HIGH UTILITY ITEMSETS IDENTIFICATION IN BIG DATA

by

Ashish Tamrakar

Bachelor Degree in Computer Engineering
Tribhuvan University, Kathmandu, Nepal
2012

A thesis submitted in partial fulfillment of
the requirements for the

Master of Science in Computer Science

Department of Computer Science
Howard R. Hughes College of Engineering
The Graduate College

University of Nevada, Las Vegas
May 2017

The Graduate College

We recommend the thesis prepared under our supervision by

Ashish Tamrakar

entitled

High Utility Itemsets Identification in Big Data

be accepted in partial fulfillment of the requirements for the degree of

Master of Science in Computer Science

Department of Computer Science

Justin Zhan, Ph.D., Committee Chair

Laxmi Gewali, Ph.D., Committee Member

Fatma Nasoz, Ph.D., Committee Member

Ge Lin Kan, Ph.D., Graduate College Representative

Kathryn Hausbeck Korgan, Ph.D., Interim Graduate College Dean

May 2017

Abstract

High utility item set mining is an important data mining problem which considers profit factors besides quantity from the transactional database. It helps find the most valuable products/items that are difficult to track using only the mere frequent data mining set. An item that has a high-profit value might be rare in the transactional database despite its tremendous importance. While there are many existing algorithms which generate comparatively large candidate sets while finding high utility itemsets, the major focus is to reduce the computational time significantly with the introduction of pruning strategies. Another aspect of high utility itemset mining is to compute the large dataset. There are very few algorithms that can handle large dataset to find high utility itemset mining in a parallel (distributed) system.

In this thesis, there are two proposed methods: High utility itemset mining using pruning strategies approach (HUI-PR) and Parallel EFIM (EFIM-Par). In the method I, the proposed algorithm constructs the candidate sets in the form of a tree structure, which traverses the itemsets with high transaction weighted utility (HTWUs). It uses a pruning strategies to reduce the computational time by refraining the visit to unnecessary nodes of an itemset to reduce the search space. It significantly minimizes the transaction database generated on each node. In the method II, the distributed approach is proposed dividing the search space among different worker nodes to compute high utility itemsets which are aggregated to find the result. The experimental results for both methods show that they significantly improve the execution time for computing the high utility itemsets.

Acknowledgements

I would like to express my sincere gratitude to my advisor, Dr. Justin Zhan, for his continuous motivation, guidance and support to drive me into research not only for my thesis but throughout my Master's program.

Furthermore, I would like to acknowledge Dr. Laxmi Gewali, Dr. Fatma Nasoz, and Dr. Ge Lin Kan for their support and being part of my thesis committee. I am thankful to Dr. Ajoy Datta who has always been available to me whenever I needed guidance since the beginning of my Master's program.

I am truly grateful to all my Big Data Hub labmates for their support. I am thankful to Dr. Ming-Tai Wu (Jimmy) and Pradip Singh Maharjan who have provided me with insightful suggestions and comments throughout my thesis.

I must thank my dad Anand Krishna Tamrakar, my mom Shanti Tamrakar and my sister Arati Tamrakar for their unconditional love, support, and care without whom I would not have been here. And, I am especially thankful to Shuveksha Tuladhar, for her continuous support and motivation.

Last but not the least, I am thankful to all my friends, seniors, and juniors who made my time here in UNLV very enjoyable one.

Ashish Tamrakar

University of Nevada, Las Vegas

May 2017

Table of Contents

Abstract iii

Acknowledgements iv

Table of Contents v

List of Tables vii

List of Figures viii

List of Algorithms ix

Chapter 1 Introduction 1

1.1 Objective 2

1.2 Outline 3

Chapter 2 Background and Preliminaries 4

2.1 Related Work 4

2.2 Preliminaries 6

2.2.1 High Utility Itemset Mining 6

2.2.2 Distributed Systems 12

Chapter 3 Proposed System 14

3.1 Method I - Pruning Strategies Approach 14

3.1.1 Construction of 1-HTWUIs Tree of Items 14

3.1.2 Node Selection Rule 15

3.1.3 Construction of Sub-tree of Itemsets 15

3.1.4 Pruning Strategies 16

3.1.5 Detailed Algorithm (HUI-PR) 18
3.2 Method II - Distributed EFIM 19
3.2.1 Generating 1-HTWUIs with their corresponding TWU 19
3.2.2 Generating Revised Transactional Database 20
3.2.3 Finding Local Utility and Sub-tree Utility of 1-HTWUIs 20
3.2.4 Sub-tree Assignment to Worker Nodes 21
3.2.5 Node Data Generation 22
3.2.6 Mining High Utility Itemsets 23
3.2.7 Overall Flow of EFIM Parallel Algorithm 24

Chapter 4 Experimental Results **26**
4.1 Datasets 26
4.2 HUI-PR vs EFIM 27
4.2.1 Comparison of Computational Time 27
4.2.2 Comparison of HUIs 28
4.2.3 Comparison of Candidate Sets 28
4.3 EFIM-Par vs EFIM 34
4.3.1 Comparison of Computational Time 34
4.3.2 Comparison of HUIs 34

Chapter 5 Conclusion and Future Work **38**

Bibliography **40**

Curriculum Vitae **43**

List of Tables

2.1 A Transactional Database D 6

2.2 A Profit Table 6

2.3 Transaction Weighted Utility of 1-TWU Items 9

2.4 Revised Transactional Database 9

2.5 Projected Database for Itemset D of 1-HTWUIs 10

4.1 Datasets Characteristics 27

4.2 Total Number of HUIs found in HUI-PR and EFIM 31

4.3 Total Number of Transactions Pruned in HUI-PR 31

4.4 Total Number of HUIs found in EFIM-Par and PHUI-Miner 37

List of Figures

2.1 Construction of Tree Structure of Itemsets 10

3.1 Construction of 1-HTWUI Tree of Items 15

3.2 Overall Flow Diagram of EFIM Parallel Algorithm 25

4.1 Comparison of computational time between HUI-PR and EFIM w.r.t. variants of minimum threshold for different datasets 29

4.2 Comparison of computational time with state-of-the-art algorithms w.r.t. variants of minimum threshold 30

4.3 Comparison of candidate sets between HUI-PR and EFIM w.r.t. variants of minimum threshold 32

4.4 Comparison of candidate sets with state-of-the-art algorithms w.r.t. variants of minimum threshold 33

4.5 Comparison of computational time between EFIM-Par and PHUI-Miner w.r.t. variants of minimum threshold for different datasets 35

4.5 Comparison of computational time between EFIM-Par and PHUI-Miner w.r.t. variants of minimum threshold for different datasets 36

List of Algorithms

1 Build Sub-tree to determine Itemsets 16

2 Checking in Pruning Hash Table for Transaction Pruning 17

3 Insertion in Pruning Hash Table 17

4 Algorithm to find HUIs 19

5 Revised Transactional Database Generation 20

6 Assignment of Sub-tree to Worker Nodes 21

7 Node Data Generation 22

8 Mining HUIs in Parallel 23

Chapter 1

Introduction

The challenge in big data mining has been finding the meaningful information in large datasets. A technique in data mining to discover interesting, unexpected and useful patterns of data from a large database is called pattern mining. For example, if the customer buys a mobile phone then he/she is most likely to buy phone cover and screen protector as well. These patterns are found based on the mining of large database transactions. It can also be used in the recommendation systems by accessing the history of customers and arrangement of goods in the departmental stores. In the past, most research in pattern mining focuses on Frequent Itemset Mining (FIM) and Associative Rule Mining (ARM), which are the traditional ways to find the frequent set of itemset patterns which are higher than the minimum support threshold[CHY96]. These mining patterns occur frequently within a huge transaction database. Apriori algorithm was proposed for frequent itemset mining which scans database in multiple scans and large candidate sets were generated [AS94]. To overcome the limitation of Apriori algorithm, FP-Growth was then proposed which discovers all the frequent patterns with only two scans of the transactional database [HPY00]. Although FIM was a great discovery to mine frequently occurring itemsets, it gives equal importance to all items in the transactional database. It only gives importance to quantity however the importance of profit was lacking. For example, the sale of milk and bread occurrence is frequent in the transactions of the dataset while the sale of diamond seems to be rare and it might not be reflected in the outcome of the FIM and ARM. Therefore, it is necessary to consider both profit and quantity of the itemsets. Consequently, the concept of utility mining was introduced.

Utility pattern mining has been one of the most significant research works proposed to discover the useful and profitable itemsets from the large transactional datasets[CYS03]. Utility mining includes both internal utility and external utility to compute the itemset where internal utility is represented by the quantity of an item and external utility is represented by the profit of an item[YH06]. A minimum utility threshold is used to discover whether an itemset is a high utility itemset or not. Recently, many types of research have been carried out in the field of high utility itemset mining [YHB04, YH06, YL07, YRR14]. Liu et al. proposed a two-phase model which computes transaction weighted utility (TWU) and considers the transaction weighted download closure property to find high utility itemsets [LLC05]. Downward closure property defines that every sub-pattern of itemsets must also be frequent. However, this algorithm by Liu et al. generates a large number of candidates in order to find the high utility itemsets. Therefore, the performance is not optimized even for the smaller datasets. It takes a lot of computational time and memory to process a large number of candidates. Different methods were proposed to reduce the possible number of candidate sets [TWSY10, TSWY13]. Liu et al. [LQ12] proposed an approach to find the high utility mining without candidate generation. And, Zida et al. [ZFVL$^+$15] proposed different upper-bound pruning to reduce candidate sets. Different pruning approaches have been introduced so far to reduce the number of candidate sets generation. However, these state-of-the-art algorithms perform well when the dataset is small. When the size of the dataset increases, the performance degrades. Therefore with the current era of big data, there is a need to compute datasets in multiple machines, which is possible through distributed computing.

One of the methods of distributed computing is to implement Map-Reduce framework [DG08] with Hadoop. This framework is highly scalable and fault-tolerant system and can process large datasets on multiple clusters. Hadoop framework can be implemented on less powerful and cheap machines. However, this popular framework is a disk-based paradigm and is heavily dependent on its Hadoop Distributed File System (HDFS). Another framework named Spark [ZCF$^+$10] was introduced to overcome its heavy dependency with HDFS by allowing in-memory computation. Spark framework can perform up to 100 times faster than Hadoop. Spark uses Resilient Distributed Dataset (RDD) which is an immutable data structure allowing efficient reuse of data for in-memory computation.

1.1 Objective

The objective of this thesis is to extend the state-of-the-art algorithm named EFIM: A Highly Efficient Algorithm for High-Utility Itemset Mining (EFIM) [ZFVL$^+$15] with a novel pruning strategies approach for smaller datasets. In this thesis, an algorithm named High utility itemset mining using pruning strategies approach (HUIPR) is proposed which uses a pruning hash table to reduce the searching area in EFIM algorithm and another utility bound named transaction weighted utility is also considered. These proposed pruning strategies reduce the number of candidate sets generated reducing the computational time to find the high utility itemsets. Another contribution of this thesis is to build the distributed system using Apache Spark framework from the EFIM algorithm named EFIM parallel computing (EFIMPar) for larger datasets.

1.2 Outline

In Chapter 1, the brief topic of itemset mining, its application in real world and the proposed method was described.

In Chapter 2, we will discuss the related works with our proposed methods and the background information required to understand the Itemset mining. It will also cover the distributed system and the most popular distributed computing frameworks.

In Chapter 3, we will propose two methods to find the high utility itemsets. One method will be based on the pruning strategies approach which is suitable for the smaller datasets while another method will be based on the distributed computing approach for the very large datasets. The detailed description will be provided along with the examples of these methods.

In Chapter 4, we will present the experimental results showing the difference between our methods and the state-of-the-art algorithms based on the time, accuracy and efficiency. The characteristics of datasets used for these methods will also be described.

In Chapter 5, we will summarize the proposed methods and their results along with the possible extension of this thesis.

Chapter 2

Background and Preliminaries

2.1 Related Work

Many researchers have been focusing their efforts in the field of high utility itemset mining (HUIM). HUIM started with the pattern mining concepts [CHY96, BH03, LYC05, HPY00] such as Frequent Itemset Mining (FIM) and Associative Rule Mining. The initial breakthrough came when Agrawal and Srikant [AS94] proposed a method named Apriori. However, Han et al. [HPY00] proposed the FP-Growth algorithm, with a tree-structure named FP-tree to improve performance than Apriori algorithm. FIM does not emphasize the importance of items and quantities of items. Therefore, there is the need for weighted FIM (WTI-FWI) [YR13, YRR14]. These methods that focus on weight gives importance to items.

High utility itemset mining (HUIM) [ATJL09, YH06, EGA07, YL07, LHL11, WSTY12, YHB04, TSWY13] gives the importance to the item quantities and profit value (external utility). This concept was firstly proposed by Yao et al. [YHB04]. Liu et al. [LLC05] proposed a Two-Phase algorithm based on Apriori to find high utility itemsets using multiple database scans. The initial scan generates the high transaction weighted utility items (1-HTWUIs) for the first level which accepts only items that have transaction weighted utility (TWU) higher than the threshold value. The second scan generates the candidate sets based on the 1-HTWUIs and considers only those itemsets with TWU higher than the minimum threshold. The next scan selects the high utility itemsets (HUI) with higher utility value than the minimum threshold value. This maintains the downward closure property. However, for each level of the tree, this algorithm generates a large number of candidate sets.

To reduce the overestimated utilities, different pruning approaches have been proposed [CTL09, LQ12, LHT14, Kri15, ZFVL+15, TSWY13, FVWZT14, SLL14, FVLDD16]. Liu et al. [LQ12] proposed the mining of high utility itemsets without candidate generation. A utility-list was used to store the information about utilities for itemsets. These utility-lists also helped to prune unnecessary candidates. However, this algorithm uses a large amount of memory for utility list for each itemset. Zida et al. [ZFVL+15] also use the concept of utility-lists and proposed two upper bounds named sub-tree utility and local utility for pruning the search space. These bounds are described in the following sections. It also uses the fast utility counting technique to reduce the memory usage. Fournier-Viger et al. [FVLDD16] introduced the pruning strategy of length upper bound reduction by constraining the generation of candidate sets up to given maximum length of itemsets.

Since the advent of Big data, many research works have been on computing the very large datasets using parallel computing. Initially, simple approaches for map-reduce framework have been used for frequent itemset mining[YLF10, CRC10]. There are very few algorithms proposed so far for both the frequent itemset mining and high utility itemset mining. Li et.al [LWZ+08] proposed the Parallel FP-Growth algorithm to find the frequent itemsets in a distributed approach with multiple map-reduce stages. For the parallel high utility itemset mining, Lin et.al [LWT15] proposed the parallel UP-Growth (PHUI-Growth) algorithm with counting map-reduce phase and mining phase using Hadoop framework [DG08]. Another algorithm proposed by Chen et.al [CA16] implemented the distributed approach (PHUI-Miner) on HUI-Miner algorithm [LQ12] to perform better than PHUI-Growth. PHUI-Miner replaces the Hadoop framework [DG08] with more efficient Spark framework [ZCF+10]. Spark performs much better because of its ability to perform an in-memory computation.

Hence, the number of candidate sets reduction by applying pruning rule for smaller datasets and parallel computing for high utility itemsets in large datasets play a significant role in improving the performance in the identification of high utility itemsets. Therefore, our thesis aims to construct a novel approach to generate candidate sets efficiently and to apply proposed pruning strategies to reduce the unnecessary candidate sets. The distributed approach can be used with Spark framework on state-of-the-art algorithm EFIM [ZFVL+15] to improve the computational time for finding the high utility itemsets.

Table 2.1: A Transactional Database D

TID	**Transaction (item:quantity)**	**TU**
T_1	A:3, B:2, D:2	17
T_2	A:4, C:1, D:3, E:2	15
T_3	A:2, B:1, E:3, F:5, G:2	22
T_4	B:2, C:1, E:3	16
T_5	B:1, C:1, E:1, F:1	11

Table 2.2: A Profit Table

Item	**Profit Value**
A	1
B	5
C	3
D	2
E	1
F	2
G	1

2.2 Preliminaries

2.2.1 High Utility Itemset Mining

Let us suppose the transactional database with set of transactions $D = T_1, T_2, ..., T_n$ which has the finite set of m unique items $I = i_1, i_2, ..., i_m$. Each transaction in database, $T_q \in D$ where $1 \leq q \leq n$ has a unique identifier, called its Transaction ID (TID). Each item i_j is associated with quantity, which is internal utility, and with its associated profit value, which is external utility. Internal utility is denoted by $q(i_j, T_q)$ and external utility by $pft(i_j)$. A set of k unique items $X = i_1, i_2, ..., i_k$ where $X \subseteq I$ is said to be a k-itemset, where k is the length of an itemset and an itemset X is in transaction T_q if $X \subseteq T_q$ and a minimum threshold ratio δ is defined.

An illustrative example is shown in Table 2.1 which represents the quantitative (transactional) database. There are five transactions with seven distinct items in the quantitative database. Table 2.2 represents the profit table which contains profit value for each item. The user specified threshold ratio δ is taken as 30.86% which will be threshold value of $25(TU \times \delta)$.

Definition 2.2.1 *The utility of an item i_j denoted by $u(i_j, T_q)$ in a transaction T_q is defined as,*

$$u(i_j, T_q) = q(i_j, T_q) \times pft(i_j) \tag{2.1}$$

The utility of items A, B and D in transaction T_1 are calculated using the Equation 2.1 as,

$u(A, T_1) = q(A, T_1) \times pft(A) = 3 \times 1 = 3$

$u(B, T_1) = q(B, T_1) \times pft(B) = 2 \times 5 = 10$

$u(D, T_1) = q(D, T_1) \times pft(D) = 2 \times 2 = 4$

Definition 2.2.2 *The utility of an itemset X denoted by $u(X, T_q)$ in a transaction T_q is defined as,*

$$u(X, T_q) = \sum_{i_j \subseteq X \cap X \subseteq T_q} u(i_j, T_q) \tag{2.2}$$

The utility of itemsets in transaction T_1 is calculated from Equation 2.2 as,

$u(AB, T_1) = u(A, T_1) + u(B, T_1) = 3 + 10 = 13$

$u(ABD, T_1) = u(A, T_1) + u(B, T_1) + u(D, T_1) = 3 + 10 + 4 = 17$

Definition 2.2.3 *The utility of an itemset X denoted by $u(X)$ in database D is defined as,*

$$u(X) = \sum_{X \subseteq T_q \cap T_q \in D} u(X, T_q) \tag{2.3}$$

The utility of itemsets C and D in database D is calculated from Equation 2.3 as,

$u(AB) = u(AB, T_1) + u(AB, T_3) = 13 + 7 = 20.$

Definition 2.2.4 *The transaction utility of a transaction T_q denoted by $TU(T_q)$ is defined as,*

$$TU(T_q) = \sum_{X \subseteq T_q} u(X, T_q) \tag{2.4}$$

The transaction utility of a transaction T_1 is calculated from Equation 2.4 as,

$TU(T_1) = u(A, T_1) + u(B, T_1) + u(D, T_1) = 3 + 10 + 4 = 17.$

Similarly, the total utility for other transactions are $T_2 = 15, T_3 = 22, T_4 = 16$ and $T_5 = 11$ as shown in Table 2.1.

Definition 2.2.5 *The total utility denoted by TU in database D is defined as,*

$$TU = \sum_{T_q \in D} TU(T_q) \tag{2.5}$$

The total utility is calculated from Equation 2.5 as,

$TU = 17 + 15 + 22 + 16 + 11 = 81.$

Definition 2.2.6 *The transaction weighted utility of an itemset X denoted by $TWU(X)$ in database D is defined as,*

$$TWU(X) = \sum_{X \subseteq T_q \in D} TU(T_q) \tag{2.6}$$

The transaction weighted utility for an itemset {A,B} is calculated from Equation 2.6 as,

$TWU(AB) = TU(T_1) + TU(T_3) = 17 + 22 = 39.$

Definition 2.2.7 *An itemset X in a database D is a high transaction weighted utility itemset (HTWUI) if its TWU is greater than or equal to the minimum threshold, where minimum threshold is TU multiplied by user specified threshold ratio δ as,*

$$HTWUI \leftarrow \{X | TWU(X) \geq TU \times \delta\} \tag{2.7}$$

Since an itemset $\{A, B\}$ has $TWU(AB) \geq TU \times \delta (81 \times 30.86 = 25)$, it is therefore a high transaction weighted utility itemset.

Definition 2.2.8 *An itemset X in a database D is a high utility itemset (HUI) if its utility is greater than or equal to the minimum threshold, where minimum threshold is TU multiplied by user specified threshold ratio δ as,*

$$HUI \leftarrow \{X | u(X) \geq TU \times \delta\} \tag{2.8}$$

An itemset $\{A, B\}$ has $u(AB) \leq TU \times \delta$, it is not a high utility itemset (HUI). Similarly, an itemset $\{B, E\}$ has $u(BE) \geq TU \times \delta$, it is a HUI.

Definition 2.2.9 *The total ordering denoted by $\rightarrow$ is the ordering of items in the increasing order of transaction weighted utility in the transaction.*

The transaction weighted utility for each item is as shown in the Table 2.3. The increasing order of items in terms of TWU is: G, D, F, C, A, E, B ($G \rightarrow D \rightarrow F \rightarrow C \rightarrow A \rightarrow E \rightarrow B$).

Definition 2.2.10 *The revised transaction (RT) is said to be a transaction in which all the items which have $TWU \leq TU \times \delta$ are removed and the items remaining are sorted in increasing order of TWU. The items that are removed from the transactions are said to be unpromising items.*

Table 2.3: Transaction Weighted Utility of 1-TWU Items

Itemset	**{A}**	**{B}**	**{C}**	**{D}**	**{E}**	**{F}**	**{G}**
TWU	54	66	42	32	64	33	22

Table 2.4: Revised Transactional Database

TID	**Transaction (item:utility)**
T_1	D:4, A:3, B:10
T_2	D:6, C:3, A:4, E:2
T_3	F:10, A:2, E:3, B:5
T_4	C:3, E:3, B:10
T_5	F:2, C:3, E:1, B:5

From the given illustrative example in Table 2.1 and Table 2.2, the revised transactional database after removing the unpromising items and the items arranged in increasing order of TWU are as shown in Table 2.4.

Definition 2.2.11 *The remaining utility denoted by $rem(X,T)$ in the transaction T with total ordering ($\rightarrow$) of items on itemset X is defined as,*

$$rem(X,T) = \sum_{i_j \in T \cap i_j \rightarrow z \forall z \in X} u(i_j,T) \tag{2.9}$$

From the Table 2.4, the remaining utility for the itemset $\{D,C\}$ in transaction T_2 is,

$rem(DC,T_2) = u(A,T_2) + u(E,T_2) = 4 + 2 = 6.$

Definition 2.2.12 *The extension of an itemset γ denoted by $Ex(\gamma)$ is the possible following items for the given itemset γ.*

From Figure 2.1, the extension of an itemset {A} is {B,E} and similarly, for itemset {C} is {A, E,B}.

Definition 2.2.13 *The projected database of a revised transactional database D denoted by γD of an itemset γ is as,*

$$\gamma D = \{\gamma T | T \in D \cap \gamma T \neq \phi\} \tag{2.10}$$

where, $\gamma T = \{i_j | i_j \in T \cap i_j \in Ex(\gamma)\}$ is the projection of a transaction T of an itemset γ.

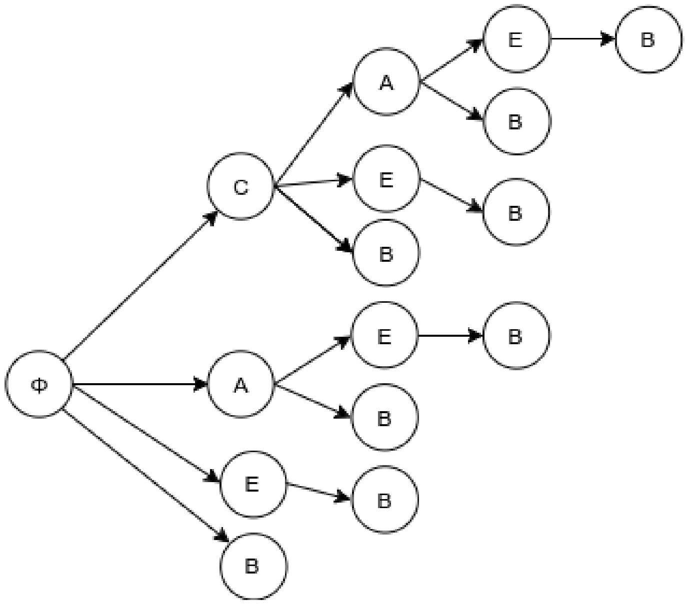


Figure 2.1: Construction of Tree Structure of Itemsets

Table 2.5: Projected Database for Itemset D of 1-HTWUIs

TID	**Transaction (item:utility)**
T_1	A:3, B:10
T_2	C:3, A:4, E:2

The projected database for the itemset D of 1-HTWUIs is as shown in the Table 2.5.

Definition 2.2.14 *The projected transaction merging is the method of merging the identical projected transactions (γT) and the utility from each transaction is merged into one as,*

$$u(i, T_m) = \sum q(i, T_k) \tag{2.11}$$

where, k is the number of identical projected transactions.

From the illustrative example from Table 2.4, considering $\gamma = \{C\}$, γD gets projected transactions of $\{A, E\}$ from T_2, $\{E, B\}$ from T_4 and $\{E, B\}$ from T_5. The projected transactions from T_4 and T_5 are merged to form a single projected transaction in the γD database. As a result, the new projected database will have $\{A, E\}$ and $\{E, B\}$ transactions.

Definition 2.2.15 *The utility-bin denoted by Ub is an array with length equal to the number of items I in the database D. For each itemset $x \in I$, the utility bin is denoted as $Ub[x]$.*

Definition 2.2.16 *The sub-tree utility denoted by $subU(\gamma, x)$ of an itemset γ and an item x which can have extension of γ is as,*

$$subU(\gamma, x) = \sum_{T\in(\gamma\cup\{x\})}[u(\gamma, T) + u(x, T) + \sum_{i_j\in T\cap E(\gamma\cup\{x\})} u(i_j, T)] \tag{2.12}$$

This sub-tree utility is one of the pruning strategies to reduce the search space. If $subU(\gamma, x) < TU \times \delta$ then, an itemset $\gamma \cup \{x\}$ can be pruned.

Referring to the Table 2.4, assuming the items are in total ordering as G, D, F, C, A, E, B, let us assume $\rho = \{\phi\}$, then the sub utility from Equation 2.12 for the following items i_j - E, D can be shown as,

$subU(\rho, \{E\}) = \sum_{T\in(\rho\cup\{E\})}[u(\rho, T)+u(\{E\}, T)+\sum_{i_j\in T\cap E(\rho\cup\{E\})} u(i_j, T)]$, where $T = T_2, T_3, T_4, T_5$

$= (0 + 2 + (0))_{T_2} + (0 + 3 + (5))_{T_3} + (0 + 3 + (10))_{T_4} + (0 + 1 + (5))_{T_5} = 29$

$subU(\rho, \{D\}) = (0 + 4 + (3 + 10))_{T_1} + (0 + 6 + (3 + 4 + 2))_{T_2} = 32$

Similarly, let us assume $\rho = \{D\}$, referring to the Table 2.5 the sub utility for items A and E are as,

$subU(\rho, \{A\}) = (4 + 3 + (10))_{T_1} + (6 + 4 + (2))_{T_2} = 29$

$subU(\rho, \{E\}) = (6 + 2 + (0))_{T_2} = 8$

Definition 2.2.17 *The local utility denoted by $locU(\gamma, x)$ for an itemset is as,*

$$locU(\gamma, x) = \sum_{T\in(\gamma\cup\{x\})}[u(\gamma, T) + re(\gamma, T)] \tag{2.13}$$

The local utility from Equation 2.13 for some of the following items i_j - E, A with $\rho = \{D\}$ can be shown as,

$locU(\rho, \{E\}) = [u(\rho, T_2) + re(\rho, T_2)]$ where T_2 is from projected transaction of ρ,

$= (6 + (3 + 4 + 2)) = 15$

$locU(\rho, \{A\}) = (4 + (3 + 10))_{T_1} + (6 + (3 + 4 + 2))_{T_2} = 32$

Definition 2.2.18 *The items are said to be itemsToKeep or follower items of an itemset if the items of 1-HTWUIs or follower items of previous itemset have the local utility value greater than threshold value.*

$$itemsToKeep(\rho) = followerItems(\rho) = \{x \in followerItems(\gamma) \mid locU(\rho, x) \geq \delta \times TU\} \tag{2.14}$$

Items to keep are computed from 1-HTWUIs for the case of root node only, and for remaining sub-trees, items to keep are computed from follower node of its parent node.

Definition 2.2.19 *The items are said to be itemsToExplore or next nodes of an itemset if the items of itemsToKeep or followerNodes have the sub-tree utility value greater than the threshold value.*

$$itemsToExplore(\rho) = nextNodes(\rho) = \{x \in followerItems(\gamma) \mid subU(\rho, x) \geq \delta \times TU\} \quad (2.15)$$

2.2.2 Distributed Systems

With the advent of big data, there is a need for large and parallel computations to find the solution in short time. Therefore, parallel computation is used to take advantage of solving the tasks by computing in parallel using cheap resources. The parallel computation is categorized into different types, but according to the hardware level parallelism, there are generally two types: shared memory and non-shared memory (distributed systems) [DG08]. In shared memory computation, there are multiple processors which concurrently access the shared memory. This model is very efficient and easy to develop. However, this model requires large memory and suffers the problem of need of large memory. In non-shared memory computation, there are different processors which have their own local memories and each processor communicates with other by passing a message through an interconnected network. This model is usually scalable and very efficient than shared memory model.

In the field of big data mining, there is a need to analyze, process and extract the information from the large data. However, there is a restriction on data because of the computation limitation by the single machine. This limitation affects the scalability of the algorithm implemented. Therefore, to process the huge amount of data and extract meaningful information, distributed systems are used. There are different distributed computing frameworks available to take advantage of scalability.

Apache Hadoop

A Java-based framework, Apache Hadoop [Apa], is a popular framework at present. This framework is highly scalable, reliable and fault-tolerant. There are two main components of Apache Hadoop.

One is the Hadoop distributed file system (HDFS), which is designed to store large datasets in a reliable manner. It stores data in different nodes by splitting as a block of the large file and it is distributed among different clusters. It is highly fault-tolerant and reliable as it replicates the file from another node even in the case of failure. Another part of Hadoop system is map-reduce which can process a large amount of data in terms of key-value pairs. There are two stages: map and reduce. The map is used to process block of data to produce the key-value pairs which are then reduced or aggregated by Reducer based on its keys.

However, there is a limitation with Hadoop system as it is based on key-value pair paradigm. Every problem needs to be formulated in terms of key-value pair solution which might be difficult for all the problems. Each map-reduce pairs are read from the disks, processed and write back into the disk. This model restricts the flexibility and performance of the Hadoop system.

Apache Spark

To overcome the limitation of Hadoop system, Apache Spark [ZCF+10] was introduced which does an in-memory computation. Unlike Hadoop system, which depends upon HDFS. Spark introduced the Resilient Distributed Datasets (RDD) abstraction which is a read-only collection of objects. These read-only objects are created by reading the disk or by transformation of other previously created RDDs. Those RDD objects created if lost can be built again, and RDDs are loaded in the memory of multiple nodes so that it can be re-used again and again in Map-reduce operations. In Apache Spark, there are one driver node (Master) and many worker nodes (Slaves) which do map-reduce operations similar to Hadoop system. However, Spark framework can operate any number of the map or reduce operations independently. In Spark framework, it is not necessary that Map operation is followed by Reduce operation unlike in Hadoop framework. These feature of Spark provides much more flexibility.

Chapter 3

Proposed System

3.1 Method I - Pruning Strategies Approach

This section describes our proposed algorithm High Utility Itemset mining using Pruning Strategies Approach (HUI-PR). This section comprises the construction of First level High Transaction Weighted Utility Itemsets (1-HTWUIs) of the given items, node selection rule in the subsequent tree structure, construction of sub-trees of itemsets and pruning strategies to reduce search space by skipping unnecessary visitation of nodes.

3.1.1 Construction of 1-HTWUIs Tree of Items

The 1-HTWUIs is constructed from a tree-structured graph. The items of the transactional database are considered for forming itemsets at level one of the tree, and these itemsets are arranged in increasing order of the Transaction-Weighted Utility (TWU). Based on the transaction weighted downward closure property[LLC05], the transaction weighted utility of a superset itemset is low. Therefore, the itemsets with TWU less than a threshold value are removed, and these removed items are known as unpromising items.

Let us take an example from the Table 2.3. There are 7 items in the transactional database D in which there is one item, $ItemG$, with TWU less than a threshold. $ItemG$ is removed for the construction of 1-HTWUIs. This pruning of items in the initial stage reduces the searching space. The remaining items with TWU, $ItemA = 54, ItemB = 66, ItemC = 42, ItemD = 32, ItemE = 64, ItemF = 33$ are arranged in the ascending order of TWU. Therefore, 1-HTWUIs have the items D, F, C, A, E, B which is shown in the Figure 3.1.

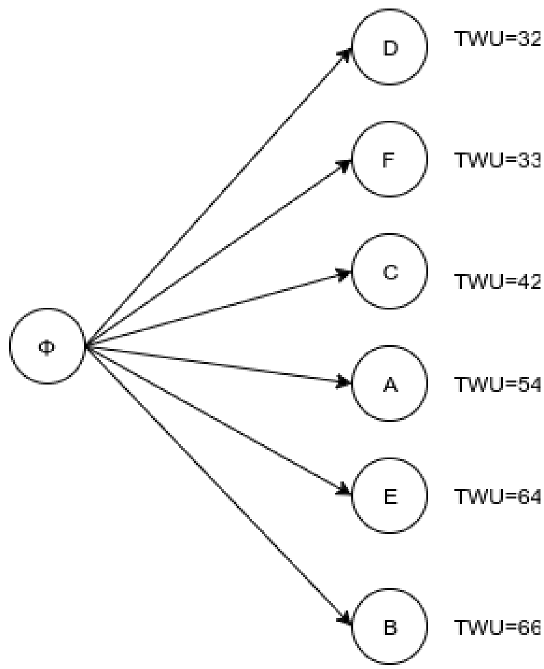


Figure 3.1: Construction of 1-HTWUI Tree of Items

3.1.2 Node Selection Rule

This section describes how our proposed algorithm chooses our nodes. The node with highest TWU is traversed first. From the Figure 2.1, the highest TWU item, $ItemB$, is traversed and then next item $ItemE$ is traversed along with its child nodes. The case is same when traversing inside the child of child nodes. For example, the child nodes of $ItemA$ are $ItemE$ and $ItemB$. The node with $ItemB$ is traversed first and then $ItemE$ is traversed. Therefore, some of the itemsets formed by traversing the tree are as $\{B\}, \{E\}, \{E, B\}, \{A\}, \{A, B\}, \{A, E\}, \{A, E, B\}$

3.1.3 Construction of Sub-tree of Itemsets

For the construction of sub-tree of itemsets, a recursive approach is used in which traversing of node starts from the node with higher TWU itemset and the next subsequent node is taken and traversed. It utilizes depth-first search strategy to traverse every node.

Different computation undergoes in the algorithm as shown in Algorithm 1 which includes the computation of projected database, checking the pruning table to prune the transactions, calculation of utility of an itemset, calculation of sub-tree utilities, local utilities and transaction weighted utilities for all its following items, next child nodes of a current itemset and follower nodes of the child node is computed and the insertion of an itemset to pruning table is also carried out in this process.

Algorithm 1: Build Sub-tree to determine Itemsets

Input: Transactional Database D, ThresholdRatio δ, Total Utility TU

Function *constructSubTree(γ, D, nextNodes(γ), followerItems(γ), δ, TU)*
  for *each item i_j in $nextNodes(\gamma)$* **do**
    $\rho \leftarrow \gamma \cup \{i_j\}$;
    while *scan each T_j in D* **do**
      if *checkPruningTable(T_j)* **then**
        continue from while loop;
      Compute ρD;
      Calculate $u(\rho)$;
    end
    if $u(\rho) \geq \delta \times TU$ **then**
      $HUIs \leftarrow \rho$;
    Calculate $subU(\rho, x), locU(\rho, x)$ and $TWU(\rho, x)$ for all the items i_j in $followerItems(\gamma)$ by scanning ρD;
    $nextNodes(\rho) = \{x \in followerItems(\gamma) | subU(\rho, x) \geq \delta \times TU\}$;
    $followerItems(\rho) = \{x \in followerItems(\gamma) | locU(\rho, x) \geq \delta \times TU\}$;
    while *scan each item i_k in $followerItems(\gamma)$* **do**
      $i_s \leftarrow \rho \cup i_k$;
      if $TWU(i_s) < \delta \times TU$ **then**
        $insertToPruningTable(i_s)$;
    end
    $constructSubTree(\rho, \rho D, nextnodes(\rho), followerItems(\rho), \delta, TU)$;
  end

3.1.4 Pruning Strategies

In our proposed algorithm, the concept of a pruning hash table is implemented. The detail of the proposed pruning hash table is explained in detail in Transaction Pruning Strategy section below. We also use different utility bounds such as sub-tree utility, local utility and transaction weighted utility to prune the branches.

Transaction Pruning Strategy

The proposed algorithm uses a transaction pruning rule to avoid the transactions which contain the itemsets in the pruning hash table to generate the projected transaction ρD.

A hash table is implemented to insert itemsets that are to be pruned. The hash table stores the itemsets with low-utility value. While traversing the different nodes, the itemset is inserted into the pruning hash table if the current itemset has transaction weighted utility lower than the threshold value. For example, if an itemset $\{A, B, C\}$ is to be inserted into the pruning table, our proposed algorithm first checks whether there is already a superset of that itemset in the hash table. If pruning hash table does not contain any superset, it then stores an $ItemA$ in the map with key as A and *null* as value. Then, another map with $ItemB$ will be inserted as the value in A and so on until all the items are stored in the pruning hash table. The algorithm to check whether the superset of an itemset is present or not is shown in Algorithm 2 and to insert an itemset in the pruning hash table is shown in Algorithm 3.

Algorithm 2: Checking in Pruning Hash Table for Transaction Pruning

```
Input: Pruning Hash Table pTable, transaction T_j
Function checkPruningTable(T_j)
    pr ← pTable;
    if pTable.size() > 0 then
        for each item i_k ∈ T_j do
            //check item in pruning table
             if i_k not in pr then
                return false;
            pr ← pr(i_k);
            if pr is null then
                return true;
        end
    return false;
```

Algorithm 3: Insertion in Pruning Hash Table

```
Input: Itemset itemset, Maximum limit of pruning table φ
Function insertIntoPruningTable(itemset)
    if checkPruningTable(itemset) is null then
        if pTable.size() < φ then
            Insert into pruning table recursively;
            pTablesize++;
            return true;
    return false;
```

Utility Based Pruning

Our proposed algorithm uses utility based pruning to prune the branches with itemsets that are not feasible. Utility based pruning includes sub-tree utility, local utility and transaction weighted utility. The transaction weighted utility of an itemset prunes the itemset that is less than the minimum threshold by inserting into the pruning hash table which is used for reducing the search space. During the generation of projected transaction, while traversing on each node, calculation of sub-tree utility and local utility are done for each of the possible follower items of that node. If any of the possible follower items have sub-tree utility less than the minimum threshold, then that item cannot be the next possible node but still has a chance to be the follower item for the next nodes of the current itemset. If those possible follower items of a node have local utility less than the minimum threshold, it cannot be the next node as well as a follower item for that node. For example, let us consider a node as $ItemA$ and suppose, there are 3 possible follower items $ItemB, ItemC$ and $ItemD$. The sub-tree utility is calculated for all its follower items B, C and D and if B has sub-tree utility greater than threshold value and C and D have sub-tree utility value less than threshold then, $ItemB$ is the next node as well as the follower item for $ItemA$ but the local utility is calculated for $ItemC$ and $ItemD$ and suppose if $ItemC$ has local utility greater than threshold value and $ItemD$ has local utility less than threshold, then $ItemC$ can be the follower item of node $ItemA$. Therefore, next node of $ItemA$ is $\{B\}$ and follower items is $\{B, C\}$.

3.1.5 Detailed Algorithm (HUI-PR)

Our detailed algorithm is shown in Algorithm 4. Our algorithm starts with reading the transactional database (TD) and threshold ratio (δ). The total utility (TU), transaction weighted utility (TWU) of items and local utility ($locU$) of all the items of a database is computed by scanning the whole transactional database. Total utility of an database, transaction weighted utility of items and local utility of items are calculated as defined in Equation 2.5, 2.4 and 2.13. 1-HTWUIs are calculated based on the transaction weighted utility obtained as described in Section 3.1.1. The follower items are 1-HTWUIs for the initial node and these items are sorted in increasing order in total ordering ($\rightarrow$) as described in Definition 2.2.9. From the list of 1-HTWUIs, those itemsets with transaction weighted utility values less than the threshold are known as unpromising items. Moreover, those unpromising items are removed from the transactions of the whole transactional database. After removing the unpromising items from the database, if there are empty transactions created, then those transactions are removed. The items of each transaction in the transactional database are sorted based on the total ordering. Calculation of sub-tree utility for each followerItems is done by scanning the whole database and based on the sub-tree utility values, next nodes of the initial root node are defined where the items must have sub-tree utility greater than the threshold. Sub-tree is constructed recursively with taking the parameters as the transactional database, nextNodes, followerItems, thresholdRatio and total utility. The algorithm to find the sub-tree of itemsets is defined in detail in Section 3.1.3.

Algorithm 4: Algorithm to find HUIs

Input : Transactional Database TD, ThresholdRatio δ
Output: High Utility Itemsets $HUIs$

Initial itemset, $\gamma = \phi$;
Calculate Total Utility TU, $TWU(\gamma, i_j)$ and local utility $locU(\gamma, i_j)$ for all $i_j \in I$ by scanning whole database TD;
Compute 1-HTWUIs itemsets,
  $followerItems(\gamma) = \{i_j | i_j \in I \cap TWU(\gamma, i_j) \geq \delta \times TU\}$
  Sort the items in $followerItems(\gamma)$ in total ordering ($\rightarrow$);
Remove the unpromising items j from the transactions T_j;
Remove the empty transaction after removing unpromising items;
Sort the items in each transaction in total ordering ($\rightarrow$);
Calculate sub-tree utility $subU(\gamma, i_j)$ for all $i_j \in followerItems(\gamma)$ by scanning database TD;
Compute next nodes to visit in reverse order,
  $nextNodes(\gamma) = \{i_j | i_j \in reverse(followerItems(\gamma)) \cap subU(\gamma, i_j) \geq \delta \times TU\}$;
$constructSubTree(\gamma, TD, nextNodes(\gamma), followerItems(\gamma), \delta, TU)$;

3.2 Method II - Distributed EFIM

This section describes our proposed algorithm EFIM Parallel (EFIM-Par) to find high utility itemsets with parallel computing using Apache Spark. This algorithm is the parallel (distributed) implementation of the algorithm EFIM [ZFVL+15]. This section comprises of generating 1-HTWUIs, generating revised transactions, finding the sub-tree utility and the local utility, assigning the sub-tree to worker nodes, node data generation, mining high utility itemsets by individual worker nodes and explanation of the overall flow of EFIM Parallel algorithm.

3.2.1 Generating 1-HTWUIs with their corresponding TWU

The Transactional Database (TD) is scanned to find out the 1-HTWUIs of items along with their transaction weighted utility. First of all, the transactional database is divided into different blocks which are computed by different worker nodes using $flatMap$ operation. The result obtained from worker nodes are reduced using $reduceByKey$ operation to get the itemTWU of items which contain items with their corresponding TWU.

3.2.2 Generating Revised Transactional Database

In this process, the Transactional Database (TD) is mapped to generate the revised transactional database using map operation. Firstly, TD is split into different blocks to distribute among worker nodes in which pruning of unpromising items are done, and then the transaction is sorted in the ascending order of the transaction weighted values of items. Besides pruning of unpromising items and sorting, there is a removal of empty transactions by using filter operation. The generated revised transactions use the functionality provided by Spark to persist the RDD so that it can be used again later. It is used later to find out the sub-tree utility of items and assignment of items to worker nodes which will be described in the later sections.

Algorithm 5: Revised Transactional Database Generation

Input : Transactional Database TD, ThresholdRatio δ, Total Utility TU
Output: Revised Transactional Database

Function *map()*
    for $k = 0$ *to* $len(TD)-1$ **do**
        // Removing unpromising items from the transaction
        $TD_k = \sum_{j=0}^{len(TD_k)-1} \{i_j | i_j \in TD_k \cap TWU(i_j) \geq \delta \times TU\}$;
        // Sort items in transaction in total ordering
        $SortItems(TD_k)$;
        if $len(TD_k)$ *== 0* **then**
            Remove TD_k;
        end
    end

3.2.3 Finding Local Utility and Sub-tree Utility of 1-HTWUIs

There are two utilities in this proposed algorithm which prunes the unnecessary visitation of the nodes. It reduces the search space significantly. It needs to be calculated in the first level of the tree in order to compute the next nodes of each item and the follower nodes of those items. The local utility is calculated as shown in Equation 2.13. For the initial case, it is same as transaction weighted utility, therefore, it uses TWU of 1-HTWUIs as described in Section 3.2.1. Another utility is the sub-tree utility which is calculated as shown in the Equation 2.12. It scans the revised transaction to generate the sub-tree utility for each item of 1-HTWUIs. It uses flatMap and Reduce operations to get the sub-tree utility values for each item.

3.2.4 Sub-tree Assignment to Worker Nodes

The proposed algorithm uses grouping strategy to assign the *itemsToExplore* and their respective sub-trees to the worker nodes. The *itemsToExplore* is computed by using the Equation 2.15. Grouping of 1-HTWUIs is as shown in the Algorithm 6. This grouping approach helps to divide our tasks among the worker nodes to be executed in distributed environment properly. The grouping is done based on the number of items to explore. Items to explore is defined in Equation 2.14.

Referring to the example in Table 2.1 and 2.2, we have 7 items in which there are 6 1-HTWUI items. From the definition of 2.2.19, we have D, F, C, A, E, B as items to explore. Let us suppose we have 3 worker nodes as $Node$ 1, $Node$ 2 and $Node$ 3. According to the Algorithm 6, the worker nodes are assigned as $D \rightarrow Node$ 1, $F \rightarrow Node$ 2, $C \rightarrow Node$ 3, $A \rightarrow Node$ 3, $E \rightarrow Node$ 2 and $B \rightarrow Node$ 1. The worker nodes are assigned to the sub-tree nodes along with their respective node data which is described in Section 3.2.5.

Algorithm 6: Assignment of Sub-tree to Worker Nodes

Input : Number of worker nodes N, Follower nodes *itemsToExplore*
Output: Hashmap(*nodeId*, *itemsToExplore*) $workerNodeMap$

Function *grouping()*
    $workerNodeMap \leftarrow map()$;
    $nodeId \leftarrow 1$;
    $incr \leftarrow 1$;
    $flag \leftarrow false$;
    for *i in itemsToExplore* **do**
        $workerNodeMap[i] \leftarrow nodeId$;
        $nodeId \leftarrow nodeId + 1$;
        if $(nodeId == 0 \mid\mid nodeId == N - 1)$ **then**
            if $(flag == false)$ **then**
                $incr \leftarrow 0$;
                $flag \leftarrow true$;
            else
                if $(nodeId == 0)$ **then**
                    $incr \leftarrow 1$;
                else
                    $incr \leftarrow -1$;
                end
                $flag \leftarrow false$;
            end
        end
    end
    return $workerNodeMap$;

3.2.5 Node Data Generation

Each worker node is assigned with a sub-tree which needs to be traversed. Each node traversal indicates the candidate generation which is needed to generate projected transaction at each node. Therefore, each worker node gets the refined transactions including the items it needs to visit which is computed using flatMap operation. Algorithm 7 shows the steps to generate the node data for the specific items assigned to the worker nodes. Each transaction is checked by the binary search to find the item assigned to worker node is present or not. If items assigned are not in the transaction, then that transaction is not added to the nodeMap. After scanning all the nodes, nodeMap is grouped by a key which is then used to mine high utility itemsets which are explained in Section 3.2.6 later.

Algorithm 7: Node Data Generation

```
Input  : Revised Transactions T, workerNodeMap
Output: Hashmap(nodeId, T_r) nodeMap
Function flatMap
    nodeMap = map();
    for i ← 0 to len(T) − 1 do
        for (nodeId, item) ← workerNodeMap do
            check = binarySearchIterative(T.itemset, item);
            if check == true then
                nodeMap ← (nodeId, T_i);
            end
        end
    end
    return nodeMap;
Function binarySearchIterative(list, target)
    left ← 0; right ← len(list) − 1;
    while (left ≤ right) do
        mid = left + (right − left) / 2;
        if (list(mid) == target) then
            return true;
        else if list(mid) ≥ target then
            right = mid − 1;
        else
            left = mid + 1;
        end
    end
    return false;
```

3.2.6 Mining High Utility Itemsets

This section describes the mining of high utility itemsets that are computed by worker nodes using generated Node data. The detailed algorithm is shown in Algorithm 8. Each worker processes to compute the high utility itemsets (HUIs) for the assigned sub-tree of items. It uses a recursive algorithm to find HUIs which generates the possible candidate sets assigned to them. Let us consider the example from the Figure 2.1 in which each item of 1st level is assigned to one worker node. Let us assume that there are 3 worker nodes, then we know from the previous Section 3.2.4, $ItemD$ is assigned to $Node1$. All the possible candidate sets for $ItemD$ are processed by $Node1$. Similarly, for the $ItemF$, possible candidate sets are processed by $Node2$ and so on.

Algorithm 8: Mining HUIs in Parallel

Input : Database d, ThresholdRatio δ, Total Utility TU, $nodeMap$, $nodeId$
Output: High Utility Itemsets $HUIs$

Function *mineHUIs*(γ, d, $itemsToExplore(\gamma)$, $itemsToKeep(\gamma)$, δ, TU, $flagFirst = true$)
    for *each item* i_j *in* $itemsToExplore(\gamma)$ **do**
        if $(flag \neq true \mid\mid nodeId == nodeMap(i))$ **then**
            $\rho \leftarrow \gamma \cup \{i_j\}$;
            while *scan each* T_j *in* γD **do**
                Compute ρD;
                Calculate $u(\rho)$;
            end
            if $u(\rho) \geq \delta \times TU$ **then**
                $HUIs \leftarrow \rho$;
            Calculate $subU(\rho, x)$ and $locU(\rho, x)$ for all the items i_j in $itemsToKeep(\gamma)$ by scanning γD;
            $itemsToExplore(\rho) = \{x \in itemsToKeep(\gamma) | subU(\rho, x) \geq \delta \times TU\}$;
            $itemsToKeep(\rho) = \{x \in itemsToKeep(\gamma) | locU(\rho, x) \geq \delta \times TU\}$;
            $mineHUIs(\rho, d, itemsToExplore(\rho), itemsToKeep(\rho), \delta, TU, false)$;
        end
    end

3.2.7 Overall Flow of EFIM Parallel Algorithm

The overall flow diagram of EFIM Parallel Algorithm is shown in Figure 3.2. It starts with reading of dataset from the file which is split into different blocks to be distributed among the worker nodes. The worker nodes work on the block of the file using $flatMap$ operation to generate the key-value pairs of items and its corresponding TWU which is then combined using $ReduceByKey$ operation to get the final $itemTWU$. The generation of 1-HTWUIs was explained in detail in the previous Section 3.2.1. The split dataset is also used to find the total utility of the transactional database to find the threshold value. This threshold value is used to find the $itemsToKeep$ by filtering out the items in 1-HTWUIs having TWU values less than the threshold value. Only those items remaining in the $itemsToKeep$ are kept in the transactions of the database. Therefore, other items not in $itemsToKeep$ known as unpromising items, are removed from the transactions, sort the items in a transaction in the total ordering and removal of empty transactions are done to get the sorted revised transactions which were described in detail in Section 3.2.2. In the next step, the sorted revised transactions are used to find the $utilityBinSU$ for each items by using $flatMap$ and $ReduceByKey$ operations. The $utilityBinSU$ contains sub-tree utility for all the items of $itemsToKeep$. The $utilityBinLU$ contains local utility for all the items which is same as the $itemTWU$. Using the $utilityBinSU$, the list containing all the items for $itemsToExplore$ is created. A sub-tree is created from the items in $itemsToExplore$.

Assignment of items of $itemsToExplore$ is done using the grouping mechanism as described in detail in Section 3.2.4. In this process, the worker node identifies the sub-tree it needs to generate. Each worker node processes to filter the transactions to produce the Node Data. Using these node data, each worker node mines the high utility itemsets forming sub-trees to generate the candidate sets. In the mining process, the nodes are pruned based on the sub-tree utility and the local utility as given in Equation 2.12 and Equation 2.13 respectively. Finally, the results obtained from the worker nodes are combined to give the aggregated high utility itemsets.

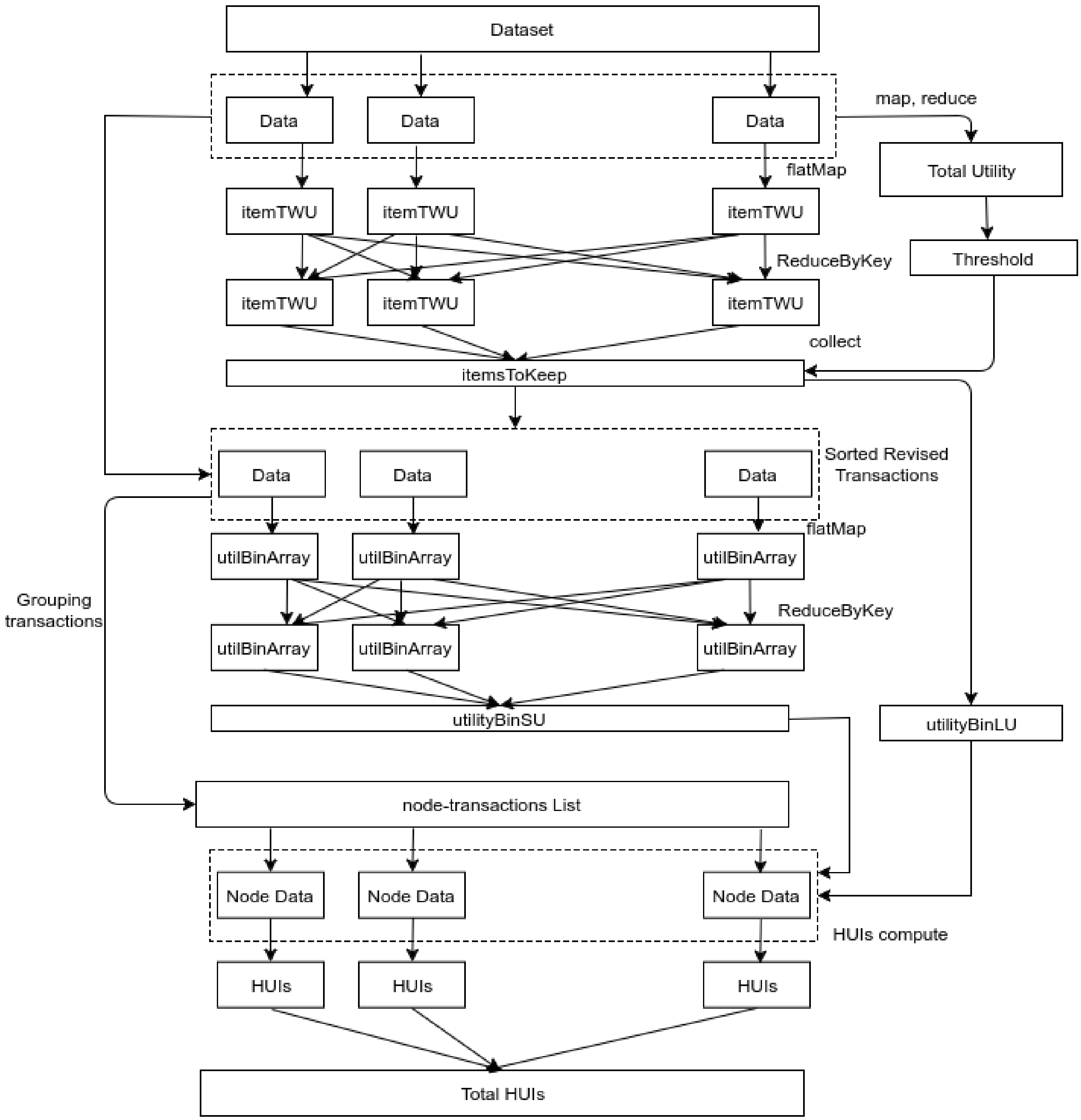


Figure 3.2: Overall Flow Diagram of EFIM Parallel Algorithm

Chapter 4

Experimental Results

The experiments were performed for our Method I with our proposed algorithm (HUI-PR) and EFIM algorithm [ZFVL+15] to find high utility itemsets on 16GB main memory in Intel Xeon(R) CPU E5-1607 0 @ 3.00 GHz x 4 on an Ubuntu 16.04 Linux Operating system. The language used to write these algorithms was Oracle Java 1.8.

For the Method II, the experiments were performed on Spark clusters with Master and all Slave nodes with 16GB main memory and Intel Xeon(R) CPU E5-2695 v4 @ 2.10 GHz x 4 with an Ubuntu 16.04 Linux Operating system. The language used to write the spark application was Scala version 2.12.1 with Spark framework version 2.0.2 to run an experiment for our proposed algorithm EFIM-Par and PHUI-Miner [CA16].

4.1 Datasets

The experiments were performed on multiple real-world datasets [Fre12, SPM]. For the method I, our experiments were conducted on smaller datasets such as Chess, Connect and Retail. For the method II, our experiments were conducted on relatively large datasets such as Connect20x, Chess30x, BMS4x, Mushroom20x. For large datasets, the small datasets such as Connect, Chess, BMS and Mushroom were multiplied to get the larger dataset. The characteristics of the datasets are shown in the Table 4.1 where $\#|D|$, $\#|I|$, $AvgLen$, $MaxLen$, $Type$ and $Scale$ represent the total number of transactions, the number of distinct items, the average size of a transaction, maximum size of a transaction, type of dataset and size of dataset respectively. For each threshold ratio of a dataset, the experimental results were executed 10 times and the average was taken.

Table 4.1: Datasets Characteristics

Dataset	$\#\|D\|$	$\#\|I\|$	AvgLen	MaxLen	Type	Scale
chess	3196	76	37	37	dense	Small
connect	67557	129	43	43	dense	Small
retail	88162	16470	10	76	sparse	Medium
$connect2x$	135114	129	43	43	dense	Large
$chess30x$	95880	76	37	37	dense	Large
$BMS4x$	238408	497	3	267	sparse	Large
$Mushroom20x$	162400	119	23	23	dense	Large

4.2 HUI-PR vs EFIM

Our proposed HUI-PR was compared with EFIM [ZFVL+15] with comparisons on the computational time, the number of high utility itemsets (HUIs) found and the number of Candidate Sets generated. These algorithms were performed on the smaller datasets.

4.2.1 Comparison of Computational Time

In this section, we compared our algorithm (HUI-PR) with the EFIM algorithm [ZFVL+15] with the real datasets (Connect, Chess, Retail). Experiments were conducted to show the effectiveness of our algorithm with the real datasets and the approach that was taken to improve the performance of an experiment. The pruning rule proposed in our algorithm HUI-PR helps to improve computational time significantly for the datasets with a large number of transactions. The proposed algorithm generates the projected transaction which reduces the number of transactions in each level. It not only reduces the number of transactions based on utility calculations but it also uses the pruning hash table to eliminate the transactions in which the itemsets in the pruning table might be a subset of items in a transaction. Therefore, it helps to check whether the items in a transaction are a superset or not in very quick time.

From the Figure 4.1, we see that HUI-PR can perform better than the EFIM algorithm. From the Figure 4.1a, for the "Connect" dataset, the threshold ratio was set from 28.90% to 29.70% as shown. When the threshold ratio was 28.90%, our algorithm HUI-PR took 1830.87 seconds while the EFIM algorithm took 1927.95 seconds. The proposed algorithms showed significant improvement in Figure 4.1c on threshold ratio 0.03%, the running time for HUI-PR was 5718.36 seconds while for EFIM algorithm, the running time was 7370.33 seconds.

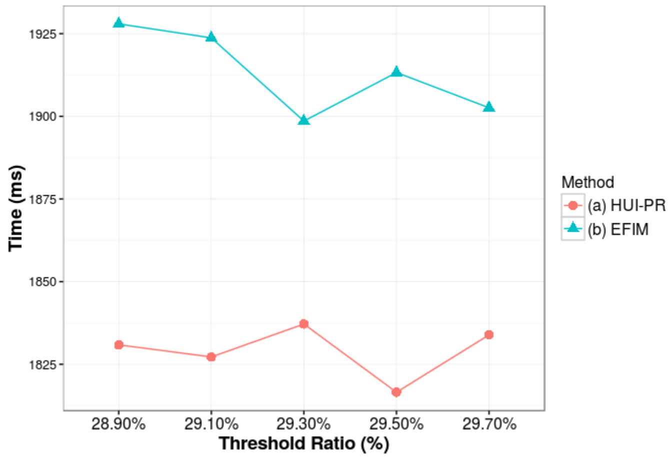
We also conducted our experiment against the other state-of-the-art algorithms: HUI-Miner [LQ12], HUP-Miner [Kri15], FHM [FVWZT14], FHM+ [FVLDD16], d^{2}HUP [Kri15, LQ12]. Our algorithm HUI-PR performs better than these state-of-the-art algorithms as shown in Figure 4.2. For the "Connect" dataset, our algorithm performed better by more than 100 times than HUI-Miner, HUP-Miner and FHM algorithms while it performed better by almost 50 times than d^{2}HUP. Similarly, for the "Chess" dataset, HUI-PR performed better by 20 times than HUI-Miner, HUP-Miner and FHM algorithms while it performed better than 7 times than d^{2}HUP. Since the time performed by FHM+ was significantly higher when it was executed with parameter $MaxLength =$ 15 for the "Chess" dataset and $MaxLength = 21$ for the "Connect" dataset. Therefore, it is not shown in the graph.

4.2.2 Comparison of HUIs

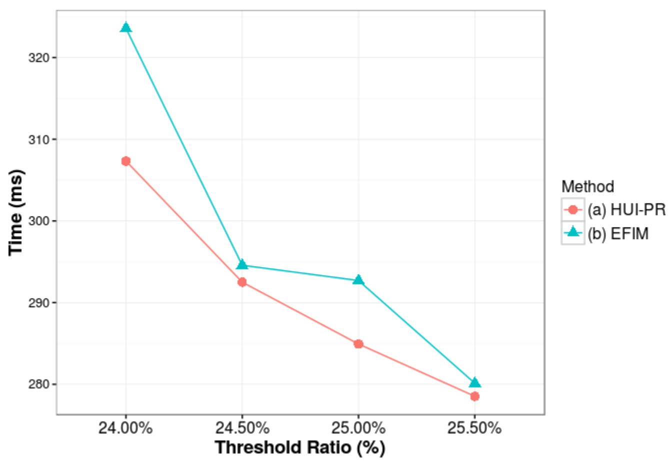
From the experiments conducted on the real-world datasets (Connect, Chess and Retail), the number of HUIs found by both the experiments are same. We recorded the number of HUIs found for the range of threshold ratio for different datasets which are shown in Table 4.2. For the "Connect" database, we got 81 HUIs for 28.90% and 4 HUIs for 29.70%. Similarly, for the "Chess" dataset, we got 342 HUIs for 24.00% and 16 HUIs for 26.00% threshold ratio. From the results obtained, we can verify that all the high utility itemsets have been found from our proposed algorithm HUI-PR.

4.2.3 Comparison of Candidate Sets

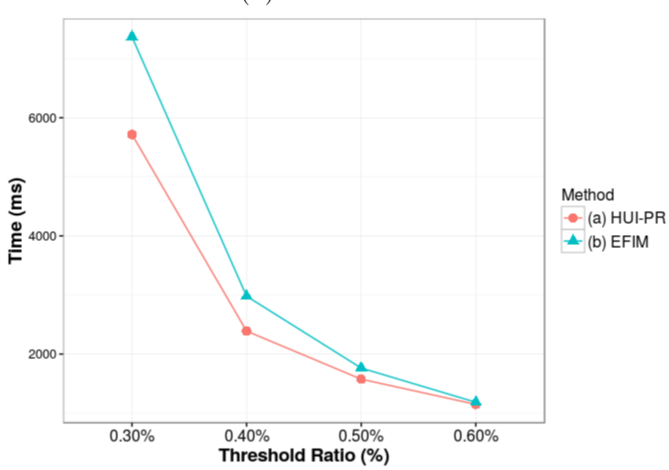
From the Figure 4.3, we compared the candidate sets obtained from HUI-PR and EFIM algorithms. The candidate sets generated in HUI-PR are lower in number than that in EFIM algorithm. The candidate sets are minimized in the HUI-PR using transaction pruning strategies with pruning hash table and utility based pruning. For the "Connect" dataset for threshold ratio 28.90%, HUI-PR generated 3007 candidate sets while the EFIM algorithm generates 3132 candidate sets. HUI-PR could generate fewer candidate sets in the "Chess" dataset. For 24.00% threshold ratio, HUI-PR generated 2933 candidate itemsets while EFIM generated 2965 number of candidate itemsets. We also compared the candidate sets obtained from state-of-the-art algorithms: HUIMiner, FHM and FHM+ as shown in Figure 4.4. The number of candidate sets generated by our algorithm HUI-PR is 8 times less than HUIMiner and FHM for the "Chess" dataset while HUI-PR generates 10 times less than HUIMiner and FHM for the "Connect" dataset.



(a) Connect Dataset

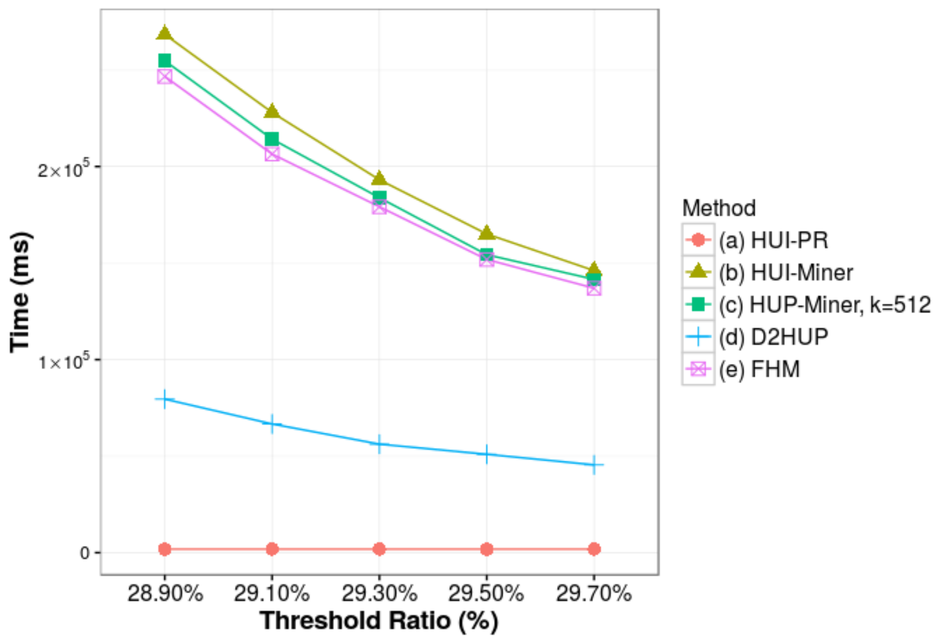


(b) Chess Dataset

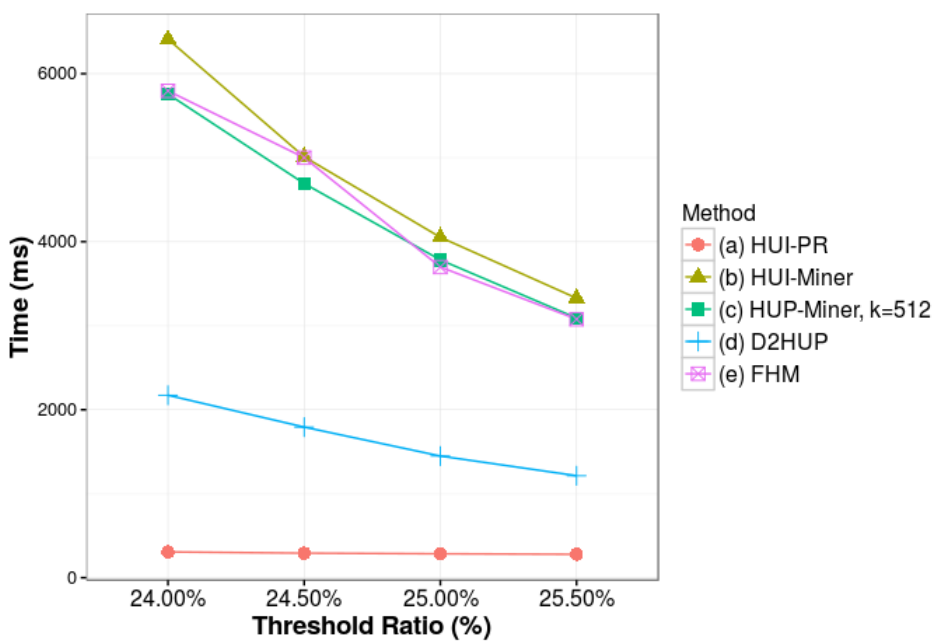


(c) Retail Dataset

Figure 4.1: Comparison of computational time between HUI-PR and EFIM w.r.t. variants of minimum threshold for different datasets



(a) Connect Dataset



(b) Chess Dataset

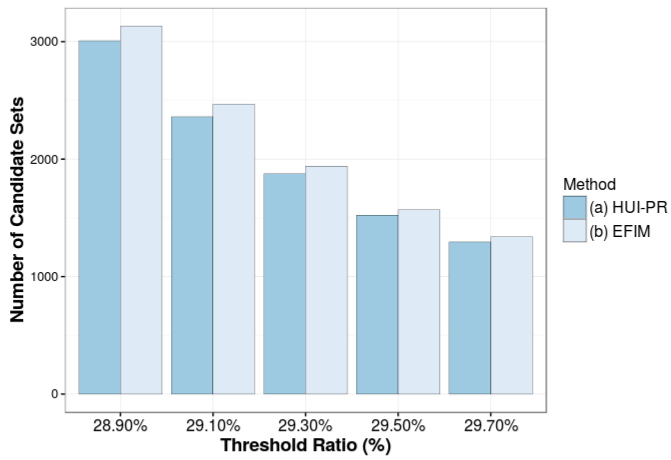
Figure 4.2: Comparison of computational time with state-of-the-art algorithms w.r.t. variants of minimum threshold

Table 4.2: Total Number of HUIs found in HUI-PR and EFIM

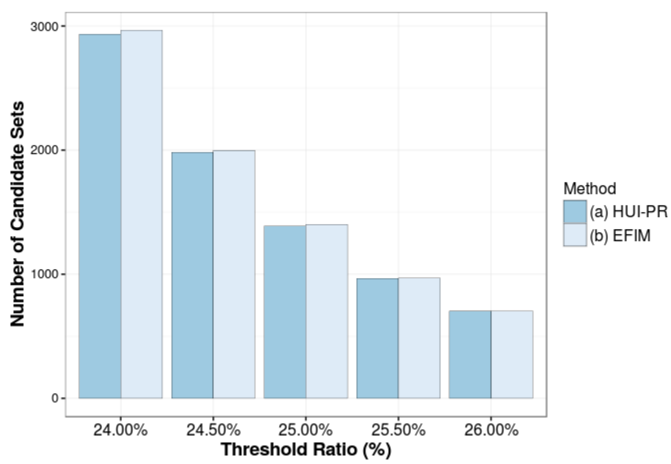
Dataset	**ThresholdRatio** δ	**# of HUIs**
Connect	28.90%	81
Connect	29.10%	40
Connect	29.30%	20
Connect	29.50%	8
Connect	29.70%	4
Chess	24.00%	342
Chess	24.50%	177
Chess	25.00%	98
Chess	25.50%	41
Chess	26.00%	16
Retail	0.30%	92
Retail	0.40%	58
Retail	0.50%	41
Retail	0.60%	30

Table 4.3: Total Number of Transactions Pruned in HUI-PR

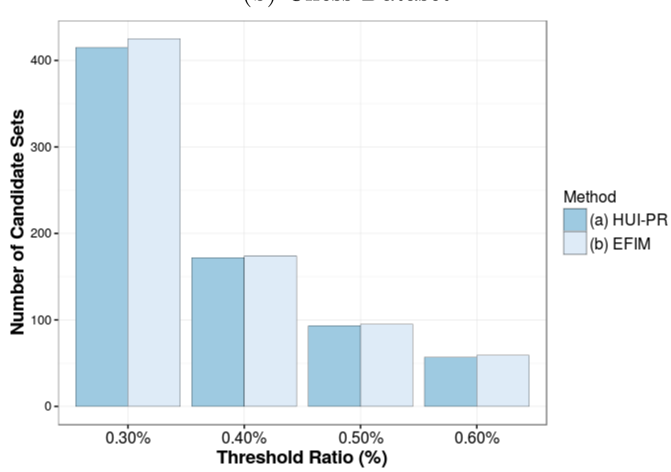
Dataset	**ThresholdRatio** δ	**# Transactions Pruned**
Connect	28.90%	556831
Connect	29.10%	550630
Connect	29.30%	550630
Connect	29.50%	550630
Connect	29.70%	550630
Chess	24.00%	24878
Chess	24.50%	30779
Chess	25.00%	27829
Chess	25.50%	26265
Chess	26.00%	26304
Retail	0.30%	670018
Retail	0.40%	245342
Retail	0.50%	117453
Retail	0.60%	61939



(a) Connect Dataset

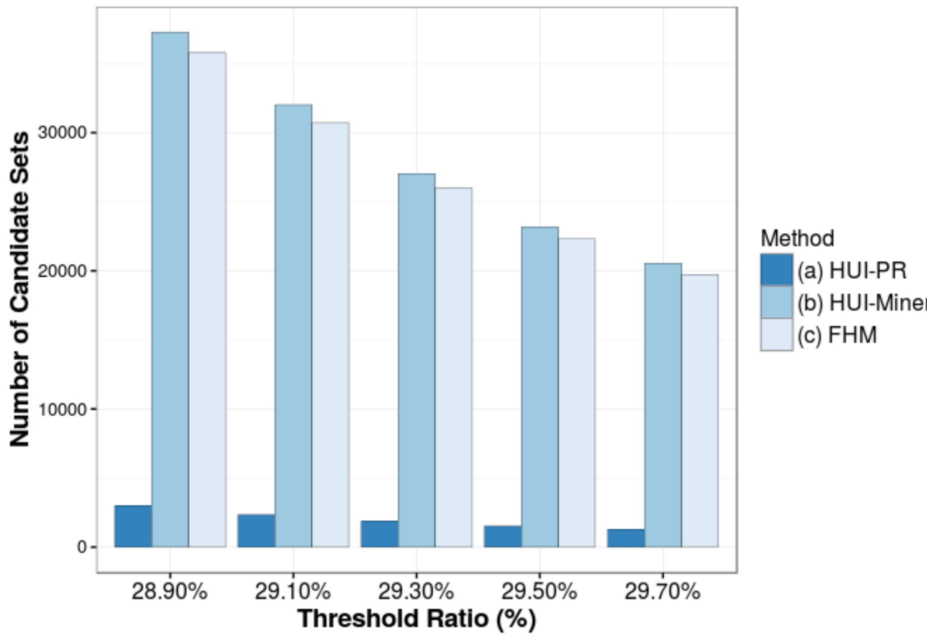


(b) Chess Dataset

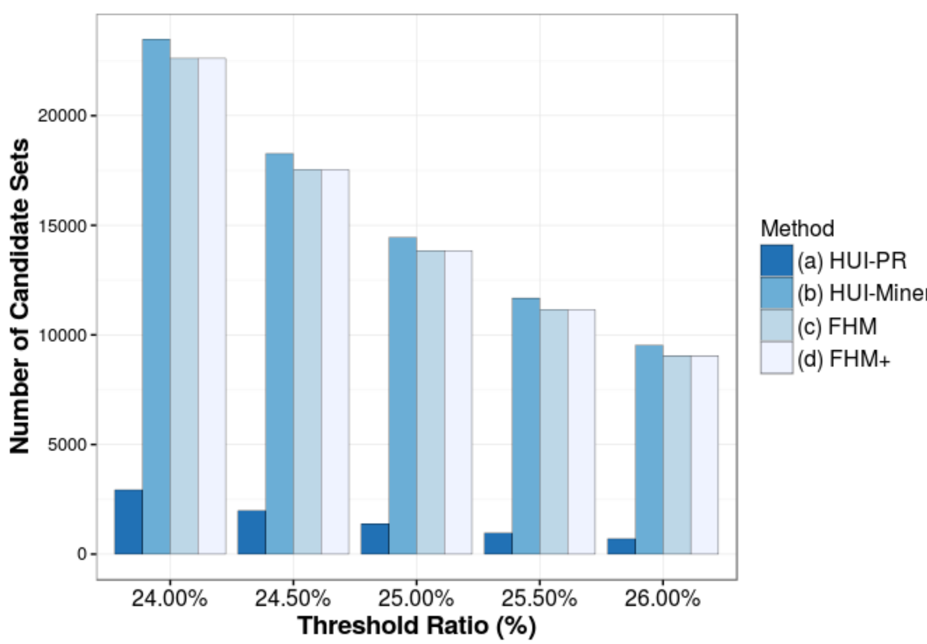


(c) Retail Dataset

Figure 4.3: Comparison of candidate sets between HUI-PR and EFIM w.r.t. variants of minimum threshold



(a) Connect Dataset



(b) Chess Dataset

Figure 4.4: Comparison of candidate sets with state-of-the-art algorithms w.r.t. variants of minimum threshold

4.3 EFIM-Par vs EFIM

We compared our proposed distributed algorithm Parallel EFIM (EFIM-Par) with Approximate parallel high utility itemset mining (PHUI-Miner) [CA16]. The computational time were recorded for the different datasets as shown in the Figure 4.5. These algorithms were performed on the larger datasets. Our algorithm were conducted on one master node and ten slave nodes in the Spark Framework.

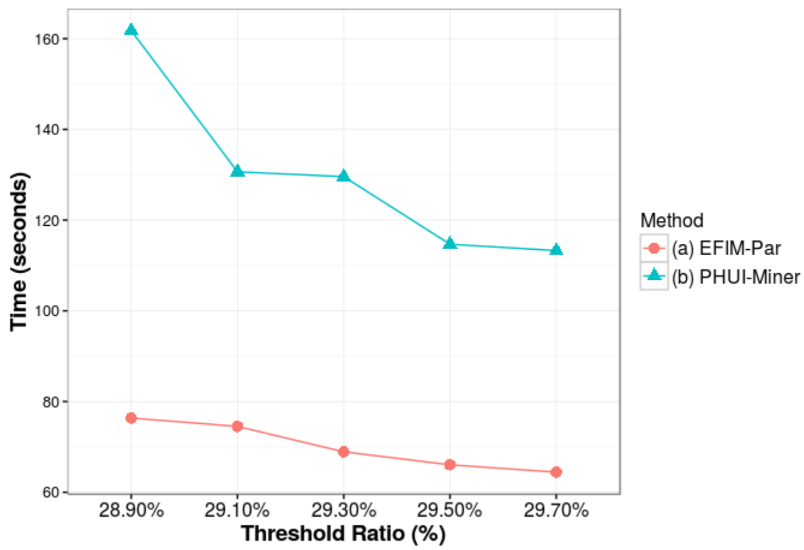
4.3.1 Comparison of Computational Time

In this section, we ran our experiments on the real-world datasets (Connect, Chess, BMS and Mushroom). However, in order to make the datasets sufficiently large, we multiplied the “Connect” dataset by a factor of 2, “Chess” dataset by a factor of 30, “BMS” dataset by a factor of 4 and “Mushroom” dataset by a factor of 20. The experiments were conducted on our proposed algorithm EFIM-Par and PHUI-Miner.

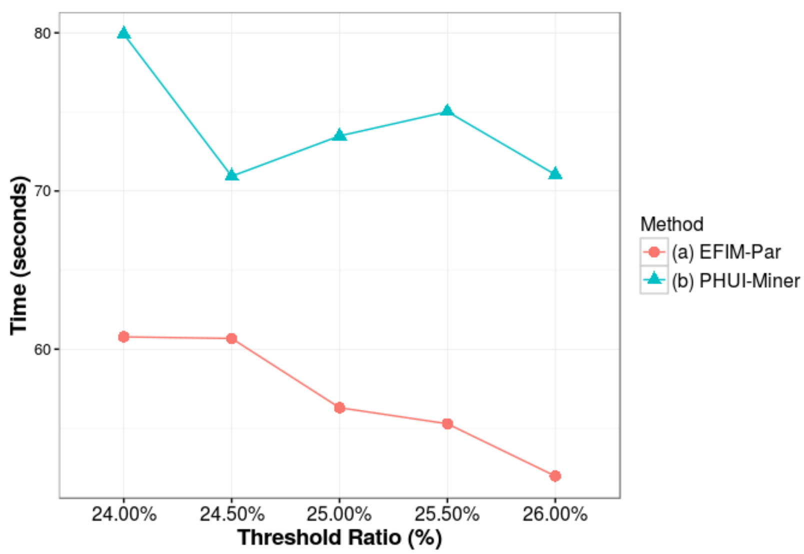
From the Figure 4.5a, our algorithm EFIM-Par took 76.36 seconds while PHUI-Miner took 161.76 seconds for the threshold ratio 28.90% for the “Connect” dataset. Similarly, for the threshold ratio 29.70%, our EFIM-Par took 64.42 seconds while PHUI-Miner took 113.27 seconds. Our algorithm EFIM-Par was able to perform around 2 times better than PHUI-Miner for the “Connect” dataset for different threshold ratio taken. From the Figure 4.5b, for the “Chess30x” dataset, our algorithm took 60.77 seconds for the threshold ratio 24.00% while PHUI-Miner took 79.19 seconds. Similarly, for the threshold ratio 26.00%, EFIM-Par took 51.97 seconds while PHUI-Miner took 71.03 seconds. Our algorithm performed almost 1.5 times better than PHUI-Miner for the “Chess30x” dataset. Similarly for the “BMS4x” dataset, our algorithm performed better for the lower threshold and almost similar for the higher threshold values. Our algorithm performed better than 1.2 times the PHUI-Miner algorithm for “Mushroom20x” dataset.

4.3.2 Comparison of HUIs

From the Table 4.4, we found that our algorithm EFIM-Par found the same number of HUIs as found by PHUI-Miner. Therefore, we can conclude our algorithm is as accurate as PHUI-Miner.

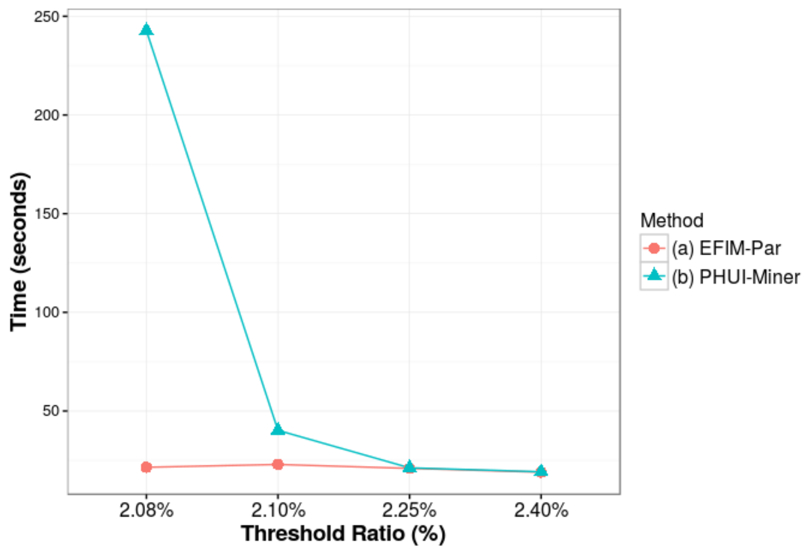


(a) Connect2x Dataset

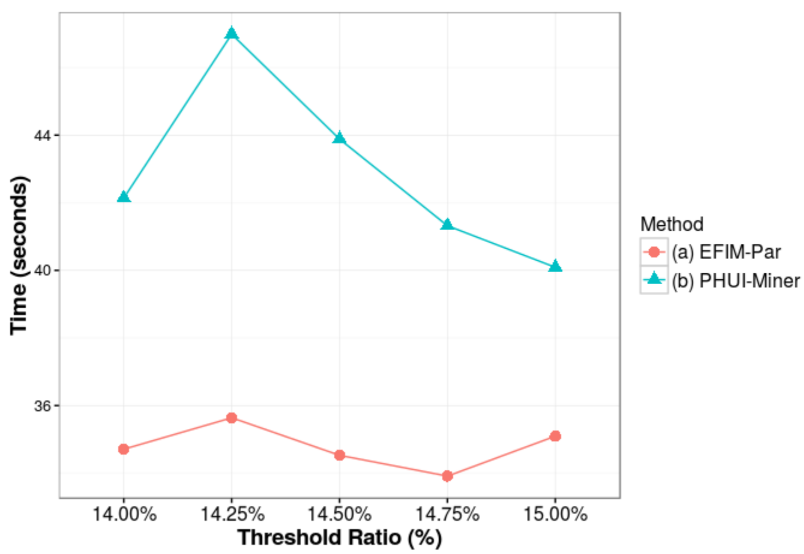


(b) Chess30x Dataset

Figure 4.5: Comparison of computational time between EFIM-Par and PHUI-Miner w.r.t. variants of minimum threshold for different datasets



(c) BMS4x Dataset



(d) Mushroom20x Dataset

Figure 4.5: Comparison of computational time between EFIM-Par and PHUI-Miner w.r.t. variants of minimum threshold for different datasets

Table 4.4: Total Number of HUIs found in EFIM-Par and PHUI-Miner

Dataset	**ThresholdRatio** δ	**# of HUIs**
$Connect2x$	28.90%	81
$Connect2x$	29.10%	40
$Connect2x$	29.30%	20
$Connect2x$	29.50%	8
$Connect2x$	29.70%	4
$Chess30x$	24.00%	342
$Chess30x$	24.50%	177
$Chess30x$	25.00%	98
$Chess30x$	25.50%	41
$Chess30x$	26.00%	16
BMS	2.08%	7
BMS	2.10%	7
BMS	2.40%	5
BMS	2.80%	3
BMS	3.00%	2
$Mushroom20x$	14.00%	67
$Mushroom20x$	14.25%	38
$Mushroom20x$	14.50%	19
$Mushroom20x$	14.75%	10
$Mushroom20x$	15.00%	2

Chapter 5

Conclusion and Future Work

In this thesis, two methods HUI-PR and EFIM-Par have been proposed. The proposed algorithm (HUI-PR) is a novel approach of pruning transactions to reduce the search space while finding high utility itemsets. HUI-PR can reduce the search space by eliminating the number of candidate sets which avoids the computation of unnecessary itemsets. HUI-PR uses a pruning hash table, which stores low-utility itemsets that are checked while generating the projected transaction in each node. This elimination helps reduce the candidate sets in HUI-PR besides different utilities such as sub-tree utility, local utility and transaction weighted utility for pruning. This approach is highly suited for comparatively smaller datasets. Another proposed algorithm (EFIM-Par) is a novel approach to mine high utility itemsets using distributed approach. Spark framework is used for the distributed computing because of its advantage over the Hadoop framework. Spark framework uses in-memory computation which is much faster than disk dependent Hadoop framework. Our algorithm divides the computation into multiple stages such that each task is divided into multiple worker nodes. In the mining stage, each work gets the task using the grouping mechanism which finds the high utility itemsets that are aggregated to find the overall high utility itemsets. An extensive experiment in various datasets with the state-of-the-art algorithm is conducted for both methods. Our experiments show that HUI-PR can perform more efficiently than other existing algorithms. HUI-PR improves the computational time for finding the high utility itemsets as it reduces the number of candidate sets. HUI-PR gains significant performance improvement in terms of computational time and a number of candidates sets generated. Our experiments for EFIM-Par shows that it performs better than PHUI-Miner. Our algorithm performs much better in terms of computation than PHUI-Miner. EFIM-Par divides the search space in an efficient way so that each worker nodes compute in faster time.

Although our proposed methods perform much better than other algorithms, these methods could be enhanced to perform at optimum level. Our algorithm (HUI-PR) finds the high utility itemsets efficiently for small datasets. However, it has lessened improvement when the datasets are very small. Therefore, an improvement could be done for very small datasets. Also, different tree construction mechanisms could be studied so that the proposed pruning strategies can work best. Our other algorithm (EFIM-Par) could be enhanced with much better grouping mechanism to divide the tasks to each worker node in an optimum manner.

Bibliography

[Apa] Apache hadoop - http://hadoop.apache.org.

[AS94] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules in large databases. In *Proceedings of the 20th International Conference on Very Large Data Bases*, VLDB '94, pages 487–499, San Francisco, CA, USA, 1994. Morgan Kaufmann Publishers Inc.

[ATJL09] C. F. Ahmed, S. K. Tanbeer, B. S. Jeong, and Y. K. Lee. Efficient tree structures for high utility pattern mining in incremental databases. *IEEE Transactions on Knowledge and Data Engineering*, 21(12):1708–1721, Dec 2009.

[BH03] Brock Barber and Howard J. Hamilton. Extracting share frequent itemsets with infrequent subsets. *Data Mining and Knowledge Discovery*, 7(2):153–185, 2003.

[CA16] Yan Chen and Aijun An. Approximate parallel high utility itemset mining. *Big Data Research*, 6:26 – 42, 2016.

[CHY96] Ming-Syan Chen, Jiawei Han, and Philip S. Yu. Data mining: An overview from a database perspective. *IEEE Trans. on Knowl. and Data Eng.*, 8(6):866–883, December 1996.

[CRC10] J. D. Cryans, S. Ratt, and R. Champagne. Adaptation of apriori to mapreduce to build a warehouse of relations between named entities across the web. In *2010 Second International Conference on Advances in Databases, Knowledge, and Data Applications*, pages 185–189, April 2010.

[CTL09] Chun-Jung Chu, Vincent S. Tseng, and Tyne Liang. An efficient algorithm for mining high utility itemsets with negative item values in large databases. *Applied Mathematics and Computation*, 215(2):767 – 778, 2009.

[CYS03] Raymond Chan, Qiang Yang, and Yi-Dong Shen. Mining high utility itemsets. In *Proceedings of the Third IEEE International Conference on Data Mining*, ICDM '03, pages 19–, Washington, DC, USA, 2003. IEEE Computer Society.

[DG08] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, January 2008.

[EGA07] A. Erwin, R. P. Gopalan, and N. R. Achuthan. Ctu-mine: An efficient high utility itemset mining algorithm using the pattern growth approach. In *Computer and Information Technology, 2007. CIT 2007. 7th IEEE International Conference on*, pages 71–76, Oct 2007.

[Fre12] Frequent itemset mining dataset repository - http://fimi.ua.ac.be/data, 2012.

[FVLDD16] Philippe Fournier-Viger, Jerry Chun-Wei Lin, Quang-Huy Duong, and Thu-Lan Dam. *FHM+:Faster High-Utility Itemset Mining Using Length Upper-Bound Reduction*, pages 115–127. Springer International Publishing, Cham, 2016.

[FVWZT14] Philippe Fournier-Viger, Cheng-Wei Wu, Souleymane Zida, and Vincent S. Tseng. *FHM: Faster High-Utility Itemset Mining Using Estimated Utility Co-occurrence Pruning*, pages 83–92. Springer International Publishing, Cham, 2014.

[HPY00] Jiawei Han, Jian Pei, and Yiwen Yin. Mining frequent patterns without candidate generation. *SIGMOD Rec.*, 29(2):1–12, May 2000.

[Kri15] Srikumar Krishnamoorthy. Pruning strategies for mining high utility itemsets. *Expert Systems with Applications*, 42(5):2371 – 2381, 2015.

[LHL11] Chun-Wei Lin, Tzung-Pei Hong, and Wen-Hsiang Lu. An effective tree structure for mining high utility itemsets. *Expert Systems with Applications*, 38(6):7419 – 7424, 2011.

[LHT14] Guo-Cheng Lan, Tzung-Pei Hong, and Vincent S. Tseng. An efficient projection-based indexing approach for mining high utility itemsets. *Knowledge and Information Systems*, 38(1):85–107, 2014.

[LLC05] Ying Liu, Wei-keng Liao, and Alok Choudhary. *A Two-Phase Algorithm for Fast Discovery of High Utility Itemsets*, pages 689–695. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.

[LQ12] Mengchi Liu and Junfeng Qu. Mining high utility itemsets without candidate generation. In *Proceedings of the 21st ACM International Conference on Information and Knowledge Management*, CIKM '12, pages 55–64, New York, NY, USA, 2012. ACM.

[LWT15] Ying Chun Lin, Cheng-Wei Wu, and Vincent S. Tseng. *Mining High Utility Itemsets in Big Data*, pages 649–661. Springer International Publishing, Cham, 2015.

[LWZ+08] Haoyuan Li, Yi Wang, Dong Zhang, Ming Zhang, and Edward Y. Chang. Pfp: Parallel fp-growth for query recommendation. In *Proceedings of the 2008 ACM Conference on Recommender Systems*, RecSys '08, pages 107–114, New York, NY, USA, 2008. ACM.

[LYC05] Yu-Chiang Li, Jieh-Shan Yeh, and Chin-Chen Chang. *Direct Candidates Generation: A Novel Algorithm for Discovering Complete Share-Frequent Itemsets*, pages 551–560. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.

[SLL14] Wei Song, Yu Liu, and Jinhong Li. Bahui: Fast and memory efficient mining of high utility itemsets based on bitmap. *Int. J. Data Warehous. Min.*, 10(1):1–15, jan 2014.

[SPM] Spmf datasets - http://www.philippe-fournier-viger.com/spmf/index.php?link=datasets.php.

[TSWY13] V. S. Tseng, B. E. Shie, C. W. Wu, and P. S. Yu. Efficient algorithms for mining high utility itemsets from transactional databases. *IEEE Transactions on Knowledge and Data Engineering*, 25(8):1772–1786, Aug 2013.

[TWSY10] Vincent S. Tseng, Cheng-Wei Wu, Bai-En Shie, and Philip S. Yu. Up-growth: An efficient algorithm for high utility itemset mining. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '10, pages 253–262, New York, NY, USA, 2010. ACM.

[WSTY12] Cheng Wei Wu, Bai-En Shie, Vincent S. Tseng, and Philip S. Yu. Mining top-k high utility itemsets. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '12, pages 78–86, New York, NY, USA, 2012. ACM.

[YH06] Hong Yao and Howard J. Hamilton. Mining itemset utilities from transaction databases. *Data Knowl. Eng.*, 59(3):603–626, December 2006.

[YHB04] Hong Yao, Howard J. Hamilton, and Cory J. Butz. A foundational approach to mining itemset utilities from databases. In *Proceedings of the Third SIAM International Conference on Data Mining*, pages 482–486, 2004.

[YL07] Show-Jane Yen and Yue-Shi Lee. *Mining High Utility Quantitative Association Rules*, pages 283–292. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[YLF10] X. Y. Yang, Z. Liu, and Y. Fu. Mapreduce as a programming model for association rules algorithm on hadoop. In *The 3rd International Conference on Information Sciences and Interaction Sciences*, pages 99–102, June 2010.

[YR13] Unil Yun and Keun Ho Ryu. Efficient mining of maximal correlated weight frequent patterns. *Intell. Data Anal.*, 17(5):917–939, September 2013.

[YRR14] Unil Yun, Heungmo Ryang, and Keun Ho Ryu. High utility itemset mining with techniques for reducing overestimated utilities and pruning candidates. *Expert Syst. Appl.*, 41(8):3861–3878, June 2014.

[ZCF+10] Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica. Spark: Cluster computing with working sets. In *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing*, HotCloud'10, pages 10–10, Berkeley, CA, USA, 2010. USENIX Association.

[ZFVL+15] Souleymane Zida, Philippe Fournier-Viger, Jerry Chun-Wei Lin, Cheng-Wei Wu, and Vincent S. Tseng. *EFIM: A Highly Efficient Algorithm for High-Utility Itemset Mining*, pages 530–546. Springer International Publishing, Cham, 2015.

Curriculum Vitae

Graduate College
University of Nevada, Las Vegas

Ashish Tamrakar

Degrees:

Bachelor Degree in Computer Engineering 2012

Tribhuvan University, Institute of Engineering, Pulchwok Campus, Nepal

Thesis Title: High Utility Itemsets Identification in Big Data

Thesis Examination Committee:

Chairperson, Dr. Justin Zhan, Ph.D.

Committee Member, Dr. Laxmi Gewali, Ph.D.

Committee Member, Dr. Fatma Nasoz, Ph.D.

Graduate Faculty Representative, Dr. Ge Lin Kan, Ph.D.