INCREMENTAL MINING OF FREQUENT PATTERNS FROM THE EDUCATIONAL DATA

By

MALAYA DUTTA BORAH

University Roll No. 2K12/PHDCO/03

Under the guidance of

Dr. Rajni Jindal

HOD, Dept. of Computer Science & Engineering,

Delhi Technological University

Submitted in fulfilment of the requirementof the degree of

Doctor of Philosophy to the

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

DELHI TECHNOLOGICAL UNIVERSITY

(FORMERLY DELHI COLLEGE OF ENGINEERING)

NEW DELHI– 110042

October, 2017

Declaration

I, Malaya Dutta Borah, a full time scholar (University Roll No. 2K12/PHDCO/03), hereby declare that the thesis titled **"Incremental Mining of Frequent Patterns from the Educational Data"** which is being submitted for the award of the degree of Doctor of Philosophy in Computer Engineering, is a record of bonafide research work carried out by me under the supervision of Dr. Rajni Jindal, HOD, Department of Computer Science & Engineering, Delhi Technological University.

I further declare that the work presented in the thesis has not been submitted to any university or institution for the award of any diploma or degree.

Date:

Place:

Malaya Dutta Borah

Roll No. 2K12/PHDCO/03

Department of Computer Science & Engineering

Delhi Technological University

(Formerly Delhi College of Engineering)

Delhi-110042

DELHI TECHNOLOGICAL UNIVERSITY

(Govt. of National Capital Territory of Delhi)
MAIN BAWANA ROAD, DELHI – 110042

Certificate

Date:

This is to certify that the work embodied in the thesis titled **"Incremental Mining of Frequent Patterns from the Educational Data"** submitted to Delhi Technological University for the award of the Degree of **Doctor of Philosophy** by **Malaya Dutta Borah**, University Roll No. 2K12/PHDCO/03 as a **full time scholar** in the Department of Computer Science & Engineering, Delhi Technological University, is an authentic work carried out by her under my guidance. This work is based on the original research and has not been submitted in full or in part for any other diploma or degree of any university to the best of my knowledge and belief.

Supervisor

Dr. Rajni Jindal

HOD, Department of Computer Science & Engineering
Delhi Technological University
(Formerly Delhi College of Engineering)
Delhi-110042

Acknowledgement

It is my great pleasure to express my thanks & gratitude to my supervisor Dr. Rajni Jindal, HOD, Department of Computer Science & Engineering, Delhi Technological University, Delhi 110042 for her valuable guidance, motivation, constant support and encouragement throughout this work.

I would like to take this opportunity to thank all the faculty members of Computer Science & Engineering Department, Delhi Technological University for their encouragement and support.

I am also thankful to all the staff members and research fellows for their untiring support.

Last but not the least I am grateful to my family members specially my husband Mr. Ganesh Chandra Deka and my daughter Tanisha Deka for their unconditional support and motivation during this work.

Date:

Place:

Malaya Dutta Borah

Roll No. 2K12/PHDCO/03

Department of Computer Science & Engineering

Delhi Technological University

(Formerly Delhi College of Engineering)

Delhi-110042

Abstract

Changes in the underlying database occur due to proliferation of innovations from heterogeneous online and offline resources. Mining frequent patterns are costly in changing databases, since it requires scanning the database from the start. Incremental Mining of frequent patterns is a solution to deal with this problem. The incremental mining process uses previous mining result to get the desired knowledge by reducing mining costs in terms of time and space. This research focuses on incremental mining of frequent patterns using four approaches.

Firstly, it is important to take into account the functionality i.e. Kind of patterns and performance (how to mine frequent patterns) of the patterns to be mined. One approach to find frequent patterns from frequently changing databases is to construct efficient data structures. A novel tree based data structure and mining algorithm called TIMFP (**T**ree for **I**ncremental **M**ining of **F**requent **P**attern) is proposed, which is compact as well as based on "Build Once and Mine Many" property.

Secondly, to measure the semantic significance of an item as per users' perspectives, a novel approach for mining high utility is presented. Focus of this research work is to introduce Average Maximum Utility (AvgMU) concept, which prunes items in early stages to avoid unnecessary staying of the items in the pruning phases.

Thirdly, this work proposes a tree based algorithm called CIFMine (**C**onstraint based **I**ncremental **F**requent Pattern **M**ining) to mine the incremental data by using a dataset filtering technique. In general, the goal of the pattern mining process is to

discover all the patterns from the data sources where not all the patterns may be suitable for the end user. This approach focuses on the constraints and specifies the desired properties of the patterns to mine that are likely to be of interest for the end user.

Lastly, we have focused on Educational Data Mining (EDM) by focusing research trends, challenges and Educational Outcomes to enhance the quality of education. A modified constraint based algorithm "Modified TIMFP" (Modified Tree for Incremental Mining of Frequent Pattern) to construct and mine incremental educational data is proposed. All the approaches are tested using Synthetic and Real dataset.

This research successfully establishes the fact that, approaches for Incremental Mining of frequent patterns are cost effective (in terms of time and space), efficient, scalable and produces optimal solutions for mining frequent patterns from different kind of data as compared to traditional frequent pattern mining approaches.

Contents

List of Abbreviations

AIEEE	All India Engineering Entrance Examination
ARM	Association Rule Mining
AvgMU	Average Maximum Utility
CanTree	Canonical-order Tree
CATS Tree	Compressed and Arranged Transaction Sequences Tree
CIFMine	Constraint based Incremental Frequent Pattern Mining
DM	Data Mining
EDM	Educational Data Mining
FHM	Fast High-Utility Miner
FP Tree	Frequent Pattern Tree
FPM	Frequent Pattern Mining
FUP	Fast UPdate
FUFP	Fast Updated FP tree
FWI	Frequent Weighted Itemsets
GMW	Global Maximum Weight
HUC-Prune	High Utility Candidates Prune
GwI-Apriori	Gap with Index Apriori
HUI-Miner	High Utility Itemset Miner
HUT	High Utility
IFPM	Incremental Frequent Pattern Mining
IHUP	Incremental High Utility Pattern
IM	Incremental Mining

IMBT	Incremental Mining Binary Tree
INFO[Node]	Information of a Node
$IWFPT_{FD}$	Incremental Weighted Frequent Pattern Tree based on Frequency Descending order
$IWFPT_{WA}$	Incremental Weighted Frequent Pattern Tree based on Weight Ascending order
TIMFP	Tree for Incremental Mining of Frequent Pattern
UP-Growth	Utility Pattern Growth
UT	Utility
UTmin	Minimum Utility support threshold
UTnew	Newly calculated Utility
MCRP-Tree	Maximum Constraint based method for generating Rare association rule Pattern mining tree
MIQ-Tree	Maximum Item Quantity Tree
MU-Strategy	Maximum Utility–Strategy
NIC	National Informatics Centre
RQs	Research Questions
SWF	Sliding Window Filtering
UWEP	Update With Early Pruning
WFIM	Weighted Frequent Itemset Mining
WIP	Weighted Interesting Patterns
WIT	Weighted Itemsets

List of Figures

List of Tables

CHAPTER-1: INTRODUCTION

This Chapter begins with a background of Data Mining. Gradually, the concept of Frequent Pattern Mining, Incremental Frequent Pattern Mining, Motivation of the Research, Problem Statement, Solution Approaches, and Contributions are discussed. The Chapter concludes with the documentation of organization of the thesis.

1.1 Background

For every organization to be in the right place at right times are the crucial factors to ensure right decision from large amount of data (Lim & Lee, 2010). To deal with the large amount of data stored in the database, **Data Mining (DM)** has been introduced. A widely accepted definition of DM is: "*It is the process of discovering interesting knowledge from large amounts of data stored in databases, data warehouses, or other information repositories* (Han & kamber, 2006)*"*.

In literature, Data Mining is referred to as Data Archaeology, Data Pattern Processing, Information Discovery, Information Harvesting and Knowledge Extraction (Fayyad, Piatetsky-Shapiro, & Smyth, 1996).

Data Mining deals with the data explorations model and finally provides the knowledge that helps decision makers in decision-making. Various DM techniques such as Frequent Pattern Mining, Association Rule Mining, Classification, Prediction, Clustering, and Neural Networks etc. are discussed in various literatures.

Frequent Pattern Mining (FPM) is an important research topic in the field of DM to discover ***interesting patterns*** in databases. A pattern is said to be interesting pattern if it appears at least as frequently as a predetermined minimum support threshold.

FPM deals with various data mining tasks such as discovering patterns, association (Agrawal, Imielinski & Swami, 1993) (Yu & Chen, 2005), and correlation (Brin et al., 1997) etc. Application areas of Frequent Pattern Mining include Behavior Analysis, Risk Pattern Analysis, Software bug analysis, Chemical and Biological trend analysis are a few to be mentioned.

FPM focuses on the extraction of frequent patterns from the large amount of data. In real life applications, the huge volume of data is generated from online and offline sources which are incremental in nature. Day-to-day transaction on dataset involves addition of new instances as well as deletion of obsolete instances without reforming, rescanning the original dataset in repetitive manner.

Mining frequent patterns is costly during changes of the underlying database. Most of the traditional approaches need repetitive scanning of the database as well as generating huge number of candidate sets (Han et al., 2007).

The **Incremental Mining (IM)** uses previous mining result to get the desired information by reducing mining costs in terms of **time** and **space** (Su et al., 2009).

The following are the important parameters/interestingness measures of Incremental Frequent Pattern Mining:

1. Incremental database: A database where new instances of transactions keep on adding and the obsolete instances are deleted. In literature, Incremental database is also referred as Dynamic database or Updated database. (Refer Figure 1.1).

2. Support: It reflects the usefulness of the rules. e.g. for a rule X=> Y, support can be written as: $Support(X \Rightarrow Y) = Frequency(X \cup Y) / TotalCount$
3. Confidence: It reflects the certainty of discovered rules. e.g. for a rule X=> Y Confidence can be written as: $Confidence(X \Rightarrow Y) = Support(X, Y) / Support(X)$
4. Minimum Support Threshold: The user or the domain experts define a threshold, which is known as Minimum Support Threshold.
5. Pattern: A Pattern is an item or combination of items in a database that occurs more often than expected.
6. Frequent Pattern: A Pattern whose frequency of occurrence is above or equal to the Minimum Support Threshold is known as a Frequent Pattern (Agrawal & Srikant, 1994).

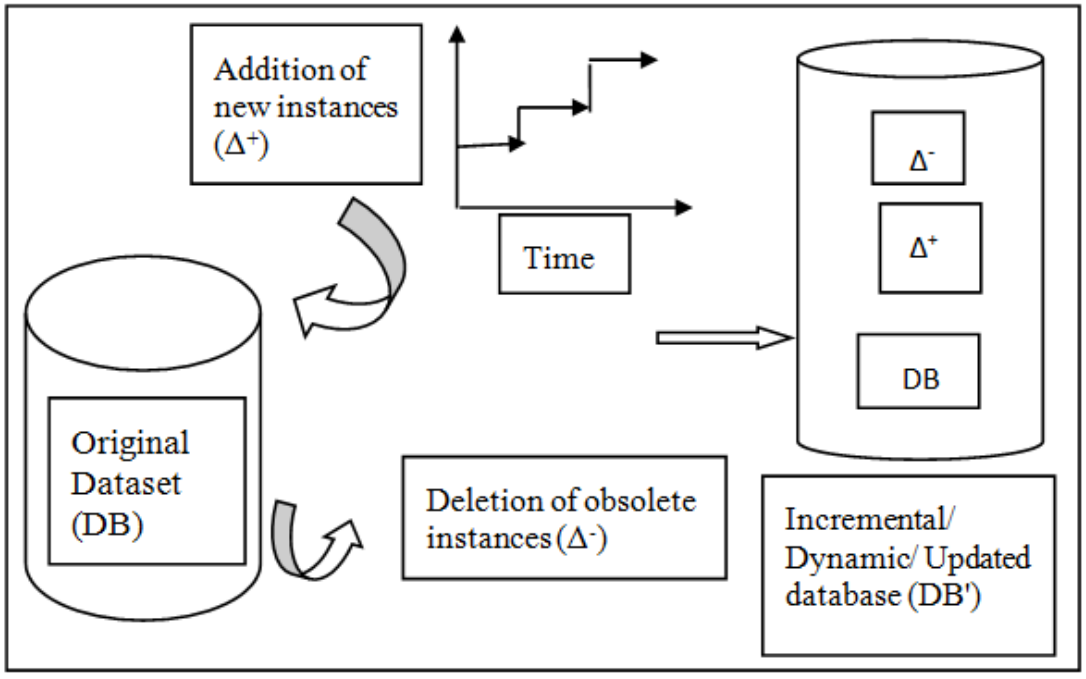


Figure 1.1: Incremental database

As the database are frequently changed, there is a possibility that some previous frequent patterns become invalid while some new interesting frequent patterns appear in the incremental database. In such type of situation, the important issue is **how to utilize**

the mining result to efficiently reflect these changes. Thus, mining of growing databases is an interesting area of research in Data Mining.

The following issues are to be addressed to effectively extract knowledge from the growing databases:

- Flexibility of exploring growing databases (to find frequent patterns, flexibility to changes in minimum support value etc.)
- Cost of mining (I/O as well as memory, No. of scans, generation of candidate key etc.)
- Updating of the knowledge base (suitable data structure, updating frequent patterns, effective data mining rules etc.)

To address these issues, a new Data Mining technique called **Incremental Frequent Pattern Mining (IFPM)** has emerged. IFPM is a way of extracting frequent patterns from incremental database. Instead of scanning repeatedly the original database, this mining process uses previous mining patterns to get the resultant pattern by reducing costs in terms of **running time and space** (Refer Figure 1.2).

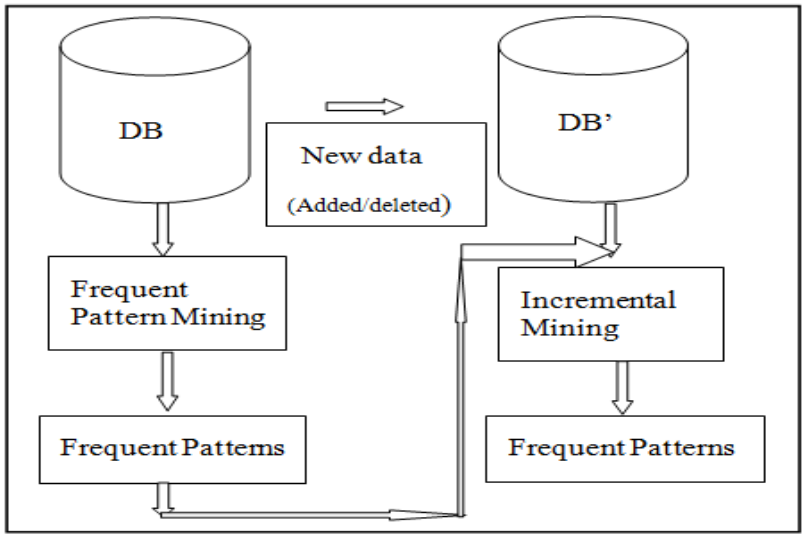


Figure 1.2: Incremental Frequent Pattern Mining

1.2 Motivation

Researchers have reported several approaches for Incremental Mining of frequent patterns. One approach to incremental mining of frequent patterns is to construct efficient data structures. Researchers have found that tree based data structures are most suitable for mining frequent patterns from incremental data such as Compressed and Arranged Transaction Sequences Tree (CATS) (Cheung & Zaïane, 2003), Canonical-order Tree (CanTree) (Leung et al., 2007), Fast Updated Frequent Pattern tree (FUFP) (Hong et al., 2008), Pre-large itemsets-FUFP (Pre-FUFP) (Lin et al., 2009), Incremental Mining Binary Tree (IMBT) (Yang &Yang, 2009) etc.

The common assumption of the above mentioned approaches is that, all patterns are equal in weight (Chuang, Huang, & Chen, 2008), but in real life applications it is not always possible to treat all patterns in equal or binary way, since the semantic significance of an item from the users' perspectives is highly significant.

In order to address the above mentioned shortcoming, high utility mining (Lin, Lan, & Hong, 2012; Liu & Qu, 2012; Tseng et al., 2013; Lee, Park, & Moon, 2013; Song, Liu, & Li, 2014; Yun, Ryang, & Ryu, 2014) has emerged as an important research topic. High utility is the relative importance (Interest/Intensity) of each item greater than or equal to the user defined minimum utility threshold, equivalent to weighted frequent pattern in dynamic environments (Li, Yeh, & Chang, 2008; Ahmed et al., 2009; Tseng et al., 2010). In general, the goal of the frequent / high utility pattern mining is to discover all the patterns from the data sources.

Most of the existing algorithms generate large number of potentially high utility patterns and require longer time for pruning using "Global Maximum Weight (GMW)".

Introducing constraints in frequent patterns of incremental data can help users extract exactly what patterns they want, instead of generating the whole patterns. The goal of the pattern mining process is to discover all the patterns from the data sources where not all the patterns may be suitable for the end user.

Most of the existing constraint based algorithms require several passes to scan the dataset even though user is interested in only few patterns (e.g. Gap with Index Apriori (GwI-Apriori), Maximum Constraint based method for generating Rare association rule Pattern mining tree (MCRP-Tree) etc.), resulting in large computing time, since they do not filter out the initial undesired transactions.

Many researchers have tried various approaches for incremental mining of frequent patterns using different types of data. Apart from the business data, the researcher focus on educational data i.e. mining educational data to define goal-directed practices for ensuring educational institution's success at all levels (Romero &Ventura, 2013).

The real world dynamic environment needs to consider the data which are incremental in nature. The motivation behind this research work is to find out the research gaps of the previous research works and find out optimal solutions in Incremental Mining of Frequent Patterns with a focus on educational data.

1.3 Problem Statement

To find out the Incremental Frequent Patterns using tree based approaches by applying semantic significance and various constraints from incremental data. The problem statement can be subdivided as follows:

i. Let I be a set of N items $I= \{I\_1, I\_2 \ldots I\_N\}$ and a Database DB= $\{T\_1, T\_2 \ldots T\_M\}$ be a set of M transactions. For each transaction T_i ($1 \leq i \leq M$), $T\_i \in I$, an

incremental portion (Δ_1^+) of the database consists of some T_i in DB' and Δ_1^- be the deleted portion of the DB' and Minimum Support Threshold '∞'.

- The problem is to mine Incremental Frequent Patterns from a given set of transactions of database (DB) which satisfies '∞' and changing as time advances (DB').

ii. Let us integrate the weight factor to the statement (i). W be a set of 'n' positive numbers W={w_1,w_2...w_n}.

Average Maximum Utility (AvgMU) is $\sum_{I\_N=0}^{N} Weight(I\_N) \ / N$ and

New Utility is $UTnew = AvgMU * SupportCount$

- The problem is to mine the Incremental High Utility Frequent Patterns from a given set of weighted transactions that changes as time advances and also satisfies downward closure property and prunes unnecessary items in early stages applying AvgMU.

iii. Let us add the user specified length threshold 'Θ' and user specified pattern 'Ω' in statement (i) & (ii).

- The problem is to discover the complete set of patterns from the database DB' using the mining results from DB, which satisfies the support '∞' of item 'I', length constraints 'Θ' and pattern constraint 'Ω'.

iv. The problem is to generate Association Rules using frequent patterns with high confidence from educational data.

1.4 Solution Approach

We have four approaches for Incremental Mining of Frequent Patterns. The solution approach of this thesis is briefly described as below:

i. An approach for Incremental Mining of Frequent Patterns

Mining frequent patterns from incremental data is costly as most of the approaches need repetitive scanning and generates a large number of candidate keys. Hence, it is important to develop an efficient method to enhance the performance of mining. This work proposes a novel tree based data structure for mining frequent pattern of incremental data called TIMFP (Tree for Incremental Mining of Frequent Pattern) to enhance the mining performance.

The tree captures the contents of the transaction database, and arranges in tree nodes according to the property of binary search tree which is unaffected by changes in item frequency. It is easily manageable when performed 'Insertion', 'Deletion' and 'Update' operations. It satisfies the property "Build Once and Mine Many" which is suitable for incremental as well as interactive mining. We have compared TIMFP with

- Canonical-order Tree (CanTree)
- Compressed and Arranged Transaction Sequences Tree (CATS) and
- Incremental Mining Binary Tree (IMBT).

Experimental results show that, the proposed work (TIMFP) has better performance than other data structures in terms of:

- Time required for constructing the tree and
- Mining frequent patterns from the tree**.**

ii. An approach for Incremental High Utility Frequent Pattern Mining

Mining high utility pattern has become prominent as it provides semantic significance (utility/weighted patterns) associated with items in a transaction. Data analysis and respective strategies for mining high utility patterns is important in real world scenarios.

Recent researches focused on high utility pattern mining using tree based data structure. But, they suffer greater computation time, since they generate multiple tree branches.

To cope up with these problems, this work proposes a novel data structure with Average Maximum Utility (AvgMU) and mining algorithm to mine high utility patterns from incremental data that reduces tree constructions and computation time. In mining operations, AvgMU is considered instead of Global Maximum Weight (GMW), which prunes the items in early stages to avoid unnecessary staying of the items in the pruning phases.

The proposed algorithms are implemented using Synthetic, real datasets and compared with state-of-the-art tree based algorithms namely

- Utility Pattern Growth (UP-Growth)
- Utility Pattern Growth^{+} (UP-Growth^{+})
- High Utility Itemset Miner (HUI-Miner)
- Maximum Utility–Strategy1 (MU-Strategy1) and
- Maximum Utility–Strategy2 (MU-Strategy2)

It is found that the proposed tree based method has better performance (running time, scalability and memory consumption) than the other algorithms as compared in this research work.

iii. An approach for Incremental Constraint based Frequent Pattern Mining

Introducing constraints in frequent patterns of incremental data can help users extract exactly what patterns they want, instead of generating the whole patterns. The goal of the pattern mining process is to discover all the patterns from the data sources where not all the patterns may be suitable for the end user. This work proposes a tree based algorithm called CIFMine (Constraint based Incremental Frequent Pattern Mining) to mine the

incremental data by using a dataset filtering constraints like specific pattern, weight and length of the pattern and user defined minimum support threshold which reduces the size of the dataset before constructing the tree.

We have compared CIFMine with the following state-of-art algorithms

- Maximum Constraint based method for generating Rare association rule Pattern mining tree)(MCRP-Tree)
- RegularMine and
- Gap with Index Apriori (GwI-Apriori)

It was found that proposed algorithm is more suitable than listed algorithms in terms of computing time as it produces only those patterns that are likely to be of interest to the end user.

iv. Approaches for Incremental Educational Data Mining

We have focused on Educational Data Mining (EDM) and its research trends for quality education.

Firstly, this work proposes a modified Constraint based algorithm called "Modified TIMFP" (Modified Tree for Incremental Mining of Frequent Pattern) to construct and mine incremental educational data more efficiently as compared to TIMFP (Jindal & Dutta Borah, 2015a).

Secondly, we have focused on behavior pattern and participation of students in teaching / learning process. Finally, association rule have been obtained from such patterns.

1.5 Dataset Used

Researchers found that widely-accepted synthetic data or the data that collects from real environment are most suitable for performance evaluation (Sun & Bai, 2008). Hence we have considered following dataset for performance evaluation of proposed approaches in our thesis.

- **Synthetic dataset**(http://fimi.ua.ac.be/data/)**:**
    - T10-I4-D100K: Taking into consideration of the above statements, we have considered "T10-I4-D100K" dataset (Agrawal & Srikant, 1994) which is moderately sparse widely accepted synthetic dataset.
- **Real datasets** (http://fimi.ua.ac.be/data/):
    - Pumsb: This dataset contains census data for population and housing, which is dense in nature.
    - Connect: This dataset contains online connection data which are dense in nature.
    - Kosarak: This dataset contains click-stream data collected from Hungarian on-line news portal, which is sparse in nature.
    - NU-MineBench 2.0: This is a chain store dataset, which is sparse in nature (Pisharath et al., 2005).
- **Educational domain (learning) datasets:**
    - Teachable Peer Learner dataset (AlgebraI2010Dec-retry-ss) (Matsuda & Ritter, 2011): This dataset consist of students and tutor participation data in online study courses.
    - StatWay - Fall 2011: Austin Community College dataset (Koedinger et al., 2010): This dataset consist of students and tutor participation data in online study courses.

- AIEEE 2007 dataset: It is a student data set seeking admission in a particular branch through All India Engineering Entrance Examination (AIEEE). Therefore, it helps predict the approximate strength of student in the particular branch. This data set collected from National Informatics Centre (NIC) Delhi. In order to demonstrate the usefulness of our high utility mining process, we have assigned random numbers in student ranks and their preferences.

- Motivation and Metacognition in Chinese Vocabulary Learning, Experiment 3 & 5 dataset (Koedinger et al., 2010): This dataset consist of students and tutor participation data in Vocabulary Learning study courses.

1.6 Contribution of the Research

The aim of this research is to develop approaches for incremental mining of frequent patterns which will reduce:

a) Number of scans in a database
b) Number of candidate sets in mining process
c) Cost in terms of time and space

These approaches should be scalable and produce optimal solutions for mining frequent patterns from different kind of data including educational data.

We believe that this research will help the researchers to find out related advances, and limitations of each approach so that they can design a refined incremental mining algorithm which will improve the performance of an algorithm as stated above using different kind of data in different domain.

The summary of the contribution is described below:

i. A novel tree based data structure and mining algorithm called TIMFP (**T**ree for **I**ncremental **M**ining of **F**requent **P**attern) is proposed, which is compact as well as based on "Build Once and Mine Many" property.

   - This novel approach can be used by the Data Mining research community for construction of tree based data structure and mining algorithm for Incremental Mining of frequent patterns which is cost effective.

ii. To measure the semantic significance of an item as per users' perspectives, a novel approach for mining high utility is presented. Focus of this research work is to introduce Average Maximum Utility (AvgMU) concept, which prunes items in early stages to avoid unnecessary staying of the items in the pruning phases.

   - This approach helps the research community to measure the semantic significance of items and increase the mining performance by pruning unnecessary items beforehand.

iii. This work proposes a tree based algorithm called CIFMine (**C**onstraint based **I**ncremental **F**requent Pattern **M**ining) to mine the incremental data by using a dataset filtering technique. This approach focuses on the constraints and specifies the desired properties of the patterns to mine that are likely to be of interest for the end user.

   - This approach helps the research community to provide faster tree construction and mining by filtering the unnecessary items beforehand as per user defined constrain.

iv. We have focused on Educational Data Mining (EDM) by focusing research trends, challenges and educational outcomes to enhance the quality of education. A modified Constraint based algorithm "Modified TIMFP" (Modified Tree for

Incremental Mining of Frequent Pattern) to construct and mine incremental educational data is proposed.

- This work helps the EDM research community by providing target oriented decision making process which enhances the quality of the education.

1.7 Thesis Organization

The thesis is comprised of seven chapters. In the first chapter, an overview of the Data Mining, Incremental Frequent Pattern Mining, Problem Statement, Solution Approach, a brief description of Datasets used, and Research Contribution is presented. The rest of the chapters are as follows:

Chapter 2: This chapter discusses the studies of the state-of-art frequent pattern mining methods. There is a detail description about incremental Frequent Pattern Mining, High Utility Pattern Mining, Constraint Mining and Educational Data Mining algorithms.

Research gaps are identified and research questions are formed based on the study in this chapter. Two papers have been published /communicated from this chapter, namely "Predictive Analytics in Higher Educational Context". IT Professional, Publisher: IEEE Computer Society, 17 (4), 24-33, 2015, doi: 10.1109/MITP.2015.68 and "A Study of Incremental Mining of Association Rules" *(communicated to an International Journal)*.

Chapter 3: Our approach for incremental frequent pattern mining is stated in this chapter. There is a description about the limitation of the traditional frequent mining algorithms with the aim of clarifying how our incremental mining approach is suitable for mining dynamic data.

A detail discussion of proposed Tree based data structure and Mining algorithm is presented in this chapter. A paper titled "A novel approach for mining frequent patterns

from incremental data" has been published in International Journal of Data Mining Modelling and Management, Publisher: Inderscience, *8 (*3), 2016. doi:10.1504/IJDMMM .2016.079071 from this chapter.

Chapter 4: This chapter presents our approach for High Utility Pattern Mining. It discusses about how utility/weights are important in real world scenarios. We have introduced Average Maximum Utility (AvgMU) instead of Global Maximum Weight (GMW), which prunes the items in early stages and avoids unnecessary staying of the items in the pruning phases.

Proposed algorithms for construction and mining of the tree are presented in this Chapter. A paper titled "An approach for high utility incremental pattern mining." has been accepted (in press) in International Journal of Data Analysis Techniques and Strategies, Publisher: Inderscience, [Online]: http://www.inderscience.Cominfo /in eneral /forthcoming. php? Jcode = ijdats from this chapter.

- **Chapter 5**: How constraints are beneficial for incremental mining of frequent patterns are discussed in this chapter. A dataset filtering technique is integrated with the proposed algorithm stated in **Chapter 3** and found that constraints are suitable for mining incremental data.

A paper titled "Constraint based filtering for Incremental Data Mining" has been published in proceedings of the 7th IEEE International Conference on Computational Intelligence and Communication Networks, Jabalpur, India, December 12-14, 2015, from this chapter.

Chapter 6: This chapter discusses how Educational Data Mining is helpful to enhance the quality of education.

Further, the approaches and constraint for frequent pattern mining from educational data are also discussed. Three papers have been published / accepted from this chapter, namely "A Survey on Educational Data Mining and Research Trends" in International Journal of Database Management Systems, Publisher: AIRCC, 5(3), 53-73, doi: 10.5121/ ijdms; "An approach to generate students' response on learning environment using Association Rule Mining" in proceedings of the IEEE International Conference on Data Mining and Intelligent Computing (pp.1-5). Delhi, India, September 5-6, 2014, doi: 10.1109/ICDMIC.2014.6954225 and "Constraint based Frequent Pattern Mining from Incremental Educational data: Educational Data Mining Approach" Accepted for publication at the 2016 IEEE International Conference on Teaching and Learning in Education, Universiti Tenaga Nasional, Malaysia, March 1-2, 2016.

Chapter 7: The final chapter presents the conclusion and future scope of this research work. This chapter deliberates upon importance of our proposed methods for mining of frequent patterns and the need of incremental mining.

A list of published/accepted/communicated paper relating to this research work in International/National Journals/Conferences of repute is given after Chapter 7**.**

A list of references referred in this research work is given at the end of the thesis for ready reference.

CHAPTER-2: LITERATURE REVIEW

This chapter presents a literature review of Incremental Mining of Frequent Patterns, Incremental High Utility Frequent Pattern Mining, Incremental Constraint based Frequent Pattern Mining and Incremental Educational Data Mining. Research questions are formed, Research gaps are identified and future directions are provided based on the study in this chapter.

2.1 Introduction

The aim of this literature review is to make a comprehensive study of existing and ongoing research work, to explore the research gaps, and to propose solution for it. We have followed the following steps for literature review.

- Planning the review
    - Identify the need of review
- Conducting the review
    - Develop Research Questions (RQs)
- Reporting the review
    - Report review category wise
    - Identify the research gaps
- Concluding the review
    - Provide future direction

2.2 Planning the Review

In real-world applications, there are many challenges emerged recently in the field of Frequent Pattern Mining. It is mainly concerned with the following parameters:

- Changes of underlying databases as time advances
- Semantic significance associated with a pattern and
- Imposing of Constraints in a particular area

Our research is concerned with the above parameters and hence we have categorized our work into four approaches as listed below.

a) Incremental Mining of Frequent Patterns
b) Incremental High Utility Frequent Pattern Mining
c) Incremental Constraint based Frequent Pattern Mining
d) Incremental Educational Data Mining

2.3 Conducting the Review

As a first step **Research Questions (RQs)** are formulated and are listed as follows:

- **RQ#1:** What is the need for Incremental Mining?

  (Solution: *Refer Chapter 1)*

- **RQ#2:**What are the published literature/approaches for Incremental mining of frequent patterns?

  (Solution: *Refer Chapter 2)*

- **RQ#3:**Can they be classified according to functionality?

  (Solution: *Refer Chapter 3)*

- **RQ#4:** Are they able to measure the semantic significance of an item as per users' perspectives?

  ***(Solution:** Refer Chapter 4)*

- **RQ#5:** Does they satisfy users' constraints?

  ***(Solution:** Refer Chapter 5)*

- RQ#6: Are they applicable in educational context?

  ***(Solution:** Refer Chapter 6)*

We have studied the research papers published in various International/National journal and Conferences of repute.

2.4 Reporting the Review

In this section we have described the review on four approaches as listed in section 2.2 and finally we have identified the research gaps based on this study.

2.4.1 A Review on Incremental Mining of Frequent Patterns

Incremental data can be defined as dynamic data that changes with time. In real world applications, underlying database consists of Incremental data that keeps changing and requires continual updating. Mining frequent patterns from such type of changing/growing databases is costly, as most of the approaches need repetitive scanning of the databases as well as generating huge number of candidate sets (Han et al., 2007).

Frequent pattern mining is an important data mining technique that can be utilized to discover the frequent patterns from changing/growing database (Cameron & Leung, 2011). It is important to take into account the functionality (what kind of patterns) and

performance (how to mine frequent patterns) (Cheung & Zaïane, 2003). On the basis of this concept, the research works are divided into the following three categories chronologically:

- Apriori based Incremental Mining
- Partition based Incremental Mining and
- Pattern Growth based Incremental Mining

2.4.1.1 Apriori based Incremental Mining

This subsection describes a candidate set generation process based on Apriori based Incremental Mining approach. We have considered those approaches that are widely used by the research community. e.g.: Fast UPdate (FUP) (Cheung et al., 1996), Fast $UPdate_2$ (FUP_2) (Cheung, Lee & Kao, 1997), Update With Early Pruning (UWEP) (Ayan, Tansal, & Arkun, 1999), and PELICAN (Veloso et al., 2001) algorithms.

FUP is the first Apriori based Incremental Mining algorithm is proposed for the efficient maintenance of discovering association rules in large databases when new transaction data are added to a transaction database. It generates less number of candidate keys as compared to Apriori, but deletion operation is not incorporated in this algorithm.

FUP_2 is an extended version of FUP, which handles the association rules in both Insertion and Deletion operation. This algorithm assumes that the large itemsets and their Support Count in the database came from the results of previously stored mining activities. It is more desirable to store counts rather than the association rules as the association rules can be calculated from these counts efficiently. The focus of this algorithm is that, the transaction to be deleted is a small part of the database activity.

UWEP is another Apriori based algorithm that updates association rules with early pruning. It uses a dynamic look ahead strategy in updating the existing large itemsets. This

algorithm detects and removes those items that will no longer remain large after the contribution of the new set of transactions. The limitation is that it does not allow changes in minimum support value.

PELICAN algorithm is similar to FUP_2 algorithm which is based on a lattice decomposition strategy to find out the maximum frequent/ large itemset when the database is updated. Apart from that it allows changes in minimum support value and the generation of candidate sets is less, compared to FUP algorithm. Limitation of the algorithm is that, it considers only maximum frequent itemset in mining process.

2.4.1.2 Partition based Incremental Mining Algorithms

The limitation of the Apriori based Incremental Mining Algorithms are that they require multiple numbers of scans as well as generate large candidate sets. To cope up with this problem, Lee, Lin, & Chen (2001) developed a partition-based algorithm known as Sliding Window Filtering (SWF) algorithm. In this algorithm, a transaction database, 'D' is divided into 'n' number of partitions i.e. $\{P_1, P_2 \text{---} P_n\}$ and it employs filtering threshold in each partition to deal with the candidate itemset generation. The Cumulative Filter (CF) produced in processing each partition constitutes the key component to realize the incremental mining. If 'L' is a frequent itemset in 'D', then 'L' must be frequent at least in one of the 'n' number of partitions.

An extended version of SWF is proposed by Chang & Yang (2003). Two algorithms has been proposed by reusing previous mining task for generating frequent patterns, namely SWF with Frequent Itemset (FI SWF) and SWF with Candidate Itemset (CI SWF).

This approach is faster and generates fewer candidates as compared to SWF and Apriori based algorithms. The limitation of this algorithm is that it does not take into consideration the situation that the dataset to be Inserted or Deleted is either too large or too small.

2.4.1.3. Tree/Pattern Growth based Incremental Mining

The limitations of the Apriori based Incremental Mining Algorithms and Partition based Incremental Mining Algorithms are:

- Multiple no. of scans required
- Do not allow changes in minimum support value
- Large candidate sets
- Assumption that data resides only on primary memory is not possible in most of the real world applications

Taking into account of the above mentioned four limitations, Tree/Pattern Growth based Incremental Mining Algorithms has been introduced by the researchers.

An extension of Frequent Pattern Tree (FP Tree) (Han et al., 2004) data structure is Compressed and Arranged Transaction Sequences Tree (CATS Tree), proposed by Cheung & Zaïane (2003). This is a prefix tree, which improves and compresses the structure of the tree by locally optimizing the sub-trees. The major advantage of this tree is that, it enables frequent pattern mining by supporting "build once, mine many" strategy. However, the limitation of this data structure is, without prior knowledge of transactions and itemsets, building the tree is not possible.

Another drawback is adding new transactions, since this process involves:

- First, checking the common items and
- Second, new items are merged into the existing path of the tree which is computationally expensive.

In 2007, canonical tree based data structure was proposed by Leung et al. (2007), which is known as Canonical-order Tree (CanTree). In CanTree, the nodes of the tree are arranged according to Canonical/Lexicographical order. During maintenance, no adjustment of nodes is required. However, the limitation of this structure is that the tree might be larger for a long transaction with multiple branches, which takes longer traversing time.

Hong, Lin, & Wu (2008) proposed a Fast Updated FP tree (FUFP) data structure, basically an extension of the FP-tree. The difference between FP-tree and FUFP is that the links between parent and child nodes are Bi-directional, which helps faster maintenance of frequent patterns in case of deletion operation. The new frequency counts are kept in the header table for making the tree update process easier.

A modified version of the FUFP tree is proposed by Lin, Hong, & Lu (2009), based on the concept of Pre-large itemset. The Pre-large itemset can be defined as an itemset that is not actually a large item but possess a high probability of getting large in future. This property can be identified with the help of a lower and an upper support threshold. This structure is suitable for handling itemsets in case of insertion (small in original and big in updating the database) only. The limitation of this approach is that it does not allow the changes in the threshold after the database is updated.

For changing of threshold during the lifetime of the database, a data structure called "IMBT - A Binary Tree for Efficient Support Counting of Incremental Data Mining" was proposed by Yang & Yang (2009). This concept is based on binary trees for mining incremental data and requires only a single scan of the database. The limitation of this work is the creation of repeated node to construct the sub-tree. The maximum duplicities may rise up to 2^N if the tree levels ranges from 0 to N. Further, leaf nodes may not be placed in the same level. Since researchers only consider limited itemsets and a few transactions for a large dataset, the tree will be larger which is undesirable.

2.4.2 A Review on Incremental High Utility Frequent Pattern Mining

In traditional frequent pattern mining, the frequency of an itemsets is defined over the binary domain to denote the presence and absence of an item in a transaction. The common assumption is that each item in a database is equal in weight (Chuang, Huang, & Chen, 2008). As a matter of fact, in real life applications, it is not always possible to treat all patterns in an equal or binary way. It may not be a sufficient indicator to measure interestingness (Yao, Hamilton, & Butz, 2004) of items.

To measure interestingness of items, Researchers integrated a term "weight". The weight of an item can be described as Unit cost, Profit, Priority, Distance, Credit e.g. Cost of items in a transactional database, importance of the traversing different web pages as per the priority basis etc. Similarly, the share measure was proposed to reflect the impact of the quantity sold in terms of Cost or Profit of an itemset in weighted frequent pattern mining (Barber & Hamilton, 2003; Li, Yeh, & Chang, 2005a; Li, Yeh, & Chang, 2005b).

Several algorithms have been proposed by researchers to reflect the relative importance of items (Wang, Yang, & Yu, 2000; Wang et al, 2004; Yao, Hamilton, & Butz, 2004; Yun & Leggett, 2005; Yun & Leggett, 2006; Yun, 2007; McGlohon, Akoglu, & Faloutsos,

2008; Sun & Bai, 2008; Yun, 2008; Yun & Ryu, 2011; Park et al., 2012; Ahmed et al., 2012; Yun, Lee, & Ryu, 2014).

Weighted Frequent Itemset Mining (WFIM) (Yun & Leggett., 2005) is the first FP tree based algorithm that reflects the relative importance (weight) of an item. Items are assigned with fixed weight in ascending order in the tree. Weighted Interesting Patterns (WIP) (Yun & Leggett, 2006) is a remarkable algorithm which introduces a new measure, weight-confidence to mine correlated patterns with weight affinity. Researcher have pointed out that weighted frequency of an item does not have the downward closure property. The limitations of both the algorithms are:

- They are well established in static database
- Quantities of items are not considered which is important to discover itemsets with high profit.

In this framework, Vo, Coenen, & Le (2013) proposed a tree based data structure, Weighted Itemsets (WIT) which is useful for fast mining of Frequent Weighted Itemsets (FWI) from transaction databases. In such framework, two tree structures i.e. Incremental Weighted Frequent Pattern Tree based on Weight Ascending order ($IWFPT_{WA}$) & Incremental Weighted Frequent Pattern Tree based on Frequency Descending order ($IWFPT_{FD}$) and two algorithms $IWFP_{WA}$ & $IWFP_{FD}$ are proposed by Ahmed et al. (2012). Though this approach is suitable for both incremental and interactive mining, multiple branches occur during construction of the tree, hence time increases for computation (Insert, Delete etc.).

To represent high utility itemset compactly, recent high utility mining algorithms (Chan Yang, & Shen, 2003; Hu & Mojsilovic, 2007; Lin, Lan, & Hong., 2012; Liu & Qu, 2012; Lee, Park, & Moon, 2013; Tseng et al., 2013; Song, Liu, & Li, 2014; Yun, Lee, & Ryu,

2014) have emerged. In high utility mining, since the downward closure property cannot be directly applied, a two-phase algorithm is proposed by Liu, Lia, & Choudhary (2005). This algorithm can efficiently prune the number of candidates and can precisely obtain the complete set of high utility itemsets. Li, Yeh, & Chang (2008) proposed an isolated item discarding strategy to reduce the number of candidate itemsets in the first phase of level wise utility mining approach.

A special indexing structure and projection based method was developed by Lan, Hong, & Tseng (2014) which is much more computationally efficient compared to Two-phase algorithm. High utility mining in frequent closed patterns are described in Chan, Yang, & Shen (2003) and Maximal high utility itemset is described by Shie, Yu, & Tseng (2012).

Incremental High Utility Pattern (IHUP) trees (Ahmed et al., 2009), High Utility Candidates Prune (HUC-Prune) (Ahmed et al., 2011), Utility Pattern Growth (UP-Growth) (Tseng et al., 2010), Utility Pattern Growth$^+$ (UP-Growth$^+$) and Utility Pattern Tree(UP tree) (Tseng et al., 2013), Fast High-Utility Miner (FHM) (Fournier-Viger et al., 2014), Maximum Utility Growth (MU-Growth, MU-Strategy1, MU-Strategy2) and Maximum Item Quantity Tree (MIQ-Tree) (Yun, Ryang, & Ryu, 2014) are tree based approaches to discover high utility itemsets without generating candidate keys.

The problem of the above mentioned algorithms are that, most of the candidates are found out to be of not high utility after their exact utilities are computed. To cope up with this problem, a novel approach was proposed by (Liu & Qu, 2012) which is known as High Utility Itemset Miner (HUI-Miner). It uses utility-lists which are similar to the tidlists used in Eclat algorithm for mining frequent itemsets (Zaki, 2000). It is based on depth-first strategy which is better than the IHUP (Ahmed et al., 2009; Tseng et al., 2010) and UP-

Growth^{+} (Tseng et al., 2013). The d2HUP algorithm is another approach that uses irrelevant item filtering and look-ahead pruning for efficient mining.

2.4.3 A Review on Incremental Constraint based Frequent Pattern Mining

The research community has been interested in constraint based mining to address the trade-offs of Data Mining tasks such as support vs. confidence, model expressiveness vs. compute time, etc. (Bayardo, Agrawal, & Gunopulos, 2000; Bayardo, 2006). A considerable number of research work has been done in the field of constraint based pattern mining by using algorithmic approaches (Han, Lakshmanan, Ng., 1999; Brailsford, Potts, & Smith, 1999; Bucila et al., 2003; Bonchi & Goethals, 2004; Uno, Kiyomi & Arimura, 2005; Pei et al., 2007; Raedt & Zimmermann, 2007; Ruggieri, 2010; Guns, Nijssen, & Raedt, 2011). Most of the existing algorithms require several passes to scan the dataset even though the user is interested in only few patterns.

To reduce the number of repetitive database scans, Gap with Index Apriori (GwI-Apriori), an enhancement of Gap with Apriori (G-Apriori) algorithm was introduced by Chiu, Chiu, & Huang (2009). To keep the record of the occurrences of repeating patterns, GwI-Apriori uses an Index structure, which makes the mining process faster than the G-Apriori.

However, this approach not only takes large computing time, but also does not filter out the initial undesired transactions.

A tree based monotone constraint mining defined in Knijf & Feelders (2005), is an extended work of FREQT algorithm. FREQT is a tree based incremental label ordered pattern mining algorithm (Asai et al., 2002). In FREQT the tree grows by adding new nodes in the rightmost branch of the tree. If the number of nodes are very large, then there are possibilities of growing a skewed tree, which is undesirable. RegularMine (Ruggieri,

2010) is an approach to produce concise frequent pattern which performs well in post-processing of mining frequent itemsets.

A recent approach for tree based constraint mining is MCRP-Tree (Maximum Constraint based method for generating Rare association rule Pattern mining tree) (Bhatt & Patel, 2015). MCRP-Tree takes less time to build the pattern tree as compared to Apriori-based algorithms, but multiple tree branches results more pattern mining time.

2.4.4 A Review on Incremental Educational Data Mining

Educational Data Mining (EDM) has attracted a significant attention of the researchers due to its ability to suggest the most suitable future planning by mining incremental educational data in the context of quality education (Romero & Ventura, 2013) (Jindal & Dutta Borah, 2014).

In EDM, students' modelling and Intelligent Tutoring System have attracted a considerable attention (Baker & Yacef, 2009) (Romero & Ventura, 2010). To enhance the quality of learning in higher learning institutions, the concept of predictive and descriptive models are discussed by Delavari & Phon-Amnuaisuk (2008). The predictive model predicts the success rate for individual students; individual lecturer and descriptive model describe the pattern modeling of student course enrollment, course assignment policy and behavior analysis (Lee & Chen, 2012; Nelson, Nugent, & Rupp, 2012) etc.

Antunes (2008) define constraint relaxation, a methodology to acquire background knowledge for student modelling. Similarly, Constraint-based tutoring for database related language in (Mitrovic & Ohlsson, 1999) are some of the remarkable researches in constraint based EDM.

2.4.5 Identification of Research Gaps

The above discussion brings forth the following Research gaps:

- Creation of duplicate node (e.g. IMBT)
- Greater computation time required for searching common itemsets, merging and finding a suitable path (e.g. CATs) etc.
- Consumption of large amounts of memory to construct the tree (e.g. CanTree)
- An unbalanced tree structure in which leaf nodes are not at the same level (e.g. CATS, CanTree, FUFP, Pre-FUFP etc.), requiring more times to traverse the tree.
- Performance based on static dataset where incremental mining is not possible
- Generates huge candidate sets
- Requiring multiple scans degrades the mining performance
- Not fulfilling the "Build Once Mine Many" property
- Multiple branches in tree based data structure, hence increasing branch computational time (e.g. IHUP, HUC-Prune, UP-Growth, MU-Growth, MU-Strategy1, MU-Strategy2 etc.)
- Generates large number of potentially high utility itemsets
- Applying global maximum weight to maintain downward closure property leading to defining unnecessary items for a longer period in pruning stage
- Most of the existing algorithms require several passes to scan the dataset even though user is interested in only few patterns (e.g. GwI-Apriori, FREQT, MCRP-Tree etc.), resulting in large computing time (since they do not filter out the initial undesired transactions).

2.5 Concluding the Review

From the literature review it has been established that there are problems in terms of computational and algorithmic development. Hence there is a scope for develop efficient approaches to enhance the performance of Incremental Frequent Pattern Mining.

CHAPTER-3: INCREMENTAL MINING OF FREQUENT PATTERNS

It has been observed from chapter 2 that there are problems in terms of computational and algorithmic development on traditional approaches of Incremental Frequent Pattern Mining. This chapter deals with an algorithmic approach i.e. tree based algorithm for mining frequent patterns of incremental data.

The proposed tree, "**T**ree for **I**ncremental **M**ining of **F**requent **P**attern (TIMFP)" captures the contents of the database and arranges tree nodes according to the property of binary search tree that is unaffected by changes in item frequency. It satisfies the property "Build Once and Mine Many" which is suitable for Incremental as well as Interactive Mining. We compared the proposed tree with CanTree, CATS tree and IMBT data structure. From the experimental results, it is found that the time required by the proposed tree (TIMFP) for building and mining the tree is lesser compared to related data structure.

3.1 Introduction

In real world applications, underlying database consists of incremental data which keeps changing and require continual updating. Mining frequent patterns from growing databases is costly as most of the approaches need repetitive scanning of the database as well as generating huge number of candidate sets (Han et al., 2007).

Frequent pattern mining is an important data mining technique that can be utilized to discover the frequent patterns from such data (Cameron & Leung, 2011). A pattern/item is

said to be frequent if it appears frequently in a dataset with a frequency which is greater than a user-specified threshold (Agrawal, Imielinski, & Swami, 1993) (Agrawal, & Srikant, 1994).

Finding frequent itemsets is costly during changes of the underlying database. One approach to find frequent itemsets from frequently changing databases is to construct efficient data structures. Researchers have found that tree based data structures such as CATS Tree, CanTree, FUFP, Pre-FUFP; IMBT etc. are most suitable for mining frequent patterns from incremental data.

The significant problems with the above stated data structures are as follows:

- Creation of duplicate node (e.g. IMBT)
- Greater computation time required for searching common itemsets, merging and finding a suitable path (e.g. CATs) etc.
- Consumption of large amounts of memory to construct the tree (e.g. CanTree)
- An unbalanced tree structure in which leaf nodes are not at the same level (CATS, CanTree, FUFP, Pre-FUFP etc.), requiring more time to traverse the tree.

It has been observed that problems occur in terms of computational and algorithmic development. It is important to develop efficient approaches to enhance the performance of mining.

3.2 Proposed Work

Our proposed work is designed for frequent pattern mining from incremental data. To avoid repetitive scanning and generation of test candidate sets, the proposed work constructs a binary tree based data structure and mining algorithm which is suitable for both incremental and interactive mining. In case of incremental mining, it can deal with

changes of the underlying database whereas with respect to interactive mining, the data structure will be ***built only once and mined several times*** as per requirement.

The features of the proposed data structure are:

- No duplicates arise during the construction of the tree.
- Root node is selected in such a way that, Root's left hand itemsets are always less than that of the Root node and equal/greater itemsets are on the right hand side. The benefit of selecting Root node in such a manner is that it takes less searching time to perform any operation (insert/delete/update)

3.2.1 Tree Construction Process

Let I be a set of N items $I = \{I\_1, I\_2 \ldots I\_N\}$ and a database DB= $\{T\_1, T\_2 \ldots T\_M\}$ be a set of M transactions. For each transaction T_i ($1 \leq i \leq M$), $T\_i \in I$.

Other two parameters associated with the tree construction process are Support Count (frequency of items in the database) and INFO [Node]. We have used the terminology INFO [Node] to describe the information (item name and respective support count) of a tree node, i.e. INFO [Node] = INFO [Item: Support count].

Definition 3.1: Representation of Root node

Let R be the root of a binary tree $T_{B.}$ Initially INFO [R] is empty. The Root node, R can be obtained from the item at position X of the first transaction i.e. T_1 in the original database DB. Note that T_1 should be arranged in ascending order.

Where, X = ceiling of [{least significant index position + most significant index position} / 2] and INFO [R] = [Item at X: Support count].

The "least significant index position" indicates the position of the zeroth item in T_1 and the "most significant index position" indicates the position of the N-1th item (last item) in T_1.

To illustrate the construction of the Root node, let us consider the Figure 3.1(a).

In the database DB, first transaction, T_1 consists of seven items which are unsorted i.e. T_1= {I_1, I_2, I_6, I_3, I_7, I_5 and I_4}. Place them in ascending order; T_1 becomes (I_1, I_2, I_3, I_4, I_5, I_6, I_7). Take the position of least and most significant index position (I_1=0 and I_7=6). Now find out the ceiling of [{0+6)}/2] which is 3. Item I_4 lies at 3rd index position. Hence, I_4 is the Root node (refer Figure 3.1 (b)).

Definition 3.2: Representation of Left Child of the tree

If R is a Root node at position X of the transaction, then T_L will be left child node or sub tree of the R if it is in (X-1)th, (X-2)th ---0th position. From the Figure 3.1(b), I_3 is in (3-1) i.e. 2nd position of R, hence I_3 is the left child node of R. Similarly I_2 and I_1 corresponds the left child nodes/ sub tree of R.

Definition 3. 3: Representation of Right Child of the tree

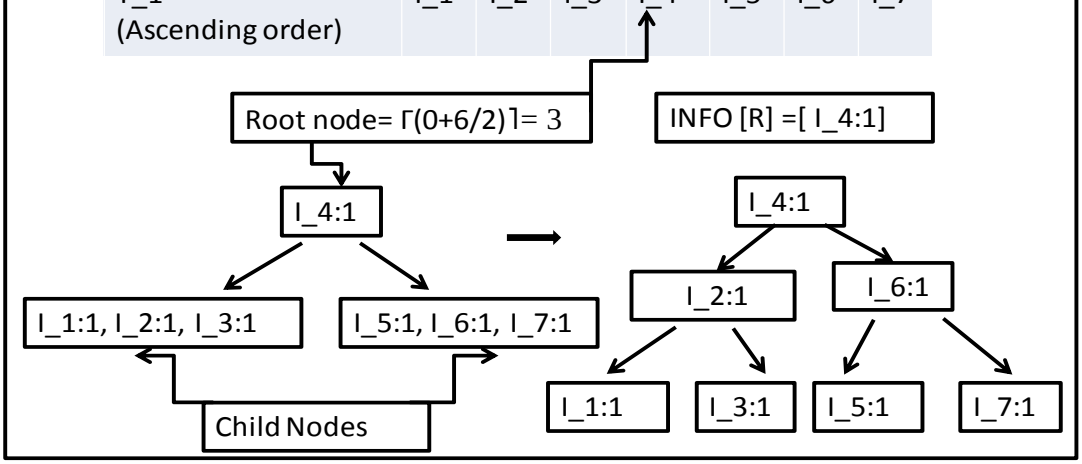
If R is a Root node at position X of the first transaction, then T_R will be right child node or sub tree of the R if it is in (X+1)th, (X+2)th --- (X+ N-1)th position. From the Figure 3.1(b), I_5 is in (3+1) i.e. 4th position. Similarly I_6 and I_7 are in the 5th & the 6th position of R. Hence I_5, I_6 and I_7 are the right child nodes /sub tree of R.

DB								
	T_5	I_2	I_6					
	T_4	I_4	I_6	I_5				
	T_3	I_1	I_2	I_3	I_4	I_7		
	T_2	I_4	I_1	I_3				
	T_1	I_1	I_2	I_6	I_3	I_7	I_5	I_4

DB :Original Database , T: Transactions, I :Itemset

(a)

Index / Transactions	0	1	2	3	4	5	6
T_1 (Unsorted)	I_1	I_2	I_6	I_3	I_7	I_5	I_4
T_1 (Ascending order)	I_1	I_2	I_3	I_4	I_5	I_6	I_7



(b)

Figure 3.1: (a) Original database (b) Representation of Root node

Further, in Figure 3.1 (b), next levels of the tree construction are done as follows:

i. We have I_1, I_2 and I_3 as left child nodes and I_5, I_6 and I_7 as right child nodes.

ii. We have applied the same procedure as defined in "*Definition 3.1: Representation of Root node"* to find out the parent node for both (left and right child nodes). The position of the parent node for the left Child nodes is 1 (ceiling of [0+2]/2. Hence I_2 is the parent node for the left sub tree. Similarly we obtained that I_6 is the parent node for the right sub tree.

iii. Further, we have applied the same procedure as stated in "*Definition 3.2*" and "*Definition 3.3*" to obtain the left hand and right hand tree for both (left and right) parent. For the parent I_2, the $(X-1)^{th}$ position is "0" which represents item I_1, hence I_1 is the left child node of I_2. Again, $(X+1)^{th}$ position of I_2 is "2" which represents the item I_3, hence I_3 is the right child node of I_2. Similarly, we obtained I_5 as left child node and I_7 as right child node of parent node I_6.

Lemma 3.1:

For any transaction in a database:

i) No duplicate nodes arise during construction of the tree and

ii) There exists a binary path which is unique w.r.t. the Root node of the tree.

Proof:

Let R is the ***Root node*** which is ***empty*** initially. Hence T_L (left sub tree) and T_R (right sub tree) = NULL and INFO [R] = NULL.

Let T_1 = {I_1, I_2---- I_M} is the first transaction taking into account to construct the tree, and is in ascending order. The Root node is chosen as per *Definition 3.1* and INFO becomes INFO [I: 1] which means the item, I is the Root node and Support Count of R is 1.

When we want to insert any item in the T_L ||T_R of the Root node, we need to check whether there exists any child node with a similar item. If exists, Support Count value is added to INFO, otherwise a new node is created and is placed in T_L ||T_R as per *Definition 3.2 & 3.3* e.g. if the order of the item is less than the R, then a new node is placed in T_L otherwise placed in T_R. For other transactions, the above mentioned approach is recursively applied to form the tree.

The Support Count of respective items will be added in the INFO of the corresponding node only. Similarly, for increment (Δ^+), corresponding support counts will be incremented, whereas for deletion (Δ^-) corresponding Support Count will be decremented. Therefore, no duplicity of nodes arises during construction of the tree. Further, the tree is formed in two ways i.e. left or right (T_L ||T_R), as a consequences, for transaction; there exists a unique binary path (at most 2 paths) w.r.t. Root node.

3.2.2 Algorithm for Tree Construction Process

This subsection briefly describes the algorithm for constructing the proposed data structure (Refer Figure 3.2). The motivation behind this algorithm is to construct a binary tree which is almost balanced in the sense that all possible leaf nodes are placed in the same level.

As an input, we have considered an original database (DB) along with incremented (Δ^+) and deleted (Δ^-) portion of DB. The output is a pattern tree.

Line 1-9 of the algorithm, tries to find out the Root node of the tree by processing the first transaction, T_1 of the DB. If an item I is a single item in T_1, then I is assigned as a Root node (line 2-4). Otherwise, T_1 is sorted according to ascending order (line 5). Root node is considered by taking the ceiling of least and the most the significant position of the item and dividing by two (line 7).

As illustrated above, the benefit of choosing Root node using this process is that all the items which is less in some order is placed in the left hand side of the root, whereas all items greater than or equal to are placed in the right side of the Root node. The information of the Root node is printed by defining the respective item and Support Count (line 9) and information on the left and right hand side of the Root node is printed as per *Definition 3.2 & 3.3* (line 10-11).

Again by scanning the rest of the transactions, Insertion and Deletion operations can be performed (line 14-20). The detail of the algorithm is shown in Figure 3.2.

```
Input : DB, Δ+, Δ-   // Where DB: Original database, Δ+: Incremental
                        portion and, Δ-: Deleted portions of database
Output:  A Pattern Tree
          1.  Begin
          2.  ∀ I, in transaction T_1 of DB do   // Root node construction
          3.  If T_1of DB contains a single item I
          4.        Print I is the root node
          5.  Else  Sort the items in T_1 according to ascending order
          6.        For i=0 to N-1 do
          7.        X = Γ(0+ (N-1)/2)
          8.        R= X [I] // Root is assigned to the Item that is in X position
          9.  Print INFO [R] = INFO [I: SuppCount]
          10. Print (X-1)th to 0th  position of root node is Left child nodes and
          11. Print (X+1)th to (X + N-1)th position of root node is right child
              nodes
          12.       End for
          13. End if
          14. Scan next transaction T_i  // Scan the next instructions
          15.        If transactions in DB // Δ+ then
          16.          Call TB_Insert () // Insert Operation
          17.        Else transactions in Δ- then;
          18.          Call TB_Delete () // Delete operation
          19.        End if
          20. End
```

Figure 3.2: Algorithm for constructing the proposed data structure, TIMFP

Procedure Insert:

Figure 3.3 describes the insertion procedure. As a first step, the scanning of the original database / incremental database is required. If there is no transaction found in the database, no 'Insert' operation takes place (line 1-5), Else, if the order of the item in every transaction to be inserted is less than the Root node, then it is placed in the left hand side

of the root (line 6-8), otherwise, the items are placed in the right hand side of the Root node (line 9-11).

If the item, I appear more than once, than the Support Count of the item increased without creating a duplicate node. Hence Support Count of INFO of **left, right** and the **Root node** increases (line 12-15).

```
Procedure T_B_Insert ( T,I, R, T_L, T_R, DB, Δ+)
1.  Begin
2.  Scan the DB/ Δ+ once
3.     If T is NULL  // if there is no transactions
4.        Return
5.     Endif
6.  Elseif order of I in T (∀ T) to be inserted < R
7.        Insert items at T_L // Insert items at L.H.S of root node
8.        Print INFO[T_L] = INFO[T_L: SuppCount]
9.        Else
10.       Insert items at T_R // Insert items at R.H.S of root node
11.       Print INFO[T_R] = INFO[T_R: SuppCount]
12. If ∀ I in ∀ T encountered >1   // if an item I encounters more than one time
                                      in all transactions
13.    Print  INFO [R] = INFO [R: SuppCount++]
14.    Print  INFO [T_L] = INFO [T_L: SuppCount++]
15.    Print INFO [T_R] = INFO [T_R: SuppCount++];
16.  End if
17. End
```

Figure 3.3: Procedure for Insert Operation

Procedure Delete:

Figure 3.4 explains the **delete** operation. First, it checks for the transactions. **I**f no transactions are available, **delete** operation is terminated in the first attempt (line 2-4). **I**f there are transactions available in the portion to be deleted from the database, then deletions take place. If the item to be **deleted** appears once or more than once, then the respective Support Count of **Root node** or **left** tree node or **right** tree node are decreased

(line 5-8). **I**f Support Count of an item is **zero**, the node is logically deleted from the list (line 9-10).

```
Procedure T_B_Delete T (I, R, INFO [], T_L, T_R, Δ¯)

1.  Begin
2.     If T is NULL  // if there is no transactions
3.        Return
4.     Endif
5.  Else if transaction T is in Δ¯
6.     Begin
7.       If item I in T encountered >=1
8.    Print  INFO [R] || INFO [T_L] || INFO [T_R] = INFO [R || T_L || T_R : SuppCount¯¯]
9.          If SuppCount of I =0 then
10. Delete I from the list
11.         End if
12.      End if
13.   End
14. End
```

Figure 3.4: Procedure for Delete Operation

3.2.3 Example: To construct tree based data structure from incremental data

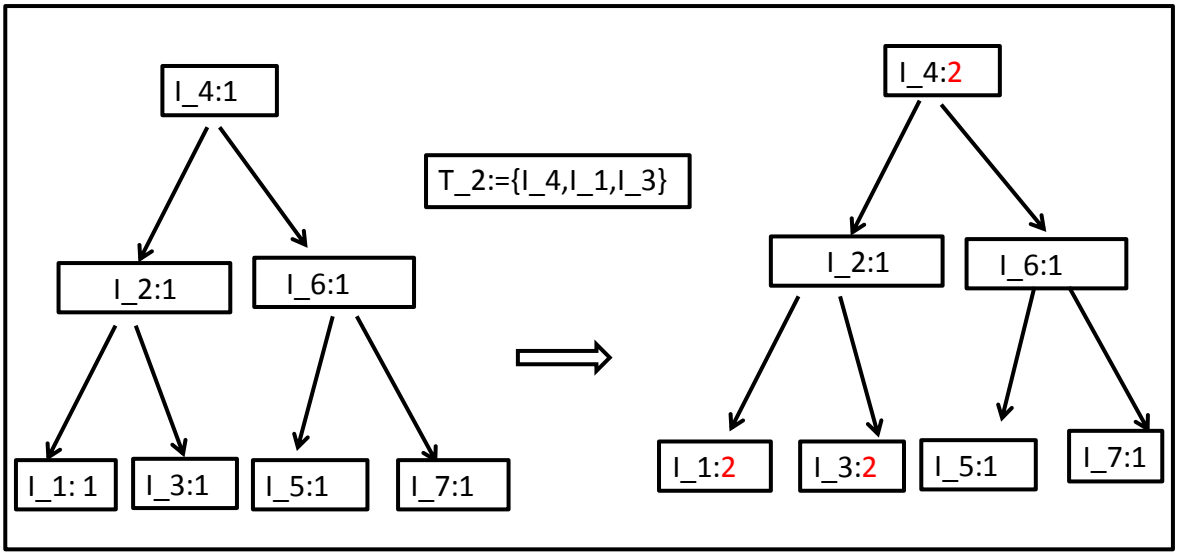
This section illustrates tree construction and related operation of the proposed algorithm. Let us consider the original database as shown in Figure 3.1 (a). The original database, DB consists of 5 transactions T_1 to T_5. The corresponding itemsets are I_1, I_2, I_3, I_4, I_5, I_6, and I_7. The construction of tree requires one scan of the original database.

Finding Root node:

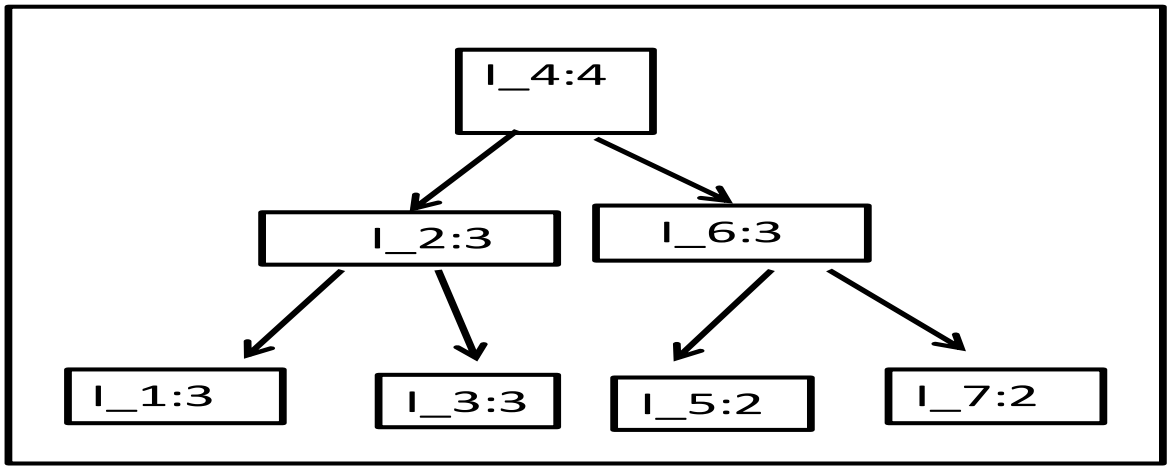
Initially Root node is empty. The Root node is obtained as per the *Definition 3.1* where Figure 3.1(b) illustrates the example of finding Root node which is I_4 and the corresponding information is INFO [I_4:1].

Insertion operation:

Finding Root node and corresponding left/ right node using transaction T_1 is shown in Figure 3.1 (b). Let us consider the next transaction T_2 which consists of 3 items, namely I_4, I_1 and I_3. Initially the INFO of I_4 was [I_4:1]. After adding the Support Count of I_4 in transaction T_2, INFO becomes [I_4:2]. Accordingly, INFO of I_1 and I_3 changes in the tree structure as shown in Figure 3.5 (a). Final tree after inserting all transactions (T_1 to T_5) in original database is shown in Figure 3.5 (b).



(a)



(b)

Figure 3.5: (a) Tree after inserting transaction T_2 (b) Final Tree in DB

Adding incremental data:

Let us consider Figure 3.6 (a) for adding incremental data. First, an incremental portion of the database (Δ_1^+) consists of two transactions T_6 and T_7. It is noted that T_6 is a short transaction containing a single item (I_9) which is initially not declared in the original database, DB.

In case of addition of transaction, there is no effect in the itemsets of the original tree. It is not necessary to rebuild the original tree, but only to update the tree. The order of Item I_9 at T_6 is greater than the Root node (I_4).

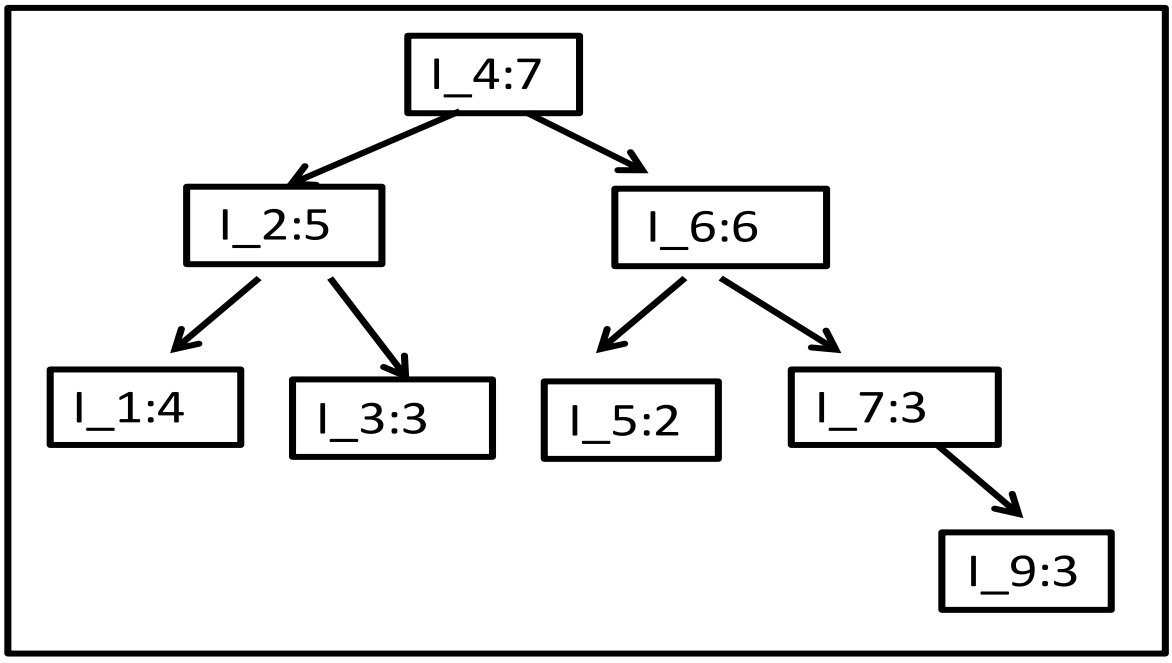
As per the *Definition 3.3*, Item I_9 is placed in the right hand side of the tree. Similarly, other transactions in Δ_1^+ and Δ_2^+ are inserted in the tree. The final tree is shown in Figure 3.6 (c).

Δ_2^+	T_9	I_4	I_6	I_9			
	T_8	I_1	I_2	I_4	I_6	I_7	I_9
Δ_1^+	T_7	I_2	I_4	I_6			
	T_6	I_9					
Δ^+ = Incremental portion of the database							

(a)

Δ_2^+	T_9	I_4	I_6	I_9				
	T_8	I_1	I_2	I_4	I_6	I_7	I_9	
Δ_1^+	T_7	I_2	I_4	I_6				
	T_6	I_9						
	T_5	I_2	I_6					
	T_4	I_4	I_6	I_5				
DB	T_3	I_1	I_2	I_3	I_4	I_7		
	T_2	I_4	I_1	I_3				
	T_1	I_1	I_2	I_6	I_3	I_7	I_5	I_4

(b)



(c)

Figure 3.6: (a) Incremental database (b) Updated database

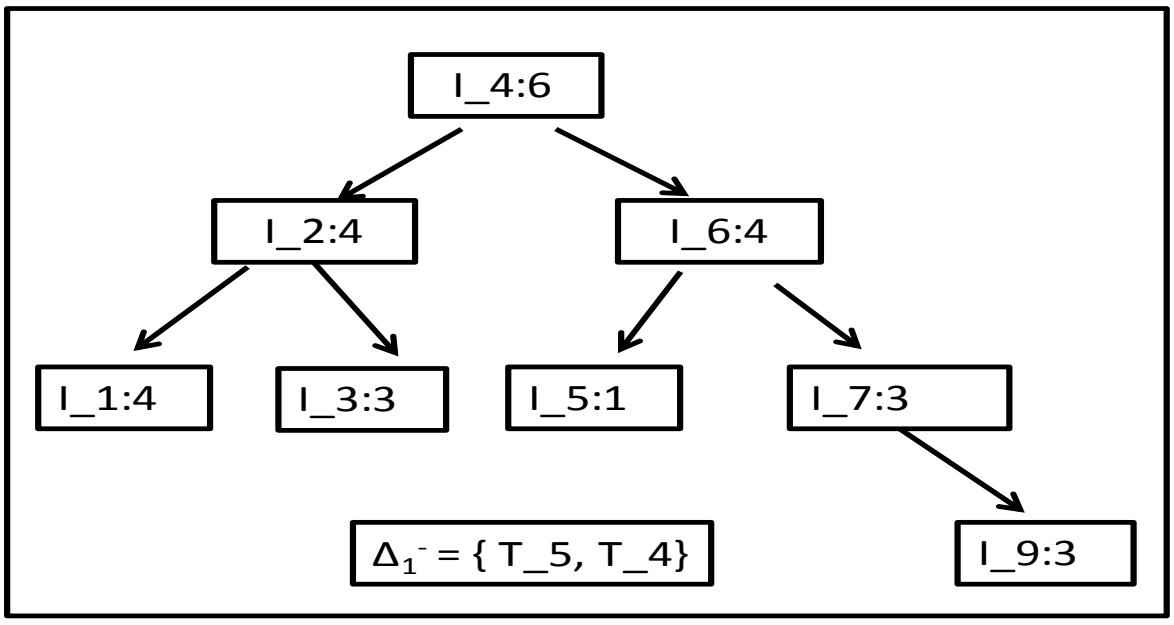
(c) Final Tree adding incremental data (Δ^+)

Deletion Operation:

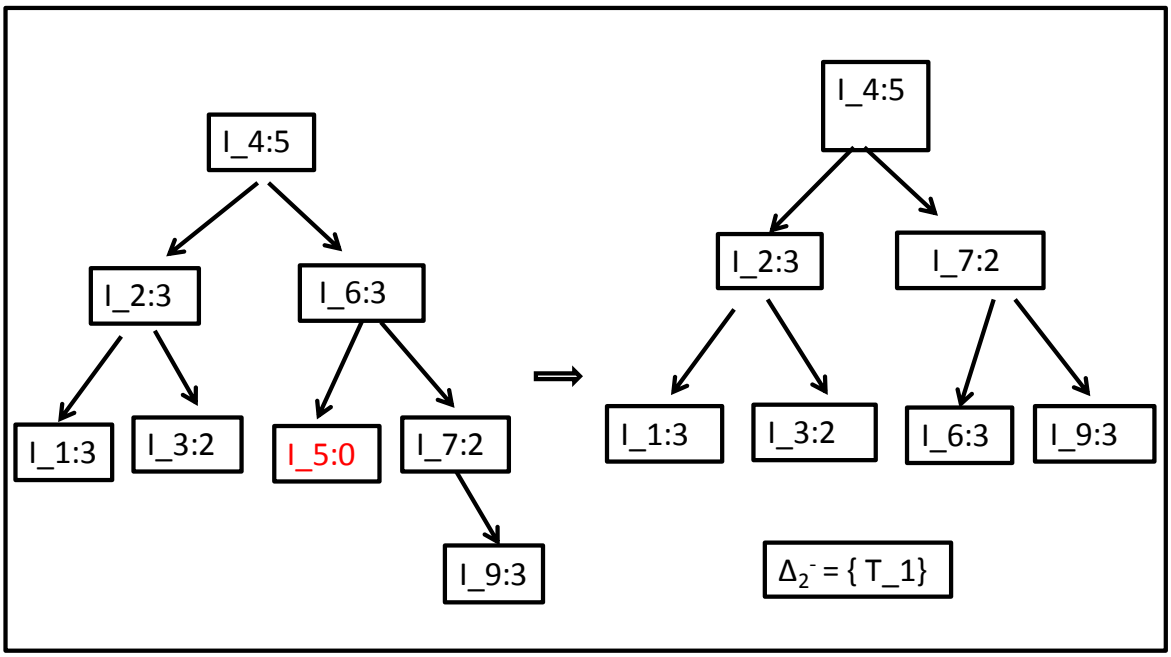
Like addition, in deletion also, there is no effect in the itemsets of the original tree. Only Support Count in INFO of the item is decremented by one for every item deletion operation. If Support Count of an item becomes zero, then the item is logically deleted from the tree.

Let us delete the transactions $\Delta_1^- = \{T\_5, T\_4\}$, $\Delta_2^- = \{T\_1\}$ and $\Delta_3^- = \{T\_9\}$ from the updated database (refer 3.6 (b)).

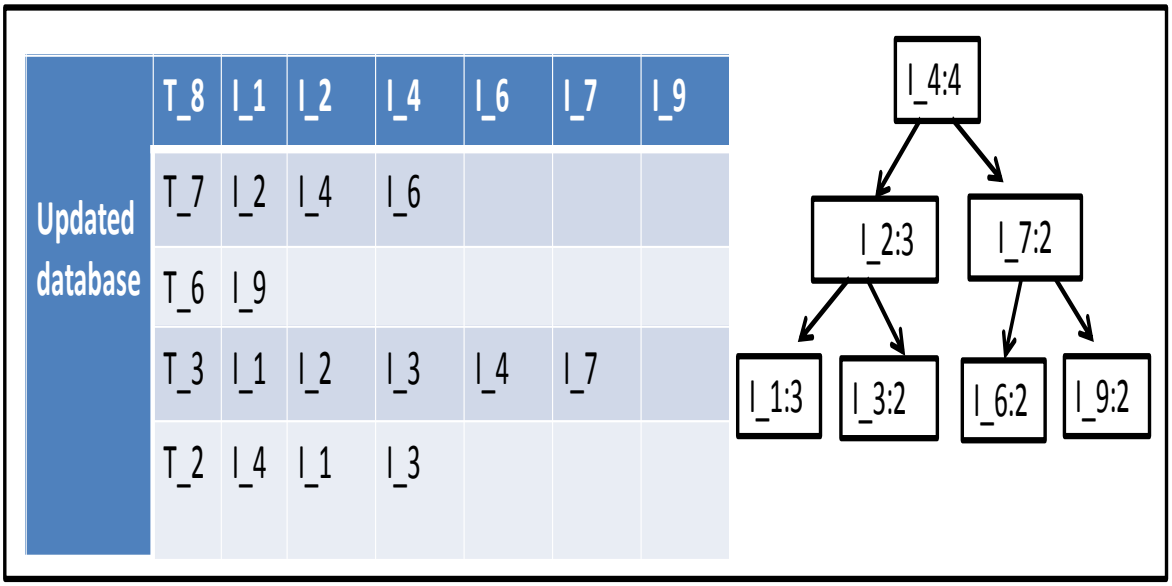
After Δ_1^- operation, it is observed that INFO [I_4:7] becomes INFO [I_4: 6] (Figure 3.7 (a)). It has been noted in Figure 3.7 (b) that item I_5 is logically deleted from the tree structure as its Support Count decreases up to zero. The final tree (after Δ_3^- operation) with updated database is shown in Figure 3.7 (c).



(a)



(b)



(c)

Figure 3.7: Deletion operations (a) Deleting T_5 & T_4 (b) Deleting T_1 and (c) Tree (after Δ_3^- operation) with updated database

3.3 Mining frequent patterns from incremental data

The proposed data structure (TIMFP) fulfils the criteria "**Build Once and Mine Many**" which means that the tree should be constructed once and it can be used for mining in future for number of times. Hence our data structure is also suitable for interactive mining. The beauty of this structure is that, frequent patterns can be mined with a variable minimum support threshold defined by user.

3.3.1 Algorithm for mining frequent patterns

The input of the algorithm is the tree data structure described in Section 3.2.2. The output of this algorithm is frequent patterns. The detail algorithm is presented in Figure 3.8.

For mining the frequent patterns of the tree, first it checks the information of the Root node (INFO[R]).

If INFO [R] is null, no further mining is required (line 2, 3), **Else** visit (traverse) the tree. Keep the INFO [R/ T_L/T_R] in the header list, if the information is greater/ equal to the user defined support threshold (UMinSupp) then update the header table (line 5-9).

This gives us the size one frequent item (line10). Next is to find out the conditional pattern base and the conditional tree of the items listed in the header table (line 14).

The conditional pattern base consists of path(s) belonging to an item (line 15, 16). It signifies the occurrence of an item. Conditional tree is derived from the conditional pattern base dataset. The way of finding out the conditional tree is similar to FP tree (Han, J. et al., 2004). Conditional tree is the tree that contains items which are greater or equal to the UMinSupp. Rest of the items are discarded (line 17-20). Frequent patterns are generated from the conditional tree (line 21-25). It is the combination of items belonging to the conditional tree for the corresponding items present in the header of the table (line 22).

```
Input: Pattern tree
Output: Frequent Patterns

1. Begin
2. If INFO [R] is NULL
3.      Print Mining not required;
4. End if
5. Else Keep the INFO [R] in the H_table; // H_table is a header table that
                                             keeps the INFO of tree nodes
6. For ∀ I in the tree do
7. If T_L || T_R are not NULL
8. Visit T_L || T_R and Update H_table = { INFO [R], INFO[T_L],INFO[T_R] } ;
9.      If INFO [R/ T_L / T_R ] >= UMinSupp then
10.      Print Size one frequent patterns ={item present in H_table};
11.     End if
12.End if
13.End for
14.Visit H_table  for mining for a size two or greater items
15.  Derive Conditional Pattern base for root node and the node belongs
     to Tree left and right
16.  Conditional Pattern Base = tree path contained in INFO[R/ T_L / T_R ]  //
     e.g. <I_1, I_2, I_4, I_7, I_9> and <I_2, I_4> are two paths for I_6
17.  Derive Conditional Tree from conditional pattern base
18.if INFO of items presents in tree path in conditional tree <= UMinSupp
19.  Discard the tree path  // e.g. I_1, I_9 < 2(UMinSupp) hence discard
     the path <I_1, I_2, I_4, I_7, I_9>
20.Else use the path for Conditional Tree // <I_2, I_4> path for I_6
21.   Update H_table with new information
22. Repeat line 11- 18 for all levels of T_L and T_R
23.Add the combination of all items presents in a particular path for a
     particular item in frequent pattern list // e.g. :frequent patterns are
     <I_2 I_6>, < I_4  I_6> and  < I_2  I_4  I_6>.
24.End if
25.End
```

Figure 3.8: Mining algorithm

Lemma 3.2:

With the increase of user defined minimum support threshold, the number of nodes participating in mining operation forming a conditional tree decreases.

Proof:

Proposed tree is an almost binary tree, hence T_L || T_R consist of at most 2 child nodes. Total nodes in T_B= (2^0 +2^1+ 2^2---+ 2^{N-1}).

Let minimum support threshold= UMinSupp and No. of nodes in the tree T_B (tree= T_L && T_R) at level 'i' is '2^i'.

For mining operation, if assigned UMinSupp= 0 then No. of nodes in T_B= (2^0 +2^1+ 2^2---+ 2^{N-1}). As keeping higher UMinSupp, the nodes with minimum INFO will be discarded

first. Hence, the nodes participating in mining operation which forms a conditional tree decrease.

3.3.2 Example: To mine frequent patterns from incremental data

Let us consider the Figure 3.7(c) to illustrate the mining example. The tree is constructed once and mining can be done several times by providing different user defined threshold (UMinSupp).

As per the algorithm, we have found that the INFO of Root node is INFO [I_4: 4], which is not NULL. Hence, mining is possible. We have added the INFO of a Root node and left/right sub tree node in the header table for construction of conditional tree (refer Figure 3.9).

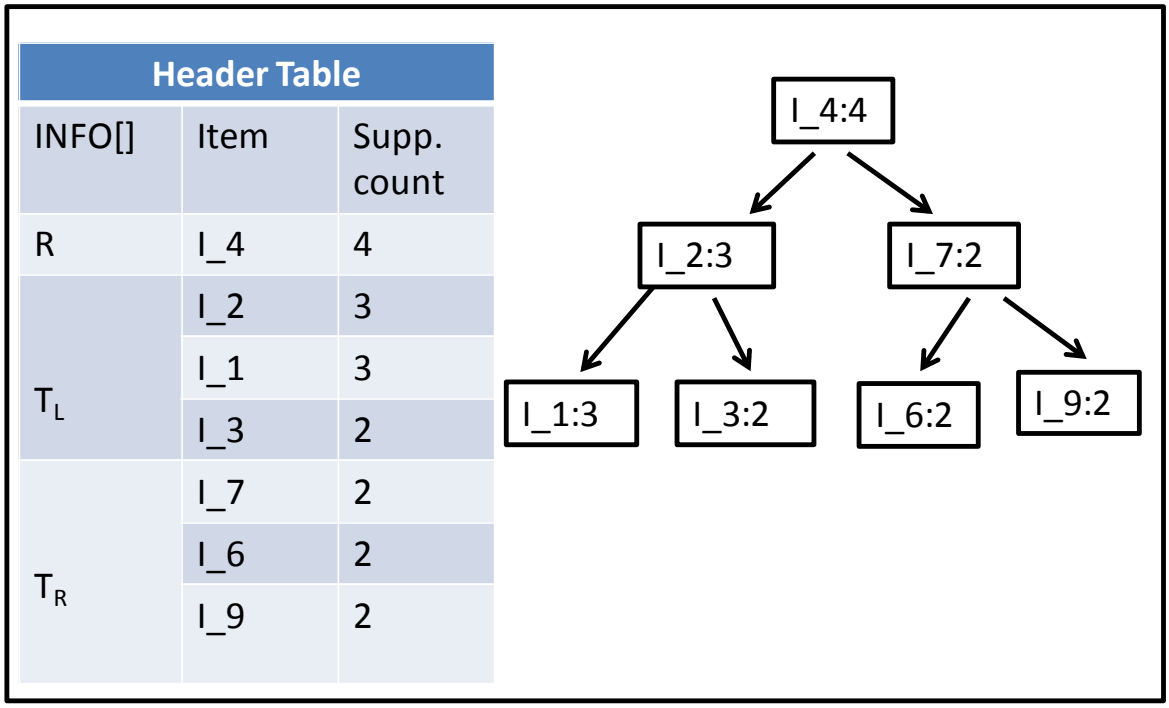


Figure 3.9: Initial header table and the tree used for mining

Let UMinSupp=2. The items shown in the header table participated in the mining operations as the INFO [R/ T_L/ T_R] > = UMinSupp. The resultant size one itemset is the

items that belong to header table. Next step is to find out the size two or more patterns from conditional tree.

As we have mentioned, the way of constructing conditional tree is similar to FP tree, hence as a first step we will find out the conditional pattern base of each item INFO listed in the header table.

Let us consider the INFO [I_6] in T_R. Conditional pattern base for I_6 is <I_1, I_2, I_4, I_7, I_9> and <I_2, I_4>.

But the conditional tree for the I_6 is only <I_2, I_4>, rest are ignored/discarded as they are less than the given UMinSupp (I_1, I_7, I_9 occurs only once). The resultant frequent patterns for I_6 are: <I_2 I_6>, < I_4 I_6> and < I_2 I_4 I_6>.

Similarly, the frequent patterns are obtained for R**oot, Left** and R**ight** sub tree nodes as shown in Table 3.1.

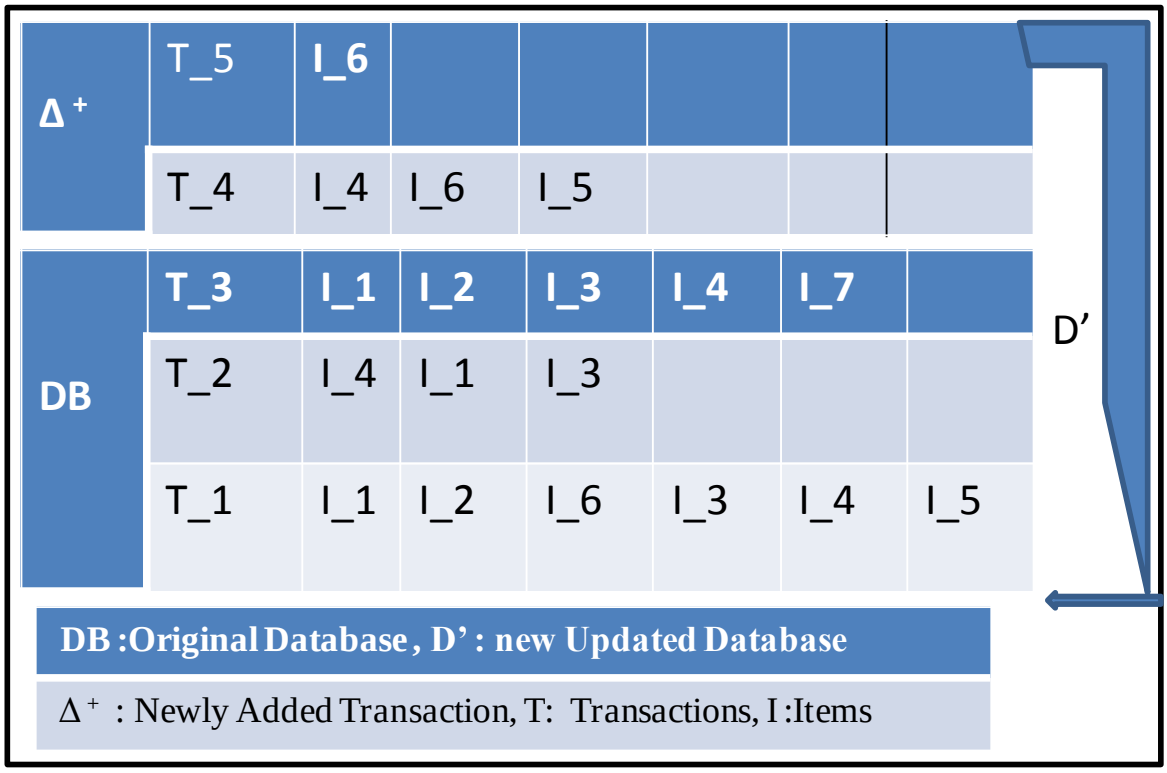
Table 3.1: Mining frequents patterns from incremental data using proposed mining algorithm

INFO []		**Conditional pattern base**	**Conditional tree**	**Frequent patterns**
R	I_4:4	<I_1, I_3>; < I_1, I_2, I_3, I_7>; <I2, I6>; < I_1, I_2, I_6, I_7, I_9>	< I_1, I_3: 2>; <I_2, I_6: 2>; <I_1, I_2, I_7: 2>	<I_1, I_4>; <I_3, I_4>; <I_2, I_4>; <I_4, I_6>; < I_1, I_3, I_4>; <I_2, I_4, I_6> ; <I_1, I_2, I_4, I_7>
T_L	I_2:3	< I_1, I_3, I_4, I_7>; < I_4, I_6>; < I_1, I_4, I_6, I_7, I_9>	< I_4, I_6: 2>; < I_1, I_4, I_7:2>	<I_2, I_4>; <I_2, I_6>; <I_2, I_4, I_6>; <I_1, I_2> ;<I_2, I_7>; < I_1, I_2, I_4, I_7>
	I_1:3	< I_3, I_4>; < I_2, I_3, I_4, I_7>; < I_2, I_4, I_6, I_7, I_9>	< I_3, I_4: 2>; <I_2, I_4, I_7: 2>	<I_1, I_3>; <I_1, I_4> <I_1, I_2>; <I_1, I_7>; <I_1, I_3, I_4>; < I_1, I_2, I_4, I_7>
	I_3:2	< I_4, I_1>; < I_1, I_2, I_4, I_7>	< I_4, I_1: 2>	< I_1, I_3>; < I_3, I_4>; < I_1, I_3>

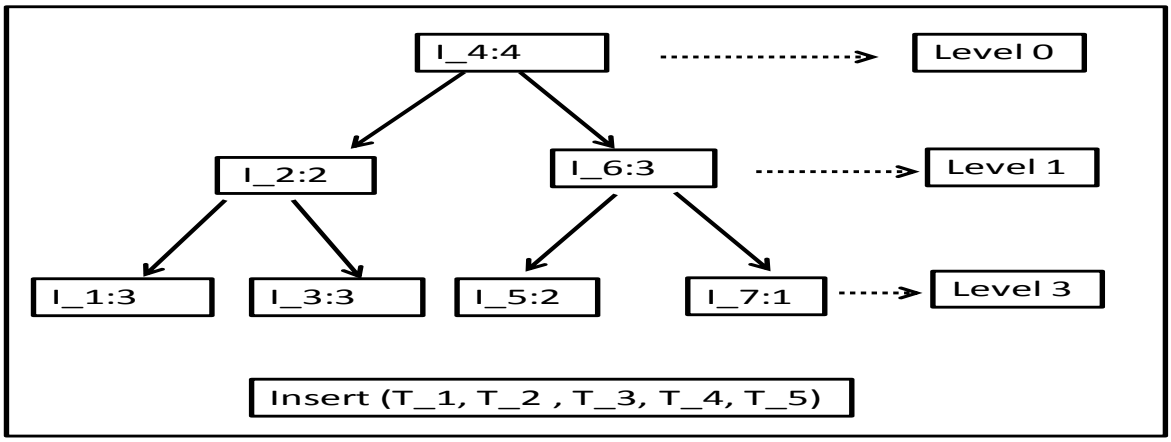
T_R	I_7:2	< I_1, I_2, I_3, I_4>; < I_1, I_2, I_4, I_6, I_9>	< I_1, I_2, I_4: 2>	< I_1, I_7>; < I_2, I_7>; < I_4, I_7>;< I_1, I_2, I_4, I_7>
	I_6:2	< I_1, I_2, I_4, I_6, I_9>; <I_2, I_4>	<I_2, I_4: 2>	<I_2, I_6>; <I_4. I_6>; <I_2, I_4, I_6>
	I_9:2	< I_1, I_2, I_4, I_6, I_7>	< - >	<I_9>

3.4 Comparative Study of proposed tree with related work

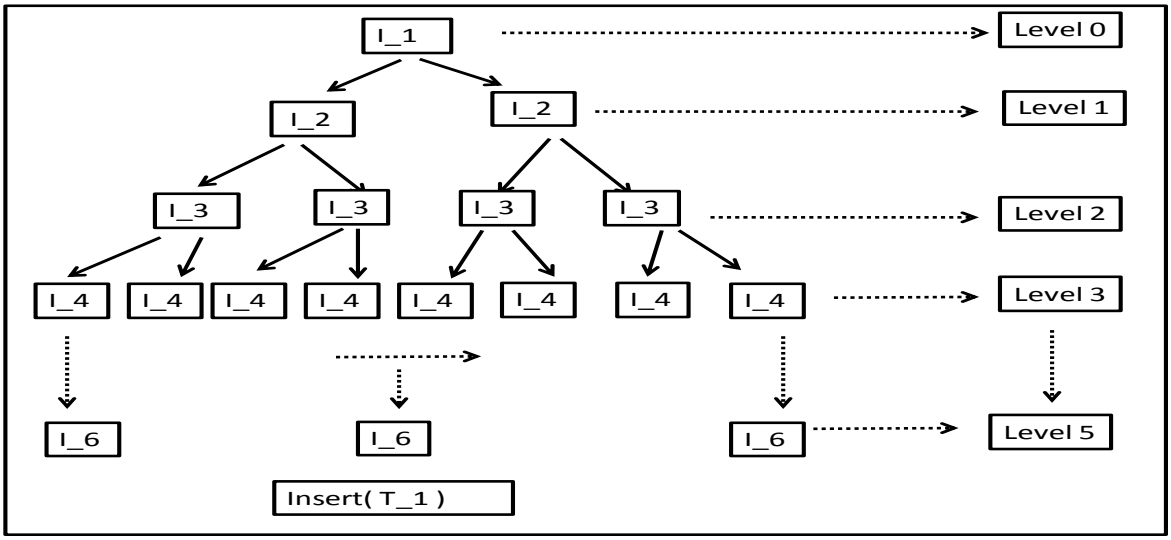
This section presents the comparative study of the proposed tree (**TIMFP**) with **CanTree, CATS tree**, and **IMBT**. Let us consider the incremental database shown in **Figure 3.10(a)** to construct the tree. We have considered only the I**nsertion** operation for each data structure in this comparison which is an identical operation for comparison among the four data structure.



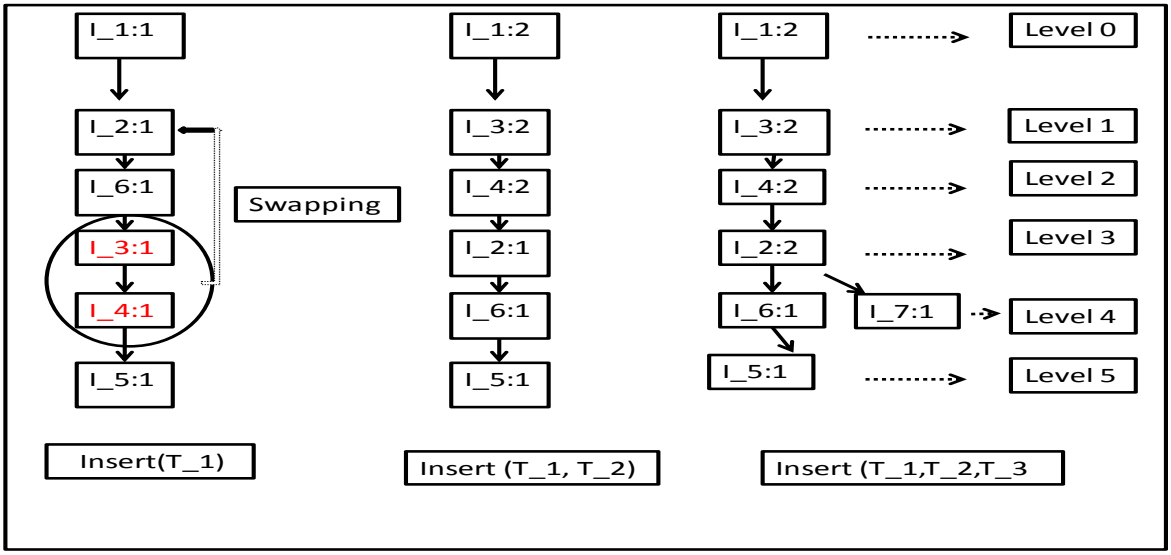
(a)


I_4:4
Level 0
I_2:2
I_6:3
Level 1
I_1:3
I_3:3
I_5:2
I_7:1
Level 3
Insert (T_1, T_2 , T_3, T_4, T_5)

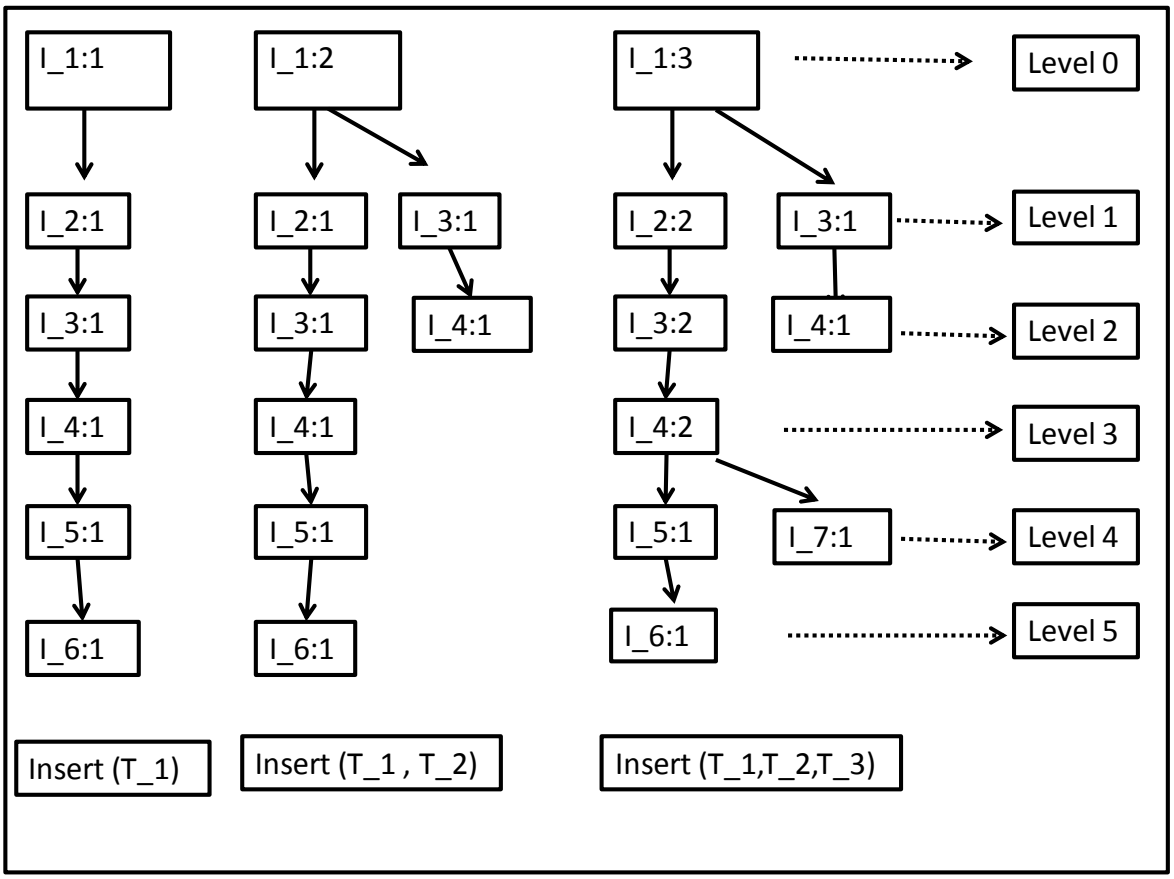
(b)


I_1
Level 0
I_2
I_2
Level 1
I_3
I_3
I_3
I_3
Level 2
I_4
I_4
I_4
I_4
I_4
I_4
I_4
I_4
Level 3
I_6
I_6
I_6
Level 5
Insert(T_1)

(c)


I_1:1
I_2:1
I_6:1
I_3:1
I_4:1
I_5:1
Swapping
Insert(T_1)
I_1:2
I_3:2
I_4:2
I_2:1
I_6:1
I_5:1
Insert (T_1, T_2)
I_1:2
Level 0
I_3:2
Level 1
I_4:2
Level 2
I_2:2
Level 3
I_6:1
I_7:1
Level 4
I_5:1
Level 5
Insert (T_1,T_2,T_3

(d)



(e)

Figure 3.10: (a) Incremental database (b) Proposed Tree (TIMFP)

(c) Partial tree of IMBT (d) CATS tree (e) CanTree

Proposed tree is shown is Figure 3.10(b) which is a binary and almost balanced tree. Partial construction of IMBT is shown in Figure 3.10(c). IMBT is partial in the sense that we have inserted only first transaction, T_1 and the tree grows with huge number of duplicate nodes (2^{Level}). This is not suitable for any long transactions.

Figure 3.10(d) shows the insertion process of **CATS** tree. We observed that the swapping is necessary to construct the tree, resulting to greater computing time. Construction of **CanTree** is shown in Figure 3.10(e). It is observed that tree level is greater than our proposed tree which takes greater time to traverse as well as to perform

other operations. It is also observed that there are possibility of developing skewed tree which is undesirable.

Comparative statement of proposed data structure is shown in the Table 3.2 and Table 3.3.

Table 3.2: Comparison of proposed data structure with related data structure

Parameters	**IMBT**	**CATS tree**	**CanTree**	**Proposed tree (TIMFP)**
Binary Tree structure	Yes	No	No	Yes
Tree branch	At least 2	More than 2	More than 2	At least 2
Leaf nodes are on the same level	No	No	No	Yes (almost)
Generation of duplicate nodes	Yes	No	No	No
Computation required for searching common itemset, paths and mining process	Short transaction, required less computation than the CATS tree as IMBT follows a binary path. But for a long transaction, required computation will be	Depend on the local support of the tree	Less than the IMBT and CATS tree, since the tree follows lexicographical order	Less than all compared data structures, since TIMFP follows a binary path without creating

	more as it creates duplicate nodes in each level of the tree			duplicate nodes
Tree size	Large as it produces duplicate nodes in both sides of the Root node of the binary tree	Compact as compared to IMBT and CanTree	Larger than CATS tree	Compact as it maintain binary order
Most suited for	Short transaction (more duplicate nodes for long transaction)	Short transaction	Short and long transaction	Short and long transaction

Table 3.3: Observations on the state of art work

Parameters	**IMBT [Figure 3.10 (c)]**	**CATS tree [Figure 3.10 (d)]**	**CanTree [Figure 3.10 (e)]**	**Proposed tree (TIMFP) [Figure 3.10 (b)]**
Number of Nodes	32	07	09	07
Height of the Tree	05	05	05	02
Level of the Tree	level 0 to level 5	level 0 to level 5	level 0 to level 5	Level 0 to level 2
Number of Transactions Inserted	01 (T_1)	03 (T_1, T_2, T_3)	03 (T_1, T_2, T_3)	05 (T_1,T_2,T_3,T_4, T_5)

From the Table 3.3, it has been observed that the proposed tree (TIMFP) has better performance than other data structures compared in this work.

In summary:

- The proposed tree has fewer nodes since it is based on the property of a binary tree.
- The height of the proposed tree is less because it follows a binary path for insertion of items, whereas other data structures compared in this work follows multiple paths or a single skewed path for I**nsertion** of items. Again, for insertion or deletion operation, only the INFO of a node (INFO [item: Support count$^{++/--}$]) is incremented or decremented without affecting the binary path, whereas for other data structures, support counts of respective nodes are incremented or decremented by effecting either local or global paths.

The only limitation of the proposed tree is that there might be a skewed path if all the items to be inserted is less or greater than the Root node which is a rare case for any transaction.

3.5 Experimental Results

3.5.1 Data and Environment for Experiment

For effective results, it is important to consider dataset from the widely accepted synthetic data source or the data collected from real life environment (Verma & Vyas, 2005) (Chang & Lee, 2005) (Hu et al., 2008). Hence we have generated synthetic data set using the same technique introduced in (Agrawal & Srikant, 1994), real data set of learning domain (Teachable Peer Learner dataset) (Matsuda & Ritter, 2011) and NU-MineBench

2.0 chain data-store dataset (Pisharath et al., 2005) to evaluate and compare the performance of the proposed data structure (**TIMFP**) with **CanTree, CATS** tree and **IMBT** data structure.

The parameters for each of the datasets are as follows:

- Dataset1: Synthetic dataset "T10-I4-D100 K" (Agrawal & Srikant, 1994), (http://fimi.ua.ac.be/data/).

  The average transaction size (T) = 10, Average size of the maximal potentially frequent/large item sets (I) = 4 and the number of transactions (D) = 100,000. The dataset was generated by setting number of items = 1000, number of maximal potentially frequent/large item sets = 2000 and 870 distinct items. This database is moderately sparse in the sense that it contains 1.14% ([Avg. Transaction/ distinct items] x 100) of distinct items in every transaction (Ahmed et al., 2012).

- Dataset2: Educational data set: Teachable Peer Learner (AlgebraI2010Dec-retry-ss) dataset (Matsuda & Ritter, 2011).

  Number of transactions = 372,519; Students' hours = 158.52 with 129 unique step knowledge components and minimum 1 step knowledge component.

- Dataset3: NU-MineBench 2.0 chain store sparse dataset (Pisharath et al., 2005) (Ahmed et al., 2009)

  Number of transactions = 1,112,949, distinct items = 46,086 with average transaction size 7.2. This database is sparse in the sense that it contains 0.0156% ([Avg. Transaction/ distinct items] x 100) of distinct items in every transaction.

The experiments were performed on an Intel 2 Duo CPU@ 2.0GHz with 3GB RAM 32 bit operating system. We used a similar development framework as used in (Yang &

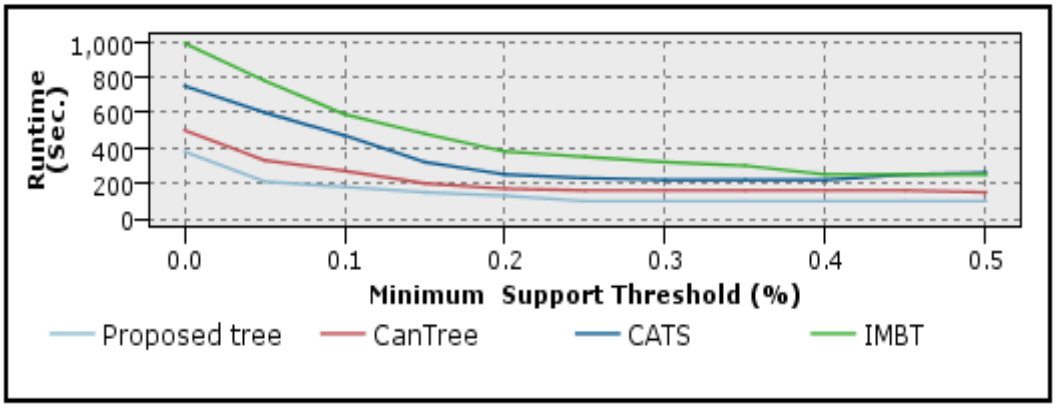
Yang, 2009). We have performed several experiments and the results are based on the average of multiple iterations.

3.5.2 Experiment 1: Performance Analysis

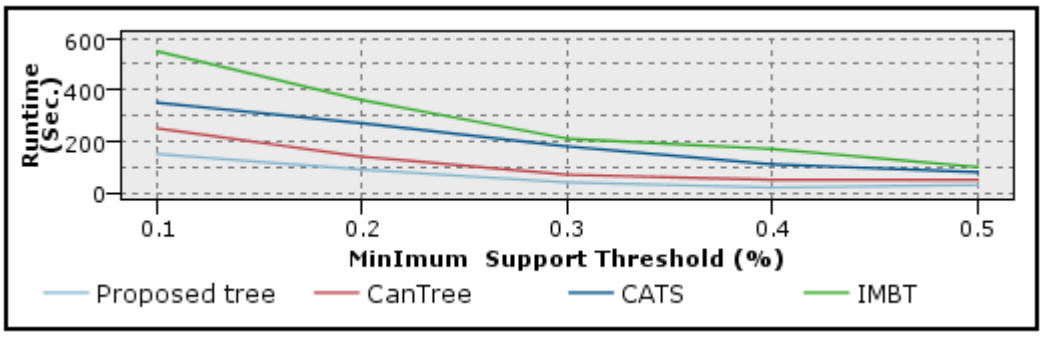
We compared the performance of the proposed data structure **TIMFP** with the following tree using variable minimum support thresholds for mining frequent patterns:

- **CanTree**
- **CATS tree and**
- **IMBT**

In our experiment we have chosen Dataset1 and Dataset2 as mentioned in section 3.5.1**.** For Dataset1 & 2, we have considered a minimum support threshold percentage (%) ranging from 0% to 0.5% similar to the experiment performed in (Leung et al., 2005). The results obtained from this experiment are shown in Figure 3.11 (a) and 3.11(b).



(a)



(b)

Figure 3.11: Effectiveness of runtime using (a) Dataset1 and (b) Dataset2

It has been found that when a minimum support threshold decreases, respective runtime increases. All the related data structure, including proposed tree, kept all items in every transaction.

IMBT took highest time as it produces duplicate nodes for each item. Similarly for both **CATS** and **CanTree**, variable tree paths (more than 2 branches) exist and require more traversing time, computation time and ultimately greater mining time.

Proposed data structure (**TIMFP**) took least run time as it is a binary tree without duplicate nodes and possible leaf nodes placed in the same level.

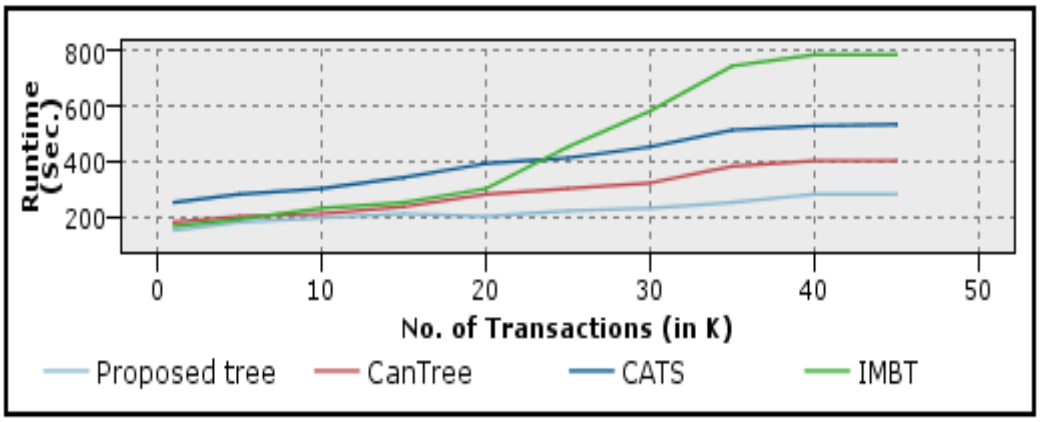
3.5.3 Experiment 2: Effectiveness of Insertion and Deletion operation

For mining incremental data, it is important to find out performance on the basis of the size of insertion (Δ^+) and deletion of transactions (Δ^-). We have considered Dataset1 and Dataset2 for this experiment. To perform the effect of Δ^+/ Δ^-, we keep the value of Δ^- as constant and by varying the value of Δ^+ and vice versa. The following parameter setting is made during this test (refer Table 3.4):

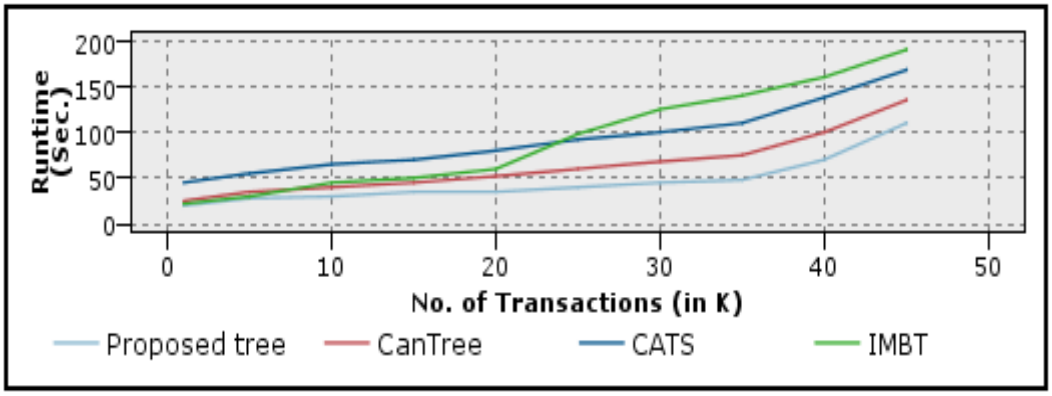
Table 3.4: Parameters setting during insertion and deletion operation

Parameters		**Dataset1**	**Dataset2**
Minimum Support Threshold		1.5%	
Initial transaction		100 K	370K
For Insertion operation	Δ^+	Variable size (in K) (1,5,10, 15, 20, 25, 30, 35, 40, 45)	
	Δ^-	Constant Size (12K)	Constant Size (12K)
For Deletion operation	Δ^+	Constant Size (12K)	Constant Size (12K)
	Δ^-	Variable size (in K) (1,5,10, 15, 20, 25, 30, 35, 40, 45)	

It is found that for both the datasets, all the tree structures are moderately efficient when 'Insert' operation of the transaction is low (e.g. 1K to 15 K). IMBT has almost similar runtime with proposed data structure with the initial construction of the tree with the least number of transitions (e.g. 1K to 05 K). When greater number of transactions inserted, it takes more time as it produces duplicate nodes. From the Figure 3.12, it is observed that proposed tree (TIMFP) is more efficient as compared to related data structures.



(a)



(b)

Figure 3.12: Effectiveness of Δ^+ using (a) Dataset1 and (b) Dataset2

Similarly, it has been observed that as the number of deleted transactions increases, the amount of time taken by the data structure is decreases in the following order:

"Proposed tree (**TIMFP**)<**CanTree**<**CATS**<**IMBT**". Since the size of the updated database decreases, proposed tree has taken lesser time.

3.5.4 Experiment 3: Scalability and Memory use of the Proposed Algorithm

Research has shown that the main memory requirement for tree data structure is low enough to use the current available memory (in GB) (Leung et al., 2007) (Ahmed & Tanbeer, 2009). Hence we have used the memory range that can cope up with current available memory.

For this experiment we have used Dataset3 which is larger than Dataset1 and Dataset2 as stated in section 3.5.1. Dataset3 has been scaled up with $\Delta^{+}x$ (range 150 to 1550) amount of transactions as shown in the **Figure 3.13.**

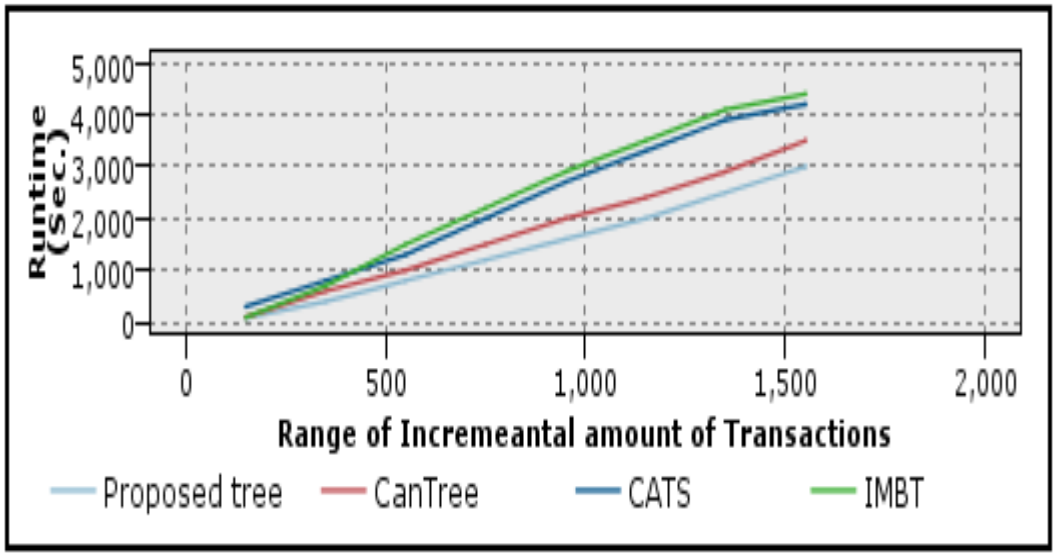


Figure 3.13: Effectiveness of $\Delta^{+}x$ amount of transactions on scalability test

From Figure 3.13, it is observed that the run time of all the data structure is linearly proportional to the Δ^{+} x, the proposed tree takes less time as compared to other data structure.

Hence from the theoretical and experimental analysis, it is found that the time required by the proposed tree (TIMFP) for building and mining is lesser compared to related tree and also suitable for incremental mining.

CHAPTER-4: INCREMENTAL HIGH UTILITY FREQUENT PATTERN MINING

Chapter 3 of the thesis proposed a tree based algorithm for mining frequent patterns of incremental data which satisfying the property "Build Once and Mine Many". The proposed work in Chapter 3 is generic in nature and can be extended for mining high utility frequent pattern.

Mining high utility patterns has become prominent since it provides semantic significance (utility/weighted patterns) associated with items in a transaction.

This chapter presents our approach for "High Utility Pattern Mining" by enhancing the concept of Incremental Mining of Frequent Patterns illustrated in chapter 3. We have introduced Average Maximum Utility (AvgMU) instead of Global Maximum Weight (GMW), which prunes the items in early stages and avoids unnecessary staying of the items in the pruning phases. Proposed algorithms for construction and mining of high utility patterns are efficient and cost effective in respect to the rest of the algorithms compared in this work.

4.1 Introduction

Most of the research on frequent pattern mining (Agrawal & Srikant, 1994; Lee, Yun, & Ryu, 2014; Pyun, Yun, & Ryu, 2014) has been applied to different kind of databases such as Transactional, Sequential, Streamline (Calders et al., 2014) and Incremental databases. The common assumption is, each item in a database is equal in weight (Chuang, Huang, & Chen, 2008). Since, some of the specific patterns may have more importance than the other patterns (Ahmed et al., 2008; Yun & Ryu, 2011; Bhandane, Shah, & Vispute , 2012; Otey

& Parthasarathy, 2004), it does not measure the semantic significances of an item as per users' perspectives.

As a matter of fact in real life applications, it is not always possible to treat all patterns in an equal or binary way. High Utility Pattern Mining has become popular in Data Mining research community as it provides semantic significance (utility/weighted patterns) associated with items in a transaction.

Some of the limitations of the existing pattern mining methods are as follows:

- Performance is based on static dataset where incremental mining is not possible. Incremental mining can be defined as the mining frequent patterns from dynamic dataset. This is a big issue for real world applications.
- Generates huge candidate sets
- Requires multiple scans hence degrading mining performance
- Does not fulfil "build once mine many" property.
- Uncertainty of processing of data due to imprecision inherent in the data (Yun, U, and Ryu, K.H., 2011).

In order to address the above mentioned shortcomings, High Utility Mining (Calders et al., 2014; Lin, Lan, & Hong,, 2012; Liu & Qu, 2012; Tseng et al., 2013; Lee, Park, & Moon, 2013; Song, Liu, & Li, 2014; Yun, Lee, & Ryu, 2014) has emerged as an important research area. High utility can be referred as the relative importance such as Interest/Intensity of each item which should be greater than or equal to the user defined minimum utility threshold equivalent to weighted frequent pattern mining in dynamic environments (Li, Yeh, & Chang, 2008; Ahmed et al., 2009; Tseng et al., 2010).

Discovering high utility pattern is highly useful in the decision making process in different domains. In the context of distance learning domain, subject credit can be

assigned as a utility value. Students can choose their high credit subject for a particular semester. For instance, the Subject 'CS' has credit value 5 which is the highest credit among the subjects listed for a semester, which can be opted by the student as per their conveniences.

In business context Cost, Profit etc. can be assigned as utility value since quantity of items does not affect the profit margin if the cost associated with items is higher.

Following are some of the examples of application areas of high utility mining:

- e-Commerce (Shie, Yu, & Tseng, 2013)
- Web click stream
- Retail & Cross sales (Barber & Hamilton, 2003), (Lee, Park, & Moon, 2013)
- Biological gene database analysis (Ahmed et al., 2009) etc.

Some of the limitations of the existing high utility pattern mining methods are:

- Multiple branches in tree based data structure, hence increasing computational time
- Generates large number of potentially high utility itemsets
- Applying Global Maximum Weight (GMW) to maintain downward closure property which requires defining unnecessary items for a longer period in pruning stage

To address the issues stated above, this work, proposes a novel binary tree based data structure for constructing tree and an algorithm for high utility pattern mining.

The focal points of the proposed method are:

- At the most two branches exist, hence less computational time
- Introducing Average Maximum Utility (AvgMU) to maintain the downward closure property which is better than Global Maximum Weight (GMW) in terms of early pruning.
- Suitable for incremental mining since the proposed method is capable of dealing with the changes of the underlying database

4.2 Proposed Work

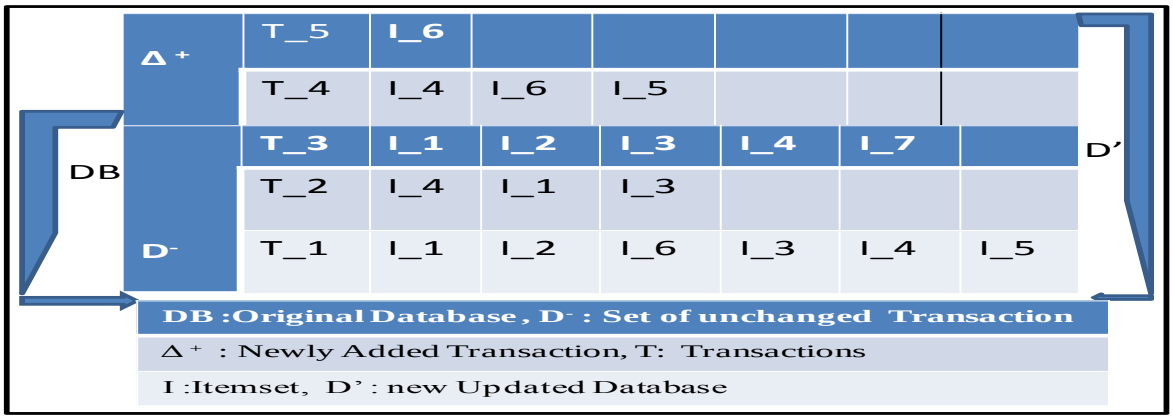
This work starts off by discussing the representation and definitions of weight, utility and, high utility followed by tree construction process with examples.

Representation of Weight, Utility/Weighted support and High Utility:

Let I be a set of N items I = {I_1, I_2…I_N} and a database DB = {T_1, T_2… T_M} be a set of M transactions for each transaction T_i ($1 \leq i \leq M$), $T_i \in I$.

Let W be a set of 'n' positive numbers called weight W = {w_1, w_2...w_n}. A pair consisting of items and corresponding weight is called a weighted item (Tao et al., 2003). For instance, (I_1, 0.75) means that item I_1 has weights of 0.75.

In general, the relative importance of items can be used as weights. When the variation of relative importance of items (e.g. Prices) is too high, it cannot be used directly as the weight (Yun, 2009). Hence normalized weight values are considered in this work. We have considered a normalized weight similar to (Yun, 2007; Ahmed et al., 2009). Figure 4.1 (a) & (b) depicts the transaction database and corresponding normalised weight table.



(a)

ITEMS	RELATIVE IMPORTANCE OF ITEMS	FREQUENCY	NORMALISED WEIGHT
I_1	750	4,000	0.75
I_2	500	2,500	0.5
I_3	800	1,000	0.80
I_4	250	10,000	0.25
I_5	300	8,000	0.30
I_6	400	20,000	0.40
I_7	600	3,500	0.60

(b)

Figure 4.1.: (a) A transactional incremental data base, (b) Items with relative importance, frequency and corresponding normalized weight

Definition 4.1: Representation of Utility: We have considered the definition of Utility (UT) and High Utility (HUT) itemset as defined in (Ahmed et al., 2009; Tseng et al., 2013) as shown in Eq. 1 below.

$$UT(I\_N) = Weight(I\_N) * Support(I\_N) \quad \text{-- (1)}$$

Where

Weight of size one item(I_N) = Normalized Weight (I_N)

Weight of two more items(I_N) = $\sum_{I\_N=1}^{N} Weight(I\_N) \ / N$

Support(I_N) = Frequency of (I_N) in DB/Δ^+/Δ^-

Example 1: Let us consider Figure 4.1 to illustrate the Eq. (1). In this example, we will calculate UT of 1-item {I_1}, 2-itemset {I_1 I_2} and 3-itemset {I_1 I_2 I_3}

Solution:

Step-1: Calculate Weight (I_N)

(a) Weight (I_1)= 0.75

(b) Weight (I_1 I_2) = (0.75 + 0.50) / 2 = 0.625

(c) Weight (I_1 I_2 I_3) = (0.75 + 0.50+0.80)/3 = 0.6833

Step-2: Calculate the support of (I_N) in DB

(a) Support (I_1) = 3

(b) Support (I_1 I_2) = 2

(c) Support (I_1 I_2 I_3) = 2

Step-3: Calculate the UT (I_N)

(a) UT (I_1) = 0.75 x 3= 2.25

(b) UT (I_1 I_2) = 0.62 x 2 =1.24

(c) UT (I_1 I_2 I_3) = 0.6833 x 2= 1.3666

Definition 4.2: Representation of High Utility Item set

An item set is said to be High Utility (HUT) item if it is greater than or equal to the user defined minimum utility threshold (UT_{min}) as shown in Eq.(2) below.

$$HUT = UT \geq UT\min \quad \text{-- (2)}$$

Definition 4.3: Representation of tree nodes

A tree node is represented by an item, Support Count and respective weight shown in Eq. (3) below.

$$NodeA = INFO[Item : SupportCount : Weight] \quad \text{-- (3)}$$

Let Item I_1 has Support Count 3 in the original database and weight is 0.75 then corresponding node is INFO [I_1: 3:0.75].

4.2.1 Tree Construction Process

The related work compared in this work is based on the concept of general tree data structure. It is observed that the number of branches grows rapidly for large databases resulting in greater computation time. To cope up with this problem, we adopt a weight ordered binary search tree i.e. tree with at most 2 branches where items with less weight are on the left hand side and greater or equal items are on the right hand side of the Root node.

The reason being used a weighted order tree is that it achieves more compression during building and mining the tree data structure.

As a first step, the process of finding out the Root node is as follows (Jindal & Borah, 2015 a):

Case I: If an initial transaction (T_1) in the original database (DB) contains items with different weight value

Solution:

Let R be the root of a binary tree T_B. Initially INFO [R] is empty. The Root node, R can be obtained from the item at position X of the first transaction i.e. T_1 in the original database DB. Note that T_1 should be arranged in weight ascending order.

Where,

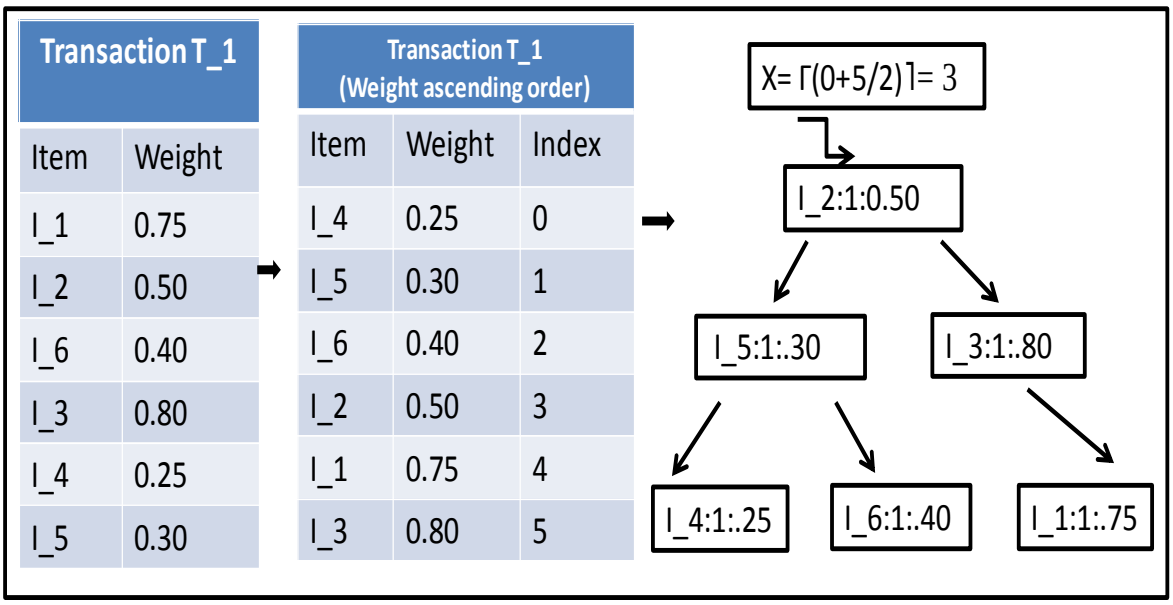
X = ceiling of [{least significant index position + most significant index position}/2] and INFO [R] = [Item at X: Support Count: Weight].

The "least significant index position" indicates the position of the zeroth item in T_1 and the "most significant index position" indicates the position of the N-1th item (last item) in T_1.

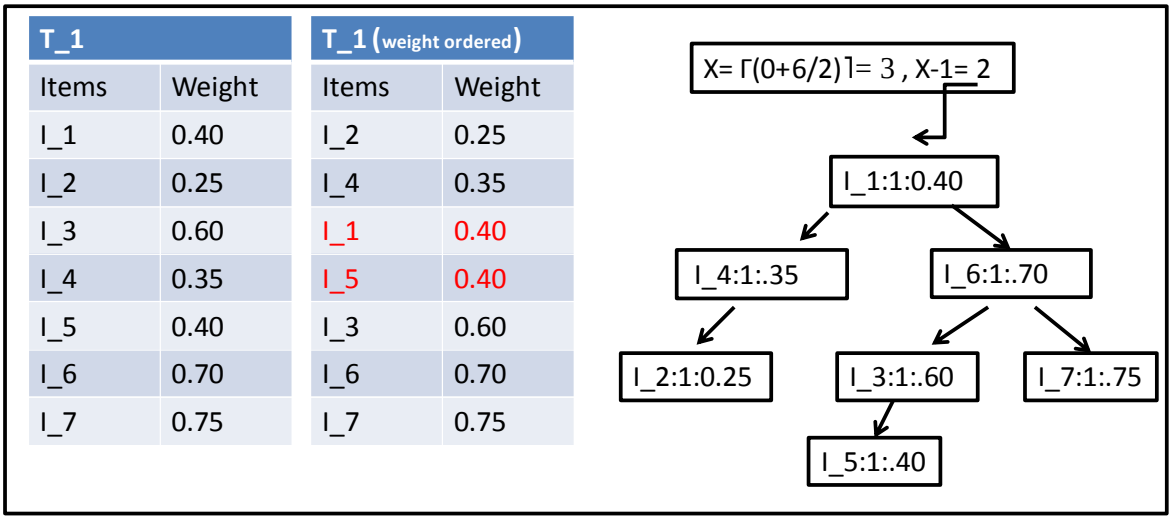
The item in that X will be the Root node. The items at (X-1)th, (X-2)th ---- 0th position will be the Left branch of the tree (T_{BL}) and the (X+1)th, (X+2)th ---- (X+ N-1)th position will be the Right branch of the tree (T_{BR}).

Example 2: To illustrate the Case-II, Let T_1 in DB consists of 6 (six) weighted items (Figure 4.2 (a)). Let the position of the items be according to weight ascending order. Hence T_1 becomes (I_4, I_5, I_6, I_2, I_1, I_3) as per weight ascending order is (0.25<0.30<0.40<0.50<0.75<0.80).

Now the X= ceiling of [(0+5) /2] which is 3. Item I_2 is at 3rd positions; hence I_2 will be the corresponding Root node. Now R= INFO [I_2:1:0.50], I_4, I_5, I_6 are the left node and I_1, I_3 are the right node.



(a)



(b)

Figure 4.2.: Root node construction as per (a) Case-I (b) Case-III

Case II: If T_1 in DB contains single item with smallest / biggest weight value.

Solution: If we consider item with smallest / biggest weight value item as a Root node, the tree will be skewed binary tree which is undesirable. Hence, proceed to the next transaction T_2 to make sure that it contains items with average or any (average, small, big) weight value and treat T_2 as the initial transaction to construct the Root node. Further construction will be done as per *Case-I*.

Case III: If T_1 in DB contains items with two equal weight values, where one lies in the Root node position and next is in left hand side of that item.

Solution: Tree construction process of this work is based on the property of binary search tree; hence the item with equal value should be placed at the right hand side of the root. Hence mark the item at $(X-1)^{th}$ position as the Root node (Figure 4.2 (b)).

4.2.2 Algorithm for Tree construction Process

This subsection briefly describes the algorithm for weighted tree as shown in the Figure 4.3.

The required input for this algorithm is original database (DB), incremental (Δ^+), deleted (Δ^-) data and weight of the items. Output is a Weighted Pattern Tree.

Line 1-20 of the algorithm tries to find out the Root node of the tree as explained in C**ase I to III**. As illustrated above, the benefit of choosing Root node using this process is that all the items which is less in some order is placed in the left hand side of the root, whereas all items greater than or equal to are placed in the right hand side of the Root node. The information of the Root node is printed by defining the respective item, Support Count and weight (line 8).

```
Input  : DB, Δ+, Δ- , W
Output: Weighted Pattern Tree

        1.  Begin
        2.  ∀ I, in transaction T_1 of DB do  // Root node construction process
        3.  If T_1of DB contains variable items with variable weights then
                                                                  // Case-I
        4.  Sort the items in T_1 according to weight ascending order
        5.      For i=0 to N-1 do
        6.          X = Γ(0+ (N-1)/2)┐
        7.          R= X [I]
        8.       Print INFO [R] = INFO [I: SuppCount: W]
        9.       Print (X-1)th to 0th  position of root node is Left child nodes and
        10.      Print (X+1)th to (X + N-1)th position of root node is right child nodes
                                                                  // Case-III
        11.     If (X-1)th position item contains similar weight value of Root node then
        12.       R=  (X-1)[I]  // Root is assigned to the Item that is in (X-1)th position
        13. Repeat line 8-10 for further construction if the tree
        14.     End for
        15.     End If
        16. End If
        17.     Else If T_1of DB contains a single item with lowest/highest weight value
        18.         Initialize T_2 as first transaction of DB   // Case-II
        19.         Repeat line 4-12 for further construction if the tree
        20.      End if
        21. If transactions in DB / Δ+ then
        22.      Call TB_Insert ()
        23. Else if transactions in Δ- then;
        24.      Call TB_Delete ()
        25. End if
    End
```

Figure 4.3: Algorithm to construct weighted tree

Again by scanning the rest of the transactions, insertion and deletion operations (on incremental data) can be performed (line 21-24) and described as follows:

***Procedure Insert* (refer Figure 4.4 (a)):**

Scan the original database/ incremental database to insert the rest of the transactions. If no transaction is found in the database, 'Insert' operation ended (line 1-5).

Else, if the order of the item in every transaction to be inserted is less than the Root node, then it is placed in the left hand side of the root (line 6-8)

Else placed in the right hand side of the Root node (line 9-11). Print the corresponding information of the node by placing the corresponding item, Support Count and weight.

If an item appears more than once, then the Support Count of the item increases without creating a duplicate node. Hence Support Count information contained in left, right and the Root node is increased (line 12-15).

Procedure T_B_Insert (T,I, R,W, T_{BL}, T_{BR}, DB, Δ^+)

1. **Begin**
2. **Scan** the DB/ Δ^+ once
3. **If** T is NULL // if there is no transactions
4. **Return**
5. **End If**
6. **Else If** order of I in T ($\forall$ T) to be inserted < R // if the ascending order items to be inserted is less than the root node, e.g. I_1 comes before I_4, where I_4 is the item defined as root node
7. **Insert** items at T_{BL} // Insert items at L.H.S of root node
8. **Print** INFO[T_{BL}] = INFO[T_{BL}: SuppCount: W]
9. **Else**
10. **Insert** items at T_{BR} // Insert items at R.H.S of root node
11. **Print** INFO[T_{BR}] = INFO[T_{BR}: SuppCount : W]
12. **If** $\forall$ I in $\forall$ T encountered >1 // if an item I encounters more than one time in all transactions , i.e. in DB, Δ^+
13. **Print** INFO [R] = INFO [R: SuppCount^{++} : W]
14. **Print** INFO [T_{BL}] = INFO [T_{BL}: SuppCount^{++} : W]
15. **Print** INFO [T_{BR}] = INFO [T_{BR}: SuppCount^{++} : W];
16. **End if**
17. **End**

(a)

```
Procedure T_B_Delete T (I, R, W, INFO [], T_BL, T_BR, Δ⁻)

1.  Begin
2.     If T is NULL  // if there is no transactions
3.        Return
4.     End If
5.  Else if transaction T is in Δ⁻
6.     Begin
7.       If item I in T encountered >=1
8.  Print  INFO [R] ||INFO [T_BL] || INFO [T_BR] =
           INFO [R||T_BL||T_BR : SuppCount⁻ : W]     // Support count decreases
9.         If SuppCount of  I =0 then
10. Delete I from the list
11.        End if
12.     End if
13.   End
14. End
```

(b)

Figure 4.4: Procedure for (a) Insert Operation (DB, Δ^+) (b) Delete Operation (DB, Δ^-)

***Procedure Delete* (refer Figure 4.4 (b)):**

If no transactions available, delete operations ended with the first attempt line (1-4). If there are transactions available in the portion to be deleted from the database (Δ^-), then deletions take place. If item encountered more than once (or may equal), then the respective Support Count of node (Root/Left/Right) decreases (line 5-8). If Support Count of an item is equal to zero, the node containing that item is logically deleted from the list (line 9-10).

Lemma 4.1: *For any weight ascending order transaction in a database, in construction of tree data structure, there exists binary path which is unique w.r.t the Root node of the tree.*

Proof:

Let R = Root node which is empty initially. Hence T_{BL} and T_{BR} = NULL and INFO [R] = NULL. Let T_1 = {I_1, I_2---- I_N} is the first transaction taking into account to construct the tree, and is in weight ascending order. The Root node is identified as per cases described in Section 4.2.1 and INFO of R becomes INFO [Item: 1: Weight] which means an item, I is the Root node with a Support Count 1 with corresponding weight. When we want to insert any item in the T_{BL} ||T_{BR} of the Root node, we need to check whether there exists any child node with a similar item. If exists, the Support Count value added to INFO, otherwise a new node is created and is placed in T_{BL} ||T_{BR}. e.g. if item < R as per weighted order, then a new node is placed in T_{BL} otherwise placed in T_{BR}. For other transactions, above approach is recursively applied to form the tree. The tree is formed in two ways left or right i.e. T_{BL} ||$T_{BR,}$ as a consequence, for any transaction; there exists a unique binary path (at most 2 paths) w.r.t. Root node.

Lemma 4.2*: For any weight ascending order transaction in a database, in construction of tree data structure, no duplicate nodes exist in the tree*.

Proof:

Let I_1, I_2---- I_N are the weight ascending itemsets in transaction T_1. The Root node is chosen as per cases described in Section 4.2.1 and INFO becomes INFO [Item:1: W] which means an item, I is the Root node with a Support Count 1 with corresponding weight. Similarly, those weights which are less than the root lie in the left side of the Root node and those which are greater/ equal to, lie in the right hand side of the Root node.

Let us consider a transaction T_2 consists of two items I_2 and I_7. Now, according to the weight I_2 and I_7 are placed in the left/right hand side of the tree and information becomes INFO [I_2] = {I_2: 1: W}, INFO [I_7] = {I_7: 1: W}. Let I_2 appears in T_3

also. Hence the Support Count of I_2 is increased by 1 and it becomes {I_2: 2: W}. Hence the node containing item I_2 will appear as a single node by increasing its Support Count without creating a duplicate node of I_2. Hence, there exist no duplicate nodes in the construction of the tree.

4.2.3 Example: To construct utility/weighted pattern tree from incremental data

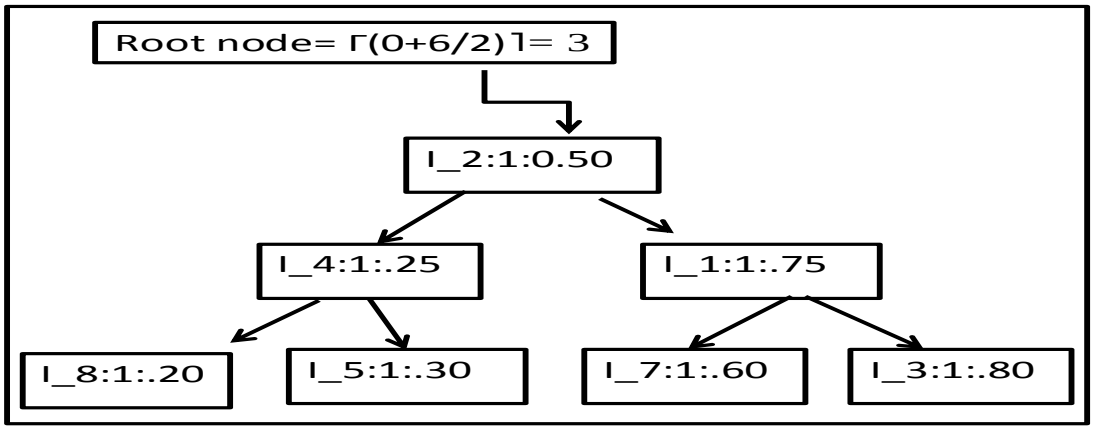
To illustrate the algorithms of tree construction, let us consider the original database with a normalized weight value as shown in Figure 4.5(a) below.

DB								
	T_4		I_2		I_4	I_5	I_6		
	T_3						I_6		
	T_2	I_1	I_2		I_4				
	T_1	I_1	I_2	I_3	I_4	I_5		I_7	I_8

DB :Original Database , T: Transactions, I:item, N.W: Normalized weight

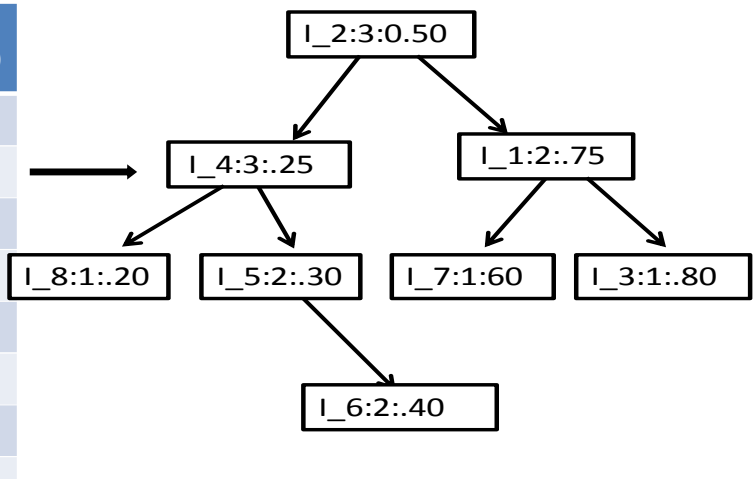
Items	N .W.
I_1	0.75
I_2	0.5
I_3	0.80
I_4	0.25
I_5	0.30
I_6	0.40
I_7	0.60
I_8	0.20

(a)



(b)

I	Supp. count	Weight (ordered)
I_8	1	0.20
I_4	3	0.25
I_5	2	0.30
I_6	2	0.40
I_2	3	0.50
I_7	1	0.60
I_1	2	0.75
I_3	1	0.80



(c)

Figure 4.5: (a) Original database (DB) (b) I_2 as Root node (c) Tree of DB

For this example, I_2 is the Root node (refer Figure 4.5 (b)). The weight of items I_4, I_8, I_5 are less than the I_2, hence left child node of I_2. Weight of I_1, I_7 and I_3 are greater than the I_2, hence right child node of I_2. Similarly, other transactions in the DB are inserted and the resultant tree is shown in the Figure 4.5 (c).

Adding incremental data:

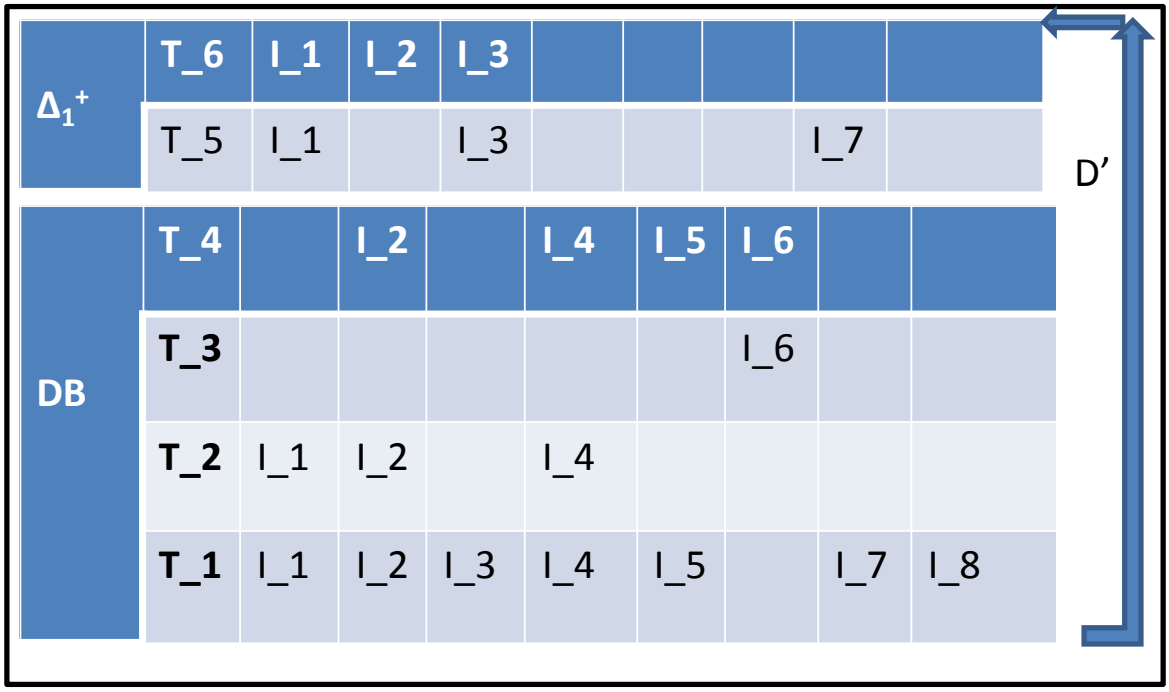
Let us insert incremental portion (Δ^+) to the original database. We have considered two possible cases as described below:

Case- I: Adding similar item sets that exist in the original data base DB

Let us consider Δ_1^+, which is the first incremental portion to the DB. The items set in Δ_1^+ are I_1, I_3 and I_7. Now DB becomes DB'(updated database) (refer Figure 4.6 (a)).

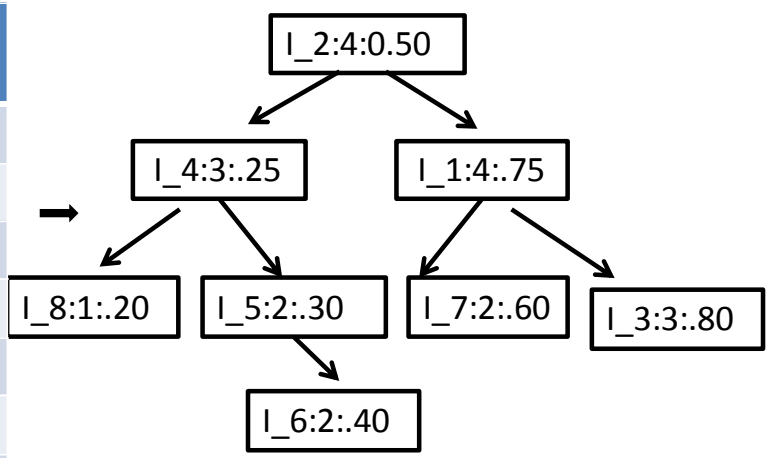
In addition of transactions, there is no effect in the itemsets of the original tree as the tree is constructed according to the weighted order. For an update of INFO of a node,

corresponding support counts are increasing in the particular node. The final tree after adding Δ_1^+ is shown in Figure 4.6 (b).



(a)

Item	Supp count	W(ordered)
I_8	1	0.20
I_4	3	0.25
I_5	2	0.30
I_6	2	0.40
I_2	4	0.50
I_7	2	0.60
I_1	4	0.75
I_3	3	0.80



(b)

Figure 4.6: (a) Adding incremental data (Δ_1^+) and (b) corresponding tree

Case- II: Adding different item with different weight value

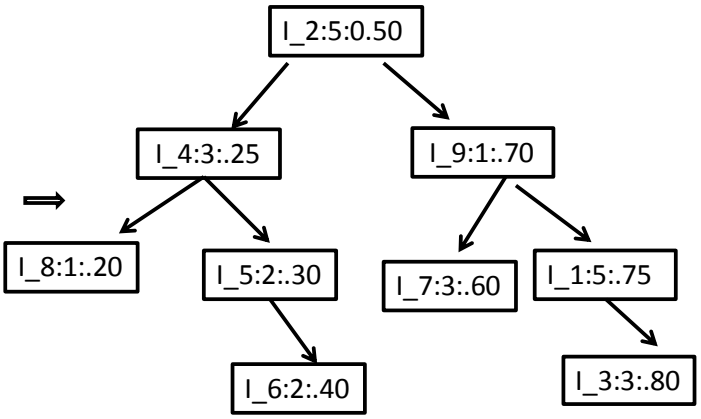
Let us consider Δ_2^+, which is the second incremental portion to the DB. The items set in Δ_2^+ are I_1, I_2, I_7 and I_9. I_9 with weight 0.70 is a different item that is not present in DB / Δ_1^+ (Figure 4.7 (a)). The resultant tree is shown in Figure 4.7 (b).

Δ_2^+	T_7	I_1	I_2					I_7	I_9
Δ_1^+	T_6	I_1	I_2	I_3					
	T_5	I_1		I_3				I_7	
DB	T_4		I_2		I_4	I_5	I_6		
	T_3						I_6		
	T_2	I_1	I_2		I_4				
	T_1	I_1	I_2	I_3	I_4	I_5		I_7	I_8

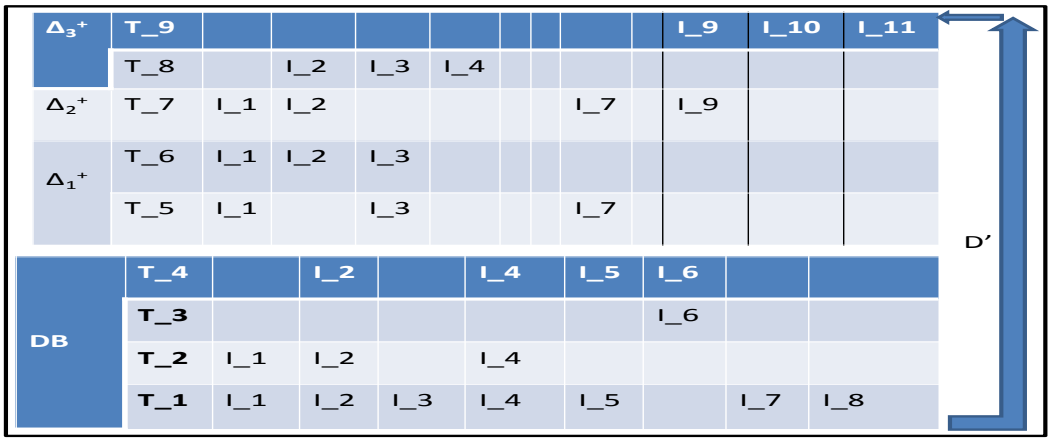
D'

(a)

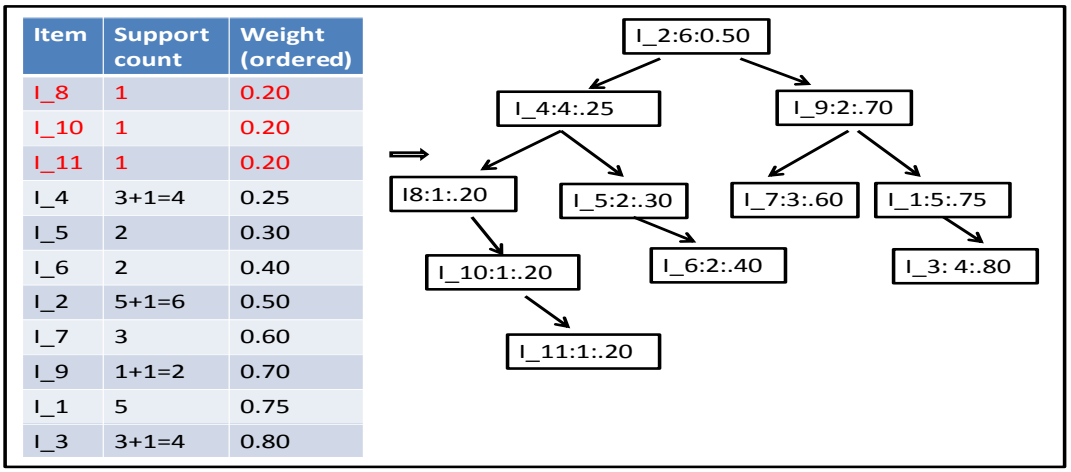
Item	Support count	Weight (ordered)
I_8	1	0.20
I_4	3	0.25
I_5	2	0.30
I_6	2	0.40
I_2	5	0.50
I_7	3	0.60
I_9	1	0.70
I_1	5	0.75
I_3	3	0.80



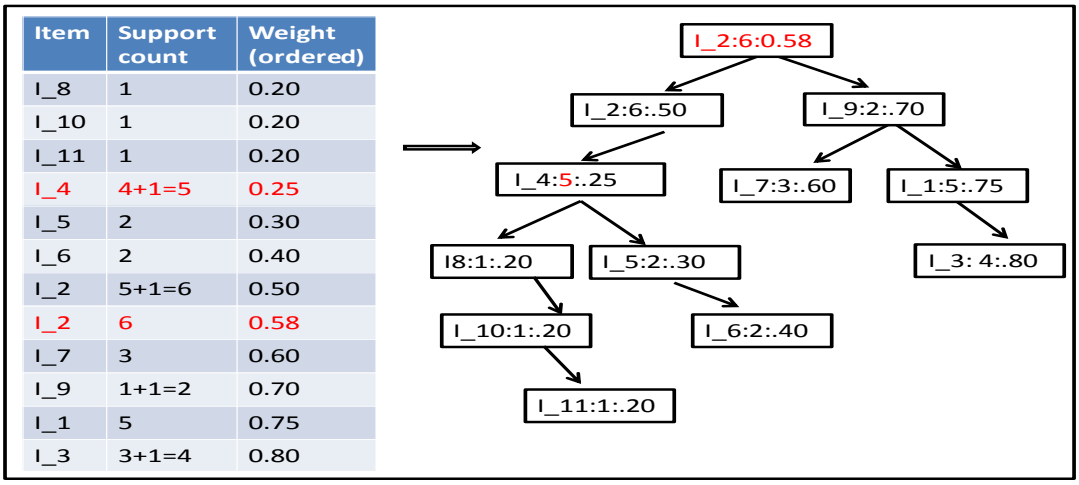
(b)



(c)



(d)



(e)

Figure 4.7: (a) Adding incremental data (Δ_2^+) and (b) corresponding tree (c) Adding incremental data (Δ_3^+) and (d) Resultant tree (e) Adding incremental data (Δ_4^+) with corresponding tree in case of changes of weight of Root Node

Case- III: Adding different item with similar frequency and weight value

Let us consider Δ_3^+, which is the third incremental portion to the DB. Δ_3^+ consists of transaction T_8 and T_9 (Figure 4.7 (c)). In T_8 there are three items I_2, I_3 and I_4 with frequency count 1 and respective weight value are assigned already in the updated dataset. In the resultant tree, only respective Support Count increases.

In transaction T_9, there are three items I_9, I_10 and I_11. Among them, two items I_10 and I_11 have similar frequency i.e. 1 and weight value, i.e. 0.20. Hence the tree grows at the right hand side of the node I_9. The resultant tree is shown in Figure 4.7 (d). Note that if the number of nodes increases with similar frequency and weight value, then local tree should be rebalanced with respect to higher frequency count.

Case- IV: Adding similar items when changes of weight take place

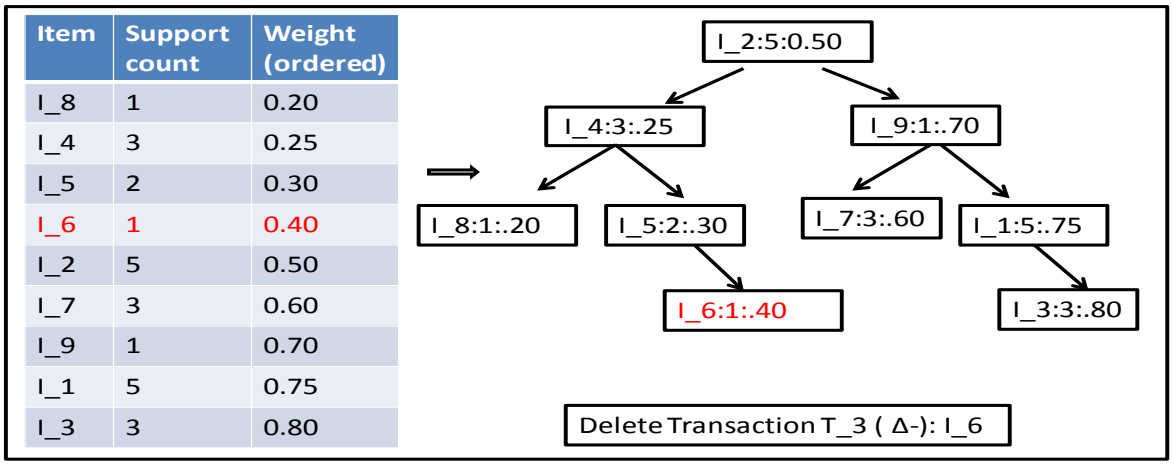
Let us consider Δ_4^+, the fourth incremental portion of the DB. It consists of one transaction T_10; where, T_10={ [I_2:6:0.58], [I_4:1:0.25]}. From the previous resultant tree (Figure 4.7 (d)), it has been observed that, Root Node is I_2 with frequency count 6 and weight 0.50 i.e., INFO [Root Node] is [I_2:6:0.50].

But from the Δ_4^+, it is seen that the weight value of I_2 changes from 0.50 to 0.58. In such type of situations, item with greater weight value will be treated Root Node and previous node will be shifted towards the left hand side of the Root Node as per the property of binary search tree. Hence I_2:6:0.58 will act as Root Node. The resultant tree is shown in Figure 4.7 (e).

Delete data from the Updated database:

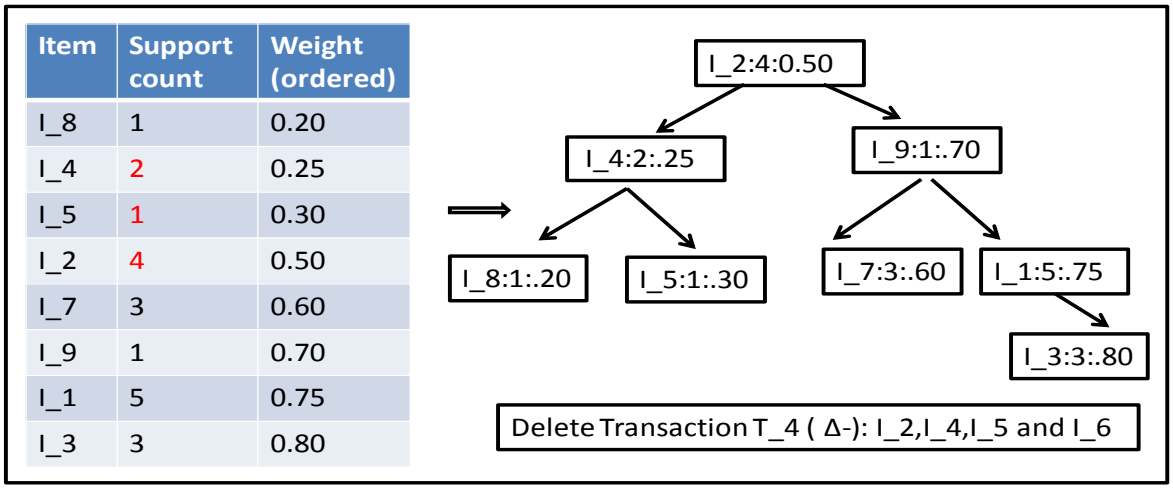
Case- I: Delete leaf node

Let us delete the transaction T_3 that consists of a single item I_6. From the tree (refer Figure 4.7 (a) & (b), it has been observed that the item I_6 is a leaf node. But INFO of I_6 is [I_6:2:.40] which means that item I_6 has 2 Support Count along with weight 0.40. But, I_6 in T_3 appears for one time. Hence we cannot delete the item directly, but Support Count of I_6 is decremented by 1 and the INFO of I_6 become [I_6: 1: 0.40]. This means that item present in T_3 is deleted, but I_6 is present in some other transaction in the database (refer Figure 4.8 (a)).



Item	Support count	Weight (ordered)
I_8	1	0.20
I_4	3	0.25
I_5	2	0.30
I_6	1	0.40
I_2	5	0.50
I_7	3	0.60
I_9	1	0.70
I_1	5	0.75
I_3	3	0.80

(a)



Item	Support count	Weight (ordered)
I_8	1	0.20
I_4	2	0.25
I_5	1	0.30
I_2	4	0.50
I_7	3	0.60
I_9	1	0.70
I_1	5	0.75
I_3	3	0.80

(b)

Figure 4.8: Tree after deleting (a) transaction T_3 and (b) transaction T_4

Case-II: Delete any node

Let us delete the transaction apply in T_4 = {I_2, I_4, I_5 and I_6}. From the Figure 4.8 (a), it has been observed that itemsets are on the left hand side of root including Root node. It is found that every itemsets Support Count decreases by one. Hence INFO of I_2 becomes [I_2:4:.50], I_4 becomes [I_4:2:.25], I_5 becomes [I_5:1:.30] and I_6 becomes [I_6:0:.40]. As the INFO [I_6] becomes zero, hence I_6 no longer valid in the tree (Figure 4.8 (b)).

Advantage of the proposed tree:

- No duplicate nodes exits
- Tree with at most two branches
- Insertion of incremental data with variable weights is possible
- Easy to Search, Insert, and Delete as the nodes with less weight lie in the left hand side of the root and greater/ equal weights lie in the right hand side of the Root node of the tree.

4.3 Mining High Utility Frequent Patterns from Incremental Data

- ***Challenges: Downward Closure Property***

The main challenge of utility mining is that it does not fulfil the downward closure property. Let us consider size one item {I_1} and {I_4} and size two item set {I_1 I_4} from the Figure 4.1 (a) and (b). As per the "Eq. (1)", UT of {I_1} = 2.25 (i.e. Weight (0.75) X Support Count (3)) and UT of {I_4} =1. 00 (i.e. Weight (0.25) X Support Count (4)). UT of {I_1 I_4} = 1.5 (i.e ([Weight of I_1 (0.75) + Weight of I_4 (0.25)] /2) X Support Count (3)). If minimum given UT threshold (UTmin) is 1.5, then the size one item {I_4} is not HUT as UT {I_4} < UTmin (i.e.1.00 <1.50) whereas item {I_1} and item set

{I_1 I_4} are HUT as UT {I_1} > UTmin (i.e 2.25>1.50) and UT {I_1 I_4} = UTmin (i.e.1.50=1.50).

- ***Solution 1: Global Maximum Weight (GMW***)

Researchers cope up with this problem by introducing Global Maximum Weight (GMW) (Yun & Leggett, 2006; Yun, 2007; Ahmed et al., 2012; Tseng et al., 2013) that is the highest normalized weight as present in the dataset. e.g. GMW from Figure 4.1 (b) is 0.80.

The downward closure property can be maintained by multiplying the Support Count of each item with GMW (Eq. (4).

$$UTnew = GMW * SupportCount \quad \text{--4}$$

Hence UTnew of item I_4 will be 3.2 (i.e. GMW (0.80) X Support Count (4)) which is HUT and cannot be pruned in early stage.

Disadvantage:

The problem of using GMW is that there is a possibility of keeping unnecessary patterns/items for a maximum period, which blocks the early pruning.

- ***Solution 2: our proposed method, Average Maximum Utility (AvgMU)***

$$AvgMU = \sum_{I\_N=0}^{N} WeightI\_N \Big/ N \quad \text{-- 5(a)}$$

$$UTnew = AvgMU * SupportCount \quad \text{-- 5(b)}$$

Let us consider the same example stated above.

We have calculated UT of {I_1} = 2.25; UT of {I_4} =1. 00 and UT of {I_1 I_4} = 1.5; If UTmin = 1.5 then size one item {I_4} is not HUT i.e.1.00 <1.50.

To satisfy the downward closure property, we have calculated AvgMu and UTnew from Eq. 5(a) and 5(b) as follows:

AvgMU = ∑ 0.75 + 0.5--- 0.60) /7= 0.514, where as GMW =0.80(i.e. the largest weight value).

UTnew of {I_4} = 0.514 X 4 =2.0; Similarly UTnew of {I_1} = 1.5 and UTnew of {I_1 I_4} = 1.5.

It has been observed that all the UTnew values are greater or equal to the user defined UTmin. Hence, the downward closure property can be maintained by applying our new approach.

Again, by applying AvgMU, we obtained UTnew of {I_4} is 2.0, whereas applying GMW, we get UT of {I_4} is 3.2.

It is observed that value obtained using AvgMU is much closer to the value of UTmin, Hence it is possible that unnecessary items can be pruned at an earlier stage (AvgMU < GMW).

So the proposed method is more applicable than the existing methods.

- ***Pruning Strategy***

Proposed algorithm performs mining HUT with only itemsets having UTnew values greater or equal to the UTmin as specified by the user or the domain expert. In this process, firstly a conditional tree is created from the conditional pattern base by pruning the items having UT values less than the UTmin.

Secondly, UTnew is calculated by the product of "AvgMU and Support Count" to prune the candidate items from the tree path. Again the items that have lower UTnew than the UTmin are pruned.

4.3.1 Algorithm for mining utility/weighted frequent patterns

To pursue this mining process, first we have adopted a Weighted Pattern Tree to show the tree construction phase efficiently. The output of this algorithm is HUT. This tree data structure can be mined many times as per the variable minimum support threshold defined by user which it is suitable for interactive mining. The mining algorithm is described in Figure 4.9.

For mining the frequent patterns of the tree, first it checks the INFO [] of the Root node. If INFO [] of the Root node is null, no further traversing is required (line 2, 3), otherwise the Root node is noted in the header table (line 5).

Multiply supports of items that are present in INFO [Node] with AvgMU to get new weight (line 6,7). Print itemsets presents in new weight as a candidate key whose new weight is greater or equal to the minimum utility support threshold (UTmin) (line 9). Update header table (line 9-13).

Derive conditional pattern base by obtaining the tree path belonging to an item. Derive the conditional tree and pattern from conditional pattern base (line 15-16).

If the utility (UT) of conditional patterns is less than the UTmin, prune the tree (line 17-18), Else update the header table (line 19). Add the combinations of all items for a particular path as a HUT (line 20).

Lemma 4.3:

With the increase of user defined minimum utility support threshold, the INFO [] of nodes i.e. Support Count and nodes participating in mining operation in T_{BL} || T_{BR} decrease.

Proof.:

Let minimum utility support threshold= UTmin and number of nodes in T (T_{BL} && T_{BR}) at the level i is 2^i. Total nodes in Tree= ($2^0 + 2^1 + 2^2 \text{---} + 2^{N-1}$). Proposed tree is a binary tree; hence T_{BL} || T_{BR} consists of at most 2 child nodes which are arranged in weighted order. For mining operation, if assigned UT_{min} = 0 then no. of nodes in T = $2^0 + 2^1 + 2^2 \text{---} + 2^{N-1}$). Else if UT_{min} is greater than zero, Support Count of particular node INFO [] will be decreased. Hence total nodes in T become ($2^0 + 2^1 + 2^2 \text{---} + 2^{N-1}$) which is less than T= ($2^0 + 2^1 + 2^2 \text{---} + 2^{N-1}$). Similarly, keeping higher UT_{min}, number of nodes in tree will be decreased.

Input: Weighted Pattern Tree
Output: HUT

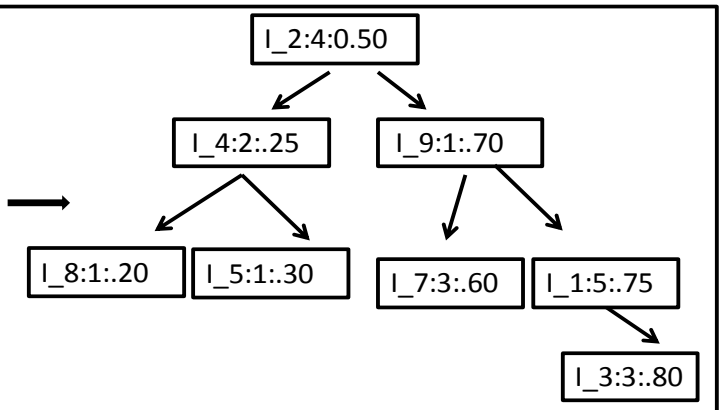
1. **Begin**
2. **If** INFO [R] is NULL // R is the root node
3. Print mining not required;
4. **End if**
5. **Else** Keep the INFO [R] in the H_table; // H_table is a header table that keeps the INFO of tree nodes
6. **For** ∀ I in the tree do
7. **Calculate** new weight (Supp count in INFO[Node] x AvgMU)
8. **If** new weight of items >= UT_{min}
9. **Print** items as a candidate keys
10. **Update** H_table by adding the items
11. **Print** Size one frequent patterns ={item present in H_table}; //
12. **End if**
13. **End for**
14. Visit H_table for mining for a size two or greater items
15. Derive conditional pattern base where conditional pattern base = tree path contained in INFO[R/ T_{BL} / T_{BR}]
16. Derive conditional tree and candidate pattern from conditional pattern base
17. **if** UT of candidate patterns <= UT_{min}
18. Pruned the tree;
19. **Else Update** H_table with new information
20. Add the combination of all items presents in a particular path for a particular item in high utility pattern/ weighted patterns (HUT)
21. **End if**
22. **End**

Figure 4.9: High Utility Pattern Mining algorithm

4.3.2 Example: To mine high utility patterns from incremental data

Let us consider the Figure 4.10 (a) and (b) to illustrate the mining of the example. The tree is constructed only once and by providing different user defined threshold (UTmin), mining can be done for several times.

Item	Support count	Weight (ordered)
I_8	1	0.20
I_4	2	0.25
I_5	1	0.30
I_2	4	0.50
I_7	3	0.60
I_9	1	0.70
I_1	5	0.75
I_3	3	0.80



(a)

AvgMU:

(0.20+0.25+0.30+0.50+0.60+0.70+0.75+0.80)/8 =**0.51**

GMW:
= **0.80**

Item	Normalized Weight	New Weight using AvgMU= (Supp Count x AvgMU)	New Weight using GMW = (Supp Count x GMW)
I_8	0.20	1 x 0.51=0.51	1 x 0.80 = 0.80
I_4	0.25	2 x 0.51=1.02	2x 0.80 =1.60
I_5	0.30	1 x 0.51=0.51	1x 0.80 = 0.80
I_2	0.50	4 x 0.51=2.04	4x 0.80 =3.20
I_7	0.60	3 x 0.51=1.53	3x 0.80 = 2.40
I_9	0.70	1 x 0.51=0.51	1 x 0.80 = 0.80
I_1	0.75	5 x 0.51=2.55	5 x 0.80 = 4.00
I_3	0.80	3 x 0.51=1.53	3 x 0.80 = 2.40

(b)

Figure 4.10: (a) Tree for mining frequent patterns (b) New weight using AvgMU and GMW

As per the mining algorithm, first we check the Root node of the original tree. The INFO [R] = INFO [I_2:4:0.50] which is not NULL, hence further mining will take place.

Add INFO [R] to header table. Now calculate AvgMU which is 0.51. Next step is to multiply AvgMU with Support Count of the itemsets. The new generated weights are shown in Figure 4.10 (b). For comparison, new weights using GMW have also been calculated.

- **Generation of candidate patterns**

Generation of candidate patterns using different UTmin is important in mining operations. We have considered three cases as shown Table 4.1, 4.2 and 4.3:

Case –I: Let the value of UT_{min} is small one

Table 4.1: Generation of candidate patterns with small UT_{min}

	UT_{min}	Candidate patterns (Size one)	Remarks
Using AvgMU	< 0.50	I_8, I_4, I_5, I_2, I_7, I_9, I_1, I_3	Generates ***an equal number*** of candidate keys for both AvgMU and GMW
Using GMW		I_8, I_4, I_5, I_2, I_7, I_9, I_1, I_3	

Case –II: Let the value of UT_{min} is medium one

Table 4.2: Generation of candidate patterns with medium UT_{min}

	UT_{min}	Candidate patterns (Size one)	Remarks
Using AvgMU	0.80	I_4, I_2, I_7, I_1, I_3	Generates ***less*** candidate keys by using AvgMU
Using GMW		I_8, I_4, I_5, I_2, I_7, I_9,I_1, I_3	

Case –III: Let the value of UT_{min} is large one

Table 4.3: Generation of candidate patterns with large $\boldsymbol{UT_{min}}$

	UT_{min}	Candidate patterns (Size one)	Remarks
Using AvgMU	1.5	I_2, I_7, I_1, I_3	Generates ***less*** candidate keys by using AvgMU
Using GMW		I_4, I_2, I_7, I_1, I_3	

It has been observed that for two cases II and III mentioned above, less candidate patterns have been generated using AvgMU. Hence the proposed data structure fulfills the criteria of generating less candidate patterns as compared to other data structure stated in this work.

- **Generation of high utility patterns**

Our goal is to find HUT. Let us consider the candidate patterns {I_2, I_7, I_1, I_3} as obtained from Table 4.3 using AvgMU. These keys are added to the header table for the generation of conditional trees and to find HUT (Table 4.4). The process of obtaining conditional tree is same as FP tree. We have considered minimum support threshold of 2 to illustrate this example.

Result: It has been observed that item sets {I_2 I_3}, {I_2 I_7}, {I_3 I_7}, {I_1 I_2 I_7} and {I_1 I_3 I_7} has been pruned. As a result, high utility pattern/ weighted patterns (HUT) are: {I_1}, {I_2}, {I_3}, {I_7}, {I_1 I_2}, {I_1 I_7}, {I_1 I_3}, and {I_1 I_2 I_3}.

Table 4.4: Example of mining process of candidate patterns {I_2, I_7, I_1, I_3}

Candidate Keys	Conditional pattern base	Conditional tree	Candidate pattern	New Utility(UTnew)	Remarks
I_2	<I_1 I_3 I_4 I_5 I_7 I_8>, <I_1 I_4>, <I_1 I_3>, <I_1 I_7 I_9>	<I_1 I_3>, <I_1 I_7>	<I_1 I_2>, <I_2 I_3>, <I_2 I_7>	UTnew {I_1 I_2} = 2.5 UTnew {I_2 I_3}=1.3 UTnew {I_2 I_7}=1.1	Passed Pruned Pruned
I_7	< I_1 I_2 I_3 I_4 I_5 I_8>, <I_1 I_3>, <I_1 I_2 I_9>	<I_1 I_3>, <I_1 I_2>	<I_1 I_7>, <I_3 I_7>, <I_2 I_7>	UTnew {I_1 I_7}=2.02 UTnew {I_3 I_7}=1.4 UTnew {I_2 I_7}=1.1	Passed Pruned Pruned
I_1	<I_2 I_3 I_4 I_5 I_7 I_8>,<I_2 I_3>, <I_3 I_7>, <I_2 I_4>, <I_2 I_7 I_9>	<I_2 I_3>, <I_3 I_7>, <I_2 I_4>, <I_2 I_7>	<I_1 I_2>, <I_1 I_3>, <I_1 I_7>, <I_1 I_2 I_3> , <I_1 I_2 I_7>, <I_1 I_3 I_7>	UTnew {I_1 I_2} = 2.5 UTnew {I_1 I_3}= 2.3 UTnew {I_1 I_7}=2.02 UTnew {I_1 I_2 I_3} =2.05 UTnew {I_1 I_2 I_7}=1.23 UTnew {I_1 I_3 I_7} =1.43	Passed Passed Passed Passed Pruned Pruned
I_3	<I_1 I_2 I_4 I_5 I_7 I_8>, <I_1 I_7>, <I_1 I_2>	<I_1 I_7>, <I_1 I_2>	<I_1 I_3>, <I_3 I_7>, <I_2 I_3>	UTnew {I_1 I_3}= 2.3 UTnew {I_3 I_7}=1.4 UTnew {I_2 I_3}=1.3	Passed Pruned Pruned

4.4 Experimental Results

The purpose of this experiment is to evaluate and compare the performance of the proposed algorithm with state of the art algorithms UP-Growth, UP-Growth+, HUI-Miner, MU-Growth Strategy1, MU-Growth Strategy2.

4.4.1 Data and Environment for Experiment

Researchers found that widely-accepted synthetic data source or the data that is collected from real environment are most suitable for performance evaluation of the proposed algorithm (Verma & Vyas, 2005; Sun & Bai, 2008). To see the effect of variable dataset as well as to indicate the strength of our proposed algorithms, we have considered sparse (less items per transactions and with large number of distinct items) and dense (many items per transactions and with small number of distinct items) dataset (Ahmed et al., 2012).

- **Synthetic dataset**(http://fimi.ua.ac.be/data/)**:**
    - T10-I4-D100 K: Taking into consideration of the above statements, we have considered "T10-I4-D100 K" which is moderately sparse, widely accepted synthetic dataset for training purpose.
- **Real datasets** (http://fimi.ua.ac.be/data/):
    - Pumsb: This dataset contains census data for population and housing, which is dense in nature.
    - Connect: This dataset contains online connection data, which is dense in nature.
    - Kosarak: It consists of click-stream data collected from Hungarian on-line news portal, which is sparse in nature.

- NU-Mine Bench 2.0 Chain Store dataset: This dataset contains Chain store data which are sparse in nature (Pisharath et al., 2005).

- **Educational domain (learning) datasets:**
  - AIEEE 2007: It is a student data set seeking admission in a particular branch through All India Engineering Entrance Examination (AIEEE). Therefore, it helps predict the approximate strength of student in the particular branch. This data set is collected from NIC Delhi. In order to demonstrate the usefulness of our high utility mining process, we have assigned random numbers in student ranks and their preferences.
  - StatWay - Fall 2011 - Austin Community College dataset (Koedinger et al., 2010): This dataset consist of students and tutor participation data in online study courses.

The dataset parameters are shown in Table 4.5 and related characteristics of different datasets are shown in Table 4.6, 4.7 and 4.8.

Table 4.5: Dataset parameters

Parameter	Meaning
$\|s\|$	Size of the Dataset
$\|T\|$	Number of transactions in transaction set
$\|I\|$	No. of Items
$\|t\|_{avg}$	Average size of a transaction
$\|t\|_{max}$	Maximum Transaction length
D_i *(%)*	The percentage of Distinct item present in every Transaction: [(|t|avg / |I|) x 100]
Type	Dense/Sparse:

	• Depends on the value of D_i *(%)* • A dataset becomes ***dense*** when it contains many items per transactions else it is ***sparse.***

Table 4.6: Characteristics of real datasets

Dataset	$\|s\|$ (in MB)	$\|T\|$	$\|I\|$	$\|t\|_{avg}$	$\|t\|_{max}$	D_i *(%)*	Type
Pumsb	14.75	4.9×10^4	2113	74	74	3.50	Dense
Connect	12.14	6.8×10^4	150	43	43	28.66	Dense
Kosarak	30.5	9.9×10^5	41270	8.1	2498	0.019	Sparse
NU-Mine Bench 2.0 Chain Store	37	11.1×10^5	46086	7.2	3520	0.0156	Sparse

Table 4.7: Characteristics of synthetic dataset

Dataset	$\|s\|$ (in MB)	$\|T\|$	$\|I\|$	$\|t\|_{avg}$	$\|t\|_{max}$	Type
T10-I4-D100K	3.83	10×10^4	1000	10	2000	Moderately Sparse

Table 4.8: Characteristics of educational domain (learning) datasets

Dataset	$\|s\|$ (in MB)	$\|T\|$	$\|I\|$	$\|t\|_{avg}$	$\|t\|_{max}$
AIEEE 2007	3.2	65,533	5	2	600
StatWay - Fall 2011 - Austin Community College' dataset	1.5	12,974	10	8	634

The experiments were performed on an Intel 2 Duo CPU @ 2.0 GHz with 3GB RAM and 32 bit operating system. We have performed several experiments and the results are based on the average of multiple runs.

4.4.2 Experiment 1: Performance Analysis

Major parameters of performance analysis are impact of candidate generation and total runtime of high utility mining. We have first presented the performance of comparing algorithms in terms of candidate generations. Secondly, we present the result in terms of total runtime of the algorithms.

For this experiment, we have considered dense dataset, moderately sparse dataset and educational domain dataset with a normalized weight of 0.93-0.96 used for Pumsb dataset and 0.8 -1.0 for Connect dataset (Yun & Ryu, 2011). For the rest of the dataset, we have used normalized weights between 0.3 and 0.6. The numbers of candidates generated with different UTmin of dense, synthetic and educational domain dataset are shown in the Tables 4.9, 4.10 and 4.11.

Table 4.9: Number of candidates generated with different UT_{min} of **dense** datasets

Datasets	UT_{min} *(%)*	**UP-Growth**	**UP-Growth$^+$**	**HUI-Miner**	**MU-Strategy 1**	**MU-Strategy 2**	**Proposed Algorithm**
Pumsb	68	6,01,200	3,56,729	3,04,780	3,00,120	1,45,872	1,00,561
	70	4,50,781	2,70,629	2,39,100	2,35,128	99,831	90,278
	72	2,23,002	1,67,936	1,50,935	1,45,815	78,500	50,423
	74	1,35,342	99,658	80,654	78,345	45,340	31,700
Connect	62	7,23,905	4,32,500	3,46,700	3,50,245	3,10,900	2,99,654
	64	4,67,842	2,78,391	2,15,278	2,00,594	1,90,728	1,50,913
	66	3,10,326	2,10,482	1,94,639	1,78,362	99,478	80,459
	68	2,00,350	1,10,450	99,730	98,381	78,367	45,532

Table 4.10: Number of candidates generated with different UT_{min} of **synthetic** datasets

Datasets	UT_{min} *(%)*	**UP-Growth**	**UP-Growth$^+$**	**HUI-Miner**	**MU-Strategy 1**	**MU-Strategy 2**	**Proposed Algorithm**
T10-I4-D100K	0.03	78,320	70,786	60,743	59,345	57,641	50,390
	0.04	55,732	50,874	45,876	44,002	42,200	40,345
	0.05	42,874	38,700	35,754	35,000	32,000	28,096
	0.06	35,980	30,674	28,900	27,300	25,900	23,450

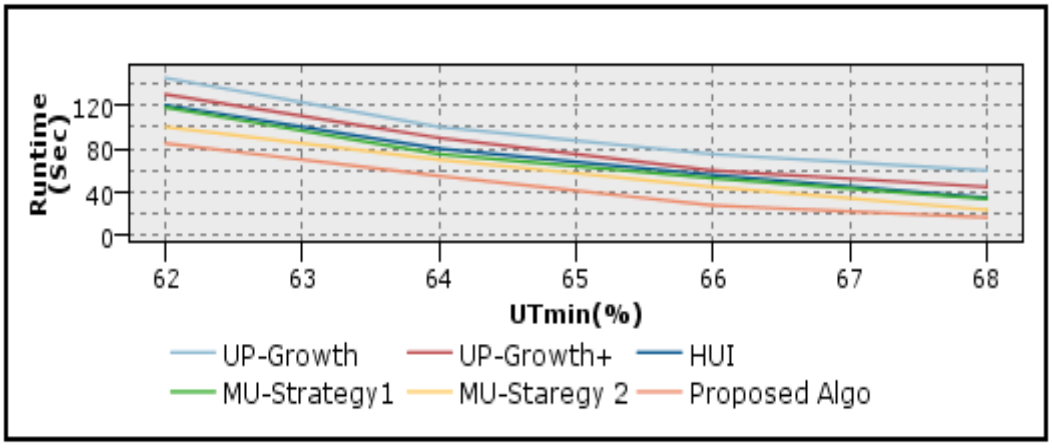
Table 4.11: Number of candidates generated with different UT_{min} of **educational domain** datasets

Datasets	UT_{min} *(%)*	**UP-Growth**	**UP-Growth$^+$**	**HUI-Miner**	**MU-Strategy 1**	**MU-Strategy 2**	**Proposed Algorithm**
AIEEE	0.5	35,080	21,320	10,200	9,876	7,856	5,301

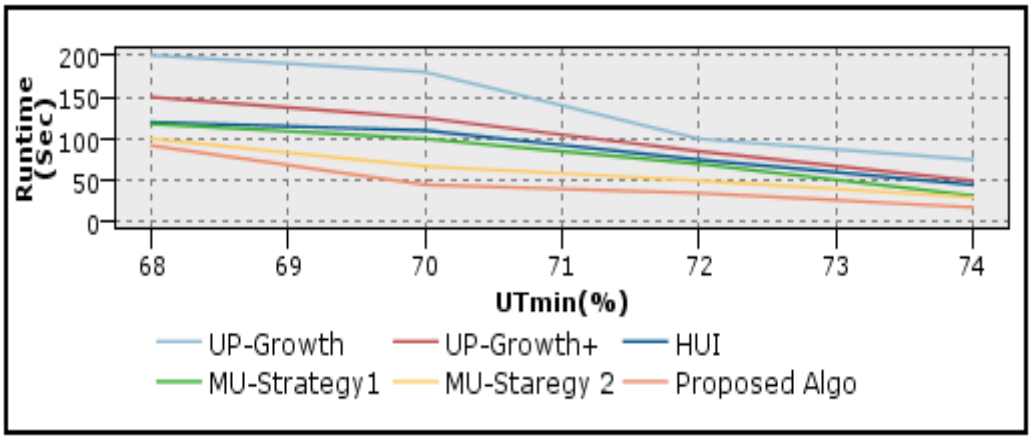
2007	1.0	12,780	6,200	4,152	3,919	2,301	1,129
	1.5	2,099	1,078	534	532	135	56
	2.0	679	200	89	88	34	18
StatWay - Fall 2011 - Austin Community College' dataset	0.5	40,318	23,378	15,782	12,387	8,152	5,823
	1.0	20,043	7,015	6,902	5,870	3,560	1,980
	1.5	5,320	1,321	1,534	1,409	903	329
	2.0	836	356	642	674	328	78

It has been observed that the compared algorithms generate much more candidates on the dense dataset as compared to the moderately sparse and educational domain dataset. As an overall performance, UP-Growth generates large candidates and proposed method generates fewer candidates among other methods.

Following figure shows the runtime performance of all the methods using different datasets. It has been observed that performance on educational domain data set takes less running time (Figure 4.13) as compared to dense dataset (Figure 4.11). The reason is that educational domain data set is smaller than the other dataset as well as has shorter size of average transaction. Overall, Proposed algorithm takes less runtime as compared to the other methods.



(a)



(b)

Figure 4.11: Total runtime on (a) Connect dataset (b) Pumsb dataset

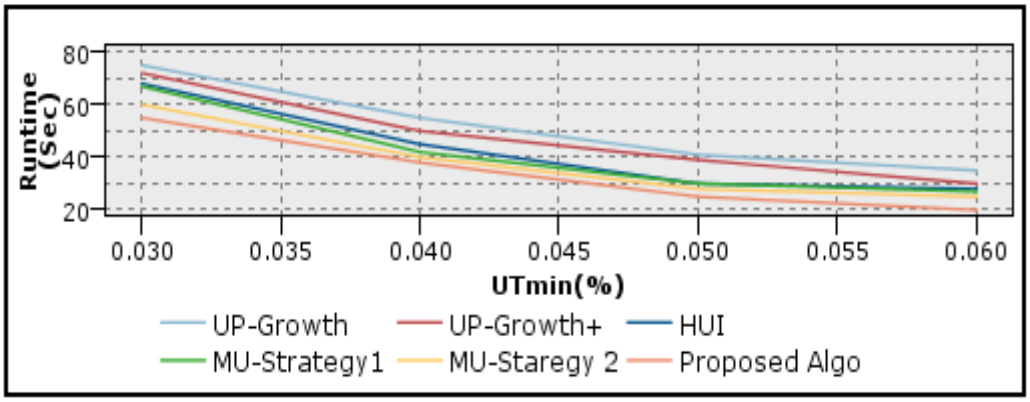
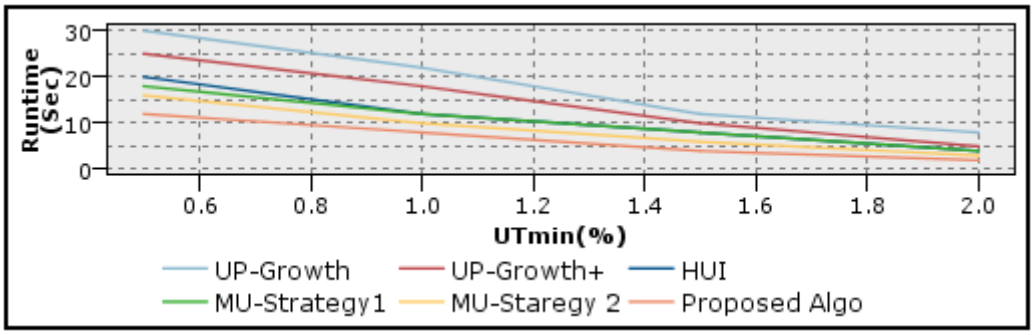
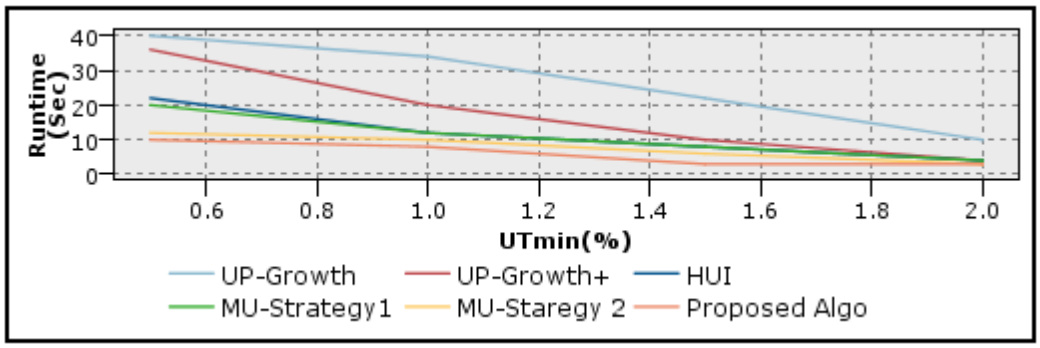


Figure 4.12: Total runtime on T10-I4-D100K dataset



(a)



(b)

Figure 4.13: Total runtime on (a) AIEEE 2007 dataset (b) StatWay - Fall 2011 - Austin Community College' dataset

4.4.3 Experiment 2: Scalability of the Proposed Algorithm

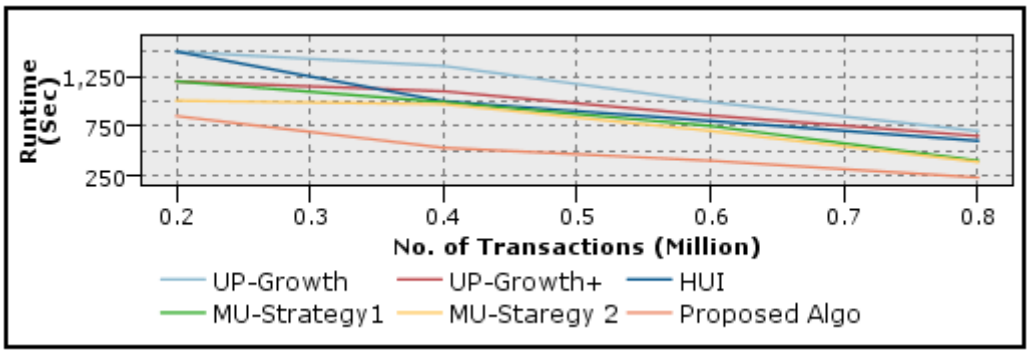
The main factors affecting the scalability factors are:

- Minimum utility support threshold
- Number of transactions to be Inserted or Deleted

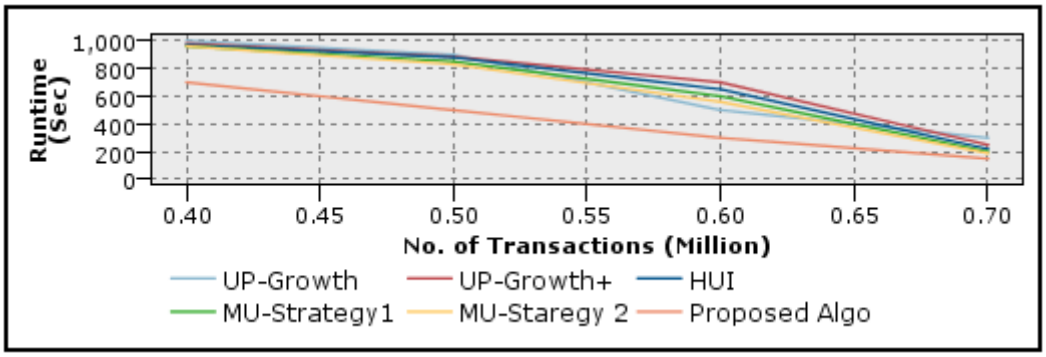
For mining incremental data it is important to find out performance on the basis of the size of insertion (Δ^+) and deletion (Δ^-) of transactions. We have considered kosarak and Nu-Mine Bench 2.0 Chain Store dataset for this experiment. To perform the effect of Δ^+/ Δ^-, we keep the value of Δ^- as constant and by varying the value of Δ^+ and vice versa. The following parameters setting are made during this test (Table 4.12):

Table 4.12: Parameters setting during incremental update

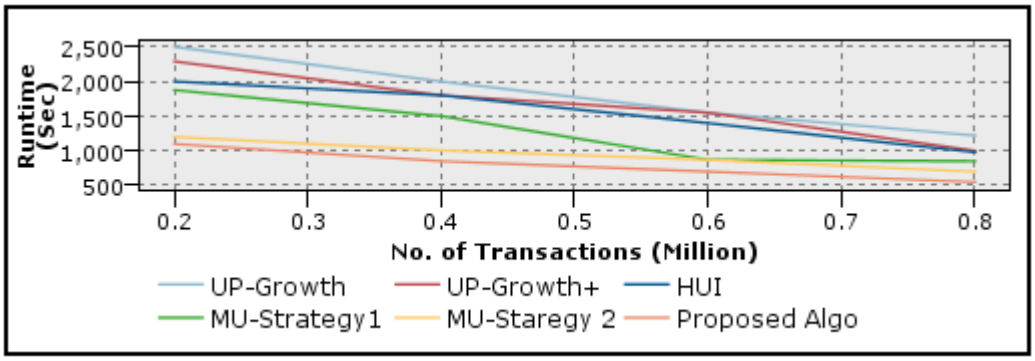
Parameters	**Kosarak and Nu-Mine Bench 2.0** **Chain Store data set**
UT_{min}	6%
Initial transaction	0.2 Million
For Insertion operation (Δ^{+}), No of transactions	0.2 Million
For Deletion operation (Δ^{-}), No of transactions	0.1 Million



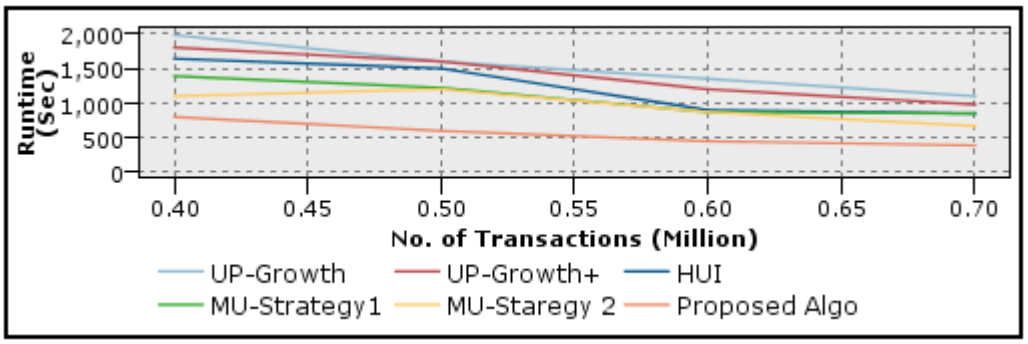
(a)



(b)



(c)



(d)

Figure 4.14: (a) Effectiveness of Δ^+ and (b) Δ^- using Kosarak data set

(c) Effectiveness of Δ^+ and (d) Δ^- using Nu-Mine Bench 2.0 Chain Store dataset

From the Figure 4.14 (a) to (d), it is observed that proposed tree construction algorithm is more efficient than the other comparison algorithms since the proposed algorithm has taken almost less time for constructing the tree in the updated dataset (Δ^+ and Δ^-).

4.4.4 Experiment 3: Memory Consumption

Much research has shown that the main memory requirement for tree based data structure is low enough to use the current available memory (in GB). Hence we have used the memory range that can cope up with current available memory.

We have analyzed the memory consumption of comparing method with moderately sparse and dense data set. The resultant memory consumption of each method is shown in Figures 4.15 and 4.16.

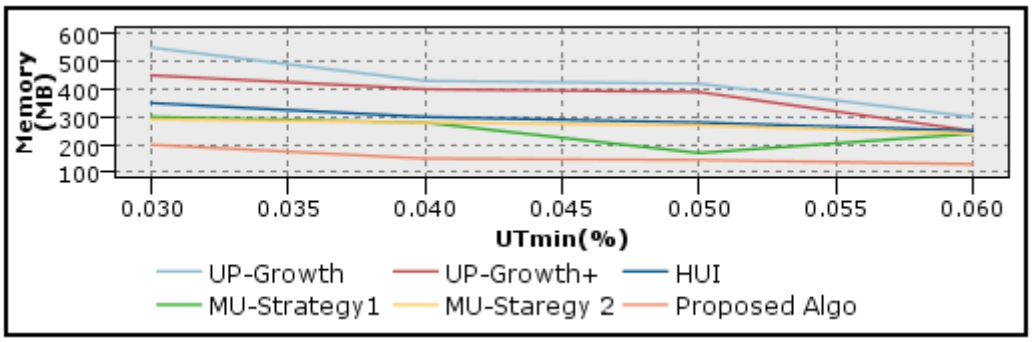


Figure 4.15: Memory consumptions under varied UT_{min} (%) on T10-I4-D100K dataset

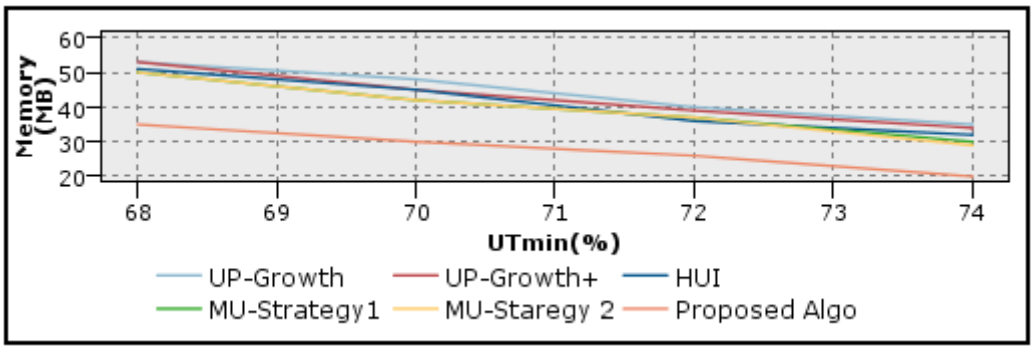


Figure 4.16: Memory consumptions under varied UT_{min} (%) on Pumsb dataset

When UTmin is low, the compared algorithms required more memory. Creation of more nodes and branches, require memory space to be increased.

From the Figure 4.15 and 4.16 above, it has been observed that there is a less memory requirement in case of the dense data set compared to the moderately sparse dataset. The reason is that many nodes are shared due to transaction having a similar form in the dense data set.

Also, memory consumption by the proposed algorithm is less as compared to the other algorithms due to its binary tree construction process where there are only two possible branches with less possible nodes.

Finally, it has been observed that the proposed work has better performance in terms of **Running time, Scalability and Memory consumption** than the other algorithms (UP-Growth, UP-Growth$^+$, HUI-Miner, MU-Strategy1, MU-Strategy2) compared in this work.

CHAPTER-5: INCREMENTAL CONSTRAINT BASED FREQUENT PATTERN MINING

Chapter 3 & 4 of the thesis deals with the Incremental Frequent/ High Utility Pattern Mining is to discover all the frequent patterns from the incremental data. To deal with the user specific mining of frequent patterns, idea proposed in the Chapter 3 & 4 can be extended by applying user-defined ***constraints***.

The goal of the Incremental Frequent Pattern Mining is to discover all the frequent patterns from the incremental/dynamic data sources where not all the generated patterns may be suitable for the end user. In such cases, constraints are applicable.

This chapter discusses how constraints are beneficial for Incremental Frequent Patterns Mining. This work proposes Tree based algorithm called CIFMine (Constraint based Incremental Frequent Pattern Mining) to mine the incremental data by using the dataset filtering technique.

5.1 Introduction

The goal of the Incremental Frequent Pattern Mining is to discover all the frequent patterns from the incremental/dynamic data sources where not all generated patterns may be suitable for the end user.

Constraint based mining focuses on the constraints and to specify the desired properties of the patterns to be mined that are likely to be of interest to the end user (Boulicant, 2008; Guns, Nijssen, & Raedt, 2013; Guzzo et al., 2013; Han, Lakshmanan, & Ng., 1999).

The ability to express and exploit constraints allows effectiveness and efficiency of the mining process (Nijssen & Guns, 2010; Pei, Han, & Wang, 2007; Raedt, Guns, & Nijssen,

2010). Some of the constraints to solve real world problems are **pattern, the length of the pattern and user defined support.**

There has been a considerable research work done in the field of constraint based pattern mining by using algorithmic approaches (Bonchi & Goethals, 2004; Brailsford, Potts & Smith, 1999; Bucila et al., 2003; Guns, Nijssen & Raedt, 2011; Raedt & Zimmermann, 2007; Ruggieri, 2010; Uno, Kiyomi & Arimura, 2005).

Han, Lakshmanan, & Ng. (1999) identified the following constraints:

- Knowledge type constraints: specify the types of knowledge to be mined.
- Data constraints: specify set of data relevant to the mining task.
- Dimension/Level constraints: specify the dimensions or the levels of data to be examined in a database or data warehouse.
- Rule constraints: specify concrete constraints of the rules to be mined.
- Interestingness constraints: specify what ranges of a measure associated with discovering patterns are interesting from a statistical point of view.

Wojciechowski & Zakrzewicz, (2002) identified the following classes of constraints:

- Database constraints: specify the source dataset
- Statistical constraints: specify thresholds for the support and confidence measures
- Pattern constraints: specify the interesting frequent patterns that should be returned by the query
- Time constraints: used in sequential pattern mining to influence the process of checking whether a given data-sequence contains a given pattern

Again, constraints can be classified from different points of view, such as application points of view and constraint –pushing point of view (Pei, Han, & Wang, 2007).

From application points of view:

- Item constraints: users are often interested in finding association rules involving only some specified items
- Length constraints: specify the requirement on the length of the patterns.
- Super-pattern constraints: to find patterns that contain a particular set of patterns as a sub pattern
- Aggregate constraints (Leung, et al., 2012): are the constraints on an aggregate of items in a pattern
- Regular expression constraints: are the constraint specified as a regular expression over the set of items using the established set of regular expression operators
- Duration constraints: are defined only in sequence databases where each transaction in every sequence database has a timestamp.

Constraint –pushing point of view:

- Prefix-monotone constraints

In this work our focus is to improve the efficiency of constraint-based frequent pattern mining by the following way:

- By applying dataset filtering techniques and,
- By introducing binary search tree based approach in incremental data

5.2 Proposed Work

The problem is to find out the frequent pattern using various constraints for incremental data using tree based method. The problem may be stated as follows:

Let 'I' be a set of 'N' items I= {I_1, I_2…I_N} and a database 'DB'= {T_1, T_2… T_M} be a set of 'M' transactions with support '∞', where each transaction T_i ($1 \leq i \leq M$), $T_i \in I$.

Let the user specified length threshold be 'Θ' and user specified pattern 'Ω'. Let Δ^+ be the newly added, and Δ^- be the deleted incremental data in the original database DB. Now DB becomes DB' by updating incremental data in the original database DB.

The problem is to discover the complete set of patterns from the database DB' using the mining results from DB, which satisfies the support '∞', item 'I' and length constraints 'Θ'.

Properties of the proposed algorithm

If 'T' does not satisfy 'Θ' & 'Ω' then any sub transaction will also not satisfy the 'Θ' and 'Ω'. Hence, eliminate T_i from DB'.

5.2.1 Algorithm for Constraint based Incremental Frequent Pattern Mining

In the proposed algorithm, CIFMine (Constraint based Incremental Frequent Pattern Mining), tree based data structure is considered to avoid the repetitive scanning of the dataset where different minimum support, item and length constraints are adopted to mine the frequent patterns from incremental data.

Proposed Algorithm for constraints based pattern mining from incremental data is shown in Figure 5.1 below:

// Step 1 to 3 for building the tree based data structure
Step1: Start filtering process. i.e. eliminate T_i from DB if T_i does not satisfies 'Θ' & 'Ω'
Step2: Find the events that appear in at least in |DB| x ∞.
Step3: Build the tree based data structure
Step4: Add/delete Incremental data (Δ+/Δ-) and update DB to DB'
Step5: Increment/decrement support
// Step 6 to 9 for mining frequent patterns from the tree
Step6: Build the conditional tree
Step7: Apply the user defined constraints
Step8: Mine frequent patterns
Step9: Print the output i.e. Constraints defined frequent patterns

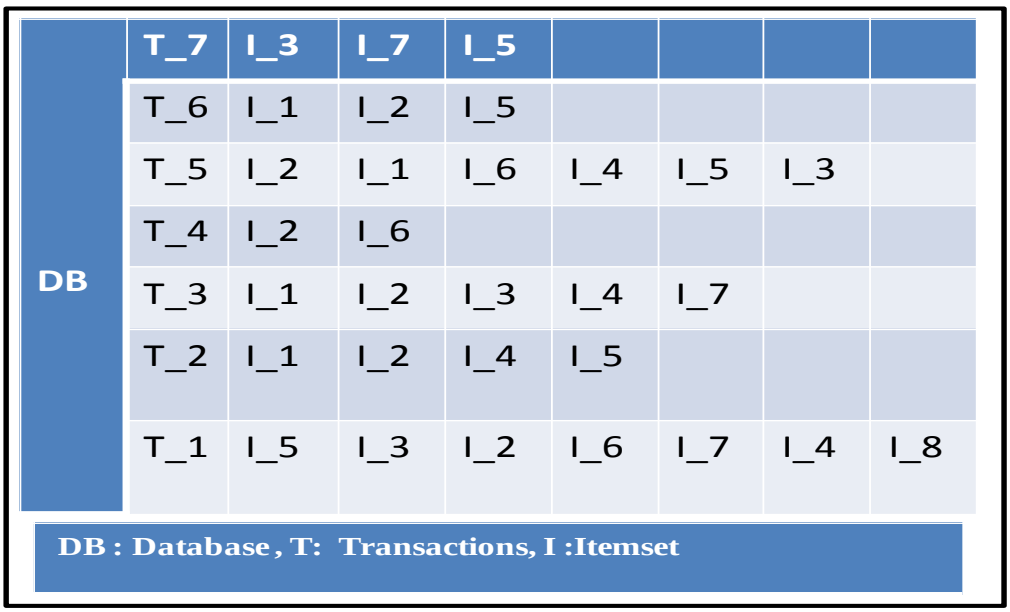
Figure 5.1: Proposed algorithm- CIFMine

In the first and second steps of the CIFMine algorithm, a reduced dataset is produced by filtering the database using user defined constraints (Saxena, De, & Jindal, 2006; Wojciechowski & Zakrzewicz, 2002). Then the tree based data structure is built which inherits the properties of Binary Search Tree (step 3) (Jindal & Dutta Borah, 2015a). Incremental data is added/ deleted (Δ^+/Δ^-) without rescanning the whole dataset as well as rebuilding the tree (step 4 & 5). The advantage of such tree is easy traversal as it is a locally optimized tree.

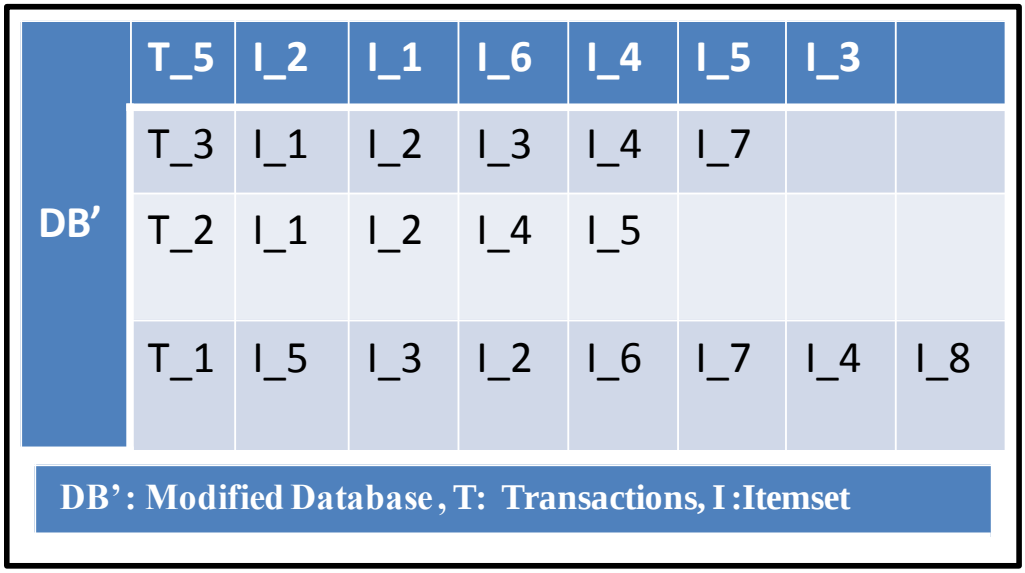
Step 6 to 9 of the algorithm as shown in Figure 5.1 above is for mining frequent patterns from the tree. The conditional tree is built by taking a count of the prefix access transaction base of a particular item. User defined constraints are applied to the conditional tree and finally prints the resultant of the frequent patterns from that tree.

5.2.2 Example: To mine constraint based frequent patterns from incremental data

To illustrate the algorithm we have considered the dataset as shown in Figure 5.2 (a).



(a)



(b)

Figure 5.2: (a) Original database (b) Reduced database

A. Tree building process using incremental data

The dataset is filtered out by taking consideration of the following constraints:

Pattern lengths >= 3; Support= 0.5 and type pattern constraint = I_1, I_2, I_4.

As the transaction ‘T_4’ does not satisfy the length constraints (lengths of T_4 are 2), filter it from the original dataset. Similarly discard transactions ‘T_6’ & ‘T_7’ that do not satisfy the type pattern constraints. Now the size of the dataset is reduced and shown in the Figure 5.2 (b). The tree is built as per definitions stated in chapter 3. The resultant tree is shown in Figure 5.3.

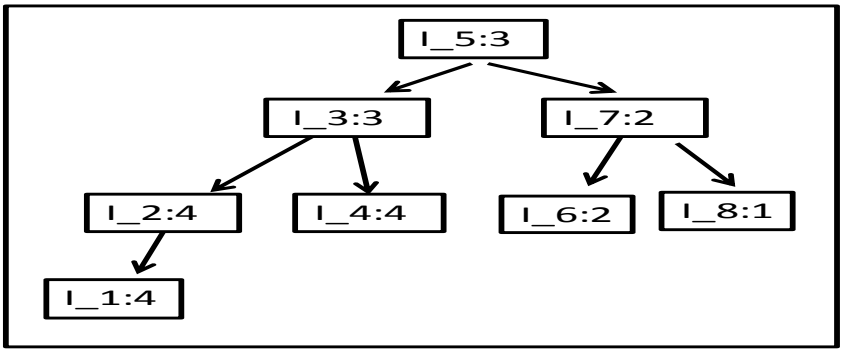


Figure 5.3: Tree obtained from Figure 5.2 (b)

Let us consider user defined item constraint where support should be greater than or equal to 1. From the Figure 5.3, item I_8 is discarded from the tree and the corresponding new tree is shown in Figure 5.4.

For adding incremental data, let us consider new transactions ‘T_8’= {I_1, I_2, I_3, I_4, I_5}; ‘T_9’= {I_4, I_5, I_6} and ‘T_10’= {I_6, I_7}. As we have added these transactions to the database shown in Figure 5.2 (b), the database grows with seven transactions.

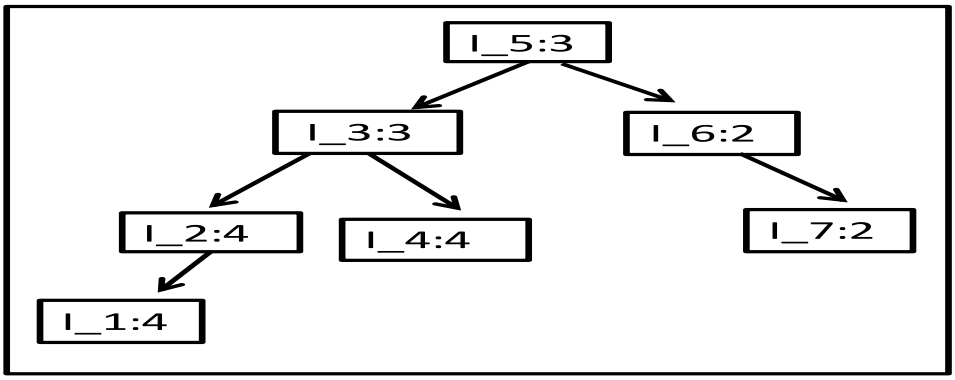


Figure 5.4: Tree after deleting item I_8

The beauty of this tree based data structure is that to add the new transaction in the tree, there is no need to scan the dataset repeatedly, instead, increase the Support Count of the existing items in the tree. The resultant tree is shown in Figure 5.5 below.

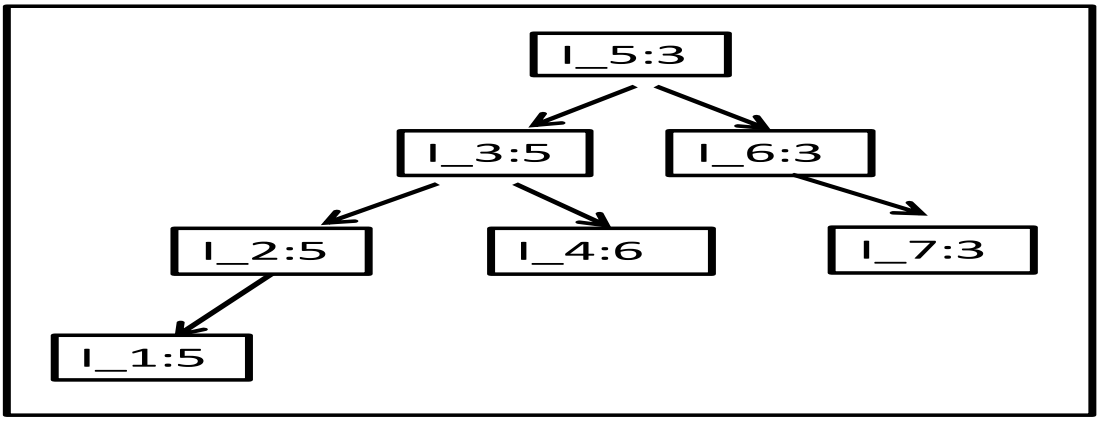


Figure 5.5: Adding incremental data

Similarly for deleting any transactions, simply decrement the Support Count of the respective item as stated in Jindal & Dutta Borah (2015a).

B. Constraint based mining from incremental data

Let us consider length constraints >3, pattern constraints= {I_2, I_3, I_4} and the prefix access transaction base for item 'I_1' for mining process. Table 5.1 below gives the resulting constraint based frequent patterns.

Table 5.1: Constraints and frequent patterns

Tree INFO [] **(refer Figure 5.5)**	**Item**	**Conditional Tree : the prefix access transaction base for I_5**	**Constraint**	**Resultant Frequent Patterns**
Root Node {I_5: 3}; Left hand side subtree	I_1	< I_5, I_3, I_2>: 1; < I_5, I_3, I_4>: 1;	Length >3; Pattern	<I_1, I_2, I_3, I_4,

nodes: {I_3: 5, I_2:5, I_1:5, I_4: 6}; Right hand side subtree nodes: { I_6:3, I_7:3}		< I_5, I_3, I_2, I_4>	constraints= { I_2, I_3, I_4}	I_5>

C. Constraint based high utility mining from incremental data

Let us revisit Chapter 4, Section 4.3.2, Figure 4.10 (a) & (b) and consider the following constarints:

- New Weight Constraint using AvgMU= 1.5
- Pattern constarint = < I_1 I_2>

As a first step, we have applied New Weight Constraint to the itemset presents in the Figure 4.10(b) and as a result, items I_8, I_4, I_5 and I_9 are filtered out. The remaing items I_2, I_7, I_1 and I_3 added to the header table for mining constraint based high utility itemset (refer Table 4.4 in Chapter 4).

Secondly we have imposed pattern constraint < I_1 I_2>. The resultant constraint based high utility patterns are listed in Table 5.2 .

The constraint based high utility itemset present in Table 5.2 is comparatively less as compared to the Table 4.4.

Hence, Constarint based mining is important as per perspective of the user.

Table 5.2: Example of Constraint based high utility mining

Candidate Keys	Conditional pattern base	Conditional tree	Candidate pattern	UTnew after imposing Pattern Constraint < I_1 I_2>.	Remarks
I_2	<I_1 I_3 I_4 I_5 I_7 I_8>, <I_1 I_4>, <I_1 I_3>, <I_1 I_7 I_9>	<I_1 I_3>, <I_1 I_7>	<I_1 I_2>, <I_2 I_3>, <I_2 I_7>	UTnew {**I_1 I_2}** = 2.5	Passed
I_7	< I_1 I_2 I_3 I_4 I_5 I_8>, <I_1 I_3>, <I_1 I_2 I_9>	<I_1 I_3>, <I_1 I_2>	<I_1 I_7>, <I_3 I_7>, <I_2 I_7>	----------	----------
I_1	<I_2 I_3 I_4 I_5 I_7 I_8>,<I_2 I_3>, <I_3 I_7>, <I_2 I_4>, <I_2 I_7 I_9>	<I_2 I_3>, <I_3 I_7>, <I_2 I_4>, <I_2 I_7>	<I_1 I_2>, <I_1 I_3>, <I_1 I_7>, <I_1 I_2 I_3> , <I_1 I_2 I_7>, <I_1 I_3 I_7>	UTnew {**I_1 I_2**} = 2.5 UTnew {**I_1 I_2** I_3} =2.05 UTnew {**I_1 I_2** I_7 } =1.23	Passed Passed Pruned
I_3	<I_1 I_2 I_4 I_5 I_7 I_8>, <I_1 I_7>, <I_1 I_2>	<I_1 I_7>, <I_1 I_2>	<I_1 I_3>, <I_3 I_7>, <I_2 I_3>	----------	----------

5.3 Experimental Results

5.3.1 Data and Environment for Experiment

To compare the performance of the proposed algorithm with state of the art algorithms MCRP-Tree (Bhatt & Patel, 2015), RegularMine (Raedt & Zimmermann, 2007), GwI-Apriori (Chiu, Chiu, & Huang, 2009), we have done experiments on Intel 2 Duo CPU@2. 0GHz with 3GB RAM and 32 bit operating system. We have performed several experiments and the results are based on the average of multiple runs.

For this experiment, we have considered widely used three datasets namely T10-I4-D100 K (synthetic dataset), Pumsb (real dataset: census data from Public Use Micro data Sample) and Kosarak dataset. The characteristics of these datasets are listed in table 5.2 below.

Table 5.3: Dataset parameters

Dataset	**Number of transactions in transaction set [$\|T\|$]**	**No. of Items [$\|I\|$]**	**Average size of a transaction [$\|t\|_{avg}$]**	**Maximum Transaction length [$\|t\|_{max}$]**	**Type (Dense/ Sparse)**
T10-I4-D100K	10×10^4	1000	10	2000	Moderately Sparse
Pumsb	4.9×10^4	2113	74	74	Dense
Kosarak	9.9×10^5	41270	8.1	2498	Sparse

5.3.2 Experiment 1: Performance Analysis

To evaluate the performance (efficiency and effectiveness) of the proposed algorithm, first we compare state of the art algorithms without constraints. From the Figure 5.6, it can be seen that number of candidate generation of CIFMine is less than the MCRP-Tree, RegularMine and GwI-Apriori. The reason is that CIFMine filtered out the initial transactions that are not used for future reference.

Hence, the proposed algorithm, CIFMine is more efficient than other state of the art algorithm compared in this work

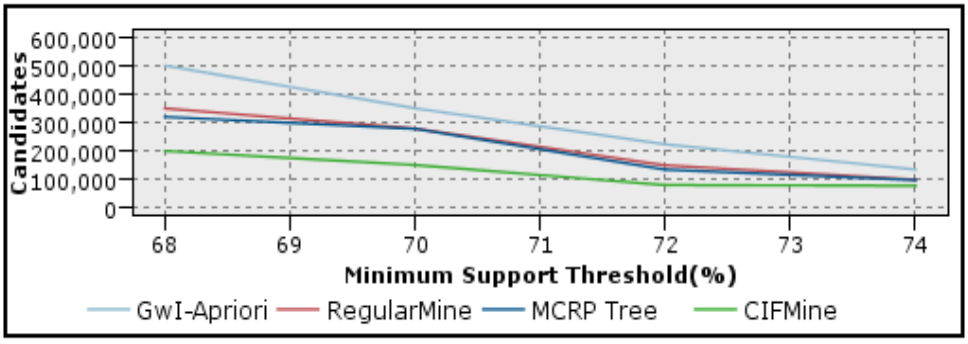


Figure 5.6: Candidates generated with different Minimum Support of Pumsb dataset

Secondly, we have considered user defined constraints such as length, pattern and minimum support constraint to measure the performance of the proposed algorithm. For a desired pattern, once the constraint is satisfied, the checking of a superset of the pattern is not required. Hence, for a higher rate of selectivity of constraints, the required running time is less.

It has been observed that time taken to mine the desired pattern by CIFMine is less as compared to the other algorithms (Figure 5.7).

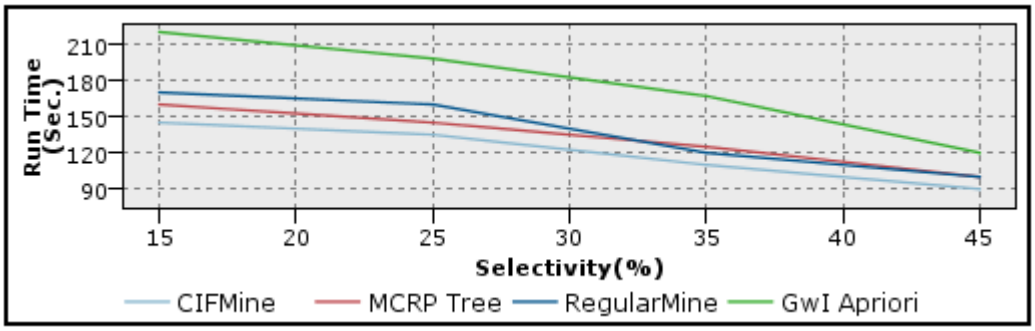
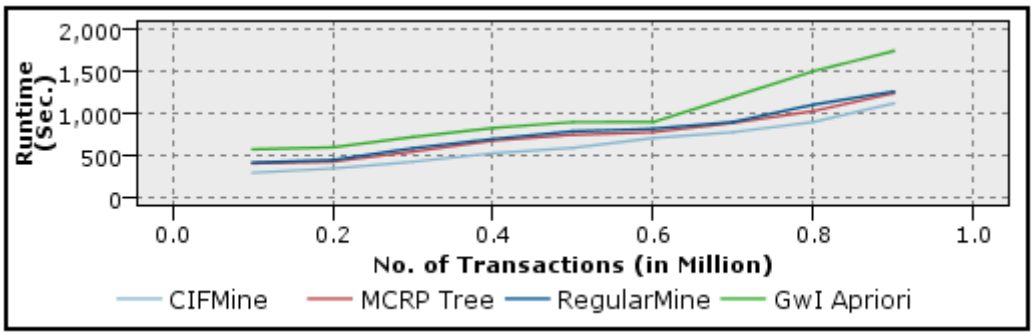


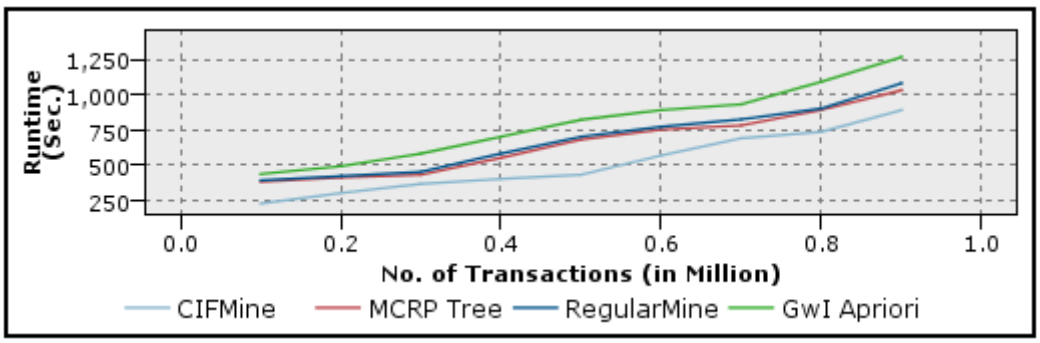
Figure 5.7: Runtime with selectivity of constraints using T10-I4_D100K dataset

5.3.3 Experiment 2: Scalability of the Proposed Algorithm

To measure the scalability of the proposed algorithm, effect of incremental data is important. As an incremental data we have considered kosarak dataset with Δ^+ (No of transactions For Insertion operation) = 0.2 Million and Δ^- (No. of transactions for Deletion operation) = 0.1 Million. From the experiment, it is found that proposed CIFMine is more scalable than the others compared in this work (Figure 5.8 (a) & (b)).



(a)



(b)

Figure 5.8: Effect of incremental data (a) Δ^+ and (b) Δ^-

From the above results, it has been observed that introducing constraints in frequent patterns from incremental data can help users extract exactly what patterns they want instead of generating the whole patterns.

The proposed work reduces the initial dataset by defining dataset filtering technique and generates less number of candidate keys in the mining process. It can cope up with incremental data, avoiding rebuilding the tree. Hence, it is efficient in terms of memory used.

The proposed algorithm, CIFMine, is an efficient algorithm as compared to the state-of-art algorithms MCRP-Tree, RegularMine, and GwI-Apriori, in terms of computing time as it produces only those patterns which are likely to be of interest to the end user.

CHAPTER-6: INCREMENTAL EDUCATIONAL DATA MINING

Educational Data Mining (EDM) is an emerging field exploring data in educational context by applying different Data Mining Techniques/Tools. EDM provides intrinsic knowledge of teaching and learning process for effective education planning. It defines goal-directed practices for ensuring institutional success at all levels.

Frequent pattern mining is one of the prominent research areas of the EDM research community to measure interestingness of the end user in terms of frequent patterns. This Chapter illustrates a modified tree based algorithm for mining frequent patterns of educational data incrementally. The modified algorithm is similar to the algorithm present in Chapter 5 but focuses on educational data.

Also presents an application to find out the strong association of students' participation in an online environment.

6.1 Introduction

Educational Data Mining (EDM) has acquired a significant attention of the research community for its ability to suggest the most suitable future planning by mining incremental educational data in context of quality education (Jindal & Dutta Borah, 2015b; Romero & Ventura, 2007). Incremental educational data is generated from heterogeneous resource where the addition and deletion of data take place without reforming, rescanning of the original dataset in a repetitive manner.

As an interdisciplinary research field, EDM provides goal-directed practices for ensuring success at all levels of the stakeholder of the education. There are various techniques to obtain the goal of the educational data mining such as Frequent pattern mining, Association Rule Mining, Relationship Mining, Incremental Mining, Classification, Prediction, Clustering, Neural Network and Web mining to name a few.

Frequent pattern mining is one of the prominent research areas of the EDM research community to measure interestingness of the end user in terms of frequent patterns, sequence of patterns (Sequential pattern mining) (Srikant & Agrawal, 1996) and association rules (Agrawal & Srikant, 1994). In the context of EDM, frequent pattern mining is suitable for behavior modelling of students learning environment to enhance quality education of Learning Management System (Jindal, & Dutta Borah, 2014; Mercheron, & Yacef, 2008).

Though the goal of the frequent pattern mining is to discover all the patterns from the source dataset, but not all discovered patterns are interesting to the end user. Constraint based frequent pattern mining is the solution to generate patterns which are particularly of interest to the end user.

Constraints such as Duration, Pattern, Length of patterns, Data, Rule, Dimension, Knowledge and Interestingness constraints are used to reduce the search space and hence to obtain substantial gain in the mining process (Guns, Nijssen, & Raedt, 2013; Guzzo et al., 2013; Han, Lakshmanan, & Ng, 1999).

In this chapter, we have divided our work into two parts.

i) Firstly, this work is focuses on a modified tree based algorithm for mining frequent patterns of educational data incremental. This approach is quite different from other related work in the sense that it basically depends on evaluating knowledge (constraints) rather

than the generative knowledge (rule set) (Mitrovic & Ohlsson, 1999): Focal points of the work are:

- To improve the efficiency of tree based algorithm in terms of running time
- To mine the frequent patterns from the educational data using duration (time) and access pattern constraint

ii) Secondly, applicability of EDM in learning environment, in the context of the stakeholder of education. Focal points of the proposed work are:

- Assessing the interestingness of patterns and
- Finding suitable rules using Association Rule Mining technique.

6.2 Modified TIMFP Algorithm

It is important to introduce constraints for faster response to solve a problem or to find the patterns that are frequently accessed by students from a solution space on the online environment.

In the context of EDM, the problem is to find out the students frequent patterns as well as to improve the running time by introducing constraints in TIMFP (Tree for Incremental Mining of Frequent Pattern) introduced by Jindal & Dutta Borah (2015 a). The problem may be stated as follows:

Let a database DB contains M number of students' records (T_1 to T_M: set of transactions) with N attributes (I_1 to I_N: set of items) with support '∞'. Let the user specified length threshold be 'Θ' and user specified pattern be 'Ω'. Let Δ^+ be the newly

added, and Δ^- be the deleted incremental educational data in DB. Now DB becomes DB$^{'}$ by updating incremental data.

The problem is to discover the complete set of patterns from the database DB'using the mining results from DB, which satisfies the support '∞', item /pattern 'I' and duration constraints 'Θ'.

The modified algorithm is shown in Figure 6.1 below. User defined constraints (data set filtering) (Saxena, De, & Jindal, 2006) are applied to obtain a reduced dataset (refer Step 1 &2) similar to the technique stated in CIFMine algorithm (Dutta Borah & Jindal, 2016). The TIMFP is used to build the tree based data structure and to mine the patterns from the educational original data as well as from incremental data (Δ^+/Δ^-) with constraints defined by the end user (step 3 to 5). The modified TIMFP algorithm is hybrid algorithm that takes property of TIMFP (refer Chapter 3) and CIFMine (refer Chapter 5) algorithms and we have applied in educational data.

Step 1: Start filtering to eliminate T_i Student record from DB that does not satisfies 'Θ' & 'Ω'
Step 2: Call CIFMine // (refer Chapter 5)
Step 3: Call TIMFP // (refer Chapter 3)
Step 4: Apply user defined constraints
Step 5: Mine frequent patterns from educational data incrementally

Figure 6.1: Modified TIMFP Algorithm for mining educational data

6.2.1 Experimental Results

6.2.1.1 Dataset Used

To evaluate the effectiveness of the proposed modified algorithm we have considered synthetic dataset- T10-I4-D100 K and educational dataset - Teachable Peer Learner dataset (AlgebraI2010Dec-retry-ss) as used in Chapter 3.

I. ***Statics of synthetic dataset***

- Number of transactions =10 x 10^4
- No. of Items =1000
- Average size of a transaction =10
- Maximum Transaction length = 2000

II. ***Statics of educational dataset***

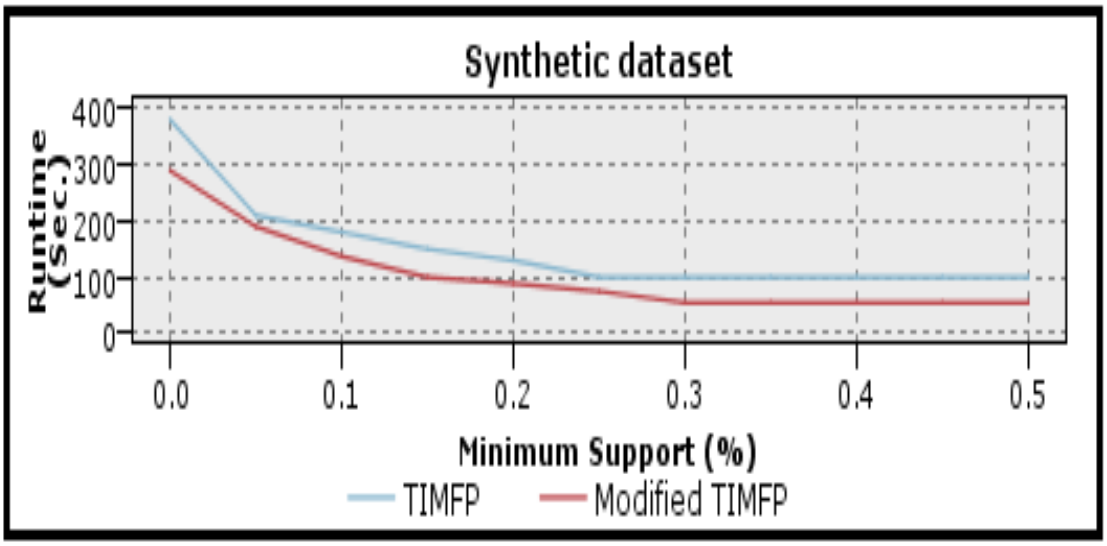
- Total Number of Transactions = 372,519
- Number of Students =145
- Number of Unique Steps =8210
- Total Number of Steps =17718
- Total Student Hours =158.52

6.2.1.2 Experiment 1: Performance Analysis

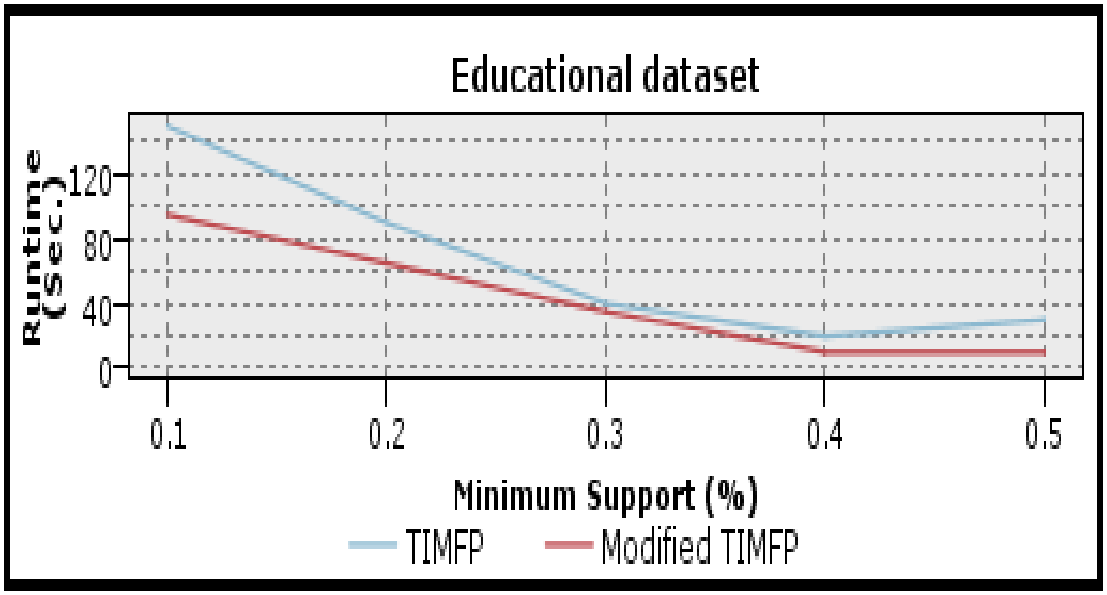
To measure the effectiveness of the proposed modified algorithm, we have run the algorithm both in synthetic and educational dataset. We have considered two constraints:

- Duration : time for which online session is maintained per students and
- Patterns : access pattern of the students

After applying Modified TIMFP it has been observed that the production of candidate keys as well as the running time is reduced as compared to TIMFP. The result is shown in Figure 6.2 (a) & (b).



(a)

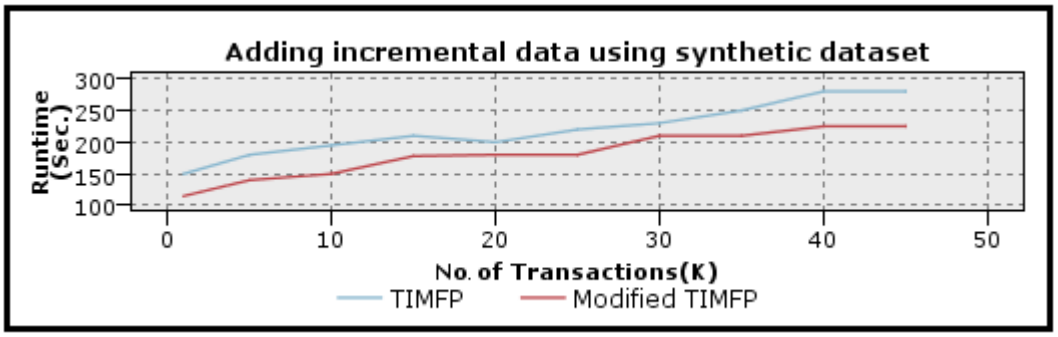


(b)

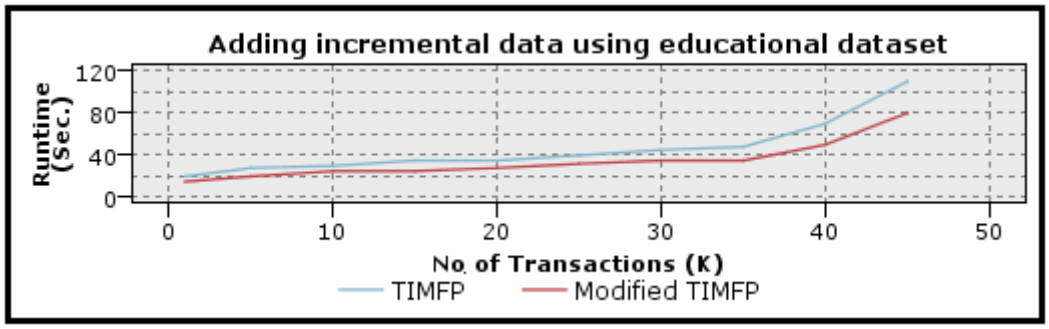
Figure 6.2: Runtime comparison (a) using synthetic dataset (b) using educational dataset

6.2.1.3 Experiment 2: Scalability of the Proposed Algorithm

To measure the scalability in terms of incremental data, we have taken similar data parameter as listed in Chapter 3. From the Figure 6.3 (a) & (b) it has been observed that proposed modified TIMFP is more scalable than the TIMFP in terms of addition or deletion of incremental data.



(a)



(b)

Figure 6.3: Scalability of Δ^+ using (a) synthetic dataset (b) educational dataset

From the experiment it has been observed that Modified TIMFP is more effective and scalable than TIMFP in terms of running time.

As mentioned in previous Ssections & Chapters of our thesis, we dealt with the Mining Frequent Patterns (incremental). It can be extended to generate rules by generating high

confidence rules from each frequent itemset. Hence, our focus is to mine strong association in context of EDM.

6.3 Applicability of EDM in Learning Environment

It is quite challenging to find out interestingness, an association of students' participation and obtain their response to a particular event on online teaching and learning environment (Gowda et al., 2013; Wang et al., 2012). Association Rule Mining (ARM) is one of the prominent technique that helps to find out the items frequently appearing in the dataset, relationships among different attributes in a learning environment.

In this part of the thesis, we are exploring the applicability of ARM in learning environment, measuring learner's beliefs, and interests in a particular lesson and finally generating students' response to online learning environment.

6.3.1 Approach used to generate Students' Response

To enhance the quality and cost effectiveness of the education, it is necessary to measure the expectations of students in the learning environment. Learning is a process that is governed by many factors such as (Romero & Ventura, 2007; Romero & Ventura, 2013):

- Skills
- Attitudes
- Knowledge
- Participation and
- Better understanding of the learning process by students

Though all the above mentioned factors are important, **students' participation and a better understanding of the learning process** should be considered as an integral part in an online environment.

The approach used for generating Students' response in a particular subject is shown in Figure 6.4.. The steps involved are as follows:

- Input data from learning lab
- Pre-processing
- Capture the knowledge component and build the model
- Pattern Evaluation and Generate Students Response

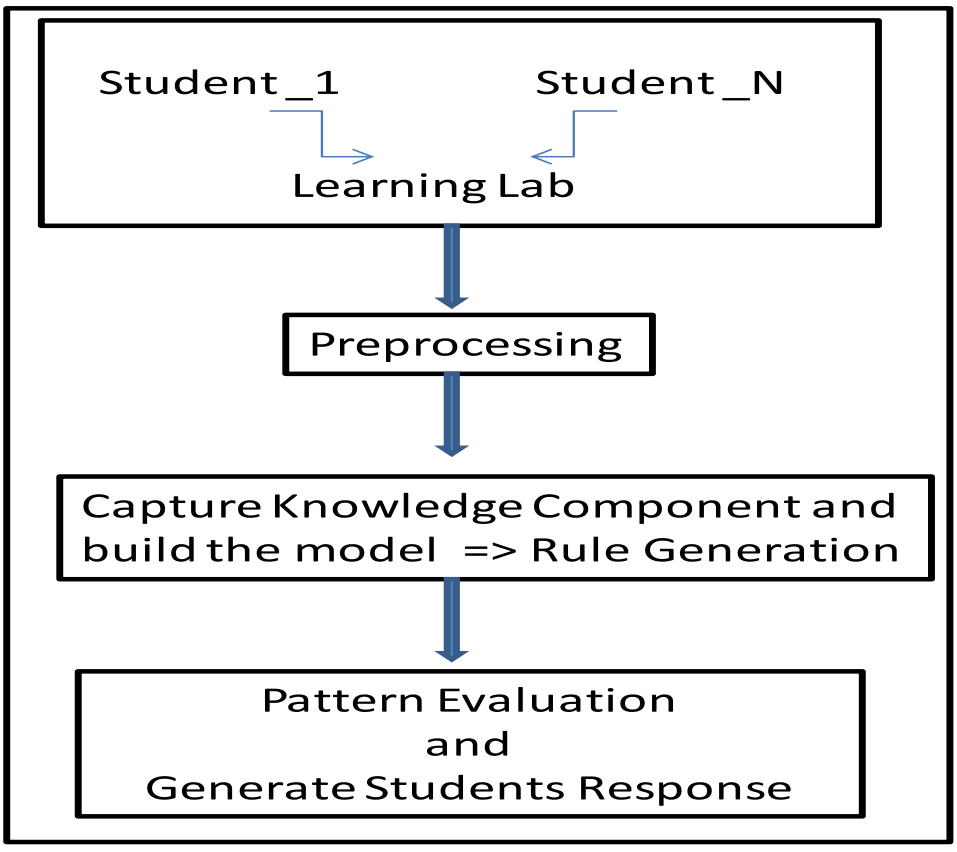


Figure 6.4: Workflow for Generating Students Response

6.3.2 Input data from learning lab

We used the "Motivation and Metacognition in Chinese Vocabulary Learning, Experiment 3 (27,421 transactions) & 5 (72,249 transactions) dataset" accessed via DataShop (Koedinger et al., 2010). Some of the attributes in the dataset are shown in Table 6.1:

Table 6.1: Attributes in learning dataset

Sl. No	Attributes	Descriptions
1.	Row	1 to N
2.	Sample name	All data (Vocabulary Learning) in learning lab
3.	Anon, Student Id	Student Identity
4.	Session Id	Session and lesson for a specific user
5.	Problem starting/ session Time	HH: MM: SS
6.	Time Zone	US/Eastern
7.	Duration	Seconds
8.	Student Response Type	Attempt a problem
9.	Student Response sub Type	Event/ Drill
10.	Tutor Response	Suggestions/ Feedback
11.	Problem name	Specific lesson
12.	Problem view	No of times viewing the problem
13.	Condition	Random difficulty
14.	Total Num Hints	No of hits achieved
15.	Knowledge component	Default (lesson), ChineseMM10Fall etc.

The datasets were trained and tested in SPSS Clementine 11.1 environment. A partial view of this environment is shown in Figure 6.5.

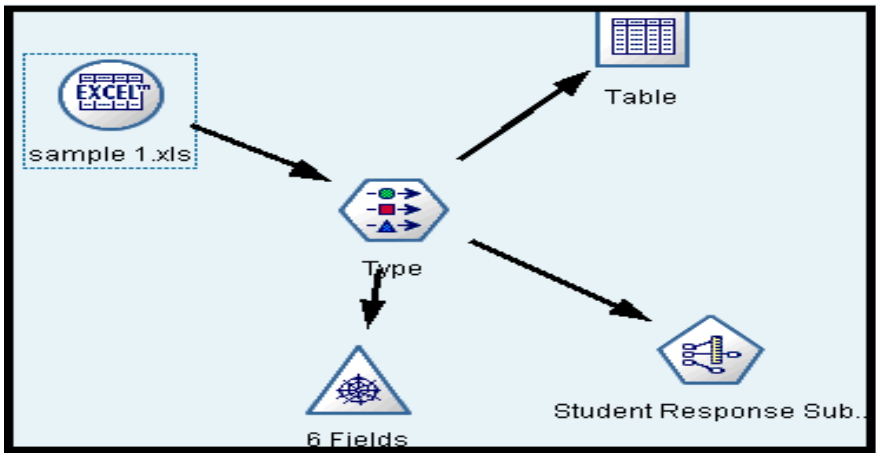


Figure 6.5: A partial view of SPSS environment

6.3.3 Preprocessing

To provide the suitable data for effective training and testing purpose, there is a necessity to cleanse, integrate and transform the original data. A major step performed during this process is as follows:

- Identify missing values, special characters, question marks, blanks etc. using "Null Rule"
- Filter out the attributes those are not so important in the model building process (ARM). E.g.: Row, Sample name etc.

The resultant snapshot is shown in Figure 6.6.

Session Id	Duration (sec)	Student Response Subtype	Level(UNIT)	Problem Name	Problem View	Problem Start Time
session-Lesson8-user4	2.000000000000000e+000	DRILL	Lesson8	9_9_Lesson8	9.000	23:21:46
session-Lesson8-user4	2.000000000000000e+000	EVENT	Lesson8	LAST_SINOGRAPH 8	10.000	23:21:40
session-Lesson8-user4	2.500000000000000e+000	DRILL	Lesson8	9_9_Lesson8	10.000	23:21:51
session-Lesson8-user4	2.500000000000000e+000	EVENT	Lesson8	LAST_SINOGRAPH 8	10.000	23:21:46
session-Lesson8-user4	3.000000000000000e+000	DRILL	Lesson8	1_1_Lesson8	8.000	23:21:51
session-Lesson8-user4	1.000000000000000e+000	EVENT	Lesson8	LAST_SINOGRAPH 0	8.000	23:21:54
session-Lesson8-user4	1.000000000000000e+000	DRILL	Lesson8	12_12_Lesson8	8.000	23:21:57
session-Lesson8-user4	1.000000000000000e+000	EVENT	Lesson8	LAST_SINOGRAPH ...	8.000	23:21:57
session-Lesson8-user4	4.000000000000000e+000	DRILL	Lesson8	4_4_Lesson8	7.000	23:22:05
session-Lesson8-user4	4.000000000000000e+000	EVENT	Lesson8	LAST_SINOGRAPH 3	7.000	23:21:57
session-Lesson8-user4	7.000000000000000e+000	DRILL	Lesson8	15_15_Lesson8	7.000	23:22:19
session-Lesson8-user4	7.000000000000000e+000	EVENT	Lesson8	LAST_SINOGRAPH ...	7.000	23:22:05
session-Lesson8-user4	5.000000000000000e+000	EVENT	Lesson8	RETIRED_SINOGR...	4.000	23:22:19
session-Lesson8-user4	6.000000000000000e+000	DRILL	Lesson8	2_2_Lesson8	10.000	23:22:30
session-Lesson8-user4	1.000000000000000e+000	EVENT	Lesson8	LAST_SINOGRAPH 1	10.000	23:22:30
session-Lesson8-user4	3.000000000000000e+000	DRILL	Lesson8	14_14_Lesson8	10.000	23:22:37
session-Lesson8-user4	3.000000000000000e+000	EVENT	Lesson8	LAST_SINOGRAPH ...	11.000	23:22:37
session-Lesson0-user4	6.000000000000000e+000	DRILL	Lesson0	0_0_Lesson0	10.000	23:22:37
session-Lesson8-user4	1.000000000000000e+000	EVENT	Lesson8	LAST_SINOGRAPH 7	10.000	23:22:43
session-Lesson8-user4	5.000000000000000e+000	EVENT	Lesson8	RETIRED_SINOGR...	5.000	23:22:49

Figure 6.6: Resultant list after pre-processing step

6.3.4 Capture the knowledge component and build the model

In an online learning environment, it is important for the educators to understand students' behaviour and related view in a particular subject matter. This will help for future enhancements of the teaching program. To understand their participation, following parameters are important:

- Capture: Capture students' interaction/ knowledge component (input data collected from learning lab)
- Build the Model: Apply DM technique (e.g. ARM) and
- Provide learning outcome (Generate students' response)

After capturing the components that is useful to provide knowledge to the user (refer Figure 6.6) from learning lab, the next step is to build the model using the ARM technique (Apriori algorithm). "Building the model" is an important step as it fully explains the relationships among important parameters. It is a two way approach:

- Generate all itemsets whose support is greater than or equal to the user defined minimum support and
- Generate Association Rules from each frequent itemset

The appropriate values for the support and confidence thresholds of association rules depend on the application and the data.

6.3.5 Pattern Evaluation and Generate Students Response

An ARM technique has a capacity to generate high number of patterns. A Research question based on this matter is that "are all of the patterns interesting?" Han et al. (2007) defines that a pattern is interesting if it is easily understood, valid on test data, potentially useful, novel in nature and ultimately provide knowledge to the user.

"Although frequent patterns are interesting in their own right, the end goal of association analysis is often the efficient generation of association rules (Agrawal, Imielinski, & Swami, 1993)".

In (Steinbach et al., 2000) stated that the rules generated from high confidence from each frequent pattern are efficient. Hence we have considered 100% confidence to efficiently derive the association among learning parameters (refer Figure 6.7)

Consequent	Antecedent	Support %	Confidence %
Student Response S...	Outcome = EVENT	54.43	100.0
Student Response S...	Level(UNIT) Outcome = EVENT	20.51	100.0
Student Response S...	Duration (sec) = 1.0... Outcome = EVENT	19.8	100.0
Student Response S...	Duration (sec) = 1.0... Problem View < 10.5... Outcome = EVENT	15.33	100.0
Student Response S...	Duration (sec) = 1.0... Level(UNIT) Outcome = EVENT	6.97	100.0
Student Response S...	Session Id = sessio... Outcome = EVENT	1.59	100.0
Student Response S...	Session Id = sessio... Outcome = EVENT	1.59	100.0
Student Response S...	Session Id = sessio... Outcome = EVENT	1.33	100.0

Figure 6.7: Outcome after applying ARM technique

Mapping of associations between different parameters is shown in Figure 6.8. It shows that students' response for a particular lesson is tightly associated with outcome "Event"

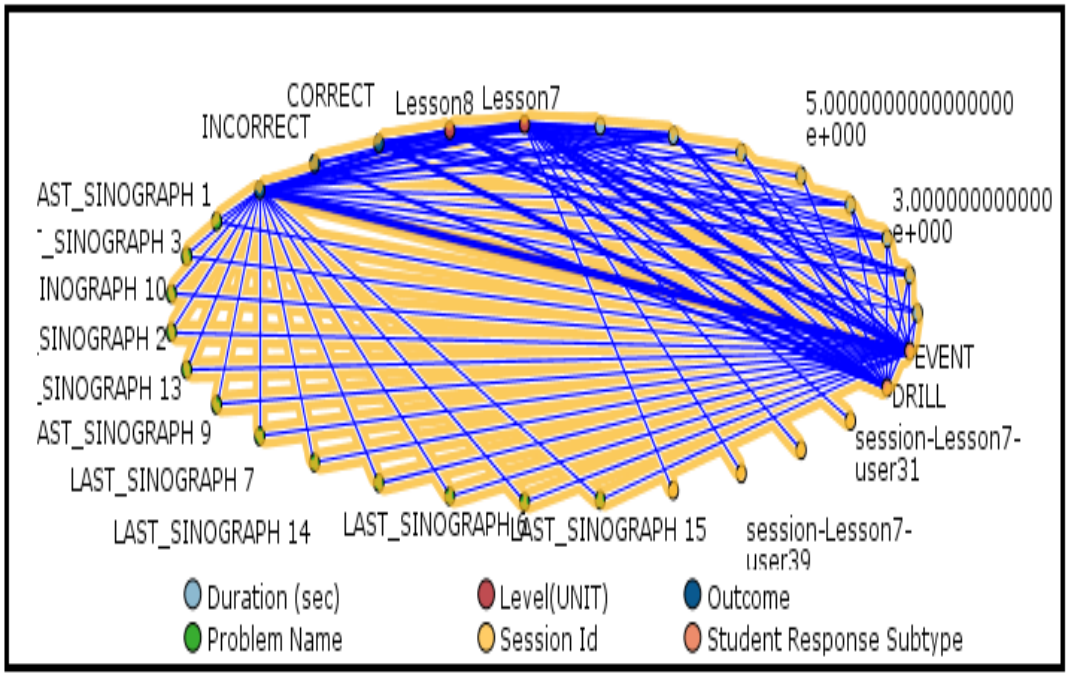


Figure 6.8: Association between Consequent and Antecedent

Finally resultant rules to generate appropriate students' response are shown in Figure 6.9. “Duration, outcome, problem view and event” are identified as main parameter to generate desired result i.e. student response.

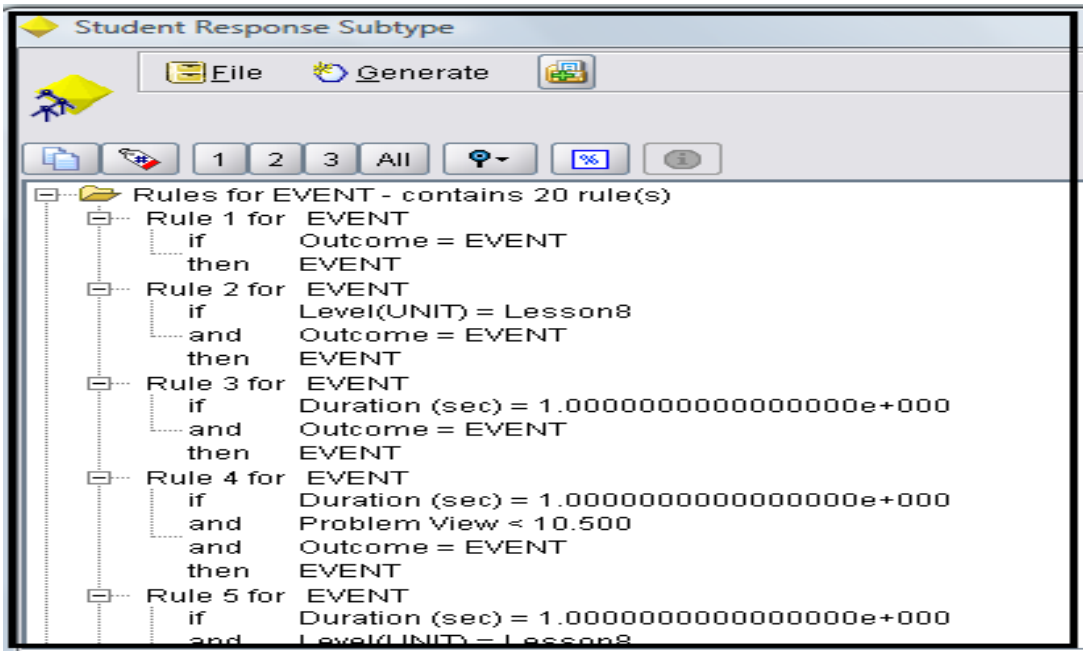


Figure 6.9: Important resultant Rules

6.3.6 Results

As an outcome of this approach, twenty numbers of association rules were generated. We have found that "problem view" and "duration" of attempting a lesson are most interesting parameters during rule generation (refer rule 3, 4 and 5 from Figure 6.9). Summarizing the resultant association rules, we get that student response for a particular lesson is mainly based on the following categories:

- Students with "outcome=event" is directly associated with "student response to event"
- Those students were exercising lesson 8 (assigned lesson) and "outcome = event" is associated with "student response to an event" and
- Duration to hit a particular lesson is 1.0 sec and viewing a problem is less than 10 then "student response is associated with event"

Educators' can use the discovered knowledge of the rules for decision making process.

This work discusses about the generation of students' response for a particular lesson by:

- Assessing the interestingness of patterns and
- Finding suitable rules using Association Rule Mining technique.

The finding of the work was that ARM generates lots of rules where all are not so important to derive proper knowledge. Hence this work considers rules having 100% confidence.

The knowledge discovered from this rule is beneficial for students as well as educators/ education planners such that

- Students should improve their learning by exercising the target lesson within the less amount of time

- Educators can monitor students' participation. E.g.: identifying learners patterns that actively participating, finding frequent mistake level and risk level of students and
- Providing the guidance/feedback to the students for their future participation and knowledge improvement.

CHAPTER-7: CONCLUSION AND FUTURE WORK

This chapter presents the Goal-related issues addressed, main contributions and future work of this research.

7.1 Goal-related Issues Addressed

This thesis presents a systematic way to address the frequent pattern mining problem using incremental frequent pattern mining. The following goal-related issues have been addressed:

- **Cost of Frequent Pattern Mining**: Mining frequent patterns are costly during changes of the underlying database. Most of the traditional approaches need repetitive scanning of the database as well as generating huge number of candidate sets. In such type of situation, the important issue is **how to utilize the mining result efficiently to reflect these changes**. Thus, mining of growing databases is an interesting area of research in Data Mining.

- **Problems in terms of Computational and Algorithmic Development:** This causes Creation of duplicate node, Greater computation time required for Searching common itemsets, Merging and finding a suitable path, Consumption of large amounts of memory to construct the tree and greater mining time etc. It is important to develop efficient approaches to enhance the performance of mining.

- **Importance of Semantic Significance:** In Frequent pattern Mining, the common assumption is that each item in a database is equal in weight. As a matter of fact, in real life applications, it is not always possible to treat all patterns in an equal or binary way. Utility/weight associated with items is important in real life applications and it provides Semantic Significance of items presents in a database.

- **Importance of Constraints:** It is not always suitable for end users that all generated patterns are equally important. Constrains are important in cases where end user wishes to mine patterns that are interesting and specific to their application point of view.

7.2 Contribution of the Research

This research work has offered a solution for addressing incremental frequent pattern mining method. We have considered mainly four approaches to fulfil our research objective, namely:

- **Incremental Mining of Frequent Patterns**
- **Incremental High Utility Frequent Pattern Mining**
- **Incremental Constraint based Frequent Pattern Mining and**
- **Incremental Educational Data Mining**.

First, this work proposes a novel tree based data structure for mining frequent pattern of incremental data called TIMFP which is compact and suitable for incremental and interactive mining **(Chapter 3)**.

Secondly, another data structure with Average Maximum Utility (AvgMU) and mining algorithm to mine high utility patterns from incremental data that reduces tree constructions and computation time is presented **(Chapter 4)**.

Thirdly, a tree based algorithm called CIFMine to mine the incremental data by using a dataset filtering constraints like specific pattern, the length of the pattern and user defined minimum support threshold which reduces the size of the dataset before constructing the tree is stated **(Chapter 5)**.

Lastly, a modified algorithm “Modified TIMFP” to construct and mine incremental educational data is presented. Also an application has been exhibited to find out the strong association of students' participation in an online environment **(Chapter 6)**.

The approaches presented in this research have been successfully validated on a number of benchmark synthetic and real-world datasets that are mostly used by the Data Mining research community.

The research work has successfully established the fact that, Incremental Mining of Frequent Patterns approaches used in this work is cost effective and efficient as compared to traditional frequent pattern mining approaches. The application area of the work includes Business data as well as Educational data of high impact and real world importance.

7.3 Application of the Research

Our research is applicable in diverse areas such as:

- Business: Pattern and trend analysis of customers. e.g.: Shopping patterns of customers which can help the retailer to know the need of customers and as a future action stores layout may change accordingly. The customer data keeps growing with time. This growing data can be mined incrementally using our research; Stock analysis by investors. e.g.: Investors can find out stocks with good performance etc.

- Education: Behavior analysis of students both in offline and online environment; Identification of the risk level of students; Enrollment Management (branch of study, courses, and programs in which they can enrol) and Curriculum Development which can help planners design a curriculum focused on the demands of the future workforce etc. so that successful, goal-oriented and quality education can be obtained.

- Banking and Finance: Patterns and association analysis specially loan payment and credit analysis of customers; unusual pattern analysis etc.

7.4 Future Work

This thesis throws light on a number of future research directions.

Firstly, this thesis provides a direction on incremental frequent pattern mining by analysis of performance in terms of running time and space. Last chapter also provides mining association with educational data, hence there is a future direction of focus particularly on Association Rule Mining in different kind of data.

Secondly, the concept of Incremental Mining can be enhanced to Distributed Data Mining.

Thirdly, we have considered database perspective by considering different kinds of data and analysis of performance of proposed algorithms. As a future work, the complexity of each algorithm can be derived.

Lastly, as a future work, we will integrate Incremental mining with the concept of Bigdata Analytics for mining frequent patterns. Further, a model of skill development for formal and Non-formal education will be constructed applying EDM.

Publications from the thesis

Papers Published in International Journals:

- Jindal, R. & Dutta Borah, M. (2016). A novel approach for mining frequent patterns from incremental data. *International Journal of Data Mining, Modelling and Management*, Inderscience Publications, *8*(3), doi: 10.1504 /IJDMMM.2016. 079071. ***(Google Scholar Citation: 02 till 02/10/2017).***

  Index in **Scopus** (Elsevier), Emerging Sources Citation Index (Thomson Reuters), DBLP Computer Science Bibliography, Google Scholar, Inspec (Institution of Engineering and Technology). Impact factor: 0.39)

- Jindal, R. & Dutta Borah, M. (2016). An approach for high utility incremental pattern mining. *International Journal of Data Analysis Techniques and Stratigies*, Inderscience publisher, [In press] [Online]: http://www.inderscience.com /info/ingeneral/ forth coming .php? jcode=ijdats.

  (Index in **Scopus** (Elsevier), ACM Digital Library, Thomson Reuters, DBLP Computer Science Bibliography, Google Scholar etc. Impact factor: 0.21)

- Jindal, R. & Dutta Borah, M. (2015). Predictive Analytics in Higher Educational Context. *IT Professional*, Publisher: IEEE Computer Society, *17*(4), 24-33. doi: 10.1109/MITP.2015.68. ***(Google Scholar Citation: 10 till 02/10/2017).***

  (Index in **SCI Expanded, Scopus**, Thomson Reuters. Impact factor: 0.819)

- Jindal, R. & Dutta Borah, M. (2013). A Survey on Educational Data Mining and Research Trends. *International Journal of Database Management Systems*, Publisher: AIRCC, *5*(3), 53-73. doi:10.5121/ijdms. ***(Google Scholar Citation: 46 till 02/10/2017).***

  (Index in Google Scholar, DOAJ, getCITED, ProQuest etc. Impact factor: 0.36)

Papers Communicated to International Journal:

- Jindal, R. & Dutta Borah, M. (2017). A Study of Incremental Mining of Association Rules. **(*Communicated to an International Journal)***

Papers Accepted/Published in International Conferences (IEEE):

- Dutta Borah, M. & Jindal, R. (2016). Constraint based Frequent Pattern Mining from Incremental Educational data: Educational Data Mining Approach. Accepted for publication at the *2016 IEEE International Conference on Teaching and Learning in Education*, Universiti Tenaga Nasional, Malaysia, March 1-2.
- Dutta Borah, M. & Jindal, R. (2015). Constraint based filtering for Incremental Data Mining", In *proceedings of the 7th IEEE International Conference on Computational Intelligence and Communication Networks*, Jabalpur, India, December 12-14.
- Jindal, R. & Dutta Borah, M. (2014). An approach to generate students' response on learning environment using Association Rule Mining. In *proceedings of the IEEE International Conference on Data Mining and Intelligent Computing* (pp. 1-5), Delhi, India, September 5-6. doi: 10.1109/ICDMIC.2014.6954225.
- Jindal, R. and Dutta Borah, M. (2012). Predicting Time Complexity and Accuracy of Branch Decision using Modified Heuristic Function. In *proceedings of the IEEE 2012 Nirma University International Conference on Engineering*, Ahmadabad, India, December 6-8.

References

Agrawal, R., Imielinski, T., & Swami, A. (1993). Mining association rules between sets of items in very large databases. In *proceedings of the ACM SIGMOD Conference on Management of Data* (pp. 207-216). Washington, D.C., May 26-28.

Agrawal, R., & Srikant, R. (1994). Fast algorithms for mining association rules. In *proceedings of the 20th International Conference on Very Large Databases* (pp. 487-499). Santiago, Chile, September 12-15.

Ahmed, C.F., Tanbeer, S.K., Jeong, B.S., & Lee, Y.K. (2008). Handling dynamic weights in weighted frequent pattern mining. *IEICE-Transactions on Information and Systems, E91-D* (11), 2578-2588. doi: 10.1093/ietisy/e91-d.11.2578.

Ahmed, C.F., Tanbeer, S.K., Jeong, B.S., & Lee, Y.K. (2009). Efficient Tree Structures for High Utility Pattern Mining in Incremental Databases. *IEEE Transaction on Knowledge and Data Engineering, 21*(12), 1708-1721.

Ahmed, C.F., Tanbeer, S.K., Jeong, B.S., & Lee, Y.K. (2011). HUC-Prune: An efficient candidate pruning technique to mine high utility patterns. *Applied Intelligence*, *34*, 181-198. doi: 10.1007/s10489-009-0188-5.

Ahmed, C.F., Tanbeer, S.K., Jeong, B.S., Lee, Y.K., & Choi, H.J. (2012). Single-pass incremental and interactive mining for weighted frequent patterns. *Expert Systems with Applications, 39*(9), 7976-7994. doi: 10.1016/j.eswa.2012.01.117.

Antunes, C. (2008). Acquiring Background Knowledge for Intelligent Tutoring Systems. In *proceedings of the International Conference on Educational Data Mining*, Montreal (pp. 18-27). Canada, June 20-21.

Asai, T., Abe, K., Kawasoe, S., Arimura, H., Sakamoto, H., & Arikawa, S. (2002). Efficient substructure discovery from large semi-structured data. In *proceedings of the 2nd SIAM International Conference on Data Mining* (pp. 158-174), USA, April 11-13.

Ayan, N.F., Tansal, A.U., Arkun, M.E.(1999). An efficient algorithm to update large itemsets with early pruning. In *proceedings of the 5th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp.287-291),USA, doi: 10.1145/312129.312252.

Baker, R.S.J.D., & Yacef, K. (2009). The state of Educational Data Mining in 2009: A review and future vision. *Journal of Educational Data Mining, 1*(1), 3-17.

Barber, B., & Hamilton, H.J. (2003). Extracting share frequent itemsets with infrequent subsets. *Data Mining and Knowledge Discovery, 7*(2), 153-185. doi: 10.1023/A: 1022419032620.

Bayardo, R., Agrawal, R., & Gunopulos, D. (2000). Constraint-based rule mining in Large Dense Databases. *Data Mining and Knowledge Discovery*, *4*, 217-240.

Bayardo, R. (2006). The Many Roles of Constraints in Data Mining. *Letter from the Guest Editor. SIGKDD Explorations*, *4*(1), 1-2.

Bhandane, C., Shah, K., & Vispute, P. (2012). An efficient parallel approach for frequent itemset mining of Incremental data. *International Journal of Scientific & Engineering Research*, *3*(2), 1-5.

Bhatt, U.Y., & Patel, P.A. (2015). Mining Interesting Rare Items with Maximum Constraint Model Based on Tree Structure. In *proceedings of 2015 Fifth*

International Conference on Communication Systems and Network Technologies (pp. 1065-1070). Gwalior, India, April 4-6. doi:10.1109/CSNT.2015.190.

Bonchi, F., & Goethals, B. (2004). FP-bonsai: the art of growing and pruning small FP-trees. *Advances in Knowledge Discovery and Data Mining*. *Lecture Notes in Computer Science*, *3056*, 155–160. doi: 10.1007/978-3-540-24775-3_19.

Boulicant, J.F., (2008). If constraint-based mining is the answer: what is the constraint? (Invited talk), In *IEEE International Conference on Data Mining Workshops*, doi:10.1109/ICDMW.2008.96.

Brailsford, S.C., Potts, C.N., & Smith, B.M. (1999). Constraint satisfaction problems: algorithms and applications. *European Journal of Operational Research*, *119*(3), 557-581. doi:10.1016/S0377-2217(98)00364-6.

Brin, S., Motwani, R., & Silverstein, C. (1997). Beyond market baskets generalizing association rules to correlations. In *proceedings of ACM SIGMOD International Conference on Management of data (*pp. 265 -276*)*.Tucson, Arizona, USA, May 13-15. doi: 10.1145/253260.253327.

Bucila, C., Gehrke, J., Kifer, D., & White, W.M. (2003). DualMiner: a dual-pruning algorithm for itemsets with constraints. *Data Mining and Knowledge Discovery*, *7*(3), 241-272.

Calders, T., Dextersb, N., Gillisc, J. J. M.,& Goethalsb, B. (2014). Mining frequent itemsets in a stream. *Information Systems*, *39*,233-255. doi:10.1016/j.is.2012.01.005.

Cameron, J.J., & Leung, C.K. (2011). Mining Frequent Patterns from Precise and Uncertain Data. *Journal of Systems and Computer, 1*(1), 3-22.

Chang,C.H., & Yang S.H., (2003). Enhancing SWF for incremental Association Mining by itemset Maintenance. In *proceedings of the 7th Pacific-Asia Conference on Knowledge Discovery and Data Mining* (pp. 301-312), Seol, Koria, April 30-May 2, doi: 10.1.1.7.7783

Chan, R., Yang, Q., & Shen, Y. (2003). Mining high utility Itemsets. In *proceedings of 3rd IEEE International Conference on Data Mining* (pp. 19-26). Florida, November 19-22. doi: 10.1109/ICDM.2003.1250893.

Chang, J.H & Lee, W.S., (2005). estWin: Online data stream mining of recent frequent itemsets by sliding window method. *Journal of Information Science*, *31*(2): 76-90, doi:10.1177/0165551505050785.

Cheung, W., Han, J., Ng, T., Wong, C.Y. (1996). Maintenance of discovering association rules in large databases: an incremental updating technique. In *proceedings of 12th IEEE International Conference on Data Engineering (*pp.106-114), New Orleans, L.A., February 26-March 01. doi: 10.1109/ICDE.1996.492094.

Cheung,W., Lee S.D, & Kao B. (1997). A general incremental technique for maintaining discovered association rules. *In proceedings of 5th International Conference on Database Systems for Advanced Applications* (pp.185-194), Australia.

Chuang, K.T., Huang, J.L., & Chen, M.S. (2008). Mining top-k frequent patterns in the presence of the memory constraint. *The International Journal on Very Large Data Bases*, *17*(5), 1321-1344. doi: 10.1007/s00778-007-0078-6.

Cheung, W., & Zaïane, O.R. (2003). Incremental Mining of Frequent Patterns without Candidate Generation or Support Constraint. In *proceedings of IEEE 7th*

International Conference on Database Engineering and Applications Symposium (pp. 111-116). Hong Kong, July 18. doi: 10.1109/IDEAS.2003.1214917.

Chiu, S.Y, Chiu, S.C. & Huang, J.L. (2009). On Mining Repeating Pattern with Gap Constraint. In *proceedings of 11th IEEE International Conference on High Performance Computing and Communications* (pp.557-562), Korea University, Seoul, Korea, June 25-27. doi:10.1109/HPCC.2009.65.

Delavari, N., & Phon-Amnuaisuk, S. (2008). Data Mining Application in Higher Learning Institutions, *Informatics in Education. 7*(1), 31-54.

Dutta Borah, M. & Jindal, R. (2016). Constraint based Frequent Pattern Mining from Incremental Educational data: Educational Data Mining Approach. Accepted for publication at the *2016 IEEE International Conference on Teaching and Learning in Education*, Universiti Tenaga Nasional, Malaysia, March 1-2

Fayyad, U.M., Piatetsky-Shapiro, G. & Smyth, P. (1996). From Data Mining to Knowledge Discovery in Databases. *Artificial Intelligence Magazine, 17*(3), 37-54.

Fournier-Viger, P., Wu, C., Zida, S., & Tseng, V. S. (2014). FHM: Faster high-utility itemset mining using estimated utility co-occurrence pruning. *In Foundations of Intelligent Systems. Lecture notes in computer science,* Springer International Publishing, *8502*, 83-92. doi: 10.1007/978-3-319-08326-1_9.

Gowda, S.M., Baker, R.S.J.d., Corbett, A.T., Rossi, L.M., (2013). "Towards Automatically Detecting Whether Student Learning is Shallow. *International Journal of Artificial Intelligence in Education, 23*(1), 50-70.

Guns, T., Nijssen, S., & Raedt, L.D. (2011). Itemset Mining: a Constraint Programming Perspective. *Artificial Intelligence, 175*, 1951-1983. doi:10.1016 /j.artint.2011.05.002.

Guns, T., Nijssen, S., & Raedt, L.D., (2013). K-Pattern Set Mining under Constraints. *IEEE transactions on Knowledge and Data Engineering, 25*(02), 402-418.

Guzzo, A., Moccia, L., Sacca, D., Serra, E.,(2013). Solving Inverse Frequent Itemset Mining with Infrequency Constarints via Large- Scale Linear programs. *ACM Transaction on Knowledge Discovery from Data, 7*(4),18:1-18:39.

Han, J., Lakshmanan, L.V.S, & Ng, R.T. (1999). Constraint-Based Multidimensional Data Mining. *Computer, 32*(8), 46-50. doi: 10.1109/2.781634.

Han, J., Pei, J., Yin, Y., & Mao, R., (2004). Mining Frequent Patterns without Candidate Generation: A Frequent-Pattern Tree Approach. *Data Mining and Knowledge Discovery, 8*, 53–87.

Han, J., Cheng, H., Xin, D., & Yan, X. (2007). Frequent pattern mining: current status and future directions. *Data Mining and Knowledge Discovery, 15*(1), 55-86. doi:10.1007/s10618-006-0059-1.

Han, J., & kamber, M. (2006). Data Mining Concepts and Techniques. Morgan Kaufmann Publishers, An imprint of Elsevier, San Francisco, CA 94111.

Hong, T.P., Lin, C.W., & Wu, Y.L. (2008). Incrementally fast updated frequent pattern trees. *Expert Systems with Applications, 34*, 2424-2435. doi: 10.1016 /j.eswa. 200.04.009.

Hu, J., & Mojsilovic, A. (2007). High-utility pattern mining: A method for discovery of high-utility item sets. *Pattern Recognition, 40*(11), 3317-3324. doi: 10.1016/j.patcog.2007.02.003.

Hu, T., Sunget, S.Y., Xiong, H, & Fu,Q., (2008). Discovery of maximum length frequent patterns. *Information Science, 178,* 69-87.

Jindal, R. & Dutta Borah, M. (2014). An approach to generate students' response on learning environment using Association Rule Mining. In *proceedings of the International Conference on Data Mining and Intelligent Computing* (pp. 1-5). Delhi, India, September 5-6. doi: 10.1109/ICDMIC.2014.6954225.

Jindal, R. & Dutta Borah, M. (2016). A novel approach for mining frequent patterns from incremental data. *International Journal of Data Mining, Modelling and Management*, Inderscience Publications, *8*(3), doi: 10.1504/IJDMMM.2016. 079071.

Jindal, R. & Dutta Borah, M. (2015 b). Predictive Analytics in a higher education context. IT Pro, IEEE Computer Society, *17*(4), 24-33. doi: 10.1109/MITP.2015.68.

Koedinger, K.R., Baker, R.S.J.d., Cunningham, K., Skogsholm, A., Leber, B., Stamper, J. (2010). A Data Repository for the EDM community: The PSLC DataShop. In Romero, C., Ventura, S., Pechenizkiy, M., Baker, R.S.J.d. (Eds.) Handbook of Educational Data Mining. Boca Raton, FL: CRC Press.

Knijf, J.D. & Feelders, A. (2005). Monotone Constraints in Frequent Tree Mining. In *proceedings of the Benelearn 2005 Annual Machine Learning Conference of Belgium and the Netherlands* (pp. 13-19). University of Twente, Enschede, The Netherlands, February 17-18.

Lan, G.C., Hong, T.P., & Tseng, V.S. (2014). An efficient projection-based indexing approach for mining high utility itemsets. *Knowledge and Information Systems, 38,* 85-107. doi:10.1007/s10115-012-0492-y.

Lee, G., & Chen, Y.C. (2012). Protecting sensitive knowledge in association pattern mining. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery, 2*(1), 60-68. doi:10.1002/widm.50.

Lee, C.H., Lin, C.R., & Chen, M.S. (2001). Sliding-window filtering: An efficient algorithm for incremental mining. In *proceedings of the 10th International Conference on Information and Knowledge Management* (pp. 263-270), November 5-10, Atlanta, Georgia.

Lee, D., Park, S.H., & Moon, S. (2013). Utility-based association rule mining: A marketing solution for cross-selling. *Expert Systems with Applications*, *40*(7), 2715-2725. doi: 10.1016/j.eswa.2012.11.021.

Lee, G., Yun, U., & Ryu, K.(2014). Sliding window based weighted maximal frequent pattern mining over data streams. *Expert Systems with Applications*, *41*(2), 694-708. doi:10.1016/j.eswa.2013.07.094.

Leung, C.K.S., Khan, Q.I, Li, Z. & Hoque, T. (2007). CanTree: a canonical-order tree for incremental frequent- pattern mining. *Knowledge and Information Systems*, *11*(3), 287-311. doi:10.1007/s/0115-006-0032-8.

Leung, C.K.S., Jiang, F., Sun,L., Wang,Y., (2012). A Constrained Frequent Pattern Mining System for handling Aggregate Constraints. In p*roceedings of the 16th*

International Database Engineering & Applications Symposium (pp.14-23), Prague, Czech Republic, August 8-10. doi: 10.1145/2351476.2351479, 2012

Li, Y.C., Yeh, J.S., & Chang, C.C. (2005a). Direct candidates generation: A novel algorithm for discovering complete share-frequent itemsets. In *proceedings of the 2nd International Conference on Fuzzy Systems and Knowledge Discovery* (pp. 551-560). Changsha, China, August 27-29. doi: 10.1007/11540007_67.

Li, Y.C., Yeh, J.S., & Chang, C.C. (2005b). A Fast algorithms for mining Share-Frequent Itemsets. In *proceedings of the 7th Asia Pacific Conference on Web Technology* (pp. 417- 428). Shanghai, China, March 29 - April 1.

Li, Y.C. , Yeh, J.S., & Chang, C.C. (2008). Isolated Items Discarding Strategy for Discovering High Utility Itemsets. *Data and Knowledge Engineering*. *64*(1), 198-217, doi: 10.1016/j.datak.2007.06.009.

Lim, A.H.L, & Lee, C.S. (2010). Processing online analytics with classification and association rule mining. *Knowledge-Based Systems*, 23(3), 248-255, doi:10.1016/j. knosys.2010.01.006

Lin, C.W., Hong, T.P., & Lu, W.H. (2009). The Pre-FUFP algorithm for incremental mining. *Expert Systems with Applications,* 36(5), 9498–9505. doi:10.1016/j.eswa. 2008.03.014.

Lin, C.W, Lan, G.C., & Hong, T.P. (2012). An incremental mining algorithm for high utility itemsets. *Expert Systems with Applications,* 39(8), 7173–7180. doi: 10.1016 /j . eswa.2012 .01.072.

Liu, Y., Liao, W.K. & Choudhary. A. (2005). A fast high utility itemsets mining algorithm. In *proceedings of 1st international workshop on Utility- Based Data Mining* (pp. 90–99), Chicago, Illinois, USA, August 21. doi:10.1145/1089827.1089839.

Liu, M., & Qu, J.F. (2012). Mining high utility itemsets without candidate generation. In *proceedings of the International Conference on Information and Knowledge Management* (pp. 55-64). Maui Hawaii, October 29- November 2.

Matsuda, N., & Ritter, S. (2011). AlgebraI2010Dec-retry-ss. Dataset 475 in DataShop [online] https://pslcdatashop.web.cmu.edu/DatasetInfo?datasetId=475 (accessed on January 1, 2014).

McGlohon, M., Akoglu, L., & Faloutsos, C. (2008). Weighted graphs and disconnected components: patterns and a generator. In *proceedings of the 14th ACM SIGKDD International Conference on Knowledge discovery and data mining* (pp. 424-532). Lasvegas, August 24-27. doi: 10.1145/1401890.1401955.

Mercheron, A., & Yacef, K. (2008). Interestingness Measures for Association Rules in Educational Data. In *proceedings of the 1st International Conference* on Educational Data Mining (pp. 57-66).Canada, June 20-21.

Mitrovic, A., & Ohlsson, S. (1999). Evaluation of a Constraint-Based Tutor for a Database language. *International Journal of Artificial Intelligence in Education, 10,* 238-256.

Nelson, B., Nugent, R., Rupp, A.A. (2012). On Instructional Utility, Statistical Methodology, and the Added Value of ECD: Lessons Learned from the Special Issue. *Journal of Educational Data Mining*. *4*(1), 224-230.

Nijssen, S., Guns, T., (2010). Integrating constraint programming and itemset mining. In *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases* (pp.1-16). Casa Convalescència, Spain, September 20 - 24.

Otey, M.E, & Parthasarathy, S. (2004). Parallel and Distributed Methods for Incremental Frequent Itemset Mining. *IEEE transactions on Systems, Man, and Cybernetics-Part B: Cybernetics*, *34*(6), 2439-2450.

Park, E., Kim, Y., Kim, I., Yoon, J., Lim, J., & Kim, U. (2012). CWFM: Closed Contingency Weighted Frequent Itemsets Mining. *Advances in Data Mining, Applications and Theoretical Aspects, Lecture Notes in Computer Science*, *7377*, 166-177. doi: 10.1007/978-3-642-31488-9_14.

Pei, J., Han, J., & Wang, W. (2007). Constraint-based sequential pattern mining: the pattern growth methods. *Journal of Intelligent Information System*, *28*, 133-160. doi: 10.1007/s10844-006-0006-z.

Pisharath, J., Liu, Y., Liao, W.K., Choudhary, A., Memik, G., & Parhi, J. (2005). NU-MineBench version 2.0 source code and datasets. [online] http://cucis.ece. northwestern.edu/techreports/pdf/CUCIS-2004-08-001.pdf, (accessed on 2nd January 2014).

Pyun, G., Yun, U., & Ryu, K. (2014). Efficient frequent pattern mining based on linear prefix tree. *Knowledge-Based Systems*, *55*,125-139. doi:10.1016 /j.knosys. 2013.10.013.

Raedt, L.D, Guns, T., & Nijssen, S. (2010). Constraint programming for data mining and machine learning. In *proceedings of the Association for the Advancement of*

Artificial Intelligence conference (pp. 1671-1675). Atlanta, Georgia, USA, July 11-15.

Raedt, L.D., & Zimmermann, A. (2007). Constraint-based pattern set mining. In *proceedings of the Seventh SIAM International Conference on Data Mining* (pp. 1-12). Minnesota, April 26-28.

Romero, C., & Ventura, S. (2007). Educational data mining: A survey from 1995 to 2005. *Expert Systems with Applications, 33*(1), 1135-146. doi:10.1016/j.eswa.2006.04.005.

Romero, C., & Ventura, S. (2010). Educational Data Mining: A review of the state of the Art. *IEEE Transaction on System Man and Cyber.-Part C: Application and review, 40* (6), 601-618. doi: 10.1109/TSMCC.2010.2053532.

Romero, C., & Ventura, S. (2013). Data Mining in Education. *WIREs Data Mining and Knowledge Discovery, 3*(1), 12-27. doi: 10.1002/widm.1075.

Ruggieri, S. (2010). Frequent Regular Itemset Mining. In *proceedings of the 16th ACM International Conference on Knowledge Discovery and Data Mining* (pp. 263-272). Washington, DC, July 25-28.

Saxena, P.C., De, Asok, Jindal,R. (2006). Constraint Based Dataset Filtering for Mining Access Patterns from Web Logs. *Journal of the CSI, 36*(4), 8-12.

Shie, B.E., Yu, P. S., & Tseng, V. S. (2012). Efficient algorithms for mining maximal high utility itemsets from data streams with different models. *Expert Systems with Applications, 39*(17), 12947-12960. doi: 10.1016/j.eswa.2012.05.035.s

Song, W., Liu, Y., & Li, J. (2014). BAHUI: Fast and memory efficient mining of high utility itemsets based on bitmap. *International Journal of Data Warehousing and Mining, 10*(1), 1-15. doi: 10.4018/ijdwm.2014010101.

Srikant, R, & Agrawal, R. (1996). Mining Sequential Patterns: generalizations and performance improvements. In *proceedings of the International Conference on Extending Database Technology* (pp. 3-17), Avignon, France, March 25-29.

Steinbach, M., Tan, P. N, Xiong, H. & Kumar,V.(2000). Mathematics Subject Classification. Primary 62-07, 68P99; Secondary 62P10, 2007 American Mathematical Society, Accessed on : Jan 2014, Available in :.researchgate.net% 2Fpublication % 2F2287874 92_Objective _measures _for_ association_pattern_ analysis%2Ffile%2F72e7e5182f2bdd2555.pdf&ei=n_FYU4n9BoXPrQff3oHoDQ& usg=AFQjCNFH9c65SyAyW51VaTbEPB2zA_AnJg&bvm=bv.65397613,d.bmk

Su, M.Y., Yu, C.J., & Lin, C.Y. (2009). A real time network intrusion detection system for large-scale attacks based on an incremental mining approach. *Computers & Security, Science Direct, 28*(5), 301-309. doi: 10.1016/j.cose.2008.12.001.

Sun, K., & Bai, F. (2008). Mining Weighted Association Rules without Pre assigned Weights. *IEEE Transactions on Knowledge and Data Engineering, 20* (4), 489-495.

Tseng, V.S., Wu, C.W., Shie, B.E., & Yu, P.S. (2010). UP-Growth: An Efficient Algorithm for High Utility Itemsets Mining. In p*roceedings of the 16th ACM SIGKDD Conf. Knowledge Discovery and Data Mining* (pp. 253-262). Washington, DC, USA, July 25-28.

Tseng, V.S., Shie, B.E, Wu, C.W., & Yu, P.S. (2013). Efficient Algorithms for Mining High Utility Itemsets from Transactional Databases. *IEEE Transactions on Knowledge and Data Engineering*, *25*(8), 1772-1786.

Uno, T., Kiyomi, M. & Arimura, H. (2005). LCM ver.3: collaboration of array, bitmap and prefix tree for frequent itemset mining. In *proceedings of 1st International Workshop on Open Source Data Mining* (pp. 77-86). doi: 10.1145/1133905.1133916.

Verma, K & Vyas, O.P.,(2005). Efficient calendar based temporal association rule. SIGMOD Record, *34*(3), 63-70.

Veloso, A., Pôssas, B., Meira, W., Márcio, Jr., Carvalho, B. (2001). Knowledge management in association rule mining, In *proceedings of Integrating Data Mining and Knowledge Management, held in conjunction with the 2001 IEEE International Conference on Data Mining*, doi: 10.1.1.63.9197.

Vo, B., Coenen, F., & Le, B. (2013). A new method for mining frequent weighted itemsets based on WIT-trees. *Expert Systems with Applications*, *40*(4), 1256-1264, doi:10.1016/j.eswa.2012.08.065.

Wang, W., Yang, J. & Yu, P. (2000). Efficient mining of Weighted Association Rules (WAR). In *proceedings of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining* (pp. 270-274). Boston,USA.

Wang, W., Yang, J., & Yu, P. (2004). WAR: Weighted Association Rules for Item Intensities. *Knowledge and Information Systems, 6*(2), 203-229. doi: 10.10 07/s 101 15-003-0108-7.

Wang, J., Lu,Z., Wu, W., & Li,Y. (2012). The Application of Data Mining Technology based on Teaching Information. In *proceedings of the 7th International Conference on Computer Science & Education* (pp.652-657), Melbourne, VIC, July 14-17, doi: 10.11 09 /ICCSE.2012.6295159.

Wojciechowski, M. & Zakrzewicz, M., (2002). Dataset Filtering Techniques in Constraint-Based Frequent Pattern Mining. *Pattern Detection and Discovery, Lecture Notes in Computer Science*, *2447*, 77-91. doi. 10.1007/3-540-45728-3_7.

Yang, C.H., & Yang, D.L. (2009). IMBT-A Binary Tree for Efficient Support Counting of Incremental Data Mining. In *proceedings of International Conference on Computational Science and Engineering* (pp. 324-329). IEEE Computer Society Washington, DC, USA, August 29-31. doi: 10.1109/CSE.2009.360.

Yao, H., Hamilton, H. J., & Butz, C. J. (2004). A foundational approach to mining itemset utilities from databases. In *proceedings of 3rd SIAM International Conference Data Mining* (pp. 482-486). San Francisco, CA, May 1-3.

Yu, C., & Chen, Y. (2005). Mining sequential patterns from multidimensional sequence data. *IEEE Transactions on Knowledge and Data Engineering*, *17*(1), 136-140. doi: 10.1109/TKDE.2005.13.

Yun, U., & Leggett, J.J. (2005). WFIM: Weighted Frequent Itemset Mining with a Weight Range and a Minimum Weight. In *proceedings of the SIAM International Conference on Data Mining* (pp. 636-640). Newport Beach, CA, April 21-23. doi: 10.1137/1.9781611972757.76.

Yun, U., & Leggett, J.J. (2006). WIP: Mining Weighted Interesting Patterns with a Strong Weight and/or Support Affinity. In *proceedings of the SIAM International Conference on Data Mining* (pp. 623-627). Bethesda, April 20-22.

Yun, U. (2007). Efficient Mining of Weighted interesting Patterns with a Strong Weight and/or Support Affinity. *Information Sciences, 177*(17), 3477-3499. doi:10.1016/j. ins. 2007.03.018.

Yun, U. (2008). An Efficient Mining of Weighted Frequent Patterns with Length Decreasing Support Constraints. *Knowledge-Based Systems, 21*(8), 741-752. doi:10.10 16 /j.knosys.2008.03.059.

Yun, U., & Ryu, K.H. (2011). Approximate weighted frequent pattern mining with without noisy environments. *Knowledge-Based Systems, 24*(1), 73-82. doi: 10.1016/j. knosys.2010. 07.007.

Yun, U., Lee, G., & Ryu, K.H. (2014). Mining maximal frequent patterns by considering weight conditions over data streams. *Knowledge-Based Systems, 55,* 49-65. doi: 10.10 16/j.knosys.2013.10.011.

Yun, U., Ryang, H., & Ryu, K.H. (2014). High Utility itemset mining with techniques for reducing overestimated utilities and pruning candidates. *Expert Systems with Applications, 41*(8), 3861-3878. doi: 10.1016/j.eswa.2013.11.038.

Zaki, M.J. (2000). Scalable algorithms for Association Mining. *IEEE Transactions on Knowledge and Data Engineering, 12*(3), 372-390. doi: 10.1109/69.846291.

Biography of Author

Malaya Dutta Borah is a full time research scholar in the Department of Computer Science & Engineering at Delhi Technological University (formerly Delhi College of Engineering), New Delhi, India. She did her M.E. from Delhi College of Engineering (now Delhi Technological University) under Delhi University, New Delhi in 2011.

Prior to this she worked at Delhi Technological University (2007-2012) as Assistant Professor (Contractual Appointment); Assam Engineering College (2005-2006) as Guest Lecturer; Central IT College Assam (2005-2006) as Lecturer and Community Information Centre Dhakuakhana, a CIC project under the Ministry of Information & Communication Technology, Govt of India (2002-2005) as CIC Operator.

Her research interests lie in the area of Data Mining, Cloud Computing, ICT and e-Governance. She has authored/co-authored around 26 research papers in International/National Journals/Conferences of repute. She has guided one M.Tech and four B.Tech final year projects so far. She is also associate member of CSI (Membership No.: 00173524) and member of IEEE (Membership No.: 94117178).