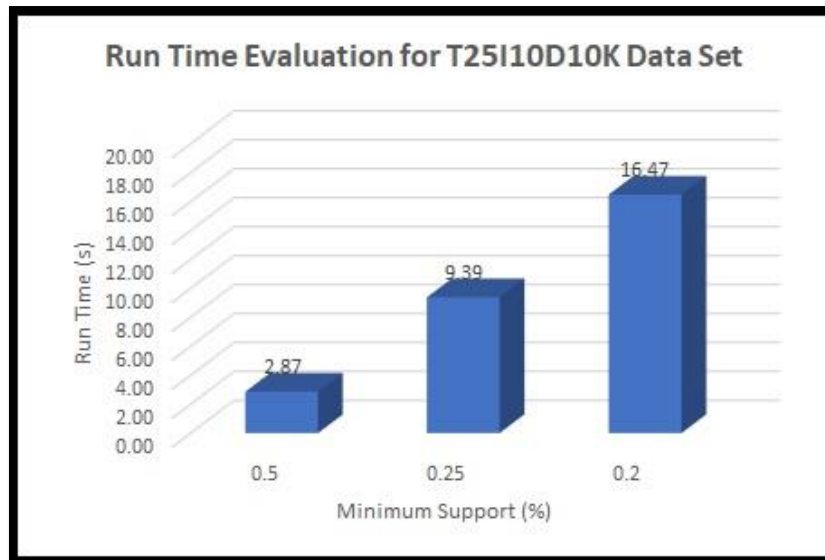


Figure 5.15: Run Time Evaluation for T25I10D10K Data Set by FP-NoSQL

For the T10I4D100K data set, when the values of minimum support are set between 1 % (10102 counts) to 0.02 % (202 counts), the Apriori algorithm is able to mine the frequent patterns within the range of 1304 seconds to 20 thousands seconds for the minimum supports between 1 % (10102 counts) to 0.1 % (1010 counts). When the value of minimum support is set at 0.08 % (808 counts) or lower, mining of the frequent patterns by the Apriori algorithm needs to be terminated due to its very long run time for candidate itemset generation and test. In contrast, with the same values of minimum support, the EFP algorithm is able to mine the frequent patterns within the range of 2174 seconds to almost 70 thousands seconds. However, the FP-NoSQL algorithm is able to mine the frequent patterns at much shorter run times between 5.69 seconds and 113.05 seconds, using the same values of minimum support.

Similarly, FP-NoSQL is capable of outperforming Apriori and EFP with such a major difference of run time because FP-NoSQL mines the frequent patterns and consolidates them into the FP-DB simultaneously, instead of inserting them one by one through a repeating loop of database scanning. The following code is the method used in Algorithm 4 of FP-NoSQL to concatenate the frequent patterns being mined so that it can be imported into the FP-DB at the same time:

```

While NOT End of Resultset Do

    pat_count++

    If cur_pat != next_pat Then

        documents += "({'\"Item\":" + item

            + "\",\"Frequency\":" + pat_count

            + "\",\"Connection\":" + con_path

            + "\"}'),"

        pat_count = 0

    End If

End While

```

When the FP-NoSQL algorithm loops through the list of frequent patterns, a counter (*pat\_count*) is used to count the unique frequent patterns. If the current frequent pattern (*cur\_pat*) is not equivalent to the next frequent pattern (*next\_pat*), then the value of *pat\_count* is considered as the frequency of the frequent pattern. Then, the current item (*item*) is concatenated with its connection path (*con\_path*) and frequency (*pat\_count*) to the *documents* string for inserting into the FP-Collection after all the frequent patterns have been processed. Therefore, the total run time of mining the frequent patterns can be greatly reduced since the data warehouse and FP-DB are not being connected and accessed by the FP-NoSQL algorithm several times during the entire FPM process.

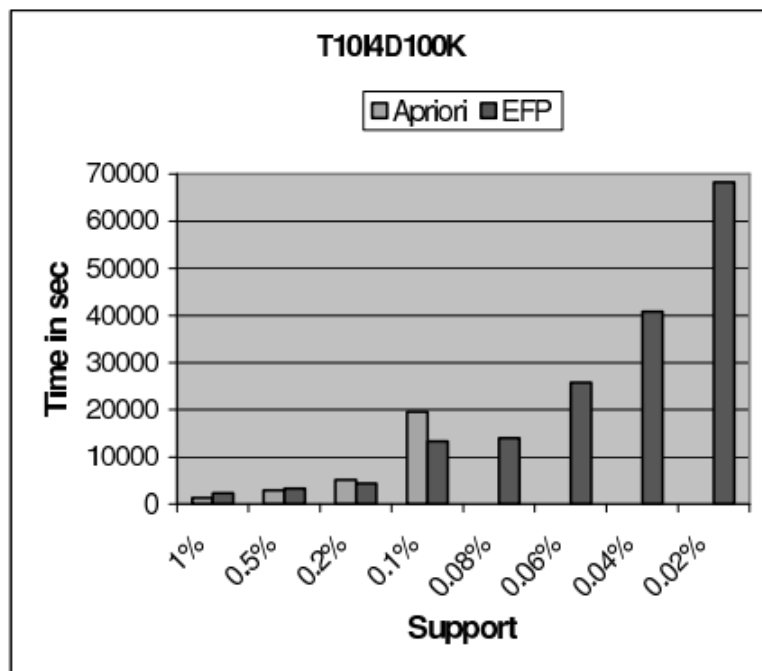


Figure 5.16: Run Time Evaluation for T10I4D100K Data Set by EFP Algorithm (Shang, 2005)

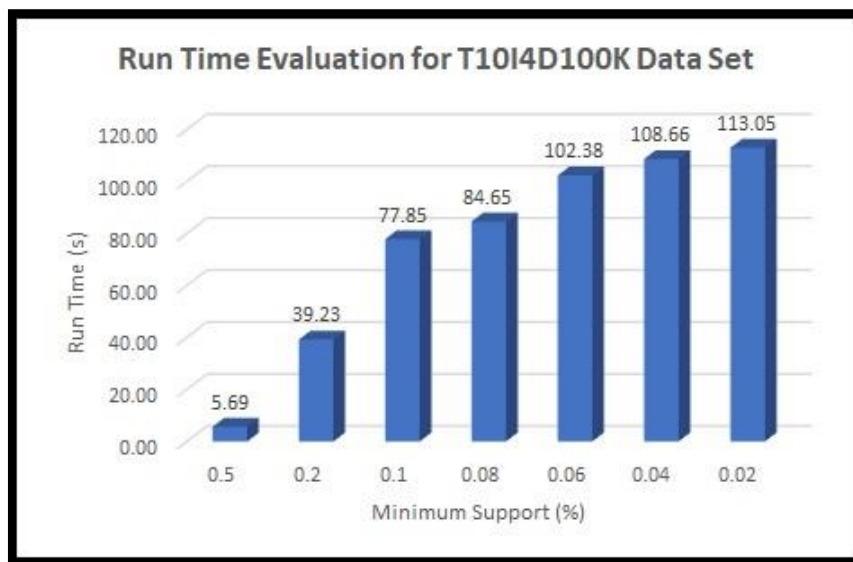


Figure 5.17: Run Time Evaluation for T10I4D100K Data Set by FP-NoSQL

## 5.5 Evaluating Memory Usage of FP-NoSQL without FP-DB and with FP-DB

To evaluate the performance of an algorithm, measuring the memory usage is another common method apart from calculating its run time. In this study, three experiments for measuring the memory usage of some existing Frequent Pattern Mining (FPM) algorithms have been reviewed as shown in Figure 5.18, Figure 5.19 and Figure 5.20. These experiments indicated that the memory usage for mining the frequent itemsets of data is increasing exponentially especially when the amount of data is big. In Figure 5.18 and Figure 5.19, the algorithms of GMiner, BigFIM and DistEclat increased the memory usage to a few gigabytes (GB) in order to obtain a lower run time. This is because as the memory space for processing the data set is expanded, the amount of frequent itemsets that can be mined are able to be increased too. In Figure 5.20, the amount of memory usage for the Plausibility Rule Mining (PRM) and FP-Growth algorithms are much lower at around 50 to 250 megabytes (MB) compared to other algorithms in the two experiments because the pruning technique is applied in mining the frequent itemsets. However, pruning is not a suitable technique to decrease the run time of an algorithm for mining the frequent itemsets because it may prevent some significant patterns from being discovered.

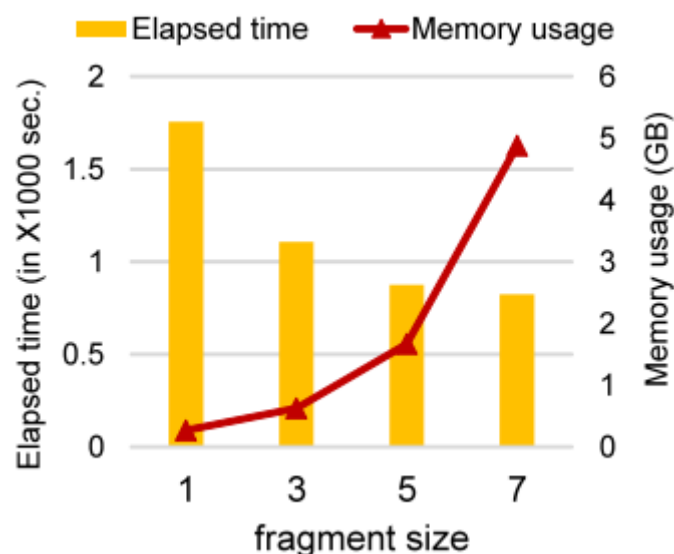


Figure 5.18: Time and Space for HIL Strategy of GMiner Algorithm  
(Chon, Hwang, & Kim, 2018)

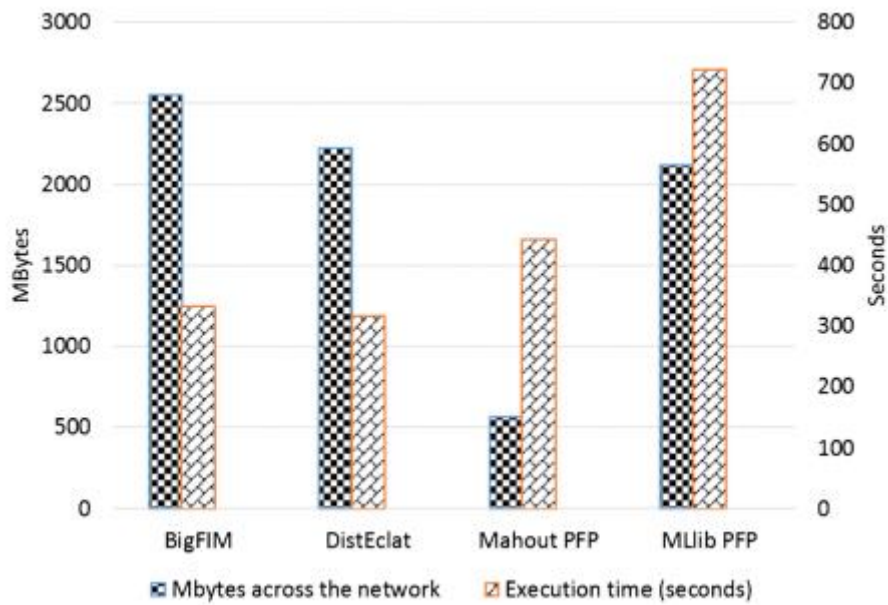


Figure 5.19: Communication Costs and Performance of FPM Algorithms (Apiletti et al., 2017)

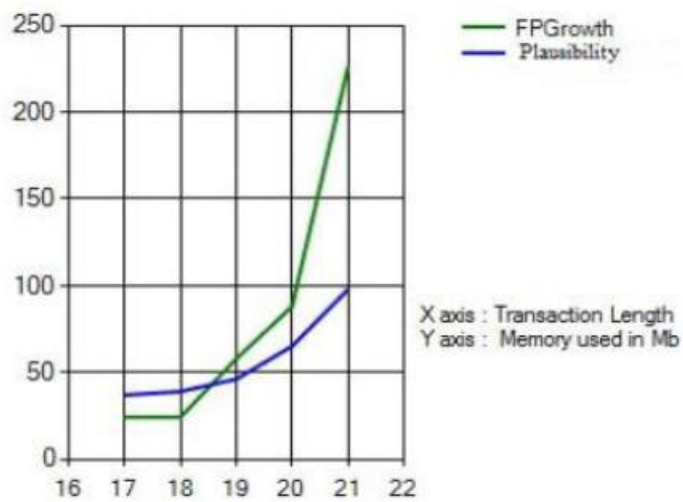


Figure 5.20: Memory Usage of PRM and FP-Growth at Support = 10% (Abraham & Joseph, 2016)

Therefore, in order to verify that the FP-NoSQL algorithm is able to mine the frequent itemsets using less memory consumption even though the size of a data set is increased, the algorithm is executed on the T25I10D10K and T10I4D100K data sets. The experiment results of the memory usage evaluation performed by the FP-NoSQL algorithm for two different scenarios are presented in Figure 5.21 and Figure 5.22. The first scenario is an FPM process that is conducted without an FP-DB, while the second scenario is an FPM process that is conducted with an FP-DB.

For the T25I10D10K data set, when the values of minimum support are set between 0.5 % (1236 counts) to 0.2 % (494 counts), in the first scenario where FPM is performed without an FP-DB, the memory usage for mining the frequent itemsets are within the range of 3.9 MB to 59.6 MB. But in the second scenario where FPM is performed with an FP-DB, the memory usage for mining the frequent itemsets are constantly maintained at 1.5 MB only.

For the T10I4D100K data set, when the values of minimum support are set between 0.5 % (5051 counts) to 0.02 % (202 counts), in the first scenario where FPM is performed without an FP-DB, the memory usage for mining the frequent itemsets are within the range of 18.6 MB to 498 MB. But in the second scenario where FPM is performed with an FP-DB, the memory usage for mining the frequent itemsets are within the range of 1.3 MB to 1.5 MB only.

In both data sets, when an FP-DB is not utilized, the memory usage for mining the frequent itemsets is increasing gradually from a few MB to a few hundreds MB as the minimum support is set to a lower value. This is because the number of frequent patterns that need to be mined will be increased when the minimum support is set to a lower value. In such a situation, patterns that appear less frequently in the data warehouse will also be included into the FPM process. Apart from this, the entire data warehouse needs to be mined in order to discover the frequent itemsets. Thus, the memory usage for mining the frequent itemsets is growing progressively as the minimum support is set to a lower value especially if the size of the entire data set is very large.



In contrast, when an FP-DB is implemented for both data sets, the memory usage for mining the frequent itemsets is maintained consistently within the range of 1.3 MB to 1.5 MB only. It is so since the entire data warehouse does not need to be mined again in order to discover the frequent itemsets. The frequent itemsets can be mined from the FP-DB directly because whatever that has been mined previously is retained in it. Apart from this, users can specify the frequent patterns and itemsets that need to be mined from the FP-DB using the appropriate NoSQL queries in the procedures of the FP-NoSQL algorithm as follows:

#### (1) Search\_All\_Items()

- `dres = col.find().sort("Frequency DESC").limit(rpd).offset(n).execute()`
- The NoSQL query in this procedure restricts the number of frequent patterns to be displayed to the users by having the ones with a higher frequency to be shown first because these are the most interested patterns that are required by the users.

#### (2) Search\_Specific\_Item()

- `dres = col.find("Item=:param1").sort("Frequency DESC").limit(rpd).offset(n).bind("param1", item).execute()`
- The NoSQL query in this procedure displays the frequent patterns that are matched to a specific item determined by the users, so that they can focus their analysis on the frequent patterns that are related to a particular item that is of interest to them.

#### (3) Search\_Specific\_Pattern()

- `dres = col.find("Connection LIKE :param1").sort("Frequency DESC").limit(rpd).offset(n).bind("param1", con_path.c_str()).execute()`

- The NoSQL query in this procedure shows the frequent patterns that are matched to a specific pattern provided by the users. This enables the users to concentrate on analyzing the frequent patterns that consist of a particular pattern that is of interest to them.

#### (4) Mine\_Frequent\_Itemsets()

- `dres = col.find("Item = :param1").bind("param1", item).execute()`
- The NoSQL query in this procedure mines the frequent itemsets that are related to an exact item requested by the users. This helps the users to retrieve only the frequent itemsets that are relevant to a specific item for data analysis.

Therefore, since the users have flexibility to specify the relevant parts of frequent patterns and itemsets for data analysis, the memory usage for mining the frequent patterns and itemsets can be greatly reduced even though the minimum support is set to a lower value in a very large data set.

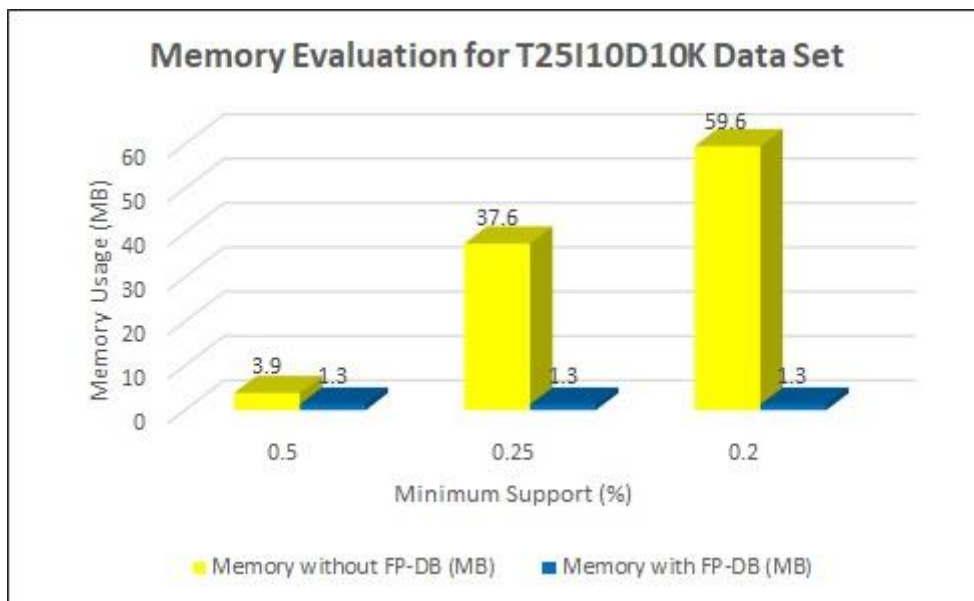


Figure 5.21: Memory Usage Evaluation for T25I10D10K Data Set

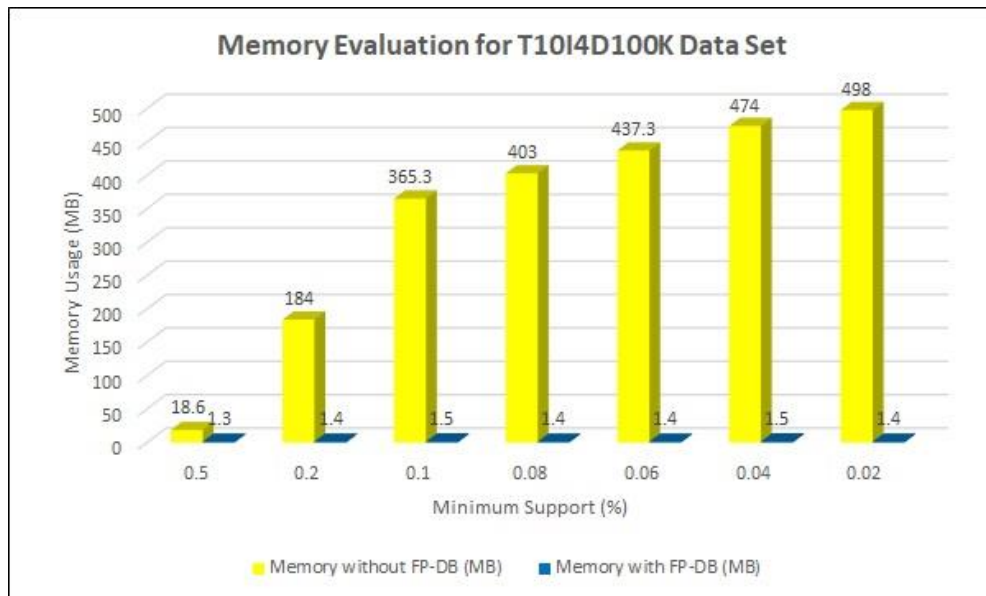


Figure 5.22: Memory Usage Evaluation for T10I4D100K Data Set

## 5.6 Evaluating FP-NoSQL with Big-O Notation

Measuring the execution time of an algorithm is not enough to determine its complexity because such metric depends upon the computer hardware being used for executing the algorithm. The problem that exists in time measurement for evaluating an algorithm is that different computers will record different values of run time. Even though the algorithm is executed in the same computer, different values of run time will be recorded when the algorithm is executed multiple times. Therefore, apart from measuring the run time for executing the algorithm, the algorithm is evaluated using a more objective analysis metrics, the Big-O Notation. The objective of this evaluation is to confirm that the FP-NoSQL algorithm has been implemented in an optimized manner to achieve a better performance in Frequent Pattern Mining (FPM).

Big-O Notation is one of the statistical measures that can be used to elaborate the complexity of an algorithm (Malik, 2019). It estimates how the run time of an algorithm grows as the size of input increases to a bigger value and allows us to eventually determine the efficiency of algorithms (Vaz, Shah, Sawhney, & Deolekar, 2017). Instead of counting the time which is so variable, the number of operations that the computer needs to perform are counted (Steele, 2019). The notation is represented

as a big “O” that is attached with a pair of opening and closing parenthesis. The connection between the input of an algorithm and the steps executed by the algorithm is represented using “n” inside the parenthesis.

The time complexity analysis with Big-O Notation is one of the most important concepts for learning how to produce efficient programming codes (Barlowe & Scott, 2015). Some of the most commonly used Big-O functions are stated in Table 5.8. In order to describe the evaluation of FP-NoSQL with the Big-O Notation, the fundamental knowledge of Big-O Notation for Constant Complexity ( $O(c)$ ), Linear Complexity ( $O(n)$ ) and Quadratic Complexity ( $O(n^2)$ ) are briefly explained from Section 5.6.1 to Section 5.6.3.

Table 5.8: Commonly Used Big-O Notations (Malik, 2019)

<b>Name</b>	<b>Big O</b>
Constant	$O(c)$
Linear	$O(n)$
Quadratic	$O(n^2)$
Cubic	$O(n^3)$
Exponential	$O(2^n)$
Logarithmic	$O(\log(n))$
Log Linear	$O(n \log(n))$

### 5.6.1 Constant Complexity – $O(c)$

The complexity of an algorithm is considered as constant if the steps required to accomplish the execution of an algorithm remain the same even though the number of inputs is changed. Constant Complexity is represented as  $O(c)$  and “c” is any object that can pass a different value to the function at different time. However, although different values are passed into the function at different times, the same number of

steps are processed by the function at every execution. For example, the following function will continue to perform two steps for printing out the square value of the first element in the array, irrespective of the input size, or the number of items in the array. Therefore, the complexity of a constant function remains constant with almost the same run time and memory consumption. An example of Constant Complexity for an algorithm is shown in Figure 5.23.

### **Big-O Function Example 1:**

```
def constant_algorithm(items):  
  
    result = items[0] * items[0]  
  
    print ()  
  
constant_algorithm([1, 2, 3, 4, 5])
```

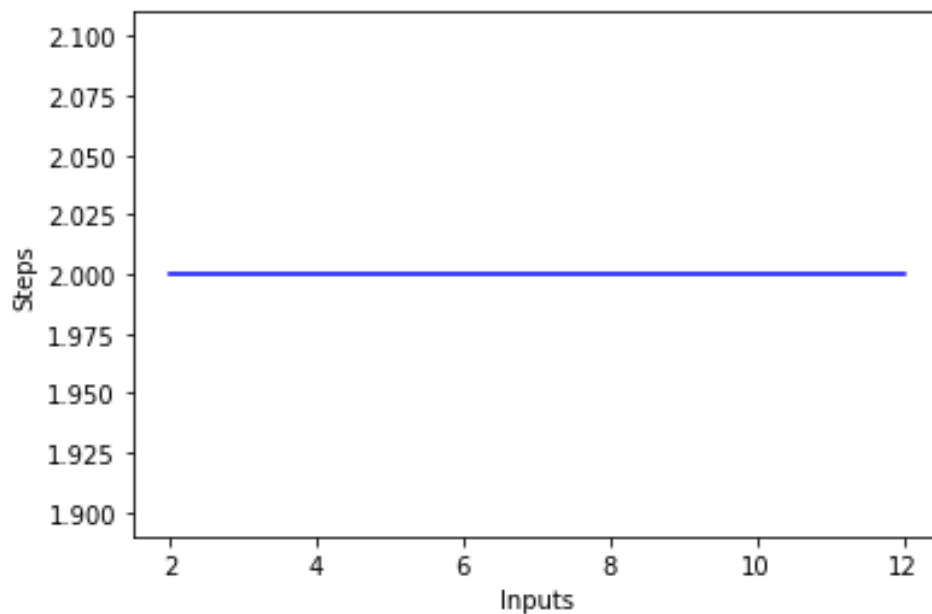


Figure 5.23: Constant Complexity (Malik, 2019)

### 5.6.2 Linear Complexity – $O(n)$

The complexity of an algorithm is considered as linear if the steps required to complete the execution of an algorithm decreases or increases in a linear manner with the number of inputs. Linear Complexity is represented as  $O(n)$  where “n” is any object that passes different values to the function at different times and causes the function to be executed with different number of iterations. For instance, the following function will increase its number of iterations for the loop if the number of items in the array is increased. Similarly, if the number of items in the array is decreased, the number of iterations for the loop is also decreased. Thus, the complexity of a linear function is having a linear relationship with the number of inputs. This causes the total amount of run time and memory consumption for the algorithm increases or decreases in a linear manner according to the number of inputs. An example of Linear Complexity for an algorithm is shown in Figure 5.24.

#### **Big-O Function Example 2:**

```
def linear_algorithm(items):  
    for item in items:  
        print(item)  
  
linear_algorithm([6, 7, 8, 9, 10])
```

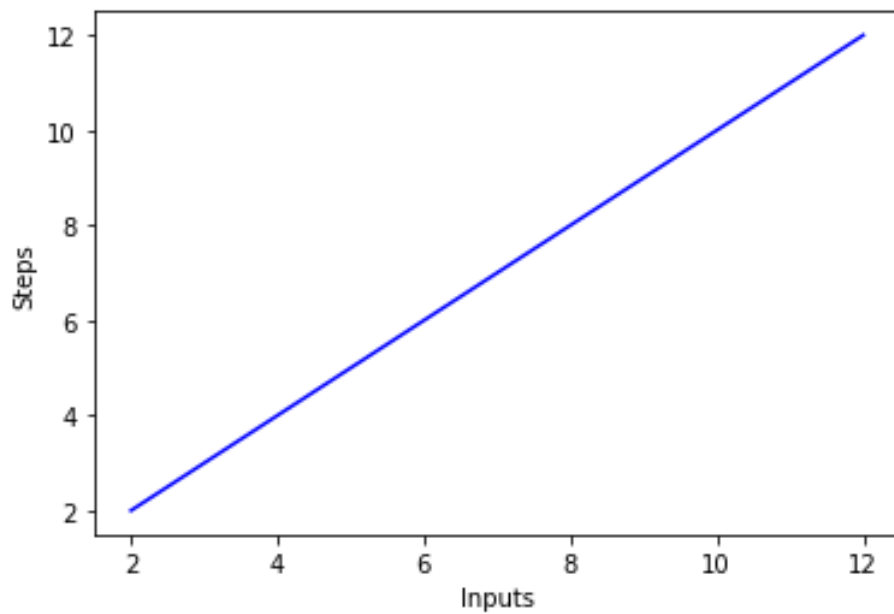


Figure 5.24: Linear Complexity (Malik, 2019)

### 5.6.3 Quadratic Complexity – $O(n^2)$

The complexity of an algorithm is considered as quadratic if the steps required to accomplish the execution of an algorithm are having a quadratic relationship with the number of inputs. Quadratic Complexity is represented as  $O(n^2)$ . Likewise, “n” is any object that passes different values to the function at different times and causes the function to be executed with different number of iterations. For example, the following function will perform  $n * n$  number of steps, where an outer loop iterates through every item of the input, followed by a nested inner loop that iterates through each item of the input again. An example of Quadratic Complexity for an algorithm is shown in Figure 5.25.

### **Big-O Function Example 3:**

```
def quadratic_algorithm(items):  
  
    for item in items:  
  
        for item2 in items:  
  
            print(item, ' ', item)  
  
quadratic_algorithm([2, 4, 6, 8, 10])
```

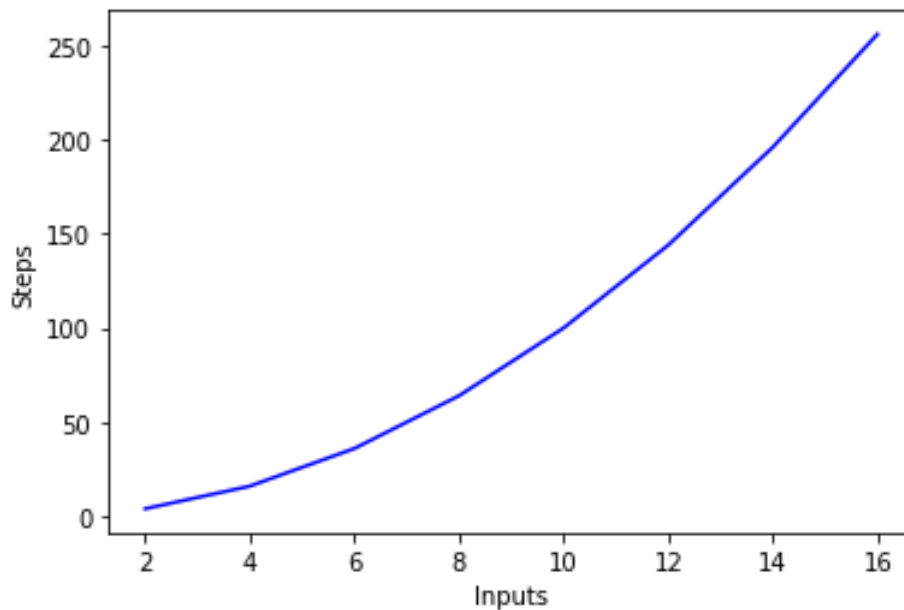


Figure 5.25: Quadratic Complexity (Malik, 2019)

### **5.6.4 Big-O Notation Analysis for FP-NoSQL**

Generally, an algorithm consists of several functions for manipulating its input. Therefore, to determine the complexity of an algorithm in a more accurate manner, different parts of the algorithm need to be identified for its complexity. Every part of



the significant FP-NoSQL algorithms stated in Chapter 4 is evaluated with the Big-O Notation as follows:

### (1) Frequent Pattern Processor

---

**Algorithm:** Construction of *Frequent Pattern* Table

**Input:** Frequent Items

**Output:** Frequent Pattern Table

---

```

1   Begin
2   Connect to Data Warehouse  O(1)
3   Select TID and Item from FrequentItem and ItemFrequency tables that
   satisfied threshold of minimum support  O(log(n))
4   While NOT End of Resultset Do  O(n)
5       If cur_id != pre_id Then  O(1)
6           con_path = "root"
7       Else
8           con_path += "~" + pre_item
9       End If
10      records += "(" + cur_item + "," + con_path + "),"  O(1)
11      pre_id = cur_id  O(1)
12      pre_item = cur_item  O(1)
13  End While
14  records = records.substr(0, records.length() - 1)  O(1)
15  Insert records into FrequentPattern table  O(log(n))
16  Disconnect from Data Warehouse  O(1)
17  End

```

---

The time complexity of the Frequent Pattern Processor algorithm is denoted as:

$$\begin{aligned}
 T(n) &= O(1) + O(\log(n)) + O(n(1 + 1 + 1 + 1)) + O(1) + O(\log(n)) + O(1) \\
 &= O(3) + O(n(4)) + 2O(\log(n)) \\
 &= O(3) + O(4n) + 2O(\log(n))
 \end{aligned}$$

$$= O(n) + O(\log(n))$$

## (2) FP-Collection Constructor

---

**Algorithm:** Construction of *Frequent Pattern Collection*

**Input:** Frequent Pattern Table

**Output:** Frequent Pattern Collection

---

```

1   Begin
2   Connect to Data Warehouse  O(1)
3   Connect to Frequent Pattern Database  O(1)
4   pat_count = 0  O(1)
5   Select pattern from FrequentPattern table  O(log(n))
6   Sort pattern in ascending order  O(n log(n))
7   While NOT End of Resultset Do  O(n)
8       pat_count++  O(1)
9       If cur_pat != next_pat Then  O(1)
10          documents += (“{“Item\”:” + item + “,\“Frequency\”:” +
pat_count + “,\“Connection\”:“” + con_path + “\”}”),”
11          pat_count = 0
12      End If
13  End While
14  documents = documents.substr(0, documents.length() - 1)  O(1)
15  Insert documents into FP-Collection  O(log(n))
16  Disconnect from Data Warehouse  O(1)
17  Disconnect from Frequent Pattern Database  O(1)
18  End

```

---

The time complexity of the FP-Collection algorithm is denoted as:

$$\begin{aligned}
 T(n) &= O(1 + 1 + 1) + O(\log(n)) + O(n \log(n)) + O(n(1 + 1)) + O(1) + \\
 &\quad O(\log(n)) + O(1 + 1) \\
 &= O(6) + O(n(2)) + 2O(\log(n)) + O(n \log(n)) \\
 &= O(6) + O(2n) + 2O(\log(n)) + O(n \log(n))
 \end{aligned}$$

$$= O(n) + O(\log(n)) + O(n \log(n))$$

When an algorithm is evaluated with the Big-O Notation, the output is always expected to be having a smooth line or a curve with minor or static slope (Devi, Selvam, & Rajagopalan, 2011). As both of the Frequent Pattern Processor and FP-Collection Constructor algorithms are evaluated with the Big-O Notation, most parts have a constant time complexity ( $O(1)$ ), whereas the parts that manipulate the data have the linear ( $O(n)$ ), logarithmic ( $O(\log(n))$ ), and log linear ( $O(n \log(n))$ ) time complexities.

For the Frequent Pattern Processor algorithm, the parts that consist of a constant time complexity ( $O(1)$ ) are the operations for connecting to the data warehouse, generating the string of records to be inserted, and disconnecting from the data warehouse. Selecting the items that satisfied the threshold of minimum support from the data warehouse has the logarithmic time complexity ( $O(\log(n))$ ) because the database table has been indexed. Inserting the records into the frequent pattern table also has the logarithmic time complexity ( $O(\log(n))$ ) because all the records are inserted into the database together at the same time. The part that consists of a linear time complexity ( $O(n)$ ) is the operation for constructing the frequent patterns. In the Frequent Pattern Processor algorithm, the  $O(n)$  operation consists of four  $O(1)$  operations that determine how the frequent patterns should be constructed by verifying the Transaction IDs and Item IDs of the data.

For the FP-Collection Constructor algorithm, the parts that consist of a constant time complexity ( $O(1)$ ) are the operations for connecting to the data warehouse and database, generating the string of documents to be inserted, and disconnecting from the data warehouse and database. Selecting the frequent patterns from the database table has the logarithmic time complexity ( $O(\log(n))$ ) because the database table has been indexed. As the frequent patterns are sorted into the ascending order, it has the log linear ( $O(n \log(n))$ ) time complexity. Similarly, inserting the documents into the frequent pattern collection has the logarithmic time complexity ( $O(\log(n))$ ) because all the documents are inserted into the data warehouse together at the same time. The

part that consists of a linear time complexity ( $O(n)$ ) is the operation for counting the frequent patterns and constructing its collection. In the FP-Collection algorithm, the  $O(n)$  operation consists of two  $O(1)$  operations that count and construct the documents of frequent patterns.

The overall performance of any algorithm evaluated with the Big-O Notation can be represented in various forms as shown in Figure 5.26 (Miller & Ranum, 2013). Generally, the growth of performance for an algorithm can be denoted as logarithmic, linear, log linear, quadratic, cubic and exponential. When the number of inputs is increased, the algorithms that possess an exponential growth for its performance are considered as the worst implemented algorithms. This is because a minor increase for the inputs will cause a major increase for the steps required to complete the execution of the algorithms in processing all the data. On the other hand, when the number of inputs is increased, the algorithms that possess a logarithmic growth for its performance are considered as the best implemented algorithms. This is because a major increase for the inputs will only cause a minor increase for the steps required to complete the execution of the algorithms in processing all the data.

The total time complexity of the Frequent Pattern Processor algorithm is evaluated to  $O(3) + O(4n) + 2O(\log(n))$ , while the total time complexity of the FP-Collection Constructor algorithm is evaluated to  $O(6) + O(2n) + 2O(\log(n)) + O(n \log(n))$ . However, both of the total time complexities of the two algorithms can be evaluated to  $O(n) + O(\log(n))$  and  $O(n) + O(\log(n)) + O(n \log(n))$  respectively. This is because when the size of inputs ( $n$ ) grows to a bigger value, the constants will become less and less important to the ultimate result. Since the FP-NoSQL algorithm is evaluated to having a linear, logarithmic or log linear time complexity relationship with its input, it can be considered as a robust algorithm for mining the frequent itemsets of data.

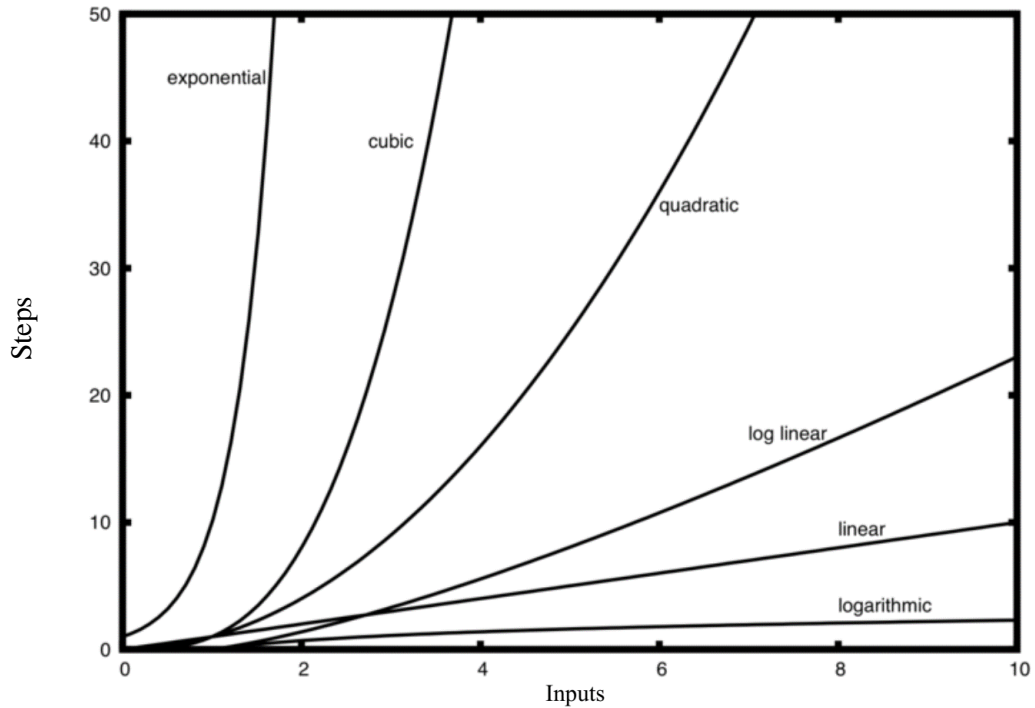


Figure 5.26: Plot of Common Big-O Functions (Miller & Ranum, 2013)

## 5.7 Chapter Summary

The FP-NoSQL algorithm is evaluated with three experiments in this research. First and foremost, four different sets of data are used to evaluate the algorithm in order to ensure that it is able to mine any kind of data set, whether it is a sparse one or a dense one. Then, another two additional sets of data are used to evaluate the algorithm so that it can be confirmed that the FP-NoSQL algorithm is able to outperform the Apriori and Extended Frequent Pattern (EFP) algorithms, which are the existing significant algorithms for Frequent Pattern Mining (FPM). Finally, the Big-O Notation is used to evaluate the FP-NoSQL algorithm in order to determine that it has been implemented in an optimized manner to achieve a better performance for mining the frequent itemsets of data. All the three experiments produced reasonable results that confirm FP-NoSQL as a suitable algorithm to be used in the area of FPM.



## CHAPTER 6

### CONCLUSION

This chapter concludes the entire work of research with a brief summary and highlights the contribution and possible work of enhancement in the near future. In Section 6.1, a summary of the research is given by addressing the research questions and research objectives together as a whole. Then, Section 6.2 describes the contribution and Section 6.3 explains the limitation of this research. Last but not least, Section 6.4 provides some possible directions for the work of this research to be further enhanced by anyone who is interested in conducting research for Frequent Pattern Mining.

#### **6.1 Summary of Research**

There are three major research questions to be addressed in this work and they are summarized into one as follows for easier discussion:

*How can the frequent patterns and itemsets be mined efficiently within a shorter run time using lesser memory consumption and be retained for further analysis even though the amount of data is big in a data set?*

In order to retain the frequent patterns that have been mined for further analysis, a Frequent Pattern Database (FP-DB) needs to be utilized for executing the process of Frequent Pattern Mining (FPM). The FP-DB is used for storing all the patterns that can be found in a data warehouse. Each unique pattern of any item is stored into the FP-DB along with its frequency of occurrence from the data warehouse. When every pattern is consolidated from the data warehouse into a Frequent Pattern Collection (FP-Collection) in the FP-DB, the frequent patterns that have been mined can be retained for further analysis even after a power failure. This is because the frequent

patterns that have been mined by the FP-NoSQL algorithm are not only stored in the Random Access Memory (RAM) but also in the database. Since the frequent patterns are stored in the FP-DB, it can be retrieved anytime from the database whenever there is a need to perform analysis on a particular set of data.

Then, for mining the frequent patterns within a shorter run time, every pattern that exists frequently among the transactions of the data warehouse is concatenated together to be inserted into the *FrequentPattern* table at the same time. The frequent patterns are processed in such a manner so that the data warehouse does not need to be scanned several times while the frequent patterns are stored into the FP-Collection of the FP-DB. As the FP-DB is not scanned multiple times when the frequent patterns are inserted into it, the time required to process all the frequent patterns have been greatly reduced. It is so because scanning through the database frequently for performing any operation of data manipulation is very costly. Apart from this, to further reduce the time for mining frequent patterns from the data warehouse, each frequent pattern is constructed directly from the *FrequentItem* table into the *FrequentPattern* table. After the whole *FrequentPattern* table is built, all the frequent patterns are sorted in the ascending order so that it can be quickly counted for storing into the FP-Collection in the FP-DB.

Last but not least, mining the frequent itemsets using lesser memory consumption is made possible with the FP-DB. This is because every unique pattern that exists frequently in the data warehouse is stored in the FP-DB together with its frequency of occurrence among all the transactions of data. Therefore, it is not necessary to mine the entire data warehouse in order to discover the frequent itemsets that are related to a specific item. Since every unique pattern that exists frequently in the data warehouse has already been mined into the FP-DB, the FP-NoSQL algorithm is only required to search through the FP-DB for frequent patterns that matched the requirements of users so that the relevant frequent itemsets can be generated easily. Hence, the memory usage for mining the frequent itemsets is significantly reduced as the amount of data to be retrieved from the FP-DB has been set appropriately with the relevant NoSQL query in the algorithm.



## 6.2 Research Contribution

Through this research, the following contributions have been accomplished:

- (1) The frequent patterns that have been mined by the current algorithms are mainly retained in the Random Access Memory (RAM). When the system is down or there is a power failure, none of the frequent patterns can be retained for data analysis. In this research, the frequent patterns that have been mined can be retained for data analysis using a Frequent Pattern Database (FP-DB). This is because the frequent patterns and their support counts are stored into a Frequent Pattern Collection (FP-Collection) of the FP-DB and it can be retrieved again even after the system is down or a power failure. Therefore, the entire data warehouse is not required to be mined again in order to construct all the frequent patterns and its support counts. Since the FP-Collection is storing the frequent patterns and their support counts, data analysis can be conducted more often at anytime.
- (2) The frequent itemsets that appear in a data set need to be mined with a longer run time by the current algorithms especially if the amount of data is big. In this research, the frequent itemsets can be mined within a shorter run time although the amount of data is big in the data warehouse. This is because every pattern that exists frequently in the data warehouse is inserted simultaneously into a frequent pattern table after they are being concatenated together. In this way, it helps to decrease the number of times for scanning the data warehouse. Hence, the total run time for constructing all the frequent patterns can be greatly reduced. As the total run time is greatly reduced, more data can be processed to identify the hidden patterns for decision making in the organization.
- (3) The frequent itemsets that appear in a data set need to be mined with more memory consumption by the current algorithms especially if the amount of data is big. In this research, the frequent itemsets can be mined with lesser memory usage even though the amount of data is big in the data warehouse. This is because the frequent patterns and their support counts can be located selectively from the FP-Collection of the FP-DB using the appropriate NoSQL queries. Thus, with the FP-Collection, only the frequent patterns that matched the requirements of the users are needed to be mined to produce the frequent itemsets. At the same time, this helps users to focus their analysis on the data that is of interest to them.

### **6.3 Research Limitation**

Although the FP-NoSQL algorithm is able to mine the frequent itemsets within a shorter run time with lesser memory usage, even if the amount of data is big in the data warehouse, there are still some limitations that are faced by the algorithm. One of the limitations in this research is where the entire FP-DB needs to be reconstructed if there are changes towards the frequency of the patterns. This is because the results of frequent itemset mining will no longer be accurate anymore as long as there are changes towards the frequency of any pattern. In addition, the entire FP-DB also needs to be reconstructed if some frequent patterns are no longer significant in the data warehouse and need to be removed. Likewise, the results of frequent itemset mining will not be accurate as well whenever the frequency of any pattern is changed, whether it has been increased or decreased.

### **6.4 Direction for Future Work**

In this research, the FP-NoSQL algorithm is designed and developed to perform the work of Frequent Pattern Mining (FPM) using one unit of computer only. To further improve the mining operations of FPM, it is still possible to reduce additional speed and memory usage for the algorithm in mining the frequent itemsets of data. Some of the methods that can be utilized to enhance the performance of the FP-NoSQL algorithm are Parallel Computing and Distributed Computing. Parallel Computing is a kind of processing in which many executions of processes are accomplished at the same time so that a computational problem can be resolved in lesser time and with greater accuracy (Kaminsky, 2016). It can be implemented within a single machine if there is a multi-core processor, or across multiple machines that are networked together to process the same task. In Parallel Computing, huge computational problems are separated into tiny ones so that it can be fixed simultaneously especially if the problems are very complicated. After a computational task is broken down into multiple subtasks to be processed individually, the final results are to be combined upon accomplishment of the entire process. Since Parallel Computing is able to speed up the processing of a computational task, it is definitely

more useful to enhance the FP-NoSQL algorithm with the capability of parallelism in mining the frequent itemsets of data. On the other hand, Distributed Computing is a type of processing where multiple components are situated on different computers that are connected together, in which communication and coordination of actions are performed by exchanging messages with one another (Coulouris, Dollimore, & Kindberg, 2011). Similarly, in Distributed Computing, a computational problem is separated into many subtasks, having each task to be solved by one or more computers. However, every processor in Distributed Computing has its own memory and data is exchanged with one another by sending messages between the processors; whereas in Parallel Computing, all processors are usually accessing a shared memory to exchange data between different processors (Papadimitriou, 1994). Since Distributed Computing consists of additional memory processing power in its hardware architecture, it is surely more beneficial to improve FP-NoSQL as a distributed algorithm in mining the frequent itemsets of data.

Apart from this, one more improvement that can be done for the FP-NoSQL algorithm is to enable the support counts of the frequent patterns to be updated without reconstructing the entire FP-DB when the frequent patterns are mined in more than one FPM process. If the support count of every frequent pattern can be updated without reconstructing the entire FP-DB, then the run time required to mine the frequent itemsets from the FP-DB can be greatly reduced. In order to achieve this capability, the relevant NoSQL queries and algorithms are required to be designed for identifying the appropriate frequent patterns where their support counts are needed to be updated. Once the relevant frequent patterns are identified, their support counts are required to be increased or decreased accordingly.

Furthermore, another improvement that can be implemented for the FP-NoSQL algorithm is to allow the frequent patterns that are no longer significant in the data warehouse to be removed from the FP-DB without reconstructing the entire FP-DB. If the unnecessary frequent patterns can be removed without reconstructing the entire FP-DB, then additional space can be allocated to store the more significant frequent patterns that are required to be mined. At the same time, this also helps to speed up

the mining process for the frequent itemsets since the search space of the frequent patterns has been reduced to a smaller area.

Finally, if all these techniques can be implemented successfully into the algorithm, the time and memory required to mine the frequent itemsets of data can be significantly reduced even though the amount of data is big and continue to increase in the data warehouse. This is because a huge data set can be divided into smaller data sets so that it can be distributed to multiple different computers in the network for mining the frequent itemsets in parallel. Performing continuous research in the area of data analytics is undeniably important and of great value since it is a capability that helps to optimize various processes of an organization in any industry like electric-power (Qing, Boyu, & Qinqian, 2017), oil and gas (Alguliyev, Aliguliyev, & Hajirahimova, 2016), restaurant (Mattera, 2018), railway engineering (Attoh-Okine, 2014), healthcare (Patil & Seshadri, 2014), media and entertainment (Suri & Singh, 2018), and finance (H. Zhang, Li, Shen, Sun, & Yang, 2015).

## REFERENCES

- [1] Abraham, S., & Joseph, S. (2016). A Coherent Rule Mining Method for Incremental Datasets based on Plausibility. *Procedia Technology*, 24, 1292-1299.
- [2] Agarwal, R. C., Aggarwal, C. C., & Prasad, V. V. V. (2000). *Depth First Generation of Long Patterns*. Paper presented at the Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Boston, Massachusetts, USA.
- [3] Agarwal, R. C., Aggarwal, C. C., & Prasad, V. V. V. (2001). A Tree Projection Algorithm for Generation of Frequent Item Sets. *Journal of Parallel and Distributed Computing*, 61(3), 350-371.
- [4] Aggarwal, A., & Toshniwal, D. (2018). *Spatio-Temporal Frequent Itemset Mining on Web Data*. Paper presented at the 2018 IEEE International Conference on Data Mining Workshops (ICDMW), Singapore.
- [5] Aggarwal, C. C. (2014a). Applications of Frequent Pattern Mining. In C. C. Aggarwal & J. Han (Eds.), *Frequent Pattern Mining* (pp. 443-461). Switzerland: Springer.
- [6] Aggarwal, C. C. (2014b). An Introduction to Frequent Pattern Mining. In C. C. Aggarwal & J. Han (Eds.), *Frequent Pattern Mining* (pp. 1-14). Switzerland: Springer.
- [7] Aggarwal, C. C., Bhuiyan, M. A., & Hasan, M. A. (2014). Frequent Pattern Mining Algorithms: A Survey. In C. C. Aggarwal & J. Han (Eds.), *Frequent Pattern Mining* (pp. 19-64). Switzerland: Springer.
- [8] Agrawal, R., & Srikant, R. (1994). *Fast Algorithms for Mining Association Rules*. Paper presented at the Proceedings of the 20th International Conference on Very Large Data Bases, Santiago, Chile.
- [9] Alguliyev, R. M., Aliguliyev, R. M., & Hajirahimova, M. S. (2016). *Big Data Integration Architectural Concepts for Oil and Gas Industry*. Paper presented at the 2016 IEEE 10th International Conference on Application of Information and Communication Technologies (AICT), Baku, Azerbaijan.

- [10] Apiletti, D., Baralis, E., Cerquitelli, T., Garza, P., Pulvirenti, F., & Venturini, L. (2017). Frequent Itemsets Mining for Big Data: A Comparative Analysis. *Big Data Research*, 9, 67-83.
- [11] Asuncion, A., & Newman, D. (2007). UCI Machine Learning Repository. Retrieved from <https://archive.ics.uci.edu/ml/index.php>
- [12] Attoh-Okine, N. (2014). *Big Data Challenges in Railway Engineering*. Paper presented at the 2014 IEEE International Conference on Big Data (Big Data), Washington, DC, USA.
- [13] Ayash, E. M. M. (2014). Research Methodologies in Computer Science and Information Systems.
- [14] Bansal, R., Gaur, N., & Narayan, S. (2016). *Outlier Detection: Applications and Techniques in Data Mining*. Paper presented at the 2016 6th International Conference - Cloud System and Big Data Engineering (Confluence), Noida, India.
- [15] Baralis, E., Cerquitelli, T., Chiusano, S., & Grand, A. (2013). *P-Mine: Parallel Itemset Mining on Large Datasets*. Paper presented at the 2013 IEEE 29th International Conference on Data Engineering Workshops (ICDEW), Brisbane, Australia.
- [16] Barlowe, S., & Scott, A. (2015). *O-Charts: Towards an Effective Toolkit for Teaching Time Complexity*. Paper presented at the 2015 IEEE Frontiers in Education Conference (FIE), El Paso, TX, USA.
- [17] Bayardo, R. (2007a). *Connect Data Set*. Retrieved from: <http://fimi.ua.ac.be/data/>
- [18] Bayardo, R. (2007b). *Pumsb Data Set*. Retrieved from: <http://fimi.ua.ac.be/data/>
- [19] Bhogal, J., & Choksi, I. (2015). *Handling Big Data using NoSQL*. Paper presented at the 2015 IEEE 29th International Conference on Advanced Information Networking and Applications Workshops, Gwangju, South Korea.
- [20] Bodon, F. (2003). *Kosarak Data Set*. Retrieved from: <http://fimi.ua.ac.be/data/>
- [21] Brijs, T. (1999). *Retail Data Set*. Retrieved from: <http://fimi.ua.ac.be/data/>
- [22] Brijs, T., Swinnen, G., Vanhoof, K., & Wets, G. (1999). *The Use of Association Rules for Product Assortment Decisions: A Case Study*. Paper

- presented at the KDD '99 Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Diego, California, USA.
- [23] Cao, J., Diao, X., Jiang, G., & Du, Y. (2010). *Data Lifecycle Process Model and Quality Improving Framework for TDQM Practices*. Paper presented at the 2010 International Conference on E-Product, E-Service and E-Entertainment, Henan.
- [24] Chang, V. (2014). The Business Intelligence as a Service in the Cloud. *Future Generation Computer Systems*, 37, 512-534.
- [25] Chaudhari, A. A., & Khanuja, H. K. (2015). *Database Transformation to Build Data-Set for Data Mining Analysis - A Review*. Paper presented at the 2015 International Conference on Computing Communication Control and Automation, Pune, India.
- [26] Chaure, T. M., & Singh, K. R. (2016). *Frequent Itemset Mining Techniques - A Technical Review*. Paper presented at the 2016 World Conference on Futuristic Trends in Research and Innovation for Social Welfare (WCFTR'16), Coimbatore, India.
- [27] Chen, F., Deng, P., Wan, J., Zhang, D., Vasilakos, A. V., & Rong, X. (2015). Data Mining for the Internet of Things: Literature Review and Challenges. *International Journal of Distributed Sensor Networks*, 2015, 1-14.
- [28] Chon, K.-W., Hwang, S.-H., & Kim, M.-S. (2018). GMiner: A Fast GPU-Based Frequent Itemset Mining Method for Large-Scale Data. *Information Sciences*, 439–440, 19-38.
- [29] Columbus, L. (2017, 13 May 2017). IBM Predicts Demand For Data Scientists Will Soar 28% By 2020. Retrieved from <https://www.forbes.com/sites/louiscolumbus/2017/05/13/ibm-predicts-demand-for-data-scientists-will-soar-28-by-2020/#57e8b54b7e3b>
- [30] Coulouris, G., Dollimore, J., & Kindberg, T. (2011). *Distributed Systems: Concepts and Design (5th Edition)*: Addison-Wesley.
- [31] CPlusPlus.com. (2018). C++ Language. Retrieved from <http://www.cplusplus.com/doc/tutorial/>

- [32] Crnkovic, G. D. (2002). *Scientific Methods in Computer Science*. Paper presented at the Proceedings of the Conference for the Promotion of Research in IT, Sweden, Skövde, Suecia.
- [33] Dave, K., Rathod, M., Sheth, P., & Sakhapara, A. (2015). Comparing the Performance of Frequent Itemsets Mining Algorithms. *International Journal of Innovative Research in Computer and Communication Engineering*, 3(3), 1968-1972.
- [34] Demeyer, S. (2011). *Research Methods in Computer Science*. Paper presented at the 2011 27th IEEE International Conference on Software Maintenance (ICSM), Williamsburg, VI, USA.
- [35] Devi, S. G., Selvam, K., & Rajagopalan, S. P. (2011). *An Abstract to Calculate Big O Factors of Time and Space Complexity of Machine Code*. Paper presented at the International Conference on Sustainable Energy and Intelligent Systems (SEISCON 2011), Chennai, India.
- [36] Djenouri, Y., Djenouri, D., Lin, J. C.-W., & Belhadi, A. (2018). Frequent Itemset Mining in Big Data With Effective Single Scan Algorithms. *IEEE Access*, 6, 68013-68026.
- [37] El-Hajj, M., & Zaiane, O. R. (2003). *COFI-Tree Mining - A New Approach to Pattern Growth with Reduced Candidacy Generation*. Paper presented at the Workshop on Frequent Itemset Mining Implementations (FIMI'03) in conjunction with IEEE-ICDM, Melbourne, Florida, USA.
- [38] Ellis, T. J., & Levy, Y. (2008). Framework of Problem-Based Research: A Guide for Novice Researchers on the Development of a Research-Worthy Problem. *Informing Science: The International Journal of an Emerging Transdiscipline*, 11, 17-33.
- [39] Endler, G. (2012). *Data Quality and Integration in Collaborative Environments*. Paper presented at the SIGMOD/PODS 2012 PhD Symposium, Scottsdale, AZ, USA.
- [40] Feddaoui, I., Felhi, F., & Akaichi, J. (2016). *EXTRACT: New Extraction Algorithm of Association Rules from Frequent Itemsets*. Paper presented at the 2016 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM), San Francisco, CA, USA.



- [41] Fernando, B., Fromont, E., & Tuytelaars, T. (2012). *Effective Use of Frequent Itemset Mining for Image Classification*. Paper presented at the 12th European Conference on Computer Vision, Florence, Italy.
- [42] Fisher, C., Lauria, E., Chengalur-Smith, S., & Wang, R. (2012). *Introduction to Information Quality*. Bloomington: AuthorHouse.
- [43] Garg, D., & Sharma, H. (2011). Comparative Analysis of Various Approaches Used in Frequent Pattern Mining. *International Journal of Advanced Computer Science and Applications (IJACSA)*, 141-147.
- [44] Giacometti, A., Li, D. H., Marcel, P., & Soulet, A. (2014). 20 Years of Pattern Mining: A Bibliometric Survey. *ACM SIGKDD Explorations Newsletter*, 15(1), 41-50.
- [45] Goethals, B. (2004). Frequent Itemset Mining Implementations Repository.
- [46] Gull, M., & Pervaiz, A. (2018). *Customer Behavior Analysis Towards Online Shopping using Data Mining*. Paper presented at the 2018 5th International Multi-Topic ICT Conference (IMTIC), Jamshoro, Pakistan.
- [47] Gullo, F. (2015). From Patterns in Data to Knowledge Discovery: What Data Mining Can Do. *Physics Procedia* 62, 18-22.
- [48] Gupta, B., & Garg, D. (2011). FP-Tree Based Algorithms Analysis FPGrowth, COFI-Tree and CT-PRO. *International Journal on Computer Science and Engineering*, 3(7), 2691-2699.
- [49] Hage, L. E. (2017). *Driving Value with Retail POS Data: Using Dayparts and Affinity Analytics to Deliver Enhanced Insights and Higher ROIs*.
- [50] Han, J., Kamber, M., & Pei, J. (2012a). Classification: Basic Concepts. In *Data Mining Concepts and Techniques* (pp. 327-392). USA: Elsevier.
- [51] Han, J., Kamber, M., & Pei, J. (2012b). Cluster Analysis: Basic Concepts and Methods. In *Data Mining Concepts and Techniques* (pp. 443-496). USA: Elsevier.
- [52] Han, J., Kamber, M., & Pei, J. (2012c). Mining Frequent Patterns, Associations, and Correlations: Basic Concepts and Methods. In *Data Mining Concepts and Techniques*. USA: Elsevier.
- [53] Han, J., Kamber, M., & Pei, J. (2012d). Outlier Detection. In *Data Mining Concepts and Techniques* (pp. 543-584). USA: Elsevier.

- [54] Han, J., Kamber, M., & Pei, J. (2012e). What Kinds of Patterns can be Mined? In *Data Mining Concepts and Techniques* (pp. 15-22). USA: Elsevier.
- [55] Han, J., Kamber, M., & Pei, J. (2012f). Why Data Mining? In *Data Mining Concepts and Techniques* (pp. 1-8). USA: Elsevier.
- [56] Han, J., & Pei, J. (2000). Mining Frequent Patterns by Pattern-Growth: Methodology and Implications. *ACM SIGKDD Explorations Newsletter - Special issue on "Scalable Data Mining Algorithms"*, 2(2), 14-20.
- [57] Han, J., Pei, J., & Yin, Y. (2000). Mining Frequent Patterns without Candidate Generation. *ACM SIGMOD Record*, 29(2), 1-12.
- [58] Han, X., Liu, X., Chen, J., Lai, G., Gao, H., & Li, J. (2019). Efficiently Mining Frequent Itemsets on Massive Data. *IEEE Access*, 7, 31409-31421.
- [59] Hashem, I. A. T., Yaqoob, I., Anuar, N. B., Mokhtar, S., Gani, A., & Khan, S. U. (2015). The Rise of “Big Data” on Cloud Computing: Review and Open Research Issues. *Information Systems*, 47, 98-115.
- [60] Hassani, H. (2017). Research Methods in Computer Science: The Challenges and Issues. *ArXiv*, 1-16.
- [61] Haupt, R., Scholtz, B., & Calitz, A. (2015). *Using Business Intelligence to Support Strategic Sustainability Information Management*. Paper presented at the 2015 Annual Research Conference on South African Institute of Computer Scientists and Information Technologists, Stellenbosch, South Africa.
- [62] Heller, M. (2017, 28 August 2017). What is SQL? Structured Query Language Explained. Retrieved from <https://www.infoworld.com/article/3219795/sql/what-is-sql-structured-query-language-explained.html>
- [63] Hemalatha, R., Krishnan, A., Senthamarai, C., & Hemamilini, R. (2005). *Frequent Pattern Discovery Based on Co-Occurrence Frequent Item Tree*. Paper presented at the Proceedings of 2005 International Conference on Intelligent Sensing and Information Processing, Chennai, India.
- [64] Henke, N., Bughin, J., Chui, M., Manyika, J., Saleh, T., Wiseman, B., & Sethupathy, G. (2016). *The Age of Analytics: Competing in a Data-Driven World*. Retrieved from

- [65] Hoseini, M. S., Shahraki, M. N., & Neysiani, B. S. (2015). *A New Algorithm for Mining Frequent Patterns in CanTree*. Paper presented at the International Conference on Knowledge-Based Engineering and Innovation, Tehran, Iran.
- [66] Hruschka, E. R., Campello, R. J. G. B., Freitas, A. A., & Carvalho, A. C. P. L. F. d. (2009). A Survey of Evolutionary Algorithms for Clustering. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 39(2), 133-155.
- [67] Inteliment. (2016, 12 August 2016). What's The Difference Between Data Analytics and Data Analysis? Retrieved from <https://www.inteliment.com/whats-the-difference-between-data-analytics-and-data-analysis/>
- [68] Jamsheela, O., & G., R. (2015). *Frequent Itemset Mining Algorithms: A Literature Survey*. Paper presented at the 2015 IEEE International Advance Computing Conference (IACC), Bangalore, India.
- [69] JayaramHariharakrishnan, Mohanavalli, S., Srividya, & Kumar, K. B. S. (2017). *Survey of Pre-processing Techniques for Mining Big Data*. Paper presented at the IEEE International Conference on Computer, Communication, and Signal Processing Chennai, India.
- [70] Jeevarathinam, R., & Thanamani, A. S. (2009). *Transaction Mapping Based Approach for Mining Software Specifications*. Paper presented at the 2009 World Congress on Nature & Biologically Inspired Computing (NaBIC), Coimbatore, India.
- [71] Jenkins, A. M. (1985). Research Methodologies And MIS Research. In *Research Methods in Information Systems* (pp. 97-109). Amsterdam: Elsevier Science Publishers.
- [72] Jesus, E., & Bernardino, J. (2014). *Open Source Business Intelligence in Manufacturing*. Paper presented at the 18th International Database Engineering & Applications Symposium, Porto, Portugal.
- [73] Jiang, H., & He, X. (2017). *An Improved Algorithm for Frequent Itemsets Mining*. Paper presented at the 2017 Fifth International Conference on Advanced Cloud and Big Data (CBD), Shanghai, China.

- [74] Jörg, T., & DeBloch, S. (2008). *Towards Generating ETL Processes for Incremental Loading*. Paper presented at the 2008 International Symposium on Database Engineering & Applications, Coimbra, Portugal.
- [75] Kabiri, A., & Chiadmi, D. (2013). Survey on ETL Processes. *Journal of Theoretical and Applied Information Technology*, 54(2), 219-229.
- [76] Kachhadiya, B. C., & Patel, B. (2018). *A Survey on Sequential Pattern Mining Algorithm for Web Log Pattern Data*. Paper presented at the 2018 2nd International Conference on Trends in Electronics and Informatics (ICOEI), Tirunelveli, India.
- [77] Kaminsky, A. (2016). *Big CPU, Big Data: Solving the World's Toughest Computational Problems with Parallel Computing*: CreateSpace Independent Publishing Platform.
- [78] Kherdekar, V. A., & Metkewar, P. S. (2016). A Technical Comprehensive Survey of ETL Tools. *International Journal of Applied Engineering Research*, 11(4), 2557-2559.
- [79] Kim, G.-W., Lee, S. H., Kim, J. H., & Son, J. H. (2010). *An Effective Algorithm for Business Process Mining Based on Modified FP-Tree Algorithm*. Paper presented at the 2010 Second International Conference on Communication Software and Networks, Singapore.
- [80] Kuramochi, M., & Karypis, G. (2001). *Frequent Subgraph Discovery*. Paper presented at the Proceedings 2001 IEEE International Conference on Data Mining, San Jose, CA, USA.
- [81] Le, T.-D. B., & Lo, D. (2015). *Beyond Support and Confidence: Exploring Interestingness Measures for Rule-Based Specification Mining*. Paper presented at the 2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER), Montreal, QC, Canada.
- [82] Lira, W. P., Alves, R., Costa, J. M. R., Pessin, G., Galvão, L., Cardoso, A. C., & Souza, C. R. B. d. (2014). A Visual-Analytics System for Railway Safety Management. *IEEE Computer Graphics and Applications*, 34(5), 52-57.
- [83] Liu, C., Yan, X., Yu, H., Han, J., & Yu, P. S. (2005). *Mining Behavior Graphs for "Backtrace" of Noncrashing Bugs*. Paper presented at the Proceedings of

- the 2005 SIAM International Conference on Data Mining, Newport Beach, USA.
- [84] Liu, S., Cui, W., Wu, Y., & Liu, M. (2014). A Survey on Information Visualization: Recent Advances and Challenges. *The Visual Computer*, 30(12), 1373-1393.
- [85] Lobel, J. (2014). Discover Deep Insights with Frequent Pattern Mining (FPM). Retrieved from <http://www.swiftiq.com/blog/discover-deep-insights-with-frequent-pattern-mining-fpm>
- [86] Malik, U. (2019, 21 January 2019). Big O Notation and Algorithm Analysis with Python Examples. Retrieved from <https://stackabuse.com/big-o-notation-and-algorithm-analysis-with-python-examples/>
- [87] Manyika, J., Chui, M., Brown, B., Bughin, J., Dobbs, R., Roxburgh, C., & Byers, A. H. (2011). *Big Data: The Next Frontier for Innovation, Competition, and Productivity*. Retrieved from
- [88] Marinov, M., Georgiev, G., & Popova, E. (2018). *NoSQL Approach for Sensor Data Storage and Retrieval*. Paper presented at the 2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), Opatija, Croatia.
- [89] Mattera, M. (2018). *SMEs Transformation through Usage and Understanding of Big Data*. Paper presented at the 2018 IEEE 3rd International Conference on Big Data Analysis (ICBDA), Shanghai, China.
- [90] McGlothlin, J. P., & Khan, L. (2013). *Managing Evolving Code Sets and Integration of Multiple Data Sources in Health Care Analytics*. Paper presented at the 2013 International Workshop on Data Management & Analytics for Healthcare, San Francisco, CA, USA.
- [91] Meenakshi, A. (2015). Survey of Frequent Pattern Mining Algorithms in Horizontal and Vertical Data Layouts. *International Journal of Advances in Computer Science and Technology*, 4(4), 48-58.
- [92] Microsoft. (2018a). Console Applications in Visual C++. Retrieved from <https://msdn.microsoft.com/en-us/library/hh875011.aspx>
- [93] Microsoft. (2018b). Microsoft Visual Studio. Retrieved from <https://visualstudio.microsoft.com>

- [94] Microsoft. (2018c). Office 365 for Home. Retrieved from <https://products.office.com/en-us/explore-office-for-home>
- [95] Midha, N., & Singh, V. (2015). A Survey on Classification Techniques in Data Mining. *International Journal of Computer Science & Management Studies (IJCSMS)*, 16(1), 9-12.
- [96] Miller, B., & Ranum, D. (2013). *Problem Solving with Algorithms and Data Structures using Python*.
- [97] Mittal, A., Nagar, A., Gupta, K., & Nahar, R. (2015). Comparative Study of Various Frequent Pattern Mining Algorithms. *International Journal of Advanced Research in Computer and Communication Engineering*, 4(4), 550-553.
- [98] MySQL. (2018a). *MySQL 5.7 Reference Manual*.
- [99] MySQL. (2018b, 16 November 2018). MySQL Shell 8.0. Retrieved from <https://dev.mysql.com/doc/mysql-shell/8.0/en/>
- [100] MySQL. (2018c, 20 November 2018). MySQL Workbench. Retrieved from <https://dev.mysql.com/doc/workbench/en/>
- [101] Nanopoulos, A., & Manolopoulos, Y. (2002). Efficient Similarity Search for Market Basket Data. *The VLDB Journal — The International Journal on Very Large Data Bases*, 11(2), 138-152.
- [102] Nath, R. P. D., Hose, K., & Pedersen, T. B. (2015). *Towards a Programmable Semantic Extract-Transform-Load Framework for Semantic Data Warehouses*. Paper presented at the ACM Eighteenth International Workshop on Data Warehousing and OLAP, Melbourne, VIC, Australia.
- [103] Oakley, R. L., Iyer, L., & A.F.Salam. (2015). *Examining the Role of Business Intelligence in Non-Profit Organizations to Support Strategic Social Goals*. Paper presented at the 48th Hawaii International Conference on System Sciences, Kauai, HI.
- [104] Papadimitriou, C. (1994). *Computational Complexity*: Addison–Wesley.
- [105] Park, D. (2017, 28 August 2017). Analysis vs. Analytics: Past vs. Future. Retrieved from [https://www.eetimes.com/author.asp?section\\_id=36&doc\\_id=1332172](https://www.eetimes.com/author.asp?section_id=36&doc_id=1332172)

- [106] Patil, H. K., & Seshadri, R. (2014). *Big Data Security and Privacy Issues in Healthcare*. Paper presented at the 2014 IEEE International Congress on Big Data, Anchorage, AK, USA.
- [107] Prajapati, H. B., Dabhi, V. K., & Bhensdadia, C. K. (2015). *Taking a Deep Breath before Jumping into Research in Computer Science and Engineering*. Paper presented at the 2015 Fifth International Conference on Advanced Computing & Communication Technologies, Haryana, India.
- [108] Praveena, M. D. A., & Bharathi, B. (2017). *A Survey Paper on Big Data Analytics*. Paper presented at the International Conference on Information, Communication & Embedded Systems (ICICES 2017), Chennai, India.
- [109] Prema, A., & Pethalakshmi, A. (2013). *Novel Approach in ETL*. Paper presented at the 2013 International Conference on Pattern Recognition, Informatics and Mobile Engineering, Salem, India.
- [110] Pyun, G., Yun, U., & Ryu, K. H. (2014). Efficient Frequent Pattern Mining based on Linear Prefix Tree. *Knowledge-Based Systems*, 55, 125-139.
- [111] Qing, L., Boyu, Z., & Qinqian, L. (2017). *Impact of Big Data on Electric-Power Industry*. Paper presented at the 2017 IEEE 2nd International Conference on Big Data Analysis (ICBDA), Beijing, China.
- [112] Qiu, Q., Fleeman, J. A., Ball, D. R., Rackliffe, G., Hou, J., & Cheim, L. (2013). *Managing Critical Transmission Infrastructure with Advanced Analytics and Smart Sensors*. Paper presented at the 2013 IEEE Power & Energy Society General Meeting, Vancouver, BC.
- [113] Radhakrishna, V., SravanKiran, V., & Ravikiran, K. (2012). *Automating ETL Process with Scripting Technology*. Paper presented at the 2012 Nirma University International Conference On Engineering, Ahmedabad.
- [114] Radhika, P., Kumar, P. P., Sailaja, S. L., & Gayatri, V. (2017). *Confrontation and Opportunities of Big Data - A Survey*. Paper presented at the 2017 International Conference on Big Data Analytics and Computational Intelligence (ICBDAC), Chirala, India.
- [115] Raskino, M. (2015). 2015 CEO Survey: Committing to Digital. Retrieved from <https://www.gartner.com/doc/3026817/-ceo-survey-committing-digital>

- [116] Rebón, F., Ocariz, G., Gerrikagoitia, J. K., & Alzua-Sorzabal, A. (2015). *Discovering insights within a Blue Ocean based on Business Intelligence*. Paper presented at the 3rd International Conference on Strategic Innovative Marketing, Madrid, Spain.
- [117] Reinsel, D., Gantz, J., & Rydning, J. (2017, 5 April 2017). Total WW Data to Reach 163ZB by 2025. Retrieved from <https://www.storagenewsletter.com/2017/04/05/total-ww-data-to-reach-163-zettabytes-by-2025-idc/>
- [118] Santos, R. J., & Bernardino, J. (2008). *Real-Time Data Warehouse Loading Methodology*. Paper presented at the 2008 International Symposium on Database Engineering & Applications, Coimbra, Portugal.
- [119] Shang, X. (2005). *SQL Based Frequent Pattern Mining*.
- [120] Sharma, N., Sawai, D., & Surve, G. (2017). *Big Data Analytics: Impacting Business in Big Way*. Paper presented at the 2017 International Conference on Data Management, Analytics and Innovation (ICDMAI), Pune, India.
- [121] Song, M., & Rajasekaran, S. (2006). A Transaction Mapping Algorithm for Frequent Itemsets Mining. *IEEE Transactions on Knowledge and Data Engineering*, 18(4), 472-481.
- [122] Steele, C. (2019). Complete Beginner's Guide to Big O Notation. <http://www.youtube.com>.
- [123] Sun, D., Teng, S., Zhang, W., & Zhu, H. (2007). *An Algorithm to Improve the Effectiveness of Apriori*. Paper presented at the 6th IEEE International Conference on Cognitive Informatics, Lake Tahoe, CA, USA.
- [124] Sun, K., & Lan, Y. (2012). *SETL: A Scalable and High Performance ETL System*. Paper presented at the 2012 3rd International Conference on System Science, Engineering Design and Manufacturing Informatization, Cheng Du.
- [125] Suri, M., & Singh, S. N. (2018). *The Role of Big Data in the Media and Entertainment Industry*. Paper presented at the 2018 4th International Conference on Computational Intelligence & Communication Technology (CICT), Ghaziabad, India.
- [126] Technopedia. (2018). What does Console Application mean? Retrieved from <https://www.techopedia.com/definition/25593/console-application-c>



- [127] Tedre, M., & Moisseinen, N. (2014). Experiments in Computing: A Survey. *The Scientific World Journal*, 2014, 1-11.
- [128] Thomsen, C., & Pedersen, T. B. (2011). *Easy and Effective Parallel Programmable ETL*. Paper presented at the 14th International Workshop on Data Warehousing and OLAP, Glasgow, Scotland, UK.
- [129] Tiwari, A., Gupta, R. K., & Agrawal, D. P. (2010). A Survey on Frequent Pattern Mining: Current Status and Challenging Issues. *Information Technology Journal*, 9(7), 1278-1293.
- [130] Tripathi, N., Vartak, D., Chaudhari, H., & Naik, S. (2018). *Estimating Frequent Products in Shopping Cart Using Data Mining*. Paper presented at the 2nd International Conference on Inventive Communication and Computational Technologies (ICICCT 2018), Coimbatore, India.
- [131] Vaz, R., Shah, V., Sawhney, A., & Deolekar, R. (2017). *Automated Big-O Analysis of Algorithms*. Paper presented at the 2017 International Conference on Nascent Technologies in the Engineering Field (ICNTE-2017), Navi Mumbai, India.
- [132] Wang, F., & Li, Y.-h. (2008). *An Improved Apriori Algorithm Based on the Matrix*. Paper presented at the 2008 International Seminar on Future BioMedical Information Engineering, Wuhan, Hubei, China.
- [133] Wang, L., Cao, S., Wan, L., & Wang, F. (2017). *Web Anomaly Detection Based on Frequent Closed Episode Rules*. Paper presented at the 2017 IEEE Trustcom/BigDataSE/ICESS, Sydney, NSW, Australia.
- [134] Wei, Y.-Q., Yang, R.-H., & Liu, P.-Y. (2009). *An Improved Apriori Algorithm for Association Rules of Mining*. Paper presented at the 2009 IEEE International Symposium on IT in Medicine & Education, Jinan, China.
- [135] Wenzhe, L., Qian, W., Yu, W., Jiadong, R., Yongqiang, C., & Changzhen, H. (2017). *Mining Frequent Patterns for Item-Oriented and Customer-Oriented Analysis*. Paper presented at the 2017 14th Web Information Systems and Applications Conference (WISA), Liuzhou, China.
- [136] Wixom, B., & Watson, H. (2010). The BI-Based Organization. *International Journal of Business Intelligence Research*, 1(1), 13-28.

- [137] Yegulalp, S. (2017). What is NoSQL? NoSQL Databases Explained. Retrieved from <https://www.infoworld.com/article/3240644/nosql/what-is-nosql-nosql-databases-explained.html>
- [138] Zaki, M. J. (2000). Scalable Algorithms for Association Mining. *IEEE Transactions on Knowledge and Data Engineering*, 12(3), 372-390.
- [139] Zhang, H., Li, Y., Shen, C., Sun, H., & Yang, Y. (2015). *The Application of Data Mining In Finance Industry Based on Big Data Background*. Paper presented at the 2015 IEEE 17th International Conference on High Performance Computing and Communications, New York, NY, USA.
- [140] Zhang, Z., Ji, G., & Tang, M. (2013). *MREClat: An Algorithm for Parallel Mining Frequent Itemsets*. Paper presented at the 2013 International Conference on Advanced Cloud and Big Data, Nanjing, China.
- [141] Zulkurnain, N. F., & Shah, A. (2017). *HYBRID: An Efficient Unifying Process to Mine Frequent Itemsets*. Paper presented at the 2017 IEEE 3rd International Conference on Engineering Technologies and Social Sciences (ICETSS), Bangkok, Thailand.

## LIST OF PUBLICATIONS

### **Journal**

Author Chin-Hoong Chee, Jafreezal Jaafar, Izzatdin Abdul Aziz,  
Mohd Hilmi Hasan, William Yeoh  
Title Algorithms for Frequent Itemset Mining: A Literature Review  
Journal Artificial Intelligence Review (2018)

### **Conference**

Author Chin-Hoong Chee, Jafreezal Jaafar, Izzatdin Abdul Aziz  
Title FP-NoSQL: An Efficient Frequent Itemset Mining Algorithm  
Using the FB-DB Approach  
Conference 2018 IEEE Conference on Big Data and Analytics (ICBDA),  
Langkawi, Malaysia, 21<sup>st</sup> – 22<sup>nd</sup> November 2018