

Efficient Algorithms For Mining Top-K High Utility Itemsets

A Project Report

Submitted in partial fulfillment of requirements to

For the award of the degree of

B. Tech in Computer Science and Engineering

By

Gadde Sai Sri Varsha (Y13CS840)

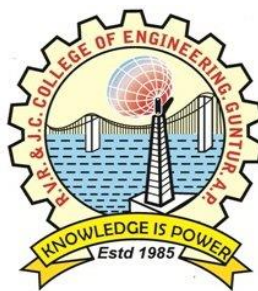
Bhimineni Sirisha (Y13CS815)

Arumbaka Jayanth (Y13CS807)

Under the Guidance of

Dr. Ch. Aparna

Associate Professor, Dept. of CSE.



MAY 2017

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

R.V.R & J.C. COLLEGE OF ENGINEERING (Autonomous)

(Affiliated to Acharya Nagarjuna University)

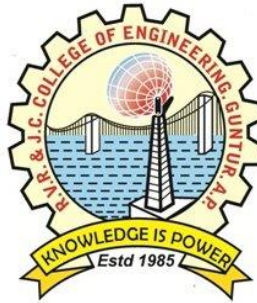
(Accredited by NBA and NAAC-‘A’ GRADE)

Chandramoulipuram::Chowdavaram,

GUNTUR – 522 019

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
R.V.R & J.C. COLLEGE OF ENGINEERING (Autonomous)

Chandramoulipuram::Chowdavaram, GUNTUR – 522 019



CERTIFICATE

This is to certify that this project work titled **“Efficient Algorithms For Mining Top-k High Utility Itemsets”** is being submitted by **Gadde Sai Sri Varsha (Y13CS840), Bhimineni Sirisha (Y13CS815), Arumbaka Jayanth (Y13CS807)**, in partial fulfillment for the award of the degree of Bachelor of Technology in computer science & Engineering is carried out by them under my guidance and supervision.

Dr. Ch. Aparna
Associate Professor, CSE.

Dr. M. Sreelatha,
Professor & Head, CSE.

ACKNOWLEDGEMENT

The successful completion of any task would be incomplete without a proper suggestion, guidance and environment. Combination of these three factors acts like backbone to our project “**Efficient Algorithms For Mining Top-k High Utility Itemsets**”.

We place on record and warmly acknowledge the continuous encouragement, invaluable supervision, timely suggestions offered by our guide **Dr. Ch. Aparna**, Associate Professor, Department of Computer Science and Engineering, at RVR & JC College of Engineering and technology, Guntur in bringing this report to a successful completion.

We are grateful to **Dr. M. Sreelatha**, Professor and Head of the Department of Computer Science and Engineering for permitting us to make use of the resources and facilities available in the department to carry out this project work successfully.

We mention our sincere thanks to our Principal, **Dr. K. Srinivasu** for providing us a supportive and stimulating environment. We would also like to express our gratitude to the Management of R.V.R & J.C College of Engineering for providing us with a pleasant environment and excellent lab facility.

Finally, we would be thankful to all the teaching and non-teaching staff of the department of Computer Science and Engineering for their cooperation given for the successful completion of project.

Gadde Sai Sri Varsha (Y13cs840)

Bhimineni Sirisha (Y13cs815)

Arumbaka Jayanth (Y13cs807)

ABSTRACT

High utility itemsets mining identifies itemsets whose utility satisfies a given threshold. In this high utility itemset mining threshold value has to be specified by the user which is tedious process since users need domain knowledge to specify the threshold value correctly. The main objective of the present work is to study a new framework for identifying the high utility itemsets without setting the minimum utility threshold min_util . Two efficient algorithms are proposed in our work named TKU (mining top-k utility itemsets) which is a two phase algorithm and another algorithm is TKO (mining top-k utility itemsets in one phase) which is a one phase algorithm. These algorithms will be used to identify the high utility itemsets without the need to set minimum utility threshold value. Empirical evaluations on both real and synthetic datasets will be discussed.

CONTENTS

	Page No
Abstract	IV
List of Tables	VII
List of Figures	VIII
List of Abbreviations	IX
Chapter 1: Introduction	
1.1 Background	1
1.2 Problem Definition	3
1.3 Significance of the work	3
Chapter 2: Literature Review	4
Chapter 3: System Analysis	
3.1. Requirements Model	
3.1.1. Functional Requirements	10
3.1.2. System Requirement Specification	10
3.2 UML diagrams for the Project work	
3.2.1. Use case Diagram	11
3.2.2. Sequence Diagram	12
3.2.3 Collaboration Diagram	13
3.2.4 Activity Diagram	15
3.2.5 Class Diagram	16
3.2.6 Component Diagram	17
3.2.7 Deployment Diagram	17
Chapter 4: System Design	
4.1 Workflow of the proposed System	18

4.2	Module Description	
4.2.1	Module1: TKU	24
4.2.2	Module2 : TKO	25
 Chapter 5: Implementation		
5.1	Algorithms	27
5.2	Datasets used	35
5.3	Metrics calculated	36
 Chapter 6: Testing		
6.1	Testing	41
6.2	Test Cases	42
 Chapter 7: Results		
7.1	Actual results of the work	45
7.2	Analysis of the results obtained	47
 Chapter 8: Conclusion & Future work		
	References	49

LIST OF TABLES

Table no	Description	Page no
Table 4.1	Transaction Data Base and Profit Table	19
Table 4.2	Items and Maximum and Minimum Utility values	22
Table 4.3	Initial Utility List	23
Table 5.1	Pre-Evaluation step	29
Table 5.2	MIU values of descendants of node $N_{\{C\}}$	30
Table 5.3	Characteristics Of datasets	35
Table 5.4	Strategies used by the algorithms	36
Table 5.5	Number of candidates	37

LIST OF FIGURES

Figure No	Description	Page no
Figure 4.1	A UP-Tree after inserting all transactions	21
Figure 4.2	Initial Utility List	23
Figure 5.1	Performance of the algorithms on Food mart dataset	36
Figure 5.2	Performance of the algorithms on Mushroom dataset	38
Figure 5.3	Run-time of TKU and TKO algorithms	39
Figure 5.4	Memory usage of TKU and TKO algorithms	40
Figure 5.5	Scalability of TKU and TKO algorithms	40

LIST OF ABBREVIATIONS

Abbreviation	Expansion
HUI	High Utility Itemset
TU	Transaction Utility
TWW	Transaction Weighted Utilization.
UP	Utility pattern Tree
MIU	Minimum Utility of an item.
MAU	Maximum utility of an item.
RU	Remaining utility of an item.
ESTU	Estimated Transaction Utility

CHAPTER 1

INTRODUCTION

1.1 BACKGROUND

The title of the work is “Mining Top-k High Utility itemsets using efficient algorithms”. An itemset is non-empty set of items. An item set with k different items is called K itemset. The items which appear frequently in a transaction are called frequent item sets.

Frequent itemset mining method doesn't consider the external utility of an item i.e., the profit that the store gets by selling that item it considers only the internal utility of an item i.e., number of times the item appears in the transaction. In order to overcome this high utility itemset mining is introduced, which considers both the internal and external utility of an item.

In high utility itemset mining, a transaction database which has finite set of transactions of items is considered and transaction utility of each and every transaction is calculated. A user defined value which is called minimum threshold is considered to determine high utility itemset. An itemset is called an high utility itemset if it's transaction utility is not less than the user defined minimum threshold value ,otherwise itemset is called the low utility itemset. The performance of this algorithm depends on the choice of minimum utility threshold value. To overcome this problem a new frame work for mining top-k high utility itemsets (where K defines the desired number of high utility itemsets) has been proposed.

High utility itemset mining:

The traditional FIM (frequent itemset mining) considers only the internal utility (quantity) not the external utility (profit) of an item. Hence, it cannot satisfy the requirement of users who desire to discover itemsets with high utilities such as high profits. To address these issues, utility mining emerges as an important topic in data mining and has received extensive attention in recent years. In utility mining, each item is associated with a utility(e.g. unit profit) and count of item in each transaction(e.g. Quantity) .An itemset is called high utility itemset if its utility is greater than the minimum utility threshold min_util. High utility itemset mining has received lots of attention and many efficient algorithms have been developed

such as UP growth and D^2 HUP and HUI miner. These algorithms can be generally categorized into two types: Two phase algorithm and one-phase algorithm. The main characteristic of two-phase algorithms is that they consist of two phases. In the first phase, they generate a set of candidates that are potential high utility itemsets. In the second phase, they calculate the exact utility of each candidate found in the first phase to identify high utility itemsets. IHUP, IIDS and UP-Growth are two-phase based algorithms.

On the Contrary side, the main characteristic of one-phase algorithms is that they discover high utility itemsets using only one phase and produce no candidates. Although the above studies may perform well in some applications, they are not developed for top-k high utility itemset mining and still suffer from the subtle problem of setting appropriate thresholds.

Top-k pattern mining:

Many Studies have been proposed to mine different kinds of top-k patterns, such as top-k frequent itemsets and top-k frequent closed itemsets and top-k closed sequential patterns, top-k association rules and what distinguishes each top-k pattern mining algorithm is the type of patterns discovered, as well as the data structures and search strategies that are employed. Some algorithms use a rule of expansion strategy for finding patterns while others rely on a pattern growth search using structures such as FP tree. The choice of data structures and search strategy affect the efficiency of a top-k pattern mining algorithm in terms of both memory and execution time.

However, the above algorithms discover top-k patterns according to traditional measures instead of the utility measure. As a consequence, they may miss patterns yielding high utility.

In high utility itemset mining to identify high utility itemsets minimum utility threshold value has to be set. Setting this threshold value is difficult and doesn't produce efficient results. In this work problem of setting the minimum utility threshold value, has been reduced. In the previous methods of Top-k mining they have considered only the quantities not the utility measures, so as a result they may miss patterns yielding high utility. So a new method which overcomes these problems has been proposed.

1.2 PROBLEM DEFINITION

Previous algorithms of mining high utility itemsets require the user to specify the value for k , which indicates the desired number of high utility itemsets from the entire transaction database. Setting the too low value for minimum utility threshold results in generating many high utility itemsets, on the other hand setting too high value generates no high utility itemset.

1. How to identify high utility itemsets without setting minimum utility threshold?
2. How to obtain desired number of itemsets that user need from entire transaction database?

1.3 SIGNIFICANCE OF THE WORK

In the high utility itemset mining method the minimum utility threshold value has to be set. Setting the minimum utility threshold value by trial and error is a difficult problem for users. If this threshold value is set too low then many high utility itemsets will be generated which will cause the mining process to be very in-efficient. On the other hand setting minimum utility threshold value too high then no high utility itemsets will be generated. In this work top- k high utility itemsets are identified without specifying the minimum utility threshold value. In top- k high utility itemsets mining, k value is specified by the user and this k indicates the desired number of high utility itemsets that user need from the entire transactions in the database.

CHAPTER 2

LITERATURE REVIEW

Efficient tree structures for high-utility pattern mining in incremental databases:

Ahmed.C, Tanbeer.S, et.al [1]. Recently, high utility pattern (HUP) mining is one of the most important research issues in data mining due to its ability to consider the non-binary frequency values of items in transactions and different profit values for every item. On the other hand, incremental and interactive data mining provide the ability to use previous data structures and mining results in order to reduce unnecessary calculations when a database is updated, or when the minimum threshold is changed. In this work, three novel tree structures to efficiently perform incremental and interactive HUP mining is proposed. The first tree structure, Incremental HUP Lexicographic Tree (IHUP_L-Tree), is arranged according to an item's lexicographic order. It can capture the incremental data without any restructuring operation. The second tree structure is the IHUP transaction frequency tree (IHUP_{TF}-Tree), which obtains a compact size by arranging items according to their transaction frequency (descending order). To reduce the mining time, the third tree, IHUP-transaction-weighted utilization tree (IHUP_{TWU}-Tree) is designed based on the TWU value of items in descending order. Extensive performance analyses show that our tree structures are very efficient and scalable for incremental and interactive HUP mining.

Applying the maximum utility measure in high utility sequential pattern mining:

Lan.G, Hong.T, et.al [7]. Recently, high utility sequential pattern mining has been an emerging popular issue due to the consideration of quantities, profits and time orders of items. The utilities of sub-sequences in sequences in the existing approach are difficult to be calculated due to the three kinds of utility calculations. To simplify the utility calculation, this work then presents a maximum utility measure, which is derived from the principle of traditional sequential pattern mining that the count of a subsequence in the sequence is only regarded as one. Hence, the maximum measure is properly used to simplify the utility calculation for sub-sequences in mining. Meanwhile, an effective upper-bound model is designed to avoid information losing in mining, and also an effective projection-based pruning

strategy is designed as well to cause more accurate sequence-utility upper-bounds of sub-sequences. The indexing strategy is also developed to quickly find the relevant sequences for prefixes in mining, and thus unnecessary search time can be reduced. Finally, the experimental results on several datasets show the proposed approach has good performance in both pruning effectiveness and execution efficiency.

Efficient updating of discovered high-utility itemsets for transaction deletion in dynamic databases:

Lin.C, Hong.T, et.al [8]. Most algorithms related to association rule mining are designed to discover frequent itemsets from a binary database. Other factors such as profit, cost, or quantity are not concerned in binary databases. Utility mining was thus proposed to measure the utility values of purchased items for finding high-utility itemsets from a static database. In real-world applications, transactions are changed whether insertion or deletion in a dynamic database. An existing maintenance approach for handling high-utility itemsets in dynamic databases with transaction deletion must rescan the database when necessary. In this work, an efficient algorithm, called PRE-HUI-DEL, for updating high-utility itemsets based on the pre-large concept for transaction deletion is proposed. The pre-large concept is used to partition transaction-weighted utilization itemsets into three sets with nine cases according to whether they have large (high), pre-large, or small transaction-weighted utilization in the original database and in the deleted transactions. Specific procedures are then applied to each case for maintaining and updating the discovered high-utility itemsets. Experimental results show that the proposed PRE-HUI-DEL algorithm outperforms a batch two-phase algorithm and a FUP2-based algorithm in maintaining high-utility itemsets.

Direct discovery of high utility itemsets without candidate generation:

Liu.J, Wang.K, et.al [9]. Utility mining emerged recently to address the limitation of frequent itemset mining by introducing interestingness measures that reflect both the statistical significance and the user's expectation. Among utility mining problems, utility mining with the itemset share framework is a hard one as no anti-monotone property holds with the interestingness measure. The state-of-the-art works on this problem all employ a two-phase, candidate generation approach, which suffers from the scalability issue due to the huge

number of candidates. This paper proposes a high utility itemset growth approach that works in a single phase without generating candidates. Our basic approach is to enumerate itemsets by prefix extensions, to prune search space by utility upper bounding, and to maintain original utility information in the mining process by a novel data structure. Such a data structure enables us to compute a tight bound for powerful pruning and to directly identify high utility itemsets in an efficient and scalable way. Further enhance the efficiency significantly by introducing recursive irrelevant item filtering with sparse data, and a look-ahead strategy with dense data. Extensive experiments on sparse and dense, synthetic and real data suggest that our algorithm outperforms the state-of-the-art algorithms over one order of magnitude.

Top-k high utility pattern mining with effective threshold raising Strategies:

Ryang.H and Yun.U, [12]. In pattern mining, users generally set a minimum threshold to find useful patterns from databases. As a result, patterns with higher values than the user-given threshold are discovered. However, it is hard for the users to determine an appropriate minimum threshold. The reason for this is that they cannot predict the exact number of patterns mined by the threshold and control the mining result precisely, which can lead to performance degradation. To address this issue, top-k mining has been proposed for discovering patterns from ones with the highest value to ones with the kth highest value with setting the desired number of patterns, k. Top-k utility mining has emerged to consider characteristics of real-world databases such as relative importance of items and item quantities with the advantages of top-k mining. Although a relevant algorithm has been suggested in recent years, it generates a huge number of candidate patterns, which results in an enormous amount of execution time. In this work, efficient algorithm for mining top-k high utility patterns with highly decreased candidates is proposed. For this purpose, three strategies that can reduce the search space by raising a minimum threshold effectively in the construction of a global tree, where they utilize exact and pre-evaluated utilities of itemsets are developed. Moreover, a strategy to identify actual top-k high utility patterns from candidates with the exact and pre-calculated utilities is suggested. Comprehensive experimental results on both real and synthetic datasets show that our algorithm with the strategies outperforms state-of-the-art methods.

Discovering high utility itemsets with multiple minimum supports:

Ryang.H, Yun.U, et.al [13]. Generally, association rule mining uses only a single minimum support threshold for the whole database. This model implicitly assumes that all items in the database have the same nature. In real applications, however, each item can have different nature such as medical datasets which contain information of both diseases and symptoms or status related to the diseases. Therefore, association rule mining needs to consider multiple minimum supports. Association rule mining with multiple minimum supports discovers all item rules by reflecting their characteristics. Although this model can identify meaningful association rules including rare item rules, not only the importance of items such as fatality rate of diseases but also attribute of items such as duration of symptoms are not considered since it treats each item with equal importance and represents the occurrences of items in transactions as binary values. In this paper, a novel tree structure, called MHU-Tree (Multiple item supports with High Utility Tree), which is constructed with a single scan are proposed. Moreover, an algorithm, named MHU-Growth (Multiple item supports with High Utility Growth), for mining high utility itemsets with multiple minimum supports is proposed. Experimental results show that MHU-Growth outperforms the previous algorithm on both real and synthetic datasets, and can discover useful rules from a medical dataset.

Mining high utility mobile sequential patterns in mobile commerce environments:

Shie.B, Hsiao.H, et.al [14]. Mining user behaviours in mobile environments is an emerging and important topic in data mining fields. Previous researches have combined moving paths and purchase transactions to find mobile sequential patterns. However, these patterns cannot reflect actual profits of items in transaction databases. In this work, a new problem of mining high utility mobile sequential patterns by integrating mobile data mining with utility mining is explored. To the best of our knowledge, this is the first work that combines mobility patterns with high utility patterns to find high utility mobile sequential patterns, which are mobile sequential patterns with their utilities. Two tree-based methods are proposed for mining high utility mobile sequential patterns. A series of analyses on the performance of the two algorithms are conducted through experimental evaluations. The results show that the proposed algorithms deliver better performance than the state-of-the-art one under various conditions.

Mining top-k high utility itemsets:

Wu.C, Shie.B, et.al [18]. Online high utility itemset mining over data streams has been studied recently. However, the existing methods are not designed for producing top-k patterns. Since there could be a large number of high utility patterns, finding only top-k patterns is more attractive than producing all the patterns whose utility is above a threshold. A challenge with finding top-k high utility itemsets over data streams is that it is not easy for users to determine a proper minimum utility threshold in order for the method to work efficiently. In this paper, a new method (named T-HUDS) for finding top-k high utility patterns over sliding windows of a data stream is proposed. The method is based on a compressed tree structure, called HUDS-tree, that can be used to efficiently find potential top-k high utility itemsets over sliding windows. T-HUDS uses a new utility estimation model to more effectively prune the search space. Several strategies for initializing and dynamically adjusting the minimum utility threshold is proposed. No top-k high utility itemset is missed by the proposed method. Experimental results on real and synthetic datasets show that our strategies and new utility estimation model work very effectively and that T-HUDS outperforms two state-of-the-art high utility itemset algorithms substantially in terms of execution time and memory storage.

Efficient mining of a concise and lossless representation of high utility itemsets:

Wu.C, Fournier-Viger.P, et.al [19]. Mining high utility item sets from transactional databases is an important data mining task, which refers to the discovery of item sets with high utilities (e.g. high profits). Although several studies have been carried out, current methods may present too many high utility item sets for users, which degrades the performance of the mining task in terms of execution and memory efficiency. To achieve high efficiency for the mining task and provide a concise mining result to users, a novel framework in this paper for mining closed+ high utility item sets, which serves as a compact and loss less representation of high utility item sets is proposed. An efficient algorithm called CHUD (Closed+ High Utility item set Discovery) for mining closed+ high utility item sets is presented. Further, a method called DAHU (Derive All High Utility item sets) is proposed to recover all high utility item sets from the set of closed+ high utility item sets without accessing the original database. Results of experiments on real and synthetic datasets show that CHUD and DAHU are very efficient with a massive reduction (up to 800 times in our experiments) in the number of high

utility item sets. In addition, when all high utility item sets are recovered by DAHU, the approach combining CHUD and DAHU also outperforms the state-of-the-art algorithms in mining high utility item sets.

Mining Top-K strongly correlated pairs in large databases:

Xiong.H, Brodie.M, et.al [20]. Recently, there has been considerable interest in computing strongly correlated pairs in large databases. Most previous studies require the specification of a minimum correlation threshold to perform the computation. However, it may be difficult for users to provide an appropriate threshold in practice, since different data sets typically have different characteristics. To this end, a 2-D monotone property of an upper bound of Pearson's correlation coefficient is proposed and develop an efficient algorithm, called TOP-COP to exploit this property to effectively prune many pairs even without computing their correlation coefficients. Our experimental results show that the TOP-COP algorithm can be orders of magnitude faster than brute-force alternatives for mining the top-k strongly correlated pairs.

CHAPTER 3

SYSTEM ANALYSIS

3.1 REQUIREMENT MODEL

Requirements analysis, also called requirements engineering, is the process of determining user expectations for a new or modified product. These features, called requirements, must be quantifiable, relevant and detailed. In software engineering, such requirements are often called functional specifications.

Requirements analysis is critical to the success or failure of a systems or software project. The requirements should be documented, actionable, measurable, testable, traceable, related to identified business needs or opportunities, and defined to a level of detail sufficient for system design

3.1.1 FUNCTIONAL REQUIREMENTS

Input:

- A Transaction Database
- A Profit Database
- The number of desired itemsets K.

Output:

List of K high utility itemsets.

3.1.2 SYSTEM REQUIREMENT SPECIFICATION

Hardware Requirements:

- | | | |
|-------------|---|-------------|
| • Processor | - | Pentium –IV |
| • Speed | - | 1.1 Ghz |
| • RAM | - | 256 MB(min) |
| • Hard Disk | - | 20 GB |

- Key Board - Standard Windows Keyboard
- Mouse - Two or Three Button Mouse
- Monitor - SVGA

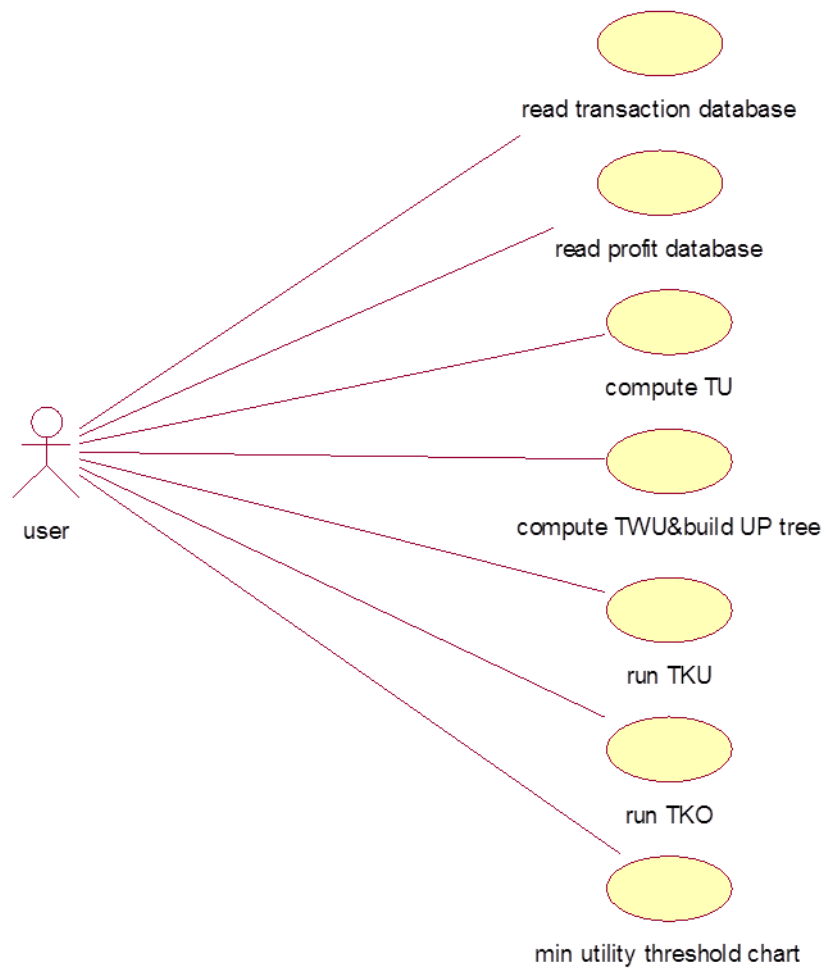
Software Requirements:

- Operating System - Windows XP
- Programming Language - JAVA

3.2. UML DIAGRAMS

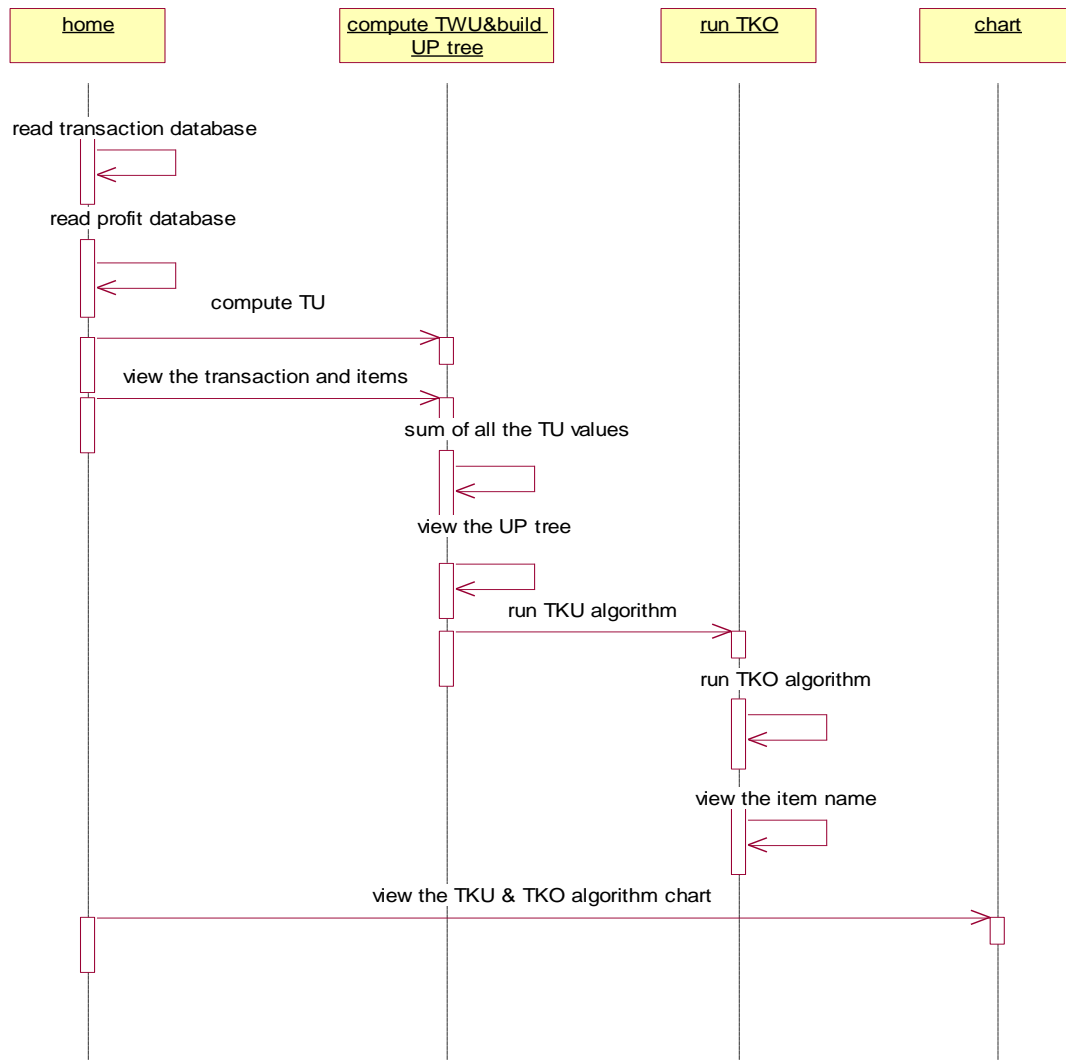
3.2.1 Use case diagram:

In this use-case diagram seven use-cases have been identified. The first two use-cases are used for reading the transactions and profit from the database which are the inputs to the algorithms. The next two use-cases are used for calculating the transaction utility (TU) and transaction weighted utilization (TWU). The last three use-cases are used for running TKU and TKO algorithms and comparison of those algorithms.



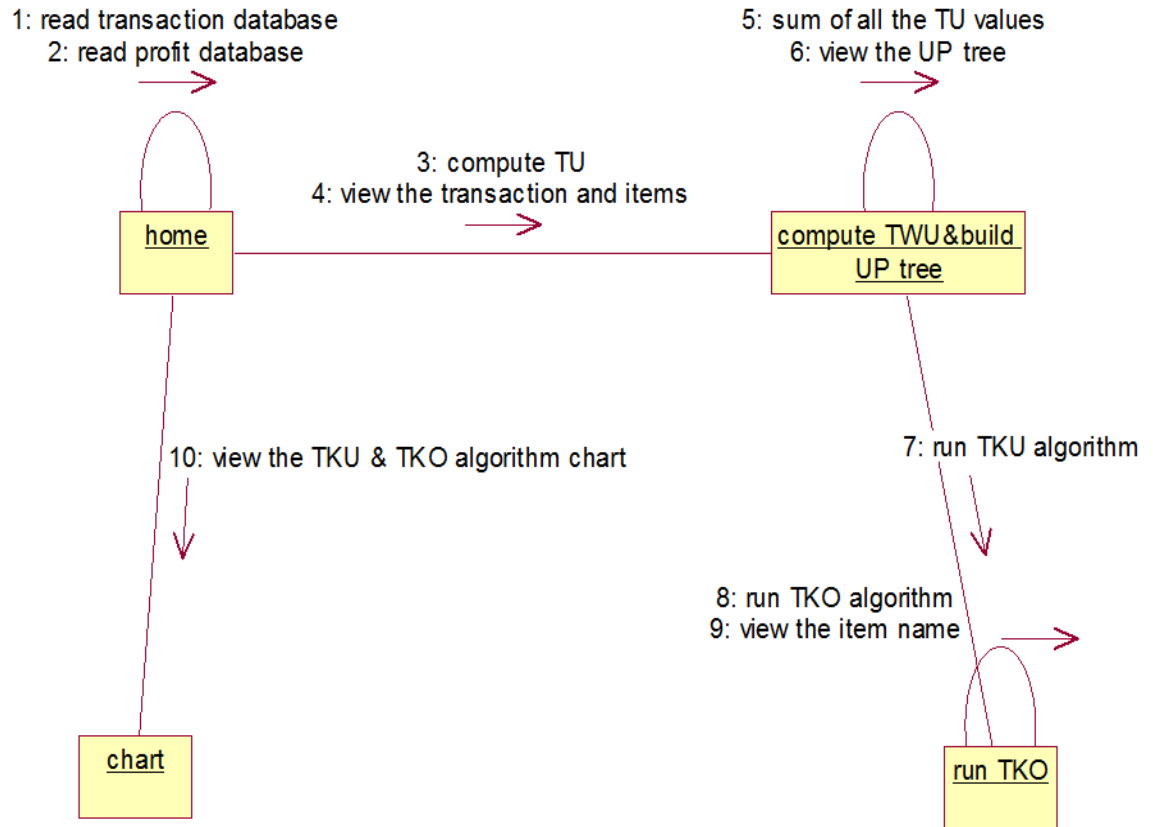
3.2.2 Sequence Diagram:

In this sequence diagram four objects are interacting with each other. The four objects are home or the user, compute TWU and build UP tree and two algorithms run TKU and TKO. First the user reads the transaction and profit tables from the database .Next calculate the transaction utility and build UP tree and display both of them to the user. And then user specify the K value i.e. the desired number of high utility itemsets and run the two algorithms TKU and TKO and the output is displayed to the user.



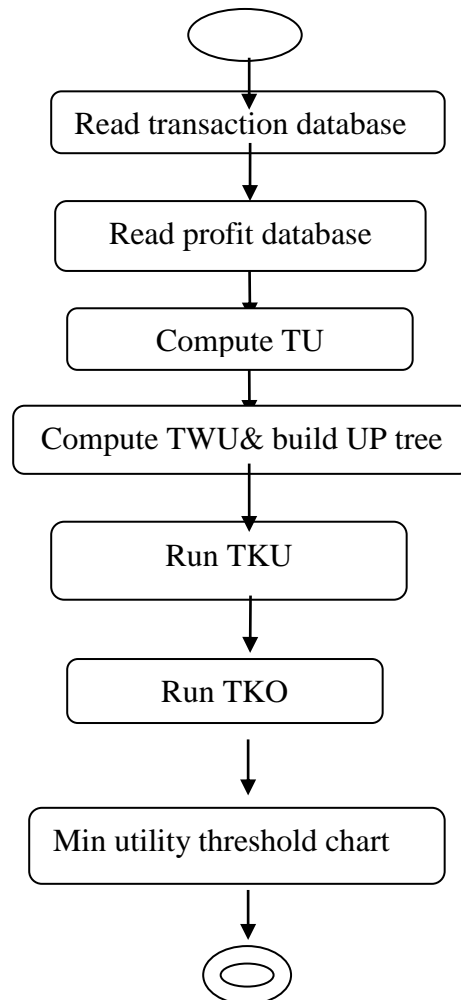
3.2.3 Collaboration diagram:

In this diagram four objects are interacting with each other .First read the transactions and profits from the database and perform the necessary calculation by interacting with the TWU and build UP tree object and run the algorithms to get result and the results of the algorithms are compared by using the chart.



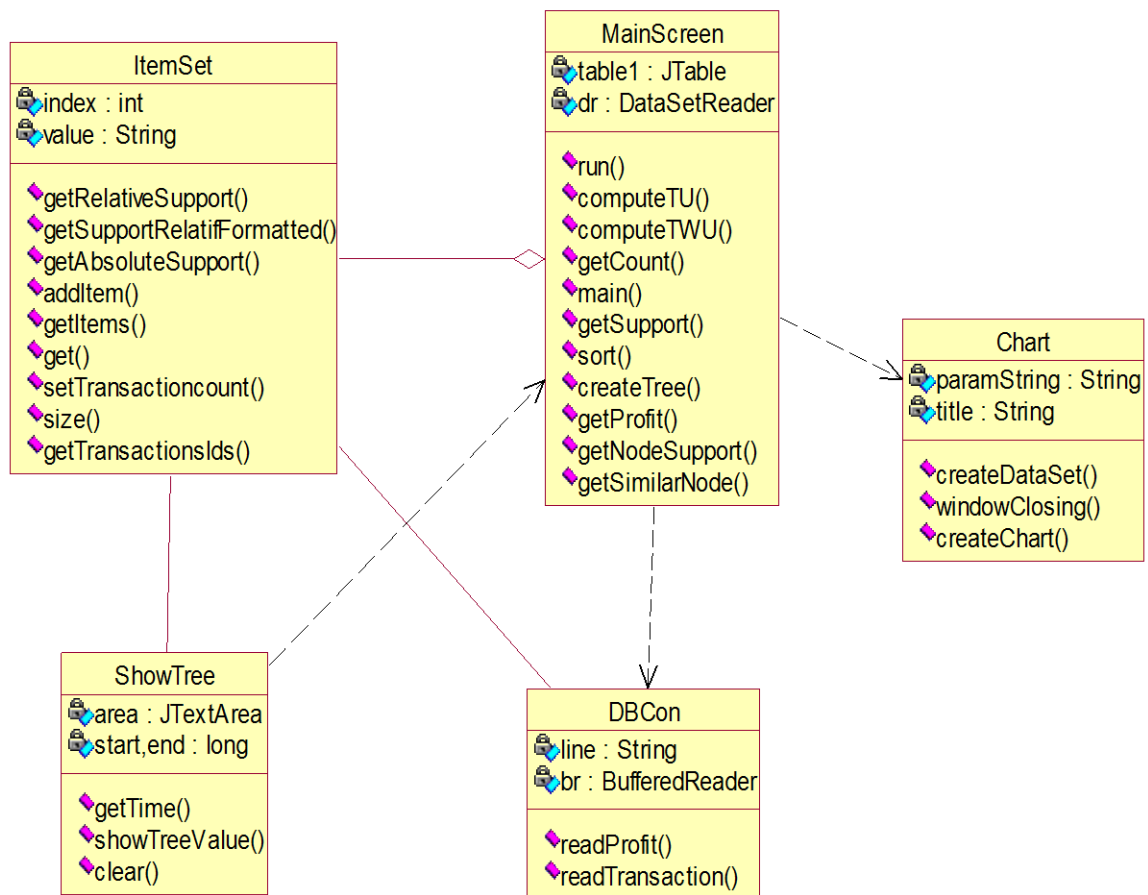
3.2.4 Activity diagram:

Activity diagram represent all the activities in the sequential order. It is basically a flow chart to represent the flow from one activity to another activity. Activity diagram starts by reading the transactions and profits from database and stop the flowchart when desired high utility itemsets are obtained and comparison is done between them.



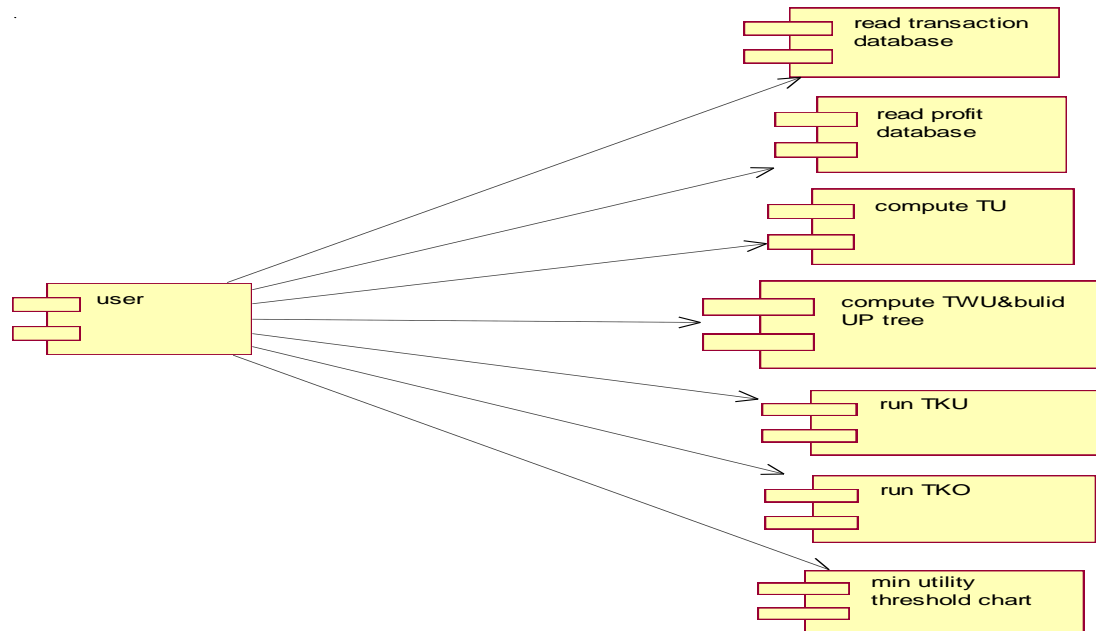
3.2.5 Class diagram:

In this diagram five classes have been identified. The five classes are Itemsets, MainScreen, Showtree, chart, Database connection. Each class has its own attributes and methods. For Example itemset class has methods like getItems, getRelativeSupport. The main screen class has methods like computeTU, computeTWU, createTree. The itemset class and mainscreen class has aggregation relationship among them. Mainscreen has dependency relationship between the chart and showtree class. Showtree class has dependency relationship with the mainscreen class. Database connection and itemset have association relationship between them.

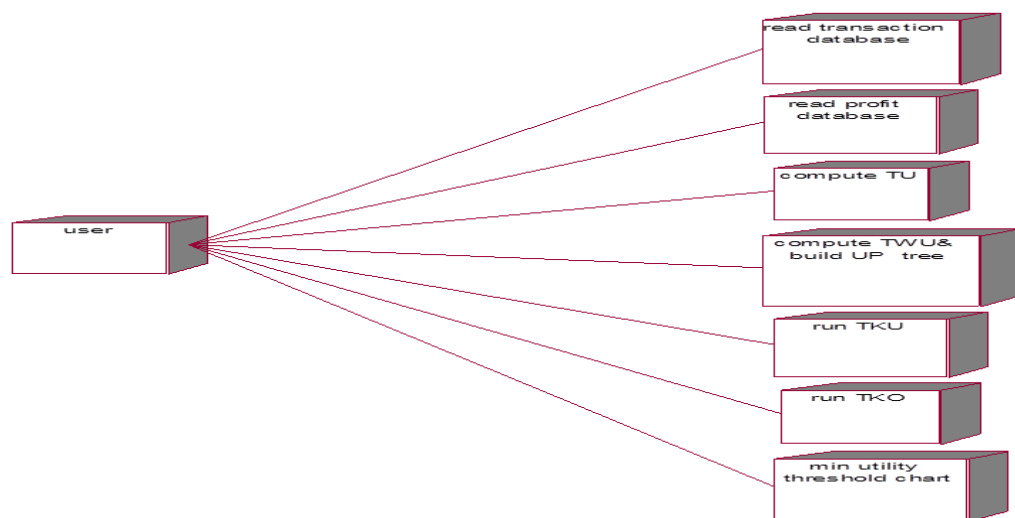


3.2.6 Component Diagram:

In this component diagram seven components are interacting with the user. User read the transactions from the database and perform the calculations and run the algorithms and identify the top-k high utility itemsets and also compare the algorithms.



3.2.7 Deployment diagram:



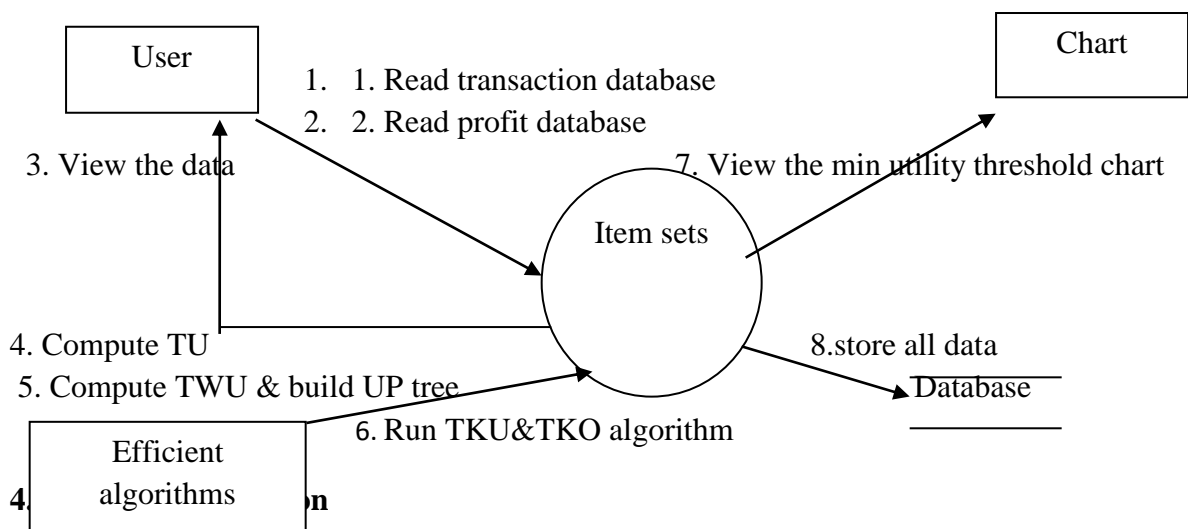
CHAPTER 4

SYSTEM DESIGN

4.1 Workflow of the proposed system

Work flow diagrams illustrate how data is processed by a system in terms of inputs and outputs. Work flow diagrams can be used to provide a clear representation of any business function. The technique starts with an overall picture of the business and continues by analysing each of the functional areas of interest. This analysis can be carried out in precisely the level of detail required. The technique exploits a method called top-down expansion to conduct the analysis in a targeted way.

As the name suggests, Work Flow Diagram is an illustration that explicates the passage of information in a process. A Work flow diagram can be easily drawn using simple symbols. Additionally, complicated processes can be easily automated by creating work flows using easy-to-use, free downloadable diagramming tools. A Work flow diagram is a model for constructing and analysing information processes. Diagram illustrates the flow of information in a process depending upon the inputs and outputs. A Work flow diagram can also be referred to as a Process Model. A work flow demonstrates business or technical process with the support of the outside data saved, plus the data flowing from the process to another and the end results.



In this work high utility itemsets are to be identified. Some of the basic terms that are related to this itemsets mining are:

1. Absolute utility of itemset.
2. Transaction utility
3. Support of an itemset

Absolute utility of an itemset:

TID	TRANSCATION	TRANSACTION UTILITY(TU)
T1	(D,1)(A,1)(C,1)	8
T2	(G,5)(A,2)(E,2)(C,6)	27
T3	(F,5)(D,6)(B,2)(A,1)(E,1)(C,1)	30
T4	(D,3)(B,4)(E,1)(C,3)	20
T5	(G,2)(B,2)(E,1)(C,2)	11

ITEM	A	B	C	D	E	F	G
PROFIT	5	2	1	2	3	1	1

Table – 4.1: Transaction Database and Profit Table

Absolute utility for itemset AC in transaction T1 is $1*5+1*1=6$ i.e., it is obtained by multiplying the quantity and its utility i.e., here it is profit.

Transaction utility:

Transaction utility of T2 is given by $2*5+6*2+2*3+5*1=27$. consider all the items that are present in the transaction T2 and calculate all the absolute utilities of the items.

Support:

Support of an itemset $AE = 2/5$. Scan in how many transactions the itemset AE appears and divide it by the total no of transactions present.

High utility itemset:

An item is called utility item set if it is not less than user specified minimum utility threshold \min_util ($0\% \leq \min_util \leq 100\%$). For example consider item set {BCD}.

$TU(BCD) = T3(BCD) + T4(BCD) = 2*2 + 1*1 + 6*2 + 4*2 + 3*1 + 3*2 = 34$. If it is less than minimum utility threshold then it is called the low utility itemset.

Techniques In Top-K High Utility Itemsets:

1. Transaction weighted utilization (TWU)
2. Utility pattern tree (UP-tree)
3. Minimum utility of an itemset.
4. Maximum utility of an itemset.
5. Initial utility Lists.

Transaction weighted utilization:

The TWU of an itemset X is sum of all the TU'S of all the transaction containing X. Transaction weighted utility of itemset AC given by

$$TWU(AC) = TU(T1) + TU(T2) + TU(T3) = 8 + 27 + 22 = 57.$$

Utility pattern tree:

A UP-Tree can be constructed by scanning the original database twice. In the first scan, the transaction utility of each transaction and TWU of each item are computed. Thus, items and their TWUs are obtained. Subsequently, items are inserted into the header table in descending order of their TWUs. During the second database scan, transactions are reorganized and then inserted into the UP-tree.

Initially, the tree is created with a root R. When a transaction is retrieved, items in the transaction are sorted in descending order of TWU. A transaction after the above reorganization is called reorganized transaction and its transaction utility is called Reorganized Transaction Utility. The items in each transaction are processed and a branch is created for each transaction.

If the same item repeats in another transaction increment the count value of that item and walso add the transaction utility of that transaction to its previous utility value.

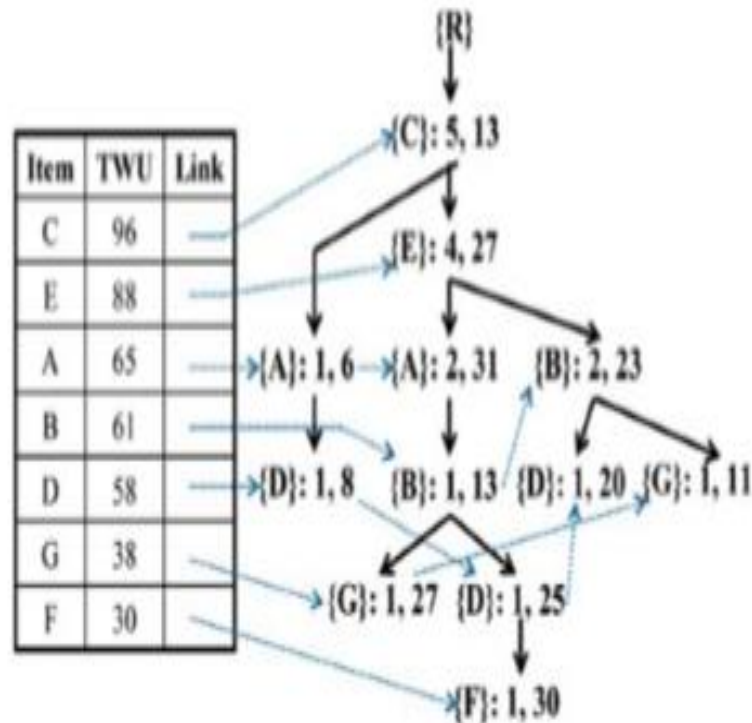


Figure – 4.1: A UP-Tree after inserting all the transactions.

Minimum utility of an itemset:

Minimum utility of an item is obtained by calculating the estimated utility of item for the transactions in which item is present and consider the minimum of them. $\text{miu}\{B\} = \min(\text{EU}(\{B\}, T_3, \text{EU}(\{B\}, T_4, \text{EU}(\{B\}, T_5) = \min\{4, 8, 4\} = 4.$

Minimum utility of an itemset is obtained by calculating the minimum utility of individual items and multiplying with their support count.
 $MIU\{BC\}=[miu(B)+miu(C)] \times SC(BC)=[4+1] \times 3=15.$

Maximum utility of an itemset:

Minimum utility of an item is obtained by calculating the estimated utility of item for each transactions in which item is present and consider the maximum of them.
 $miu\{B\}=\min((EU(\{B\},T3,EU(\{B\},T4,EU(\{B\},T5)=\min\{4,8,4\}=8$

Table – 4.2: Items and Their Minimum and Maximum Utility Values

ITEM	A	B	C	D	E	F	G
MAU	5	8	3	6	6	5	5

ITEM	A	B	C	D	E	F	G
MIU	5	4	1	2	3	5	2

Initial utility lists:

Table-4.3: Initial Utility List

TID	TRANSCATION	TRANSACTION UTILITY(TU)
T1	(D,1)(A,1)(C,1)	8
T2	(G,5)(A,2)(E,2)(C,6)	27
T3	(F,5)(D,6)(B,2)(A,1)(E,1)(C,1)	30
T4	(D,3)(B,4)(E,1)(C,3)	20
T5	(G,2)(B,2)(E,1)(C,2)	11

{G}		
2	5	22
5	2	9

{G}		
2	5	22
5	2	9

{B}		
3	4	9
4	8	6
5	4	5

{D}		
1	2	6
3	12	13
4	6	14

{A}		
1	5	1
2	10	12
3	5	4

{E}		
2	6	6
3	3	1
4	3	3
5	3	2

{C}		
1	1	0
2	6	0
3	1	0
4	3	0
5	2	0

{F}		
3	5	25

Figure –4.2: Initial Utility List

The utility-lists of items are called initial utility-lists, which can be constructed by scanning the database twice. In the first databases scan, the TWU and utility values of items are calculated. During the second database scan, items in each transaction are sorted in order of TWU values and the utility-list of each item is constructed. The utility-list of an itemset X consists of one or more tuples. Each tuple represents the information of X in a transaction Tr

and has three fields: Tid, iutil and rutil. Fields Tid and iutil respectively contains the identifier of Tr and the utility of X in Tr. Field rutil indicates the remaining utility of X in Tr.

4.2.1 MODULE1: TKU (mining top-k high utility itemsets)

The TKU algorithm adopts a compact tree -based structure named UP-tree to maintain the information of transactions and utilities of itemsets. TKU inherits useful properties from the TWU (transaction weighted utilization) model and consists of two phases: In phase 1, potential top-k high utility itemsets (PKHUIs) are generated. In phase 2, top-k HUIs are identified from the set of PKHUIs discovered in Phase 1.

In brief, this TKU algorithm is executed in three steps:

- Constructing of UP-tree (utility pattern tree)
- Generating potential top-k high utility itemsets (PKHUIs) from the constructed UP-tree
- Identifying top-k HUIs from the set of PKHUIs.

TKU is a two phase algorithm. TKU algorithm has three steps. The first step is construction of an UP-tree (utility pattern tree). A UP-Tree can be constructed by scanning the original database twice. In the first scan, the transaction utility of each transaction and TWU of each item are computed. Thus, items and their TWUs are obtained. Subsequently, items are inserted into the header table in descending order of their TWUs. During the second database scan, transactions are reorganized and then inserted into the UP-tree

During this construction of up-tree first calculate the TWU and compare that with the min_util threshold and if any item has less than min_util remove that item and that item is called unpromising item and here in the example F and G are unpromising_items. So, remove those items from the transaction in which those items are present and rearrange those transactions and calculate the utility values for reorganized transactions and next construct the up-tree as discussed above.

Generating PKHUIS from the UP-tree:

In this phase, all the PKHUIs are generated by using UP tree search procedure, which is already done by the previous UP-growth paper. Here, the extension of that will be checked and whether all the generated PKHUIs are valid or not by using the following algorithm.

The input of the algorithm is as follows:

- Database and their transactions is taken and the utilities of those transactions are calculated.
- Next input is to specify how many HUIs user want i.e., k value.

4.2.2 MODULE 2: TKO (mining high utility itemsets in one phase)

TKO algorithm uses a list based structure named utility list to store the utility information of itemsets in the database. It uses vertical data representation technique to discover top-k HUIs in only one phase. It utilizes the basic search procedure of HUI miner and its utility list structure. Whenever an itemset is generated by TKO, its utility is calculated by its utility list without scanning the original database. This is the second algorithm that is proposed and here the top-k itemsets in one phase only is identified. In this first a the list structure is constructed called utility list structure.

TID	TRANSCATION	TRANSACTION UTILITY(TU)
T1	(D,1)(A,1)(C,1)	8
T2	(G,5)(A,2)(E,2)(C,6)	27
T3	(F,5)(D,6)(B,2)(A,1)(E,1)(C,1)	30
T4	(D,3)(B,4)(E,1)(C,3)	20
T5	(G,2)(B,2)(E,1)(C,2)	11

{G}		
2	5	22
5	2	9

{G}		
2	5	22
5	2	9

{B}		
3	4	9
4	8	6
5	4	5

{D}		
1	2	6
3	12	13
4	6	14

{A}		
1	5	1
2	10	12
3	5	4

{E}		
2	6	6
3	3	1
4	3	3
5	3	2

{C}		
1	1	0
2	6	0
3	1	0
4	3	0
5	2	0

{F}		
3	5	25

The utility-list of an itemset X consists of one or more tuples. Each tuple represents the information of X in a transaction Tr and has three fields: Tid, iutil and rutil. Fields Tid and iutil respectively contains the identifier of Tr and the utility of X in Tr. Field rutil indicates the remaining utility of X in Tr

In order to calculate the remaining utility we have to know the two terms precede and succeed. Suppose in transaction T2 the succeed of item (A,2) are (E,2) and (C,6) and the precede of item (A,2) are (G,5).

Remaining utility of an item in a transaction:

The remaining utility of item D in a transaction T1 is given by

$$RU(D) = EU(\{A\}, T1) + EU(\{c\}, T1) = 5 + 1 = 6.$$

Remaining utility of item in a database:

The remaining utility of item D in a database is given by

$$RU(D) = (RU(d), T1) + RU(D), T3) + (RU(D), T4) = 6 + 13 + 14 = 33. \text{ An element with respect to the transaction Tr contains the information } (r, EU(X, Tr), RU(X, Tr)) \text{ in an utility list.}$$

CHAPTER 5

IMPLEMENTATION

5.1 ALGORITHMS

In this paper there are two algorithms first one is TKU algorithm.

TKU Algorithm:

Input: 1. A Database D
2. The number of desired HUIs k

Output: The complete set of PKHUIs C

Step1: Set $\text{min_util}_{\text{border}} \leftarrow 0$; TopK-MIU-List $\leftarrow \Phi$; C $\leftarrow \Phi$
Step2: Construct a UP-Tree by scanning database D twice;
Step3: // Apply a UP-Growth search procedure to generate PKHUIs;
Step4: For each PKHUI generated with estimated utility $\text{ESTU}(X)$ do
Step5: { If ($\text{ESTU}(X) \geq \text{min_util}_{\text{border}}$ and $\text{MAU}(X) \geq \text{min_util}_{\text{border}}$)
Step6: Output X and $\min\{\text{ESTU}(X), \text{MAU}(X)\}$; C $\leftarrow C \cup X$
Step7: If ($\text{MIU}(X) \geq \text{min_util}_{\text{border}}$)
Step8: { // Raise $\text{min_util}_{\text{border}}$ by the strategy MC;
Step9: $\text{min_util}_{\text{border}} \leftarrow \text{MC}(\text{MIU}(X), \text{TopK-MIU-List})$;
Step10: }
Step11: }

This algorithm uses an internal variable called Border minimum utility threshold (denoted as $\text{min_util}_{\text{border}}$) which is initially set to 0 and raised dynamically after a sufficient number of itemsets with higher utilities has been captured during the generation of PKHUIs. Each time a candidate itemset X is found by the UP-Growth search procedure, the TKU algorithm checks whether its estimated utility value $\text{ESTU}(X)$ is no less than $\text{min_uti}_{\text{border}}$. If

ESTU(X) is less than min_util border, X and all its concatenations are not top-k HUIs. Besides, TKU checks whether MAU(X) is no less than min_util border. If MAU(X) is smaller than min_util border, X is not a top-k HUI. Otherwise, X is considered a candidate for Phase II and it is outputted with $\min \{ESTU(X), MAU(X)\}$. If X is a valid PKHUI and MIU(X) \geq min_util border, MIU(X) can be used to raise min_util Border by the proposed strategy MC(raising the threshold by MIUs of Candidates)

Strategy Mc:

Raising the threshold by MIUs of Candidates. For any newly mined candidate itemset X, if MIU(X), ESTU(X) and MAU(X) are no less than the current min_util Border, then MIU(X) is added to Top-K-MIU-List. Each time a PKHUI X is found and its MIU is higher than min_util border, X is added into Top-K-MIU-List. If there are less than k MIU values in Top-K-MIU-List, min_util Border will not change.

Once k MIU values are found and the k-th MIU value (denoted as MIU_k-th) in Top-K-MIU-List is higher than min_util border, min_util border is raised to MIU_k-th in Top-K-MIU-List. Otherwise, if there exist more than k MUI values in Top-K-MIU-List, min_util border is raised to MIU_k-th in Top-K-MIU-List and the MIU values that are less than MIU_k-th in Top-K-MIU-List are removed. The algorithm continues searching for other PKHUIs until no candidate is found by the UP-Growth procedure.

Identifying Top-k HUIs from PKHUIs:

After identifying PKHUIs, TKU Base calculates the utility of PKHUIs by scanning the original database once, to identify the top-k HUIs. In this work, only consider a candidate itemset X if its estimated utility value reached after phase I is no less than min_util border, i.e., $\min \{ESTU(X), MAU(X)\} \geq \text{min_util border}$. In this way determine the top-k high utility itemsets using TKU algorithm which is a two phase algorithm. we propose four strategies to effectively raise min_utilBorder during different stages of the mining process. The four strategies are incorporated in TKUBase to form the advanced TKU algorithm.

Pre-Evaluation Step:

Table –5.1: Pre-Evaluation Step

Item	B	C	D	E	F	G
B	9	28	24	24	10	15
C		17	14	18	0	6
D			0	0	0	0
E				0	0	0
F					0	0
G						0

Though TKUBase provides a way to mine top-k HUIs, min_util border is set to 0 before the construction of the UP-Tree. This results in the construction of a full UP-Tree in memory, which degrades the performance of the mining task. If min_util border could be raised before the construction of the UP-Tree and prune more unpromising items [25] in transactions, the number of nodes maintained in memory could be reduced and the mining algorithm could achieve better performance. Based on this idea, we propose a strategy named PE (Pre-evaluation Step) to raise min_util border during the first scan of the database.

Strategy 2 (PE: Pre-Evaluation):

The strategy PE uses a structure named Pre-Evaluation Matrix (PEM) to store lower bounds of the utilities of certain 2-itemsets. Each entry in PEM is denoted as $PEM[x][y]$ and corresponds to the lower bound of $EU(\{x, y\})$, where $x, y \in I^*$. Initially, each value in PEM is set to 0. When a transaction $Tr = \{I_1, I_2, \dots, I_M\}$ ($I_j \in I, 1 \leq i \leq M$) is retrieved during the first database scan, the utility of $\{I_1\} \cup \{I_i\}$ ($1 \leq i \leq M$) in Tr is added to the value of the corresponding entry of $PEM[I_1][I_i]$ in PEM. After scanning all the transactions, if the k-th highest value in PEM is higher than min_util border, min_util border can be raised to the k-th highest value in PEM. The space complexity of the strategy is $O(|I^*|^2/2)$, where $|I^*|$ is the number of distinct items in the database

Raising the threshold by node utilities:

The strategy NU (raising the threshold by node utilities) is applied during the construction of the UP Tree (during the second database scan). If there are more than k nodes in the current UP-Tree and the k^{th} highest node utility value $NU_{k\text{-th}}$ is higher than min_util border. min_util border can be raised to $NU_{k\text{-th}}$. After inserting all reorganized transactions, the size of the constructed UP Tree can be further reduced by pruning items whose TWU values are less than min_util border the UP-Tree.

Raising the threshold by MIU values of descendents:

The third strategy that we propose is called MD (raising the threshold by MIU values of Descendents). It is applied after the construction of the UP-Tree and before the generation of PKHUIs.

Table – 5.2

MIU Values of Descendents of Node $N_{\{C\}}$

Descendent	$N_{\{E\}}$	$N_{\{A\}}$	$N_{\{B\}}$	$N_{\{D\}}$	$N_{\{G\}}$	$N_{\{F\}}$
SC	4	3	3	3	2	1
MIU	16	18	15	9	3	6

Strategy MD: (raising the threshold by MIU values of Descendents). Let the notation N_a represents a node of the UP-Tree such that a is the item stored in N . For each node N_a under the root of UP-Tree, the algorithm traverses the sub-tree under node N_a once to calculate the support count of the itemset $\{a[b]\}$ for every descendent node N_b of N_a . For each itemset $\{a[b]\}$, the MIU value of $\{a[b]\}$ is calculated. If the k -th highest MIU value is higher than $\text{min_util}_{\text{Border}}$, $\text{min_util}_{\text{Border}}$ can be safely raised to that value.

Raising the Threshold Using SE:

The fourth proposed strategy is called SE (raising the threshold by Sorting and calculating Exact utility of candidates), which is applied during the phase II of TKU.

Strategy 5 (SE: raising the threshold by Sorting and calculating exact utility of candidates). Let C be the set of candidates produced in Phase I. Candidates in C are sorted in descending order of their estimated utilities, i.e., $\min\{\text{ESTU}(X), \text{MAU}(X)\}$. Thus, candidates with higher estimated utility values will be considered before those having lower estimated utility values. During the phase II, if the utility of a newly considered HUI X is larger than $\text{min_util}_{\text{Border}}$, X and $\text{EU}(X)$ are inserted into a min-heap structure named Top K-HUI-List. HUIs in Top K-HUI-List are ordered by decreasing utility. Then, $\text{min_util}_{\text{Border}}$ is raised to the utility of the k th HUI in Top KHUI- List and HUIs having a utility lower than $\text{min_util}_{\text{Border}}$ are removed from Top-K-HUI-List. If the estimated utility of the current candidate Y , i.e., $\min\{\text{ESTU}(Y), \text{MAU}(Y)\}$, is less than the raised $\text{min_util}_{\text{Border}}$, Y and the remaining candidates do not need to be considered anymore because the upper bounds on their utilities are less than $\text{min_util}_{\text{Border}}$. When the algorithm completes, Top K-HUI-List captures all the top- k HUIs in the database. By this mechanism, itemsets with lower estimated utility values may not be checked if the $\text{min_util}_{\text{Border}}$ has been previously raised. Thus, the I/O cost and execution time for Phase II can be further reduced.

TKO Algorithm:

- Input:**
1. $u(p)$: utility list prefix for a prefix P ;
 2. $\text{class}[p]$: a set of itemsets w.r.t the prefix P ;
 3. $\text{ULS}[p]$: a set of utility-lists w.r.t the prefix P ;
 4. δ : border minimum utility threshold $\text{min_util}_{\text{border}}$;
 5. TopK-CL-List a list for storing candidate itemsets;

Output: Use TopK-CL-List to capture all the top- k HUIs

Step1: For each $X = \{x_1, x_2, \dots, x_L\} \in \text{Class}[p]$ do

Step2: { if($\text{SUM}(X.\text{utils}) \geq \delta$)

Step3: { // Raise $\text{min_util}_{\text{border}}$ by the strategy RUC;

Step4: $\delta \leftarrow \text{RUC}(X, \text{TopK-CL-List})$;

Step5: }

Step6: If ($\text{SUM}(X.\text{iutils}) + \text{SUM}(X.\text{rutils}) \geq \delta$)


```

Step7: { Class[X]  $\leftarrow \Phi$ ; ULS[X]  $\leftarrow \Phi$ 

Step8: For each  $Y=\{y_1, y_2, \dots, y_L\} \in \text{Class}[p] \mid y_L > x_L$  do

Step9: { Z  $\leftarrow X \cup Y$ ;

Step10:  $\text{ul}(Z) \leftarrow \text{Construct}(\text{ul}(P), X, Y, \text{ULS}[P])$ ;

Step11:  $\text{Class}[X] \leftarrow \text{Class}[X] \cup Z$ ;

Step12:  $\text{ULS}[X] \leftarrow \text{ULS}[X] \cup \text{ul}(Z)$ ;

Step13: }

Step14: TopK-HUI Search( $X, \text{ULS}[X], \text{Class}[X], \delta, \text{TopK-CL-List}$ );

Step15: }

Step16: }
```

The TKO algorithm takes as input the parameter k and a transactional database D in horizontal format. But if a database has already been transformed into vertical format such as initial utility-lists, TKO can directly use it for mining top- k HUIs.

TKO initially sets the min_util Border threshold to 0 and initializes a min-heap structure Top K-CI-List for maintaining the current top- k HUIs during the search. The algorithm then scans D twice to build the initial utility-lists. Then, TKO explores the search space of top- k HUI using a procedure that we name Top-KHUI-Search. It is the combination of a novel strategy named RUC (Raising threshold by Utility of Candidates) with the HUI-Miner search procedure. During the search, TKO updates the list of current top- k HUIs in Top-K-CI-List and gradually raises the min_utilBorder threshold by the information of Top K-CI-List. When the algorithm terminates, the Top K-CI-List captures the complete set of top- k HUIs in the database.

Strategy RUC:(Raising the threshold by the Utilities of Candidates) This strategy can be incorporated with any one phase mining algorithm where itemsets are found with their utilities. It adopts the Top K-CI-List structure to maintain top K HUIs, where itemsets are sorted by descending order of utility.

Initially, Top K-CI-List is empty. When an itemset X is found by the search procedure and its utility is no less than min_util Border , X is added to Top K-CI-List. If there are more than k itemsets already in Top K-CI-List, min_util Border can be safely raised to the utility of the k -th itemset in Top K-CI-List. After that, itemsets having a utility lower. The Raised min_util Border are removed from Top K-CI-List. For each itemset X generated by the search procedure, if its utility is no less than min_util border , the proposed RUC strategy is applied to raise min_util border , RUC is performed as follows. First, X is added into Top K-CI List. Then, if $\text{EU}(X)$ is no less than min_util border and there are more than k itemsets already in Top K-CI-List, min_util Borders raised to the utility of the k -th itemset in Top K-CI-List. The remaining itemsets having a utility lower than min_util border are removed. This ensures that all and only the top K HUIs are kept. After the above process, the strategy raises the border minimum utility threshold. It continues mining itemsets that are concatenations of an itemset X if the sum of iutils and rutils of X is no less than min_util Border (line-6) . Two ordered sets $\text{Class}[X]$ and $\text{ULS}[X]$ are created to respectively store the concatenations of X and their utility-lists.

For each itemset $Y = \{y_1, y_2, \dots, y_L\}$ in $\text{Class}[P]$ $P_1 < X_1$ and $P = \{y_1, y_2, \dots, y_{L-1}\}$, we create a candidate itemset $Z = X \cup Y$ by concatenating X with y_L and uses the Construct procedure to construct utility-list of Z (i.e., $\text{ul}(Z)$) line(9-10) Then, Z and $\text{ul}(Z)$ are respectively added to $\text{Class}[X]$ and $\text{ULS}[X]$ (Line 11-12).

After processing each itemset in $\text{Class}[P]$, the Procedure Top K-HUI-Search is called with X , $\text{Class}[X]$, min_util border and Top K-CI-List to consider $(L+1)$ itemsets that are concatenations of X . This Recursive process continues until no candidate itemset is found.

Given two itemsets X and Y and their prefix P , during the search process of Top-K-HUI-Search, the utility-list of the itemset $Z = XUY$, denoted as $\text{ul}(Z)$, obtained by applying the construct procedure. The procedure is as follows; If $X = \{x_1\}$ and $Y = \{y_1\}$ are 1-itemsets, where $x_1 < y_1$.

Let $Z = XUY = \{x_1, y_1\}$ be a 2-itemset obtained by concatenating X with y_1 . The utility-lists $\text{ul}(X)$ and $\text{ul}(Y)$ are constructed during the initial database scans. The utility-list of Z is obtained by the following process.

For each transaction Tr belongs to $g(X)$ intersection $g(Y)$, an element $(Tr, EU(Z, Tr), RU(Z, Tr))$ is created in $ul(Z)$, where $EU(Z, Tr)$ is the sum of the iutil values in elements associated with Tr in $ul(X)$ and $ul(Y)$, and where $RU(Z, Tr)$ is the rutil value associated with Tr in $ul(Y)$. In brief, $EU(Z, Tr) = EU(x1, Tr) \cup EU(y1, Tr)$ and $RU(Z, Tr) = RU(y1, Tr)$. In this way by using the TKO algorithm which is one phase algorithm we are used to identify the top-k high utility itemsets.

TKO Algorithm and effective strategies:

RUZ: (Reducing estimated utility values by using Z-elements)

The RUZ strategy is applied during the generation of candidate itemsets in the Top-K-HUI-Search procedure. For any candidate X generated by the TKO algorithm, it is not necessary to explore the search space of concatenations of X if $[NZEU(X) + RU(X)]$ is less than $min_utilBorder$.

This strategy is achieved by replacing with the following code: If $[NZEU(X) + RU(X)] \geq d$.

Strategy EPB: (Exploring the most Promising Branches first)

The EPB strategy aims at generating the candidate itemsets with the highest utility first. The reason is that if itemsets with higher utility are found earlier, TKO can raise its $min_utilBorder$ higher and earlier to prune the search space. let $Class[P] = \{X_1, X_2, \dots, X_M\}$ be the set of itemsets that share the same prefix P , where the last item of X_i precedes that of X_j ($1 \leq i < j \leq M$). For each itemset X_j in $Class[P]$, let $R = \{X_{j+1}, X_{j+2}, \dots, X_M\}$ denotes the itemsets in $Class[P]$ whose last items precedes X_j . The TKO algorithm processes itemsets in R one by one in decreasing order of their estimated utility value (i.e., the sum of utility and remaining utility). The idea is to always try to extend the itemset having the largest estimated utility value first because it is more likely to generate itemsets having a higher utility and thus to allow to raise $min_utilBorder$ more quickly for pruning the search space.

5.2 DATASETS USED

In this work the following datasets are used:

Characteristics of data sets:

Table – 5.3: Characteristics of datasets

Dataset	#Trans	Avg.Length of trans	#Items	Type
Foodmart	4,141	4.4	1,559	Sparse
Mushroom	8,124	23	119	Dense

Both synthetic and real datasets were used in the experiments. Food Mart was acquired from Microsoft FoodMart 2000 database. Retail, Mushroom, Chess and Accidents datasets were obtained from the FIMI repository. Chianstore a large dataset was obtained from NU-mine-bench. Foodmart and chainstore already contain unit profits and purchase quantities. For, other datasets unit profit of items are generated between 1 and 1000 randomly. Synthetic datasets were generated from the data generator.

5.3 METRICS TO BE CALCULATED

Performance Evaluation of TKU Base and TKU:

Table – 5.4

Algorithm	Phase 1				Phase II
	PE	NU	MD	MC	SE
TKU	Y	Y	Y	Y	Y
TKU _{NoSE}	Y	Y	Y	Y	
TKU _{Base}				Y	Y

To evaluate the performance of the proposed strategies, there are three versions of TKU that respectively name TKU, TKUNoSE and TKUBase.

- **On sparse Datasets:**

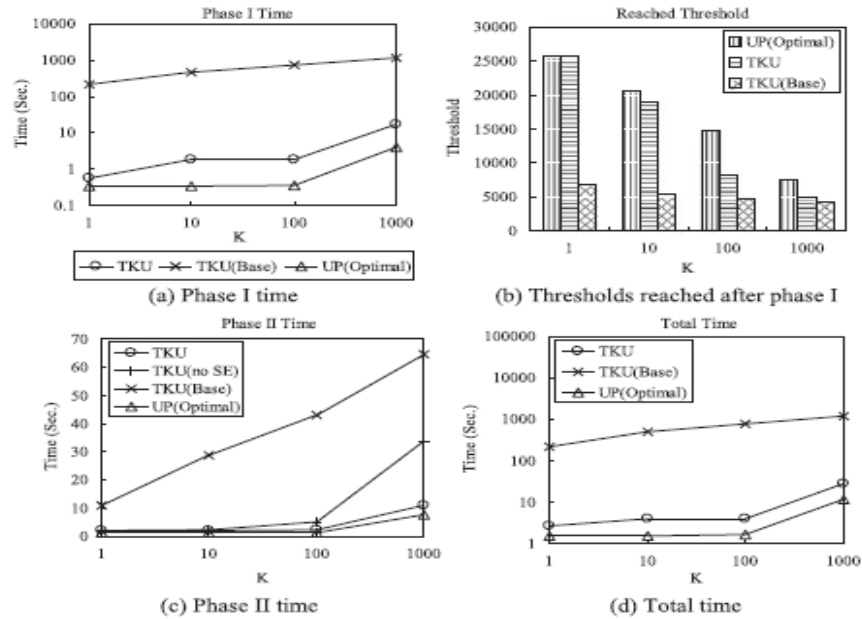


Figure – 5.1: Performance of the algorithms on Foodmart dataset

Fig. 3 shows the results on Foodmart. In Fig. 3a, the runtime of TKU for phase I is closed to that of UP(Optimal). TKUBase has the worst performance among all the algorithms. Fig. 3b shows min_utilBorder values reached by TKU and TKUBase after completing phase I, as well as the optimal threshold values used by UP-Growth. In Fig. 3b, the min_utilBorder values reached by TKU are closer to the optimal values than those reached by TKUBase. This behaviour is explained by the fact that TKUBase does not apply the strategies PE, NU and MD. Thus, it constructs a full UP-Tree using min_util border = 0. Since raising the threshold for TKUBase strictly depends on the MC strategy, its runtime is the longest. The ineffectiveness of rising the threshold for TKUBase also influences the number of candidates generated in phase I. Table 9 shows the number of candidates generated by the algorithms in phase I. In Table 9, the number of candidates produced by TKUBase is over 1,000 times larger than that of TKU when k is less than 1,000. The reason is that strategies PE, NU and MD of TKU effectively raise the threshold at different stages. Fig.3c. shows the runtime of the algorithms for Phase II. The performance of TKUNoSE is worse than TKU because the latter uses the strategy SE, which reduces the number of candidates that need to be checked in Phase II. Fig. 3d shows the overall runtimes of the algorithms. TKU is over 100 times faster than TKUBase.

On Dense datasets:

K	Mushroom		Foodmart	
	TKU	TKU _{Base}	TKU	TKU _{Base}
1	427	508,462	1,379	2,466,459
10	597,301	713,793	1,503	2,494,446
100	803,377	920,040	2,456	2,537,225
1,000	1,540,583	1,657,403	39,289	2,585,300

Table – 5.5: Number of candidates

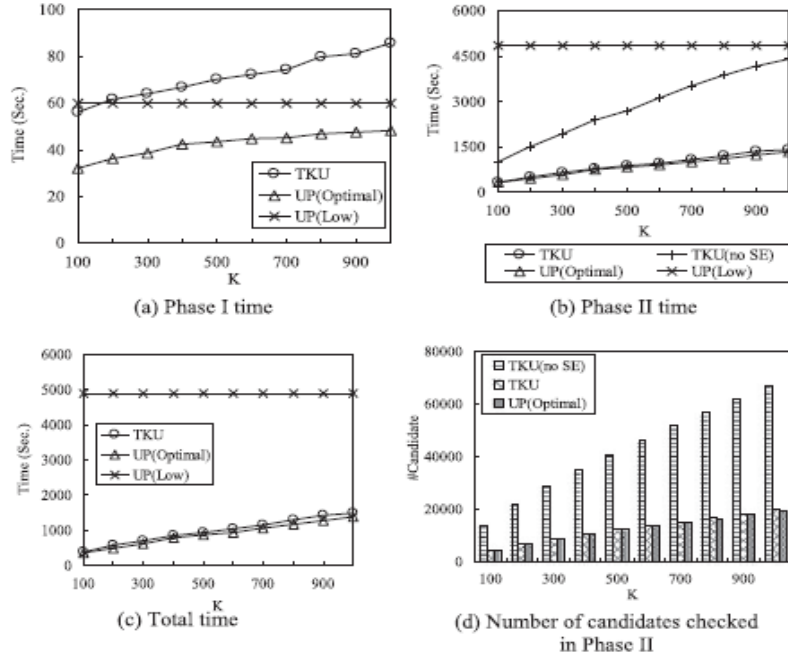


Figure – 5.2: Performance of the algorithm on Mushroom dataset

Fig. 4a shows the runtime of the algorithms for phase I. The runtime of TKU is close to that of TKUBase. This is because for dense datasets the estimated utility values of itemsets are much larger than their utilities. Thus the thresholds cannot be raised effectively in phase I. Fig. 4b shows the thresholds reached by the algorithms. In Fig. 4b, when k is larger than 1, the thresholds reached by TKU are close to that of TKUBase. Table 9 shows the number of candidates generated by the algorithms in phase I. In Table 9, we see that less candidates are produced by TKU than by TKUBase. Fig. 4c shows the runtime of the algorithms for Phase II. The runtime of TKUNoSE is the worst. This is because that, without using the SE strategy, TKUNoSE needs to check all the candidates to determine which itemsets are top- k HUIs. When k is set to 5,000, the runtime of TKUNoSE is too long to be executed (over 10,000 seconds). Fig. 4d shows the total runtime of the algorithms. In Fig. 4d, TKU is still more efficient than TKUBase.

Performance comparison of both TKU and TKO

In this we evaluate the performance of the proposed algorithms TKU and TKO.

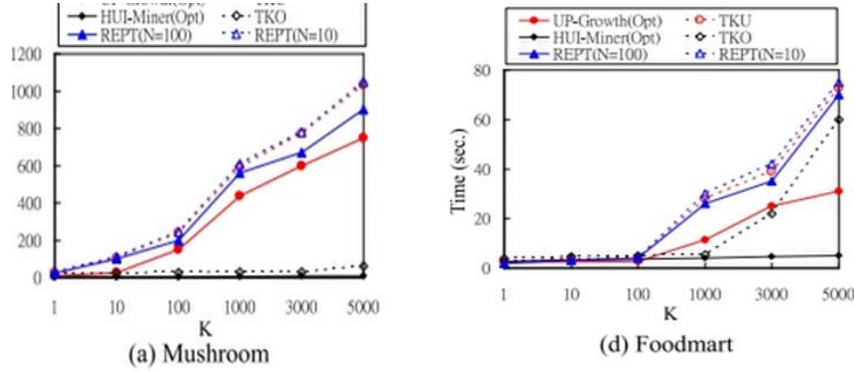


Figure – 5.3: Run-time of TKU and TKO

On Dense Datasets:

The diagrams 6a, 6b and 6c show the runtime of the algorithms of three datasets Mushroom, Chess and Accidents with varied k respectively. In these figures TKO has the best performance among top-k HUI mining algorithms. For example, on the chess dataset TKO spends only 23 seconds to complete the mining process, while TKU take more than 900 seconds. This is because TKO is one phase algorithm and TKU is two phase algorithm. Dense datasets generally contain long itemsets and transactions. TKU tend to highly overestimate the bounds of generated candidates. Whereas TKO immediately calculates the border thresholds. This avoids generating too many low utility itemsets during the mining process. The above three data sets we discussed are the dense data sets.

On sparse datasets:

The diagrams 6d, 6e and 6f show the run time of the algorithms for three datasets foodmart, retail, chainstore under varied K. In these figures, the one-phase algorithm TKO generally has the best performance. For two-phase algorithms, TKU runs slightly slower than TKO when N is set to 10. The above datasets are the sparse datasets and in both of them TKO shows the best performance.

Memory usage of algorithms:

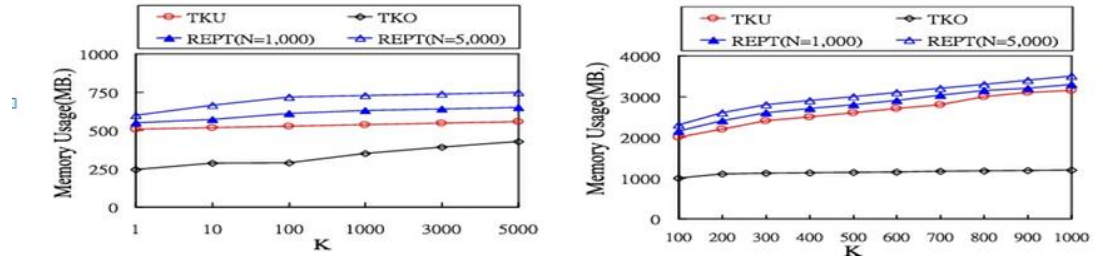


Figure – 5.4: Memory usage of TKU and TKO

The diagrams above shows the memory usage of the algorithms on datasets retail and chainstore. TKO generally uses less memory than the TKU algorithm. This is because TKU is two-phase algorithms. When they could not effectively raise the border minimum utility thresholds, they may consider too many candidates and local UP-Trees during the mining process, which causes them to consume much more memory than TKO.

Scalability of the algorithms:

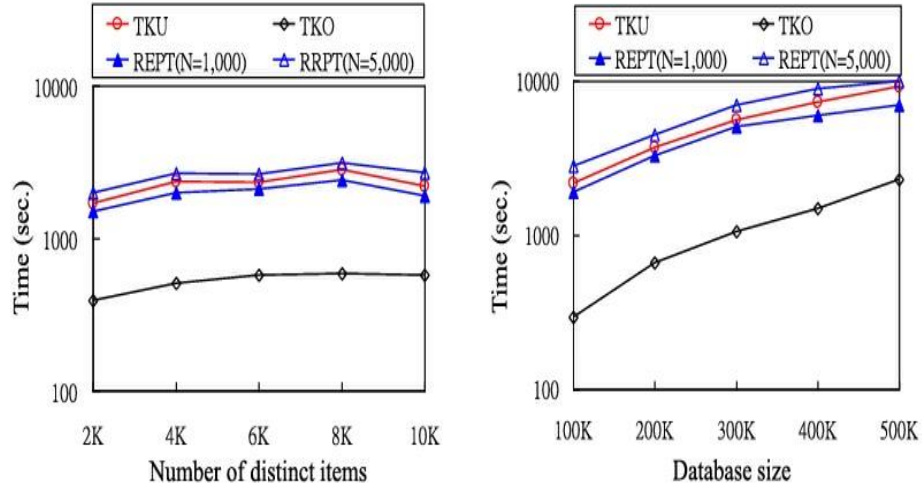


Figure – 5.6: Scalability of TKU and TKO

The scalability of the algorithms under different parameter settings has been discussed. The figure 8 shows the runtime of the algorithms on T12I8D100KQ5 when the number of distinct items is varied from 2K to 10K.

CHAPTER 6

TESTING

6.1 Testing:

Testing is a fault detection technique that tries to create failure and erroneous states in a planned way. This allows the developer to detect failures in the system before it is released to the customer. Note that this definition of testing implies that a successful test is test that identifies faults. We will use this definition throughout the definition phase. Another often used definition of testing is that “it demonstrates that faults are not present”. We will use this definition only after the system when we try to demonstrate that the delivered system fulfils the functional requirements.

If we used this second definition all the time, we would tend to select test data that have a low probability of causing the program fail. If on the other hand, the goal is to demonstrate that program has faults, we would tend to look for the test data with the higher probability of finding faults. The characteristic of good test model is that it contains test cases that identify faults. Tests should include broad range of input values, including invalid inputs and boundary cases, otherwise faults may not be detected. Unfortunately, such an approach requires extremely lengthy testing times for even small systems.

1. Test planning allocates resources and schedules the testing. This activity should occur early in the development phase s that sufficient time and work is dedicated to testing. For example, developers can design test cases as soon as the models they validate become stable.
2. Usability system tries to find faults in the user interface design the system. Often system fails to accomplish their intended purpose simply because their users are confused by their user interface and unwillingly use erroneous data.
3. Unit testing tries to find faults in participating objects and/or subsystems with respect to cases from the use case model.
4. Integration testing is the activity of finding out faults when testing the individuals tested the

components together. For example, subsystem described in the subsystem decomposition, while executing the use cases and scenarios from RAD (required analysis document).

5. Structural testing is the culmination of integration tests and structural tests exploit knowledge from the SSD (system design document) using an integration strategy described in the test plan (TP).

6. System testing tests all the components together, seen as a single system to identify faults with respect to the scenarios from the problem statement and the requirements and design goals identified in the analysis and system design, respectively.

7. Functional testing tests the requirements from RAD and the user manual.

8. Performance testing tests the non-functional requirements and additional design goals from the SSD. Functional and performance tests are done by developers.

9. Acceptance testing and installation testing check the system against the project agreement and is done by the client, if necessary, the help by the developers.

6.2 Test cases

Test Cases usually have the following components:

- Test Case Description, Specification and Test Scenario
- Initial Condition
- Steps to run the test case
- Expected behaviour/outcome

Test Case Id	Test Case Name	Test Case Desc	Test Steps			Test Case Priority	Test Status (P/F)
			Step	Expected	Actual		
Read transaction database 01	Click on Read transaction database	To Verify whether read transaction database to load the transaction data into the application or not	If not uploaded file in transaction database	Nothing will be displayed	Display the transaction table	High	P
Read profit database 02	Click on Read profit database	To Verify whether read profit database to load the profit data into the application or not	If not uploaded file in profit database	Nothing will be displayed	Display the profit table	High	P
Compute TU 03	Observe TU	To verify Computed TU or not	If you are not uploaded any database	Nothing will be displayed	It display the total or sum of TU's	High	P

Compute TWU & build UP tree 04	Observe TWU & build UP tree	Verify whether TWU & build UP tree or not	If we not generate properly in database	Nothing will be displayed	It will display sum of all the TU values of the current item in all the transactions and after view the tree also	High	P
Run TKU 05	Run TKU algorithm	Verify whether run TKU algorithm or not	If you are not run TKU algorithm	Nothing will be displayed	View the items of TWU	High	P
Run TKO 06	Run TKO algorithm	Verify whether run TKO algorithm or not	If you are not run TKO algorithm	Nothing will be displayed	View the items of TKO	High	P
Chart 07	Min utility threshold chart	To verify the graphical representation of Min utility threshold chart or not	If we are not run TWU and TKO algorithm	Nothing will be displayed	Display Minimum utility threshold comparison chart between algorithms	High	P

CHAPTER 7

RESULTS

7.1 Actual Results of The Work

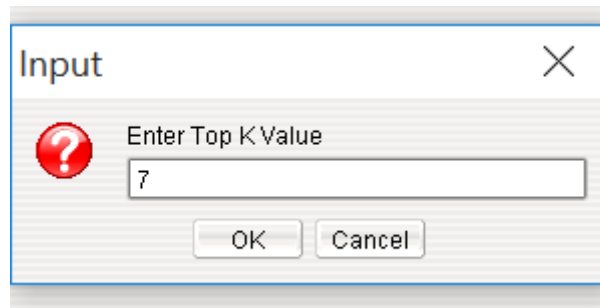
In this, first read the transactions from the database and also profit table from the database.

Next compute the transaction utility (TU) for the transactions.

Next compute the Transaction Weighted Utilization (TWU). Then run the algorithms

TKU and TKO by specifying the K value. The actual results of the algorithms are as follows.

TKU Algorithm:



A screenshot of a Windows-style input dialog box titled "Input". It contains a red question mark icon, the text "Enter Top K Value", a text input field containing the number "7", and "OK" and "Cancel" buttons at the bottom.

items		
Item Name	support count	utility value
b	224	2464
t	3196	65258
f	3196	217109
w	446	5715
l	2205	11668
n	3196	39657
b f	224	219573

TKO Algorithm:

Input

?

Enter Top K Value

10

OK

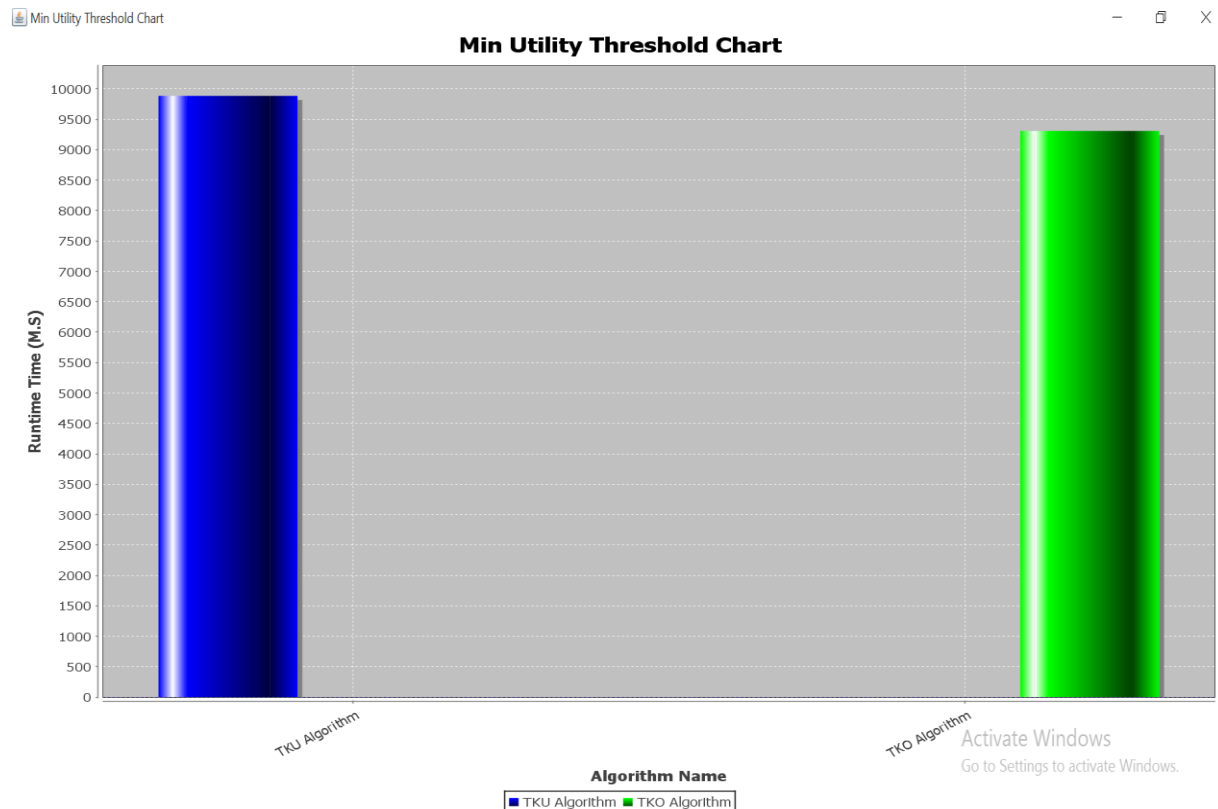
Cancel

 items

Item Name	support count	utility value
b	224	24819
t	3196	344483
f	3196	344483
w	446	50822
g	991	104758
l	2205	239725
n	3196	344483
b f	224	24819
b g	104	11339
b l	120	13480

7.2 Analysis Of The Results Obtained

In TKO the database is scanned two times. In the first scan the Transaction utility of each transaction is obtained and in the second scan the transaction weighted utilization is obtained. While TKO algorithm uses only one scan i.e. for calculating the Transaction weighted utilization. Therefore, the memory usage of TKU is more when compared to TKO and the runtime of the algorithms may also differ. Generally, TKO spends only 23 seconds on dense datasets to complete the mining process and TKU spends more than 900 seconds to complete the mining process. The comparison between runtime of the TKU and TKO algorithms on the transactional dataset taken, is shown below:



CHAPTER 8

CONCLUSION & FUTURE WORK

Top-k high utility itemsets mining, where k is the desired number of high utility itemsets to be mined is studied. Two efficient algorithms TKU (mining Top-K Utility itemsets) and TKO (mining Top-K utility itemsets in one phase) are proposed without setting minimum utility thresholds. TKU is the two-phase algorithm for mining top-k high utility itemsets, which incorporates five strategies to efficiently raise the border minimum utility threshold and further prune the search space. TKO is first one phase algorithm developed for top-k high utility mining which integrates some novel strategies to improve its performance. Empirical evaluations on different types of real and synthetic datasets show that the proposed algorithms have good scalability on large datasets and the performance of the proposed algorithms is close to the optimal case of the state-of-the art two-phase and one-phase utility mining algorithms.

Future Work:

Although a new framework for top-k HUI mining has been proposed, it has not yet been incorporated with other high utility mining tasks to discover different types of top-k high utility patterns such as top-k high utility episodes, top-k closed high utility item sets, top-k high utility web access patterns and top-k mobile high utility sequential patterns, these leave wide rooms for exploration as future work.

REFERENCES

- [1] Ahmed.C, Tanbeer.S, Jeong.B, and Lee.Y, “Efficient tree structures for high-utility pattern mining in incremental databases,”IEEE Trans. Knowl. Data Eng., vol. 21, no. 12, pp. 1708–1721, Dec.2009.
- [2] Agrawal.R and Srikant.R, “Fast algorithms for mining association rules,” in Proc. Int. Conf. Very Large Data Bases, 1994, pp. 487–499.
- [3] Chuang.K, Huang.J, and Chen.M, “Mining top-k frequent patterns in the presence of the memory constraint,” VLDB J., vol. 17,pp. 1321–1344, 2008.
- [4] Fournier-Viger.P, Wu.C, and Tseng.V.S, “Novel concise representations of high utility itemsets using generator patterns,” in Proc. Int. Conf. Adv. Data Mining Appl. Lecture Notes Comput. Sci.,2014, vol. 8933, pp. 30–43.
- [5] Fournier-Viger.P, Wu.C, and Tseng.V.S, “Mining top-k association rules,” in Proc. Int. Conf. Can. Conf. Adv. Artif. Intell., 2012, pp. 61–73.
- [6] Fournier-Viger.P and Tseng.V.S, “Mining top-k sequential rules,” in Proc. Int. Conf. Adv. Data Mining Appl., 2011, pp. 180–194.
- [7] Lan.G, Hong.T, Tseng.V.S, and Wang.S, “Applying the maximum utility measure in high utility sequential pattern mining,” Expert Syst. Appl., vol. 41, no. 11, pp. 5071–5081, 2014.
- [8] Lin.C, Hong.T, Lan.G, Wong.J, and Lin.W, “Efficient updating of discovered high-utility itemsets for transaction deletion in dynamic databases,” Adv. Eng. Informat., vol. 29, no. 1, pp. 16–27,2015.

- [9] Liu.J, Wang.K, and Fung.B, “Direct discovery of high utility itemsets without candidate generation,” in Proc. IEEE Int. Conf.Data Mining, 2012, pp. 984–989.
- [10] Liu.M and Qu.J, “Mining high utility itemsets without candidate generation,” in Proc. ACM Int. Conf. Inf. Knowl. Manag., 2012,pp. 55–64.
- [11] Quang.T, Oyanagi.S, and Yamazaki.K, “Ex Miner: An efficient algorithm for mining top-k frequent patterns,” in Proc. Int. Conf.Adv. Data Mining Appl., 2006, pp. 436 – 447.
- [12] Ryang.H and Yun.U, “Top-k high utility pattern mining with effective threshold raising Strategies,” *Knowl.-Based Syst.*, vol. 76, pp. 109–126, 2015.
- [13] Ryang.H, Yun.U, and Ryu.K, “Discovering high utility itemsets with multiple minimum supports,” *Intell. Data Anal.*, vol. 18, no. 6, pp. 1027–1047, 2014.
- [14] Shie.B, Hsiao.H, Tseng.V.S, and Yu.P.S, “Mining high utility mobile sequential patterns in mobile commerce environments,” in Proc. Int. Conf. Database Syst. Adv. Appl. Lecture Notes Comput. Sci., 2011, vol. 6587.
- [15] Tseng.V.S, Wu.C, Shie.B, and Yu.P.S, “UP-Growth: An efficient algorithm for high utility itemset mining,” in Proc. ACM SIGKDDInt. Conf. Knowl. Discovery Data Mining, 2010, pp. 253–262.
- [16] Tseng.V.S, Wu.C, Fournier-Viger.P, and Yu.P.S, “Efficient algorithms for mining the concise and lossless representation of highutility itemsets,” *IEEE Trans. Knowl. Data Eng.*, vol. 27, no. 3,pp. 726–739, Mar. 1, 2015.
- [17] Tzvetkov.P, Yan.X, and Han.J, “TSP: Mining top-k closed sequential patterns,” *Knowl. Inf. Syst.*, vol. 7, no. 4, pp. 438–457,2005.

- [18] Wu.C, Shie.B, Tseng.V.S, and Yu.P.S, “Mining top-k high utility itemsets,” in Proc. ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining, 2012.
- [19] Wu.C, Fournier-Viger.P, Yu.P.S, and Tseng.V.S, “Efficient mining of a concise and lossless representation of high utility item-sets,” in Proc. IEEE Int. Conf. Data Mining, 2011, pp. 824–833.
- [20] Xiong.H, Brodie.M, and Ma.S, “TOP-COP: Mining Top-K strongly correlated pairs in large databases,” in Proc. IEEE Int. Conf. Data Mining, 2006, pp. 1162–1166.
- [21] Yin.J, Zheng.Z, Cao.L, Song.Y, and Wei.W, “Mining top-k high utility sequential patterns,” in Proc. IEEE Int. Conf. Data Mining, 2013, pp. 1259–1264.
- [22] Yun.U and Kim.J, “A fast perturbation algorithm using tree structure for privacy preserving utility mining,” *Expert Syst. Appl.*, vol. 42, no. 3, pp. 1149–1165, 2015.
- [23] Yun.U and Ryang.H, “Incremental high utility pattern mining with static and dynamic databases,” *Appl. Intell.*, vol. 42, no. 2, pp. 323–352, 2015.
- [24] Zhu.S, Wu.J, Xiong.H, and G. Xia, “Scaling up top-k cosine similarity search,” *Data Knowl. Eng.*, vol. 70, no. 1, pp. 60–83, 2011.
- [25] Zihayat.M and An.A, “Mining top-k high utility itemsets over data streams,” *Inf. Sci.*, vol. 285, no. 20, pp. 138–161, 2014.