APPROXIMATE PARALLEL HIGH UTILITY ITEMSET MINING

YAN CHEN

A THESIS SUBMITTED TO
THE FACULTY OF GRADUATE STUDIES
IN PARTIAL FULFILMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE

GRADUATE PROGRAM IN COMPUTER SCIENCE
YORK UNIVERSITY
TORONTO, ONTARIO
OCTOBER 2015

Abstract

High utility itemset mining discovers itemsets whose utility is above a given threshold, where utilities measure the importance of itemsets. In high utility itemset mining, memory and time performance limitations cause scalability issues, when the dataset is very large. In this thesis, the problem is addressed by proposing a distributed parallel algorithm, *PHUI-Miner*, and a sampling strategy, which can be used either separately or simultaneously. *PHUI-Miner* parallelizes the state-of-the-art high utility itemset mining algorithm *HUI-Miner*. The sampling strategy investigates the required sample size of a dataset, in order to achieve a given accuracy. We also propose an approach combining sampling with *PHUI-Miner*, which provides better time performance. In our experiments, we show that *PHUI-Miner* has high performance and outperforms the state-of-the-art non-parallel algorithm. The sampling strategy achieves accuracies much higher than the guarantee. Extensive experiments are also conducted to compare the time performance of *PHUI-Miner* with and without sampling.

Acknowledgements

I would like to express my sincere gratitude to my supervisor, Prof. Aijun An, for her valuable guidance throughought my entire period of research. Her advice and support are greatly appreciated. I am also truly grateful to my committee member, Prof. Jarek Gryz, for reviewing my research and providing me his advice.

Numerous individuals provided support in different ways, which enabled me to complete this research. Special thanks to Hanwei Jin, who helped me gain the authorization to use a cluster of computers in Zhejiang University in my early stage of research. Thanks to my labmate, Morteza Zihayat, from whom I learned a lot including essential research and writing skills. I am grateful to my friends, Jing Zhang and Heidar Davoudi, for their help in solving an issue in this thesis. They both provided me with significant afflatuses. I would also like to thank Ricky Fok and Yingying Zhang for their support.

Financial support from York University is also heartily acknowledged.

Last but not least, I want to thank my family and friends for their understanding

and support. I'm specially thankful to my girlfriend, Yaying Chen, for her patience and encouragement while I took the time to complete my research.

Table of Contents

List of Tables

List of Figures

1 Introduction

Frequent pattern mining has been an important topic since the concept of frequent itemsets was first introduced by Agrawal et al [6]. Given a dataset of transactions, frequent pattern mining finds the itemsets whose support (i.e. the percentage of transactions containing the itemset) is no less than a given minimum support threshold. However, neither the number of occurrences of an item in a transaction, nor the importance of an item, is considered in frequent pattern mining. Itemsets with more occurrences or importance may be more interesting to users, since they may bring more profit.

In light of this, high utility itemset mining has been studied [9, 15, 42, 35]. In high utility itemset mining, the term utility refers to the importance of an itemset; e.g., the total profit the itemset brings. An itemset is a *High Utility Itemset* (HUI) if the utility of the itemset is no less than a given minimum threshold. High utility itemset mining focuses more on the utility values in the dataset, which are usually related to profits for the business. Such utilities are interesting to the business

owners, who could gain more profits from them. For example, supermarkets use frequent itemset mining to find merchandises customers usually buy together, so as to make recommendations to customers. However, with high utility itemset mining, supermarkets will be able to recommend not only the merchandises people usually buy together, but also the merchandises which will lead to more profits for the store.[1]

Most of the frequent pattern mining algorithms prune off itemsets in an early stage based on the popular *Apriori* property [8]: every sub-pattern of a frequent pattern must be frequent (also called the *downward closure property*). However, this property does not hold in high utility itemset mining, which makes mining high utility itemsets more challenging. The state-of-the-art approaches achieve good performance when the dataset is relatively small. However, the volume of data can grow so faster than expected, that a single machine may not be able to handle a very large amount of data.

One option to solve the problem of large volumes of data is to use parallel distributed computing techniques. The MapReduce framework [18] (e.g., Hadoop) has been a popular solution recently, which enables scalable and fault-tolerant distributed processing of huge data on large clusters. Applications in the MapReduce framework have to conform the protocols of *mapper* and *reducer* as a disk-based

[1]In Section 6.3, another example will be given to show a real world application of high utility itemset mining for news recommendation.

paradigm, which restricts the flexibility as well as the performance of the algorithm. Spark is also a distributed computing framework, which is memory-based, and thus provides performance up to 100 times faster than Hadoop for certain applications [44]. Spark uses Resilient Distributed Dataset (RDD), which is a distributed memory abstraction, for in-memory computation of data, allowing efficient reuse of data.

For very large datasets, obtaining exact results is sometimes infeasible. Recent studies focus on mining an approximate set of frequent itemsets. In most cases, approximate solutions may already be satisfactory to users. In general, approximation methods can be divided into two categories: *pattern compressing* [13, 34, 12, 43] and *sampling* [40, 37, 49, 47]. Sampling is a method that mines approximate results from a sample of the entire dataset. The most important step in sampling is to decide the size of the sample we need in order to obtain a certain accuracy, which is also the focus of our sampling strategy proposed in this thesis.

In this thesis, we address the problem of high utility itemset mining by proposing *PHUI-Miner* (Parallel High Utility Itemset Miner) and a sampling strategy. *PHUI-Miner* is a parallel distributed algorithm, which parallelizes *HUI-Miner*, a state-of-the-art algorithm for high utility itemset mining. The sampling strategy provides the required sample size for a dataset in order to achieve a given accuracy. It can be used together with any exact high utility itemset mining algorithm. To the best of our knowledge, this is the first piece of work to utilize sampling in high utility

itemset mining. Our contributions are summarized as follows:

- *PHUI-Miner*, a parallel distributed algorithm, is proposed for parallel mining of high utility itemsets without sampling, which could lead to exact results.
- We propose and prove a new theorem, which shows the relationship between the high utility itemset mining results from the whole dataset, and those from a sample of it. The theorem leads to a sampling method with theoretical guarantees on the probability that an HUI can be returned and on the utility of a returned itemset. A feature of this sampling method is that the sample size required to achieve the theoretical guarantees is independent of the size of the original data, and is thus not necessarily going up as the data set grows.
- We also propose *PHUI-Miner with sampling*, an approach combining sampling with *PHUI-Miner*, which mines an approximate set of high utility itemsets, but achieves better performances.
- Extensive experiments are conducted and the time performance and scalability of *PHUI-Miner* are evaluated. *PHUI-Miner* is demonstrated to outperform the state-of-the-art non-parallel high utility itemset mining algorithm *HUI-Miner*. The time performance of *PHUI-Miner with sampling* is also evaluated, which is shown to be better than using *PHUI-Miner* alone. Furthermore, the accuracy of the sampling strategy is evaluated with several datasets and

different parameters. Our results show that our sampling strategy achieves accuracy even higher than the expectations based on our theoretical analysis.

The thesis is organized as follows. Chapter 2 is a literature survey of related work. Chapter 3 introduces relevant definitions and a problem statement. Chapter 4 presents *PHUI-Miner*. Chapter 5 describes the sampling strategy and *PHUI-Miner with sampling*. We show experimental results in Chapter 6, and conclude the thesis in Chapter 7.

2 Related Work

Before the problem of high utility itemset mining was first proposed by Yao et al. [46], a variation of the problem, named share frequent itemsets mining, was studied by many researchers. Several algorithms have been proposed: e.g., ZP [11], ZSP [11], FSH [31], ShFSH [31], and DCG [30]. These algorithms can be used to mine high utility itemsets. However, they all use the Apriori [7] like strategy, which results in the problem of repeated database scans and large numbers of candidates. To improve the performance of these algorithms, Liu et al. proposed Two-phase [36], which uses an important utility measure, named *Transaction Weighted Utility* (TWU), for pruning the search space, since the *downward closure property* is not applicable in high utility itemset mining. Afterwards, another pruning strategy, called the *isolated items discarding strategy* (IIDS), was proposed in FUM [32] and DCG+ [32]. The number of candidates are largely reduced by these pruning strategies. However, the problem of repeated database scans is still not solved. An algorithms based on FP-Growth algorithm [22] have been proposed to mine

high utility itemsets with at most three scans of database, and thus have better performance. Examples of these algorithms include IHUP [9], HUC-Prune [10], UP-Growth [42], UP-Growth+ [41]. However, the candidate itemsets are still too many compared to the high utility itemsets. HUI-Miner [35] is one of the recent efficient algorithms proposed by Liu et al. demonstrated to have an order of magnitude better performance than other algorithms.

Parallel distributed algorithms solve the problem of mining massive datasets. Several studies [17, 28, 45, 27, 20, 23] have been done for mining frequent patterns in distributed environments, inspired by the MapReduce framework proposed by Google [18]. Some of them [17, 28, 45] use a naive approach which computes the support of every itemset in the dataset in a single MapReduce round, resulting in huge data replication. An adaption of FP-Growth algorithm to MapReduce, called PFP [27], is a more sophisticated approach. Given a minimum frequency threshold, PFP first applies a parallel and distributed counting approach to compute the frequent items. The frequent items are then partitioned into groups randomly. Subsequently, the dataset is used to generate group-dependent transactions, which are sent to reducers. Finally, the reducers use an FP-Growth like approach to generate group-dependent frequent itemsets. However, very few studies have been conducted on high utility itemset mining with distributed computing techniques so far.

As the volume of data grows, the mining task consumes more and more time. Mannila et al. [38] first suggested that sampling can be used to efficiently obtain association rules. Then Toivonen [40] presented a sampling algorithm, which builds a complete set of association rules with a probability depending on the sample size. The Chernoff bound and the union bound are used, in which the Bernoulli random variable refers to whether an itemset appears in a transaction. A number of previous works [49, 25, 51, 29, 50, 14, 16] have been focusing on improving the bound of the sample size using different techniques in association rules mining or frequent pattern mining. Sampling techniques in high utility itemset mining are more complicated since an itemset has a utility value in each transaction, instead of 0 or 1 in frequent pattern mining. There has hitherto been little research on using sampling in high utility itemset mining.

3 Preliminaries

3.1 Distributed Computing Frameworks

In data mining and other fields which require analyzing and extracting information from data, the hardware restricts the size of the data we are able to process. CPU, memory and data storage are three different kinds of resources which affect the overall scalability of algorithms. These resources are limited, so it is very hard to process a very large dataset which usually exceeds the capacity of the resources. Distributed computing frameworks solve this problem by using a cluster of computers, connected by a network, to perform computing tasks in parallel.

The most commonly known distributed computing framework is Apache Hadoop [1]. Apache Hadoop provides reliable, scalable, and distributed computing solution, which is used by many companies, including Yahoo! and Facebook.

There are two main parts in the core of Apache Hadoop, the storage part and the processing part. The storage part, also known as Hadoop Distributed File System (HDFS), stores data by splitting them into blocks and distributing them

amongst different nodes in the cluster. Each block of a file is usually replicated, and stored in several different nodes, so that data loss in HDFS is very rare in case of hardware failure. The processing part, also known as MapReduce, uses two procedures, map and reduce, for parallel processing of data. The map and reduce procedures are called mappers and reducers respectively. In mappers, a set of data is converted into tuples (key-value pairs), while reducers take output from mappers and combine tuples into smaller sets of tuples, by aggregating tuples with the same key into a single tuple. HDFS and MapReduce are inspired by the ideas proposed by Google based on the Google File System (GFS) and their proprietary MapReduce technology.

However, there are some drawbacks of Apache Hadoop, which limits the performance and flexibility of the algorithms implemented on it. On the one hand, the MapReduce paradigm requires that each mapper is followed by a reducer, and they must be programmed in a strictly pre-defined way. On the other hand, each pair of mapper and reducer in Apache Hadoop has to read data from disks, and write results back to disks, which results in a bottleneck in its performance.

In order to deal with these two drawbacks of Apache Hadoop, another distributed computing framework, Apache Spark, was developed. Instead of the two-stage disk-based MapReduce paradigm introduced in Apache Hadoop, Apache Spark uses a data abstraction, known as Resilient Distributed Datasets (RDD).

RDDs are read-only, partitioned collection of records, which are created by reading from data storages or transforming from other RDDs. [48] An RDD holds references to Partition objects, where each Partition object is a subset of the dataset represented by this RDD. RDDs are usually not in materialized form. Instead, if an RDD A is transformed from another RDD B, we only need the information of the transformation and the RDD B, in order to derive the RDD A. As a result, RDDs are only materialized when they are asked to perform a reduce operation, which aggregates data in different nodes to a single machine. Apache Spark loads data into the memories of machines in a cluster as RDDs, and uses them repeatedly for data processing tasks. Apache Spark also allows programmers to have arbitrary mappers and reducers in any order, providing a much more flexible API for its users. In an Apache Spark cluster, there is one $Master$ node and several $Worker$ nodes. The $Master$ node is responsible for allocating resources and assigning tasks to $Worker$ nodes. However, Apache Spark is only an alternative for MapReduce in Apache Hadoop. HDFS is still a state-of-the-art open source distributed data storage framework. Apache Spark has interfaces with different types of data storage, including HDFS, Cassandra [2], OpenStack Swift [3], etc. Apache Spark is able to read from these types of data storage for data processing, which makes Spark more popular. Therefore, in this thesis, Apache Spark is used as the main platform for our proposed algorithms, while HDFS is used as our data storage.

TID	Transaction	Transaction Utility
T_1	(a, 1) (c, 2) (d, 1) (g, 1)	11
T_2	(a, 2) (b, 5) (c, 3) (d, 1) (e, 2)	30
T_3	(b, 4) (c, 3) (d, 1)	16
T_4	(c, 2) (f, 1)	3
T_5	(b, 3) (c, 4) (e, 2) (g, 2)	18

(a) Transaction Dataset

Item	a	b	c	d	e	f	g
External Utility	3	2	1	5	3	1	1

(b) External Utilities

Table 3.1: An Example Transaction Database with External Utilities

3.2 High Utility Itemset Mining

Let $I^* = \{I_1, I_2, ..., I_m\}$ be a set of items. An itemset X is a set of items $\{I_{e_1}, I_{e_2}, ..., I_{e_Z}\}$, where Z is the length of X, denoted by $|X|$. A dataset D is a list of transactions $\{T_1, T_2, ..., T_n\}$, where each transaction $T_d \in D$ is an itemset.

Definition 1. (**Internal utility and external utility**) In high utility itemset mining, each item $I \in I^*$ is associated with a positive value $p(I)$, called its external utility (e.g., item profit). Each item I in transaction $T_d \in D$ is also associated with

a positive value $q(I, T_d)$, called its internal utility (e.g. purchase quantity). For example, in Table 3.1, $p(b) = 2$ and $q(b, T_2) = 5$.

Definition 2. (**Utility of an item I in transaction T_d**) Given $I \in T_d$, the utility of item I in transaction T_d is defined as $u(I, T_d) = p(I)q(I, T_d)$. For example, in Table 3.1, $u(b, T_2) = p(b)q(b, T_2) = 10$.

Definition 3. (**Utility of an itemset X in transaction T_d**) The utility of itemset X in transaction T_d is defined as $u(X, T_d) = \sum_{I \in X} u(I, T_d)$, if $X \subseteq T_d$. Otherwise, $u(X, T_d) = 0$. For example, in Table 3.1, $u(\{b, c\}, T_2) = u(b, T_2) + u(c, T_2) = 10 + 3 = 13$.

Definition 4. (**Utility of an itemset X in dataset D**) The utility of itemset X in dataset D is defined as $u_D(X) = \sum_{T_d \in D} u(X, T_d)$. For example, in Table 3.1, $u_D(\{b, c\}) = u(\{b, c\}, T_2) + u(\{b, c\}, T_3) + u(\{b, c\}, T_5) = 13 + 11 + 10 = 34$.

Definition 5. (**Utility of a transaction T_d**) The utility of transaction T_d is defined as $u(T_d) = \sum_{I \in T_d} u(I, T_d)$. For example, in Table 3.1, $u(T_4) = u(c, T_4) + u(f, T_4) = 2 + 1 = 3$.

Definition 6. (**Transaction weighted utilization of an itemset X in dataset D**) The transaction weighted utilization (TWU) of an itemset X in dataset D is defined as $twu(X) = \sum_{X \subseteq T_d, T_d \in D} u(T_d)$. For example, in Table 3.1, $twu(\{b, c\}) = u(T_2) + u(T_3) + u(T_5) = 30 + 16 + 18 = 64$.

Itemset	{a}	{b}	{c}	{d}	{e}	{f}	{g}
TWU	41	64	78	57	48	3	29

Table 3.2: Transaction Weighted Utilities (TWUs) for the Example Database

Transaction weighted utilization has *downward closure property*, which means for any itemset X and utility threshold θ, if $twu(X) < \theta$, the utility of any superset of X is lower than θ. For example, since $twu(\{b, c\}) = 64$, any superset of X will have utility lower than 64. Table 3.2 shows the TWU values for each item in the example database.

Definition 7. (**Total utility of a dataset** D) The total utility of dataset D is defined as $U_D = \sum_{T_d \in D} u(T_d)$. For example, in Table 3.1, $U_D = u(T_1) + u(T_2) + ... + U(T_5) = 84$.

Definition 8. (**Relative utility of an itemset** X **in a dataset** D) The relative utility of an itemset X in dataset D is defined as $\frac{u_D(X)}{U_D}$.

Definition 9. (**High utility itemset**) An itemset X is a *high utility itemset* (HUI) in dataset D, iff $u_D(X)$ is no less than θU_D, where θ is a user specified minimum relative utility threshold.

Given a dataset D and a user specified minimum relative utility threshold θ, the problem of high utility itemset mining is to discover all the high utility itemsets in

D.

However, mining all the high utility itemsets from a very large dataset is very time and memory consuming. Distributed computing framework and sampling based algorithms are, thus, more suitable to this task. In this thesis, distributed computing framework and sampling are both utilized in our proposed algorithms.

4 PHUI-Miner: Parallel High Utility Itemset Mining

In this chapter, we propose a parallel high utility itemset mining algorithm, named *PHUI-Miner* (Parallel High Utility Itemset Miner), which parallelizes the state-of-the-art high utility itemset mining algorithm *HUI-Miner* [35]. *PHUI-Miner* is proposed to mine exact results from a dataset, based on Apache Spark. *PHUI-Miner* adopts a way to split the search space, which is inspired by PFP [27] from Google.

Below, the *HUI-Miner* algorithm will be reviewed first, so that the reader can better understand our proposed approach. And then, our novel parallel distributed algorithm *PHUI-Miner* is elaborated.

4.1 Review of HUI-Miner

HUI-Miner mines high utility itemsets without candidate generation. It utilizes a utility-list structure to store the utility information of a database. Constructing

TID	Transaction
T_1	(a, 3) (d, 5) (c, 2)
T_2	(a, 6) (e, 6) (d, 5) (b, 10) (c, 3)
T_3	(d, 5) (b, 8) (c, 3)
T_4	(c, 2)
T_5	(e, 3) (b, 6) (c, 4)

Table 4.1: Revised Transactions

the initial utility-lists needs two database scans. The first scan of database accumulates the TWU values for all the items. During the second database scan, the unpromising items are filtered, and the rest of the items in all the transactions are sorted according to their TWU, in ascending order. The filtered and sorted transactions are called revised transactions. Simultaneously, the initial utility-lists are constructed. The structure of utility-lists is explained later in this section.

For example, in the example database in Table 3.1, if the utility threshold is 30, f and g are unpromising items since their TWU values are less than 30. The rest of the items are sorted according to their TWU ascending order: $a < e < d < b < c$. The revised transactions of the example database are shown in Table 4.1. In the utility-lists, each utility-list of itemset X has a list of elements, where each element contains three fields: *tid*, *iutil* and *rutil*. [35]

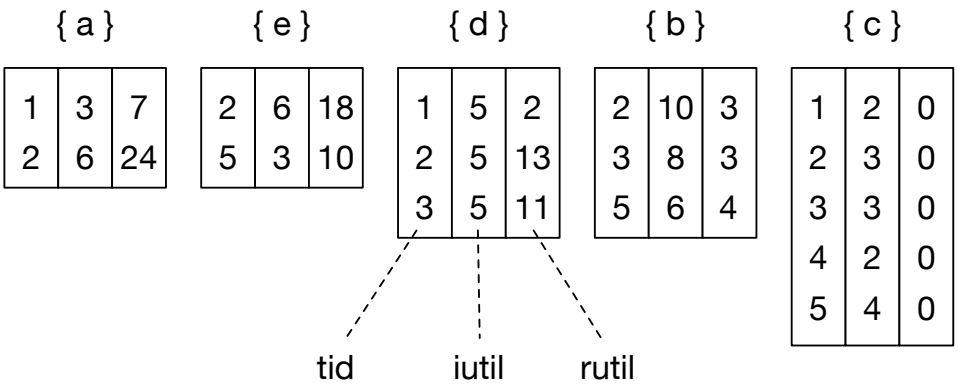


Figure 4.1: Initial Utility-Lists

- *tid* is the transaction ID of T containing X.
- *iutil* is the utility of X in T.
- *rutil* is the sum of utilities of all the items after X in T.

Figure 4.1 shows the initial utility-lists constructed by the second database scan.

Then utility-lists of k-itemsets are constructed from utility-lists of (k-1)-itemsets and (k-2)-itemsets recursively. The utility-list of itemset Pxy is constructed from utility-lists of itemsets P, Px and Py, where P is an itemset, while x and y are items. For example, to construct the utility-list of itemset *edbc*, the utility-lists of itemsets *ed*, *edb* and *edc* are needed. In the case of $k = 2$, the utility-lists of 2-itemsets are constructed from utility-lists of 1-itemsets and 0-itemsets. Since 0-itemsets are empty itemsets, the utility-lists of 0-itemsets are defined to be empty too in *HUI-Miner*. Algorithm 1 [35] shows the procedure in constructing a utility-

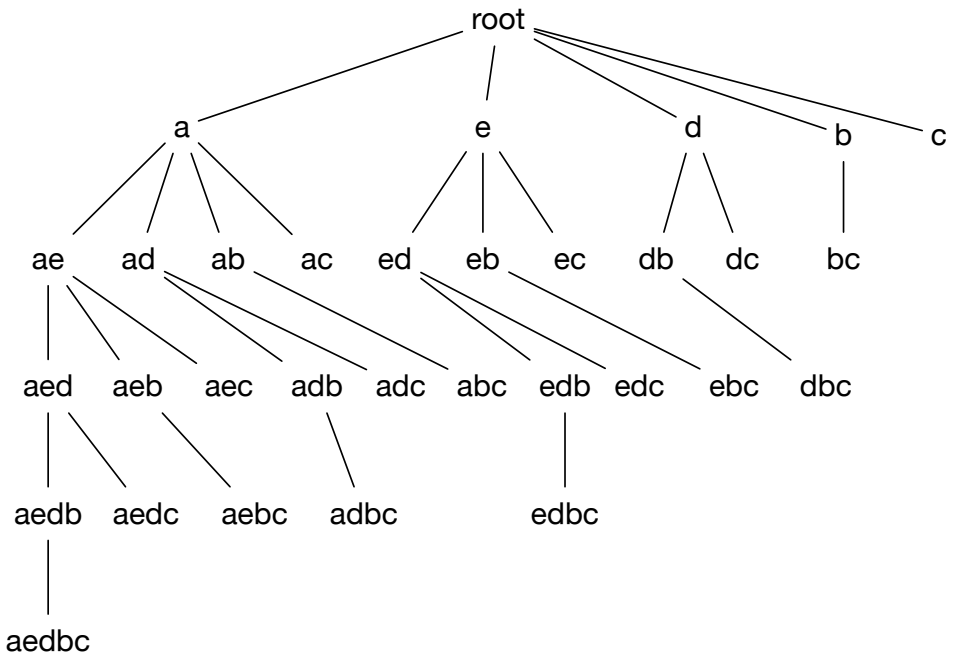


Figure 4.2: Search Space

list of a k-itemset.

The search space of high utility itemset mining can be represented as a set-enumeration tree [39]. In the tree, each node is an itemset. Given a list of items sorted in their TWU ascending order, the children of the root node is all the items. The other nodes in the tree are generated by appending an item to the itemset X in the parent node. The item is from the siblings of the parent node whose itemsets are the same as X except for the last item. The last items of those siblings are appended to X as the children of the parent node. For example, given five items with TWU ascending order $a < e < d < b < c$, the set-enumeration tree is

depicted in Figure 4.2 [35]. *HUI-Miner* mines HUIs recursively, using a depth first search in the search tree. *HUI-Miner* also prunes subtrees of the search space if it determines that all the itemsets in the subtrees are unpromising based on some criterion. Algorithm 2 [35] shows the procedure of *HUI-Miner*.

4.2 PHUI-Miner

PHUI-Miner is a distributed algorithm, which parallelizes *HUI-Miner*. In *PHUI-Miner*, the search space is divided and assigned to each node in a cluster. Each node is only responsible for mining the assigned search space, which in another word, splits the workload into different nodes in the cluster.

Figure 4.3 is the data flow of *PHUI-Miner*. Given a transaction dataset D, *PHUI-Miner* first reads the dataset from *HDFS* to different nodes. The dataset is stored in *HDFS* in a distributed way, that the dataset is split into blocks, where every block has a fixed size, defined in the configuration file of *HDFS*, except the last block. The blocks are usually replicated to ensure reliability. The dataset is read from *HDFS* in a unit of block, and the blocks will be as evenly distributed in different nodes as possible. Also, the blocks will be assigned to their local nodes if possible, to lower the communication cost. Each node stores a part of D. Then, $itemTWU$ list is built, which contains the TWU values of each item in the whole dataset. $itemTWU$ is used to revise the transactions, which prunes the unpromising

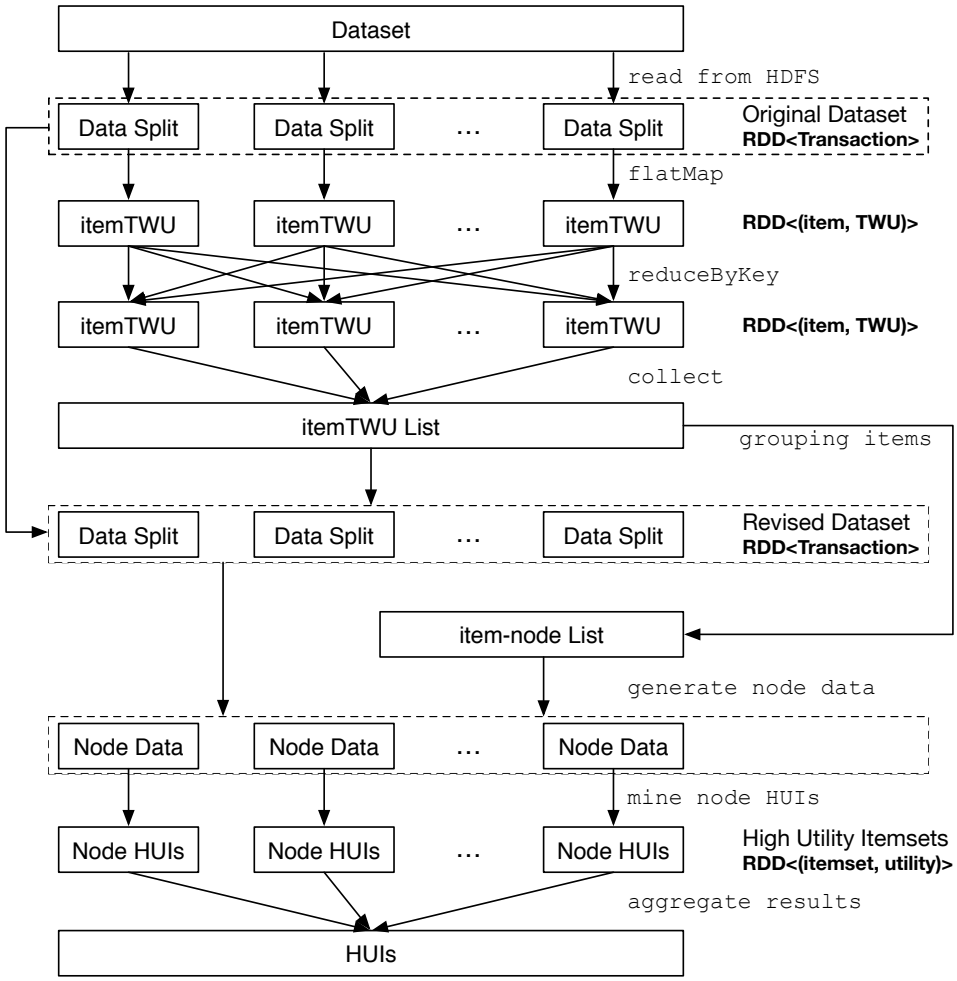


Figure 4.3: Data Flow of PHUI-Miner

items and sorts the items in ascending order according to their TWU values in all the transactions. $itemTWU$ is also used to generate an item-node list, which assigns each promising item to a node in the cluster, as described in Section 4.2.1. The item-node list is also required for generating transactions required for each node to mine their assigned search space, referred to as node data. The details of generating node data are described in Section 4.2.2. Then each node mines its node data in its assigned search space for node HUIs, as described in Section 4.2.3. Finally, node HUIs in all the nodes are aggregated directly for final results.

4.2.1 Dividing the Search Space

In *PHUI-Miner*, we use a divide and conquer strategy that divides a big task into smaller sub-tasks. In another word, the search space is split into sub-spaces. For example, in Figure 4.2, the list of items is a, e, d, b, c in TWU-ascending order. Based on this list, we divide the itemsets to be mined into the itemsets containing a, the itemsets containing e but no a, the itemsets containing d but no a or e, and so on.

In *PHUI-Miner*, each node is assigned one or more sub-tasks. For example, in Figure 4.2, if there are 2 nodes in the cluster, the items are divided into 2 groups and assigned to different nodes. Assuming items a, e, d are assigned to node 1, and b, c are assigned to node 2, node 1 will be responsible for mining all

the itemsets containing item a, the itemsets containing item e but no a and the itemsets containing item d but no a or e, while node 2 will be responsible for mining itemsets containing item b but no a, e or d and itemsets containing item c but no a, e, d or b.

The way of assigning the items to nodes affects the time performance of our algorithm, since the numbers of itemsets in different sub-spaces are different. For example, in Figure 4.2, there are $16, 8, 4, 2, 1$ itemsets in the sub-spaces a, e, d, b, c respectively. However, due to pruning in *HUI-Miner* and the fact that the items are sorted according to their TWU values, the difference of the numbers of itemsets is not as big as shown in the search space.

To split the workload to different nodes more evenly, we designed an approach, which makes the assignment of items more balanced. Suppose there are N nodes in the cluster with node id $1, 2, ..., N$, the items sorted according to their TWU ascending order are assigned one by one to nodes $1, 2, ..., N$, and then nodes $N, N-1, ..., 1$, etc. For example, if the sorted items are a, e, d, b, c and we have 2 nodes in the cluster, items a, e are assigned to nodes $1, 2$ respectively. And then, items d, b are assigned to nodes $2, 1$ and item c is assigned to node 1. So finally, items a, b, c are assigned to node 1 and items e, d are assigned to node 2. We do not guarantee that this assignment is the optimal solution. However, this approach is demonstrated to be better than random assignment in most cases according to our

experiments in Chapter 6. If in the example above, the items d, b are assigned to nodes $1, 2$ instead. The subspaces assigned to node 1 will always be much bigger than the ones assigned to node 2. So our way of assigning the subspaces is also intuitively more balanced. Algorithm 3 depicts the procedure of assigning items to different nodes. The output of this procedure is a hashmap, which maps items to it assigned nodes, called *inlist*.

We also try to further improve the balance of workloads by splitting the search space, so that each node mines itemsets with a prefix of depth d, where $d > 1$. However, in this approach, the node data generating phase requires a lot of pattern matching, which could largely reduce the time performance. Also, the itemsets with length smaller than d need to be mined in another way. We designed an algorithm which replicates the whole dataset to all the nodes, instead of generating node data for them. We conducted several experiments and found that this approach is slower than *PHUI-Miner*, but faster than *HUI-Miner* on a single machine.

4.2.2 Generating Node Data

The data for each node is generated such that, if an item is assigned to a node, the exact utility of the itemsets beginning with the item can be mined with only the data assigned to the node. So if a transaction does not contain any of the assigned items, the transaction is not included in the data for the node. If a transaction

contains some of the assigned items, only a subset of the transaction, starting from the leftmost item which is assigned to the node to the end, are included in the data for the node.

Algorithm 4 is the procedure of generating the node data. The procedure utilizes a *flatMap* operation in Spark. The *flatMap* operation allows mapping a value to 0 or more key-value pairs. For every revised transaction, it is scanned from the beginning to the end. For each item in the transaction, the node id *nid* of the item is found in *inlist*. If *nid* has not been output, the procedure outputs a key-value pair of $<nid, T>$, where T is a subset of the transaction, starting from the item to the end. The output RDD of the *flatMap* operation is then grouped according to the key values and repartitioned to each node.

4.2.3 Mining Node Data

After the node data is generated, each node mines its node data in its assigned search space for node HUIs. Specifically, each node mines HUIs beginning with the items assigned to it. Algorithm 5 illustrates the mining process. This process is very similar to Algorithm 2, except that Algorithm 5 checks whether the utility-list should be generated in Line 3.

Before we designed the algorithm *PHUI-Miner*, we also designed an algorithm parallelizing *UP-Growth* [42]. The algorithm is the same as *PHUI-Miner* except for

the *Mining Node Data* phase. In parallelized *UP-Growth*, each node in the cluster mines itemsets starting with specific items by constructing local UP-Tree [42] for those items. After discovering all the potential high utility itemsets, another step is taken to calculate the exact utility values of all the potential HUIs. However, this approach on a cluster of 20 machines is much slower than *HUI-Miner* on a single machine. As a result, we do not introduce this approach in detail in this thesis.

Algorithm 1 Constructing Utility-List for K-Itemset (Taken from [35])

Input: $P.UL$, the utility-list of itemset P

$Px.UL$, the utility-list of itemset Px

$Py.UL$, the utility-list of itemset Py

Output: $Pxy.UL$, the utility-list of itemset Pxy

1: **procedure** CONSTRUCT
2: $Pxy.UL = NULL$
3: **for** element $Ex \in Px.UL$ **do**
4: **if** $\exists Ey \in Py.UL$ and $Ex.tid == Ey.tid$ **then**
5: **if** $P.UL$ is not empty **then**
6: search such element $E \in P.UL$ that $E.tid == Ex.tid$
7: $Exy =< Ex.tid, Ex.iutil + Ey.iutil - E.iutil, Ey.rutil >$
8: **else**
9: $Exy =< Ex.tid, Ex.iutil + Ey.iutil, Ey.rutil >$
10: **end if**
11: append Exy to $Pxy.UL$
12: **end if**
13: **end for**
14: **return** $Pxy.UL$
15: **end procedure**

Algorithm 2 HUI-Miner (Taken from [35])

Input: $P.UL$, the utility-list of itemset P, initially empty

ULs, the set of utility-lists of all P's 1-extensions

$minUtil$, the minimum utility threshold

Output: all the HUIs with P as prefix

1: **procedure** HUI-MINER
2: **for** utility-list $X \in ULs$ **do**
3: **if** $SUM(X.iutil) \geq minutil$ **then**
4: output the extension associated with X
5: **end if**
6: **if** $SUM(X.iutil) + SUM(X.rutil) \geq minutil$ **then**
7: $exULs = NULL$
8: **for** utility-list Y after X in ULs **do**
9: $exULs = exULs + Construct(P.UL, X, Y)$
10: **end for**
11: HUI-Miner(X, exULs, minutil)
12: **end if**
13: **end for**
14: **end procedure**

Algorithm 3 Divide Search Space

Input: *items*, items sorted according to their TWU ascending order

N, the number of nodes

Output: *inlist*, a hashmap which maps items to nodes

```
1:  procedure
2:      mp ← empty hashmap
3:      node ← 1
4:      inc ← 1
5:      flag ← false
6:      for x in items do
7:          mp[x] ← node
8:          node ← node + inc
9:          if node is the first or last node of all the nodes then
10:             if flag then
11:                 flag ← false
12:                 if node is the first one then
13:                     inc ← 1
```

```
14:                 else
15:                     inc ← -1
16:                 end if
17:             else
18:                 flag ← true
19:                 inc ← 0
20:             end if
21:         end if
22:     end for
23:     return mp
24: end procedure
```

Algorithm 4 Generating Node Data

Input: T_i, transaction

inlist, the hashmap which maps items to nodes

Output: 0 or more <node id, transaction>

```
1: procedure FLATMAP
2:     added ← empty set
3:     for j ← 0 to |T_i| − 1 do
4:         find item T_i[j] in inlist and get node id nid
5:         if nid is not in added then
6:             add nid to added
7:             T ← subset of T_i from j to end
8:             output <nid, T>
9:         end if
10:    end for
11: end procedure
```

Algorithm 5 Mining Node Data

Input: $P.UL$, the utility-list of itemset P, initially empty

ULs, the set of utility-lists of all P's 1-extensions

$minUtil$, the minimum utility threshold

$inlist$, the item-node list

nid, the node id of the current node

Output: all the HUIs with P as prefix

```
1: procedure NODE-MINER
2:     for utility-list X ∈ ULs do
3:         if P is not empty or inlist(X.firstItem) == nid then
4:             if SUM(X.iutil) ≥ minutil then
5:                 output the extension associated with X
6:             end if
```

```
7:            if SUM(X.iutil) + SUM(X.rutil) ≥ minutil then
8:                exULs = NULL
9:                for utility-list Y after X in ULs do
10:                   exULs = exULs + Construct(P.UL, X, Y)
11:               end for
12:               Node-Miner(X, exULs, minutil, inlist, nid)
13:           end if
14:       end if
15:   end for
16: end procedure
```

5 Sampling: Approximate High Utility Itemset Mining

In the previous chapter, a parallel distributed algorithm was proposed, which splits the search space, and mines HUIs parallelly with a cluster of nodes. However, when the dataset gets bigger, the running time could sometimes be unacceptably long. One option to solve this problem is to only get an approximate set of results, instead of the exact results. In most cases, approximation results are already satisfactory to business owners.

In this chapter, a sampling strategy is proposed which determines the required sample size in order to achieve a given accuracy. A theorem will be proved which provides a theoretical bound for the error introduced by sampling.

The sampling method proposed in this thesis mines an approximate set of HUIs, by mining a subset of the whole dataset, which could lead to better time performance. An approach that combines sampling with *PHUI-Miner* proposed in Chapter 4 is also proposed in this chapter which does sampling first and then uses

PHUI-Miner to mine the sampled dataset.

We present the definitions and lemmas used in the theorem first, and then the theorem is provided. Finally, *PHUI-Miner with Sampling* is proposed.

5.1 Definitions and Lemmas

Given a dataset D, a user defined minimum threshold $\theta \in [0, 1]$, probability bound $\delta \in [0, 1)$, error bound $\epsilon \in [0, \theta]$ and probability parameter $k > 1$, the sampling strategy introduced in this chapter guarantees that itemset X is output with probability at least $1 - \delta - \frac{1}{k^2}$, if X is a HUI. To achieve this guarantee, a new theorem will be given, and proved in this section, which provides the minimum sample size required, given the parameters.

Before determining the sample size needed, another step is needed in order to get the statistics, including *total utility*, *average transaction utility*, *maximum transaction utility* and *standard deviation of transaction utilities*, of dataset D. *Total utility* is to be used in the mining process, while the other three are needed in the theorem.

Definition 10. (**Average Transaction Utility of a dataset** D) The average transaction utility of dataset D is defined as $avg_D = \frac{1}{n} \sum_{T_d \in D} u(T_d)$, where n is the number of transactions in D.

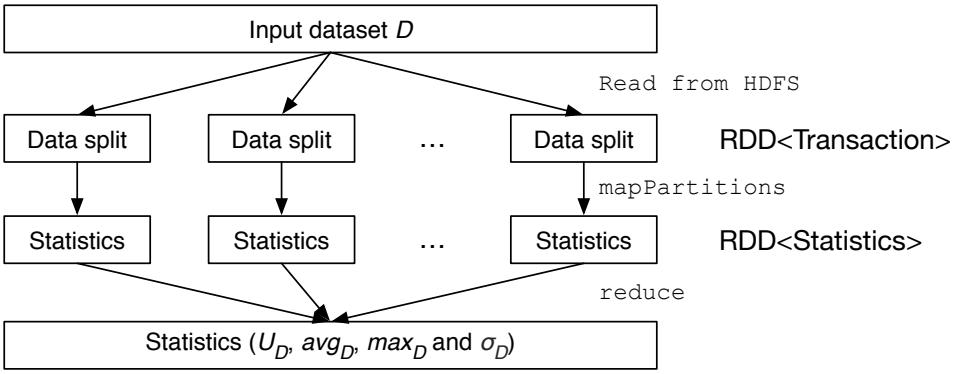


Figure 5.1: Data Flow for Getting Statistics of a Dataset

Definition 11. (**Maximum Transaction Utility of a dataset** D) The maximum transaction utility of dataset D is defined as $max_D = \max_{T_d \in D} u(T_d)$.

Definition 12. (**Standard Deviation of Transaction Utilities of a dataset** D) *The standard deviation of transaction utilities of a dataset* D *is denoted as* σ_D.

Figure 5.1 illustrates the data flow in Spark for getting the statistics of a dataset. The dataset is first read from HDFS to Spark as a RDD (Resilient Distributed Dataset) of transactions. Then the data in each node is scanned for their statistics. Finally, the statistics are combined. This process is expected to be fast, since only addition and comparison are involved here. In the case that the dataset is so huge that even determining the statistics is not feasible, sampling could also be used here. However, the discussion of this case is out of the scope of this thesis.

It is worth to mention that the way of getting *standard deviation of transaction utilities* in one pass of database scan is not very obvious. Usually, we use the

formula

$$\sigma = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \overline{x})^2} \tag{5.1}$$

to calculate the standard deviation of $x_i, i = 1, 2, ..., N$, where $\overline{x}$ is the average of these N values. This formula requires two scans of the values: one is to calculate $\overline{x}$ and the other is to calculate the final result. However, (5.1) has an algebraic identity

$$\sigma = \sqrt{\frac{1}{N-1}\left[\left(\sum_{i=1}^{N} x_i^2\right) - N\overline{x}^2\right]}, \tag{5.2}$$

which only need one scan of the values.

In order to derive the new theorem later in this section, the following lemmas are discussed first.

Lemma 1. (Hoeffding's inequality) *[24, Theorem 2] If $X_1, X_2, ..., X_n$ are independent variables, $S = X_1 + X_2 + ... + X_n$ and $a_i \leq X_i \leq b_i (i = 1, 2, ..., n)$, then for any $v > 0$,*

$$Pr\left(\frac{S}{n} - E\left(\frac{S}{n}\right) \geq v\right) \leq \exp\left(\frac{-2n^2v^2}{\sum_{i=1}^{n}(b_i - a_i)^2}\right). \tag{5.3}$$

Lemma 2. *If $X_1, X_2, ..., X_n$ are independent variables, $S = X_1 + X_2 + ... + X_n$ and $a_i \leq X_i \leq b_i (i = 1, 2, ..., n)$, then for any $t > 0$,*

$$Pr\left(|S - E(S)| \geq t\right) \leq 2\exp\left(\frac{-2t^2}{\sum_{i=1}^{n}(b_i - a_i)^2}\right) \tag{5.4}$$

Proof. Using a similar proof in [24], we have

$$Pr\left(\frac{S}{n} - E\left(\frac{S}{n}\right) \leq -v\right) \leq \exp\left(\frac{-2n^2v^2}{\sum_{i=1}^{n}(b_i - a_i)^2}\right). \tag{5.5}$$

From (5.3) and (5.5),

$$Pr\left(\left|\frac{S}{n} - E\left(\frac{S}{n}\right)\right| \geq v\right) \leq 2\exp\left(\frac{-2n^2v^2}{\sum_{i=1}^{n}(b_i - a_i)^2}\right). \tag{5.6}$$

Let $v = \frac{t}{n}$, so for any $t > 0$,

$$Pr(|S - E(S)| \geq t) \leq 2\exp\left(\frac{-2t^2}{\sum_{i=1}^{n}(b_i - a_i)^2}\right). \tag{5.7}$$

□

Lemma 3. (Chebyshev's inequality) *Let X (integrable) be a random variable with finite expected value μ and finite non-zero variance σ^2. Then for any real number $k > 0$,*

$$Pr\left(|X - \mu| \geq k\sigma\right) \leq \frac{1}{k^2} \tag{5.8}$$

Lemma 4. *Let $X_1, X_2, ..., X_N$ be a set of N independent random variables, and each X_i has the same probability distribution with mean $E(X)$ and variance σ^2. Then, the average of the N variables $\overline{X} = \frac{X_1+X_2+...+X_N}{N}$ has a distribution with mean $E(\overline{X}) = E(X)$ and variance $\sigma^2_{\overline{X}} = \frac{\sigma^2}{N}$.*

Proof. The property of means states that if X and Y are two random variables, then

$$E(X + Y) = E(X) + E(Y) \tag{5.9}$$

and

$$E(a + bX) = a + bE(X), \tag{5.10}$$

where a and b are constant values.

The property of variances states that if X and Y are two random variables, then

$$\sigma^2_{X+Y} = \sigma^2_X + \sigma^2_Y \tag{5.11}$$

and

$$\sigma^2_{a+bX} = b^2\sigma^2_X, \tag{5.12}$$

where a and b are constant values.

So,

$$\begin{aligned} E(\overline{X}) &= E\left(\frac{X_1 + X_2 + ... + X_N}{N}\right) \\ &= \frac{E(X_1) + E(X_2) + ... + E(X_N)}{N} = \frac{N \cdot E(X)}{N} = E(X) \end{aligned} \tag{5.13}$$

and

$$\sigma^2_{\overline{X}} = \sigma^2_{\frac{X_1+X_2+...+X_N}{N}} = \frac{\sigma^2_{X_1} + \sigma^2_{X_2} + ... + \sigma^2_{X_N}}{N^2} = \frac{N \cdot \sigma^2}{N^2} = \frac{\sigma^2}{N}. \tag{5.14}$$

□

In Lemma 4, $X_1, X_2, ..., X_N$ can be a sample drawn from a population. So this lemma reveals the property of the probability distribution of the sample mean.

5.2 Sampling Strategy

The main challenge in sampling methods is to determine the required sample size to achieve a given accuracy. A new theorem is proposed and proved in this section,

in order to select the sample size we need.

We derive the following theorem from Lemma 2, 3 and 4:

Theorem 1. *Given a dataset D with n transactions, and user given parameters $\epsilon \in (0,1), \delta \in (0,1), k > 1$. Let S be a random sample of D, so that the size of S, $m \geq \frac{1}{2\epsilon^2}\left(\frac{max_D}{avg_D}\right)^2 \ln\frac{2}{\delta}$, then with probability at least $1-\delta-\frac{1}{k^2}$, $\frac{u_S(X)}{U_S}$ is within the interval $\left[\frac{u_D(X)}{U_D} - \left(\epsilon + k\frac{\sigma_D}{\sqrt{m}\cdot avg_D}\right), \frac{u_D(X)}{U_D} + \left(\epsilon + k\frac{\sigma_D}{\sqrt{m}\cdot avg_D}\right)\right]$, for any itemset X.*

Proof. Suppose there are m transactions in S: $T_{g_1}, T_{g_2}, ..., T_{g_m}$. The utilities of itemset X in these transactions are $u(X, T_{g_1}), u(X, T_{g_2}), ..., u(X, T_{g_m})$, which can be viewed as independent random variables.

The values of $u(X, T_{g_i})$ are bounded:

$$u(X, T_{g_i}) \in [0, max_D], 1 \leq i \leq m. \tag{5.15}$$

Based on Definition 4,

$$u_S(X) = u(X, T_{g_1}) + u(X, T_{g_2}) + ... + u(X, T_{g_m}). \tag{5.16}$$

According to (5.4) and (5.16),

$$Pr(|u_S(X) - E[u_S(X)]| \geq t) \leq 2exp\left(-\frac{2t^2}{m \cdot {max_D}^2}\right), \forall t > 0. \tag{5.17}$$

Hence,

$$Pr\left(\left|\frac{u_S(X)}{m} - E\left[\frac{u_S(X)}{m}\right]\right| \geq \frac{t}{m}\right) \leq 2exp\left(-\frac{2t^2}{m \cdot {max_D}^2}\right), \forall t > 0. \tag{5.18}$$

From Lemma 4,

$$E\left[\frac{u_S(X)}{m}\right] = \frac{u_D(X)}{n}. \tag{5.19}$$

We have,

$$Pr\left(\left|\frac{u_S(X)}{m} - \frac{u_D(X)}{n}\right| \geq \frac{t}{m}\right) \leq 2exp\left(-\frac{2t^2}{m \cdot {max_D}^2}\right), \forall t > 0. \tag{5.20}$$

Hence,

$$Pr\left(\left|\frac{u_S(X)}{m \cdot avg_D} - \frac{u_D(X)}{n \cdot avg_D}\right| \geq \frac{t}{m \cdot avg_D}\right) \leq 2exp\left(-\frac{2t^2}{m \cdot {max_D}^2}\right), \forall t > 0. \tag{5.21}$$

Let $t = \epsilon m \cdot avg_D$, we have

$$Pr\left(\left|\frac{u_S(X)}{m \cdot avg_D} - \frac{u_D(X)}{n \cdot avg_D}\right| \geq \epsilon\right) \leq 2exp\left(-\frac{2m\epsilon^2 \cdot avg_D^2}{{max_D}^2}\right), \forall \epsilon > 0. \tag{5.22}$$

Since $m \geq \frac{1}{2\epsilon^2}\left(\frac{max_D}{avg_D}\right)^2 \ln\frac{2}{\delta}$ from the pre-condition of the theorem,

$$2exp\left(-\frac{2m\epsilon^2 \cdot avg_D^2}{{max_D}^2}\right) \leq \delta. \tag{5.23}$$

From (5.22) and (5.23),

$$Pr\left(\left|\frac{u_S(X)}{m \cdot avg_D} - \frac{u_D(X)}{n \cdot avg_D}\right| \geq \epsilon\right) \leq \delta, \forall \epsilon > 0. \tag{5.24}$$

Thus,

$$Pr\left(\left|\frac{u_S(X)}{m \cdot avg_S} \cdot \frac{avg_S}{avg_D} - \frac{u_D(X)}{n \cdot avg_D}\right| \leq \epsilon\right) \geq 1 - \delta, \forall \epsilon > 0. \tag{5.25}$$

Consequently,

$$Pr\left(\left|\frac{u_S(X)}{U_S} \cdot \frac{avg_S}{avg_D} - \frac{u_D(X)}{U_D}\right| \leq \epsilon\right) \geq 1 - \delta, \forall \epsilon > 0. \tag{5.26}$$

From Lemma 4, we also have

$$E(avg_S) = avg_D \tag{5.27}$$

and

$$\sigma_{avg_S} = \frac{\sigma_D}{\sqrt{m}}. \tag{5.28}$$

If $\sigma_D \neq 0$, from Lemma 3 we have

$$Pr\left(|avg_S - avg_D| \geq k \cdot \frac{\sigma_D}{\sqrt{m}}\right) \leq \frac{1}{k^2}, \forall k > 1, \tag{5.29}$$

where avg_S is viewed as a random variable.

So,

$$Pr\left(\left|\frac{avg_S}{avg_D} - 1\right| \geq k \cdot \frac{\sigma_D}{\sqrt{m} \cdot avg_D}\right) \leq \frac{1}{k^2}, \forall k > 1. \tag{5.30}$$

Hence,

$$Pr\left(\left|\frac{avg_S}{avg_D} - 1\right| \leq k \cdot \frac{\sigma_D}{\sqrt{m} \cdot avg_D}\right) \geq 1 - \frac{1}{k^2}, \forall k > 1. \tag{5.31}$$

In (5.26), if we denote $\left|\frac{u_S(X)}{U_S} \cdot \frac{avg_S}{avg_D} - \frac{u_D(X)}{U_D}\right| \leq \epsilon$ as event A, (5.26) is equivalent to $Pr\,(A) \geq 1 - \delta$.

In (5.31), if we denote $\left|\frac{avg_S}{avg_D} - 1\right| \leq k \cdot \frac{\sigma_D}{\sqrt{m} \cdot avg_D}$ as event B, (5.31) is equivalent to $Pr\,(B) \geq 1 - \frac{1}{k^2}$.

And,

$$Pr(A) + Pr(B) - Pr(A \wedge B) = Pr(A \vee B) \leq 1. \tag{5.32}$$

So,

$$Pr(A \wedge B) \geq Pr(A) + Pr(B) - 1 \geq 1 - \delta - \frac{1}{k^2}, \tag{5.33}$$

which is equivalent to

$$Pr\left(\left|\frac{u_S(X)}{U_S} \cdot \frac{avg_S}{avg_D} - \frac{u_D(X)}{U_D}\right| \leq \epsilon \wedge \left|\frac{avg_S}{avg_D} - 1\right| \leq k \cdot \frac{\sigma_D}{\sqrt{m} \cdot avg_D}\right) \geq 1 - \delta - \frac{1}{k^2}, \forall \epsilon > 0, k > 1. \tag{5.34}$$

Since

$$\left|\frac{u_S(X)}{U_S} \cdot \frac{avg_S}{avg_D} - \frac{u_D(X)}{U_D}\right| \leq \epsilon \wedge \left|\frac{avg_S}{avg_D} - 1\right| \leq k \cdot \frac{\sigma_D}{\sqrt{m} \cdot avg_D}$$

$$\Rightarrow$$

$$\left|\frac{u_S(X)}{U_S} - \frac{u_D(X)}{U_D}\right| \leq \epsilon + \frac{u_S(X)}{U_S} \cdot k\frac{\sigma_D}{\sqrt{m} \cdot avg_D}, \tag{5.35}$$

we have

$$Pr\left(\left|\frac{u_S(X)}{U_S} - \frac{u_D(X)}{U_D}\right| \leq \epsilon + \frac{u_S(X)}{U_S} \cdot k\frac{\sigma_D}{\sqrt{m} \cdot avg_D}\right) \geq Pr\left(\left|\frac{u_S(X)}{U_S} \cdot \frac{avg_S}{avg_D} - \frac{u_D(X)}{U_D}\right| \leq \epsilon \wedge \left|\frac{avg_S}{avg_D} - 1\right| \leq k \cdot \frac{\sigma_D}{\sqrt{m} \cdot avg_D}\right). \tag{5.36}$$

From (5.34) and (5.36),

$$Pr\left(\left|\frac{u_S(X)}{U_S} - \frac{u_D(X)}{U_D}\right| \leq \epsilon + \frac{u_S(X)}{U_S} \cdot k\frac{\sigma_D}{\sqrt{m} \cdot avg_D}\right) \geq 1 - \delta - \frac{1}{k^2}, \forall \epsilon > 0, k > 1. \tag{5.37}$$

And since

$$\frac{u_S(X)}{U_S} \leq 1, \tag{5.38}$$

consequently,

$$Pr\left(\left|\frac{u_S(X)}{U_S} - \frac{u_D(X)}{U_D}\right| \leq \epsilon + k\frac{\sigma_D}{\sqrt{m}\cdot avg_D}\right) \geq$$
$$Pr\left(\left|\frac{u_S(X)}{U_S} - \frac{u_D(X)}{U_D}\right| \leq \epsilon + \frac{u_S(X)}{U_S}\cdot k\frac{\sigma_D}{\sqrt{m}\cdot avg_D}\right). \tag{5.39}$$

Thus, from (5.37) and (5.39),

$$Pr\left(\left|\frac{u_S(X)}{U_S} - \frac{u_D(X)}{U_D}\right| \leq \epsilon + k\frac{\sigma_D}{\sqrt{m}\cdot avg_D}\right) \geq 1 - \delta - \frac{1}{k^2}, \forall\epsilon > 0, k > 1, \tag{5.40}$$

under the condition that $\sigma_D \neq 0$.

If $\sigma_D = 0$, all the transactions are having the same utility. So $avg_S = avg_D$. From (5.26),

$$Pr\left(\left|\frac{u_S(X)}{U_S} - \frac{u_D(X)}{U_D}\right| \leq \epsilon\right) \geq 1 - \delta, \forall\epsilon > 0. \tag{5.41}$$

So

$$Pr\left(\left|\frac{u_S(X)}{U_S} - \frac{u_D(X)}{U_D}\right| \leq \epsilon + k\frac{\sigma_D}{\sqrt{m}\cdot avg_D}\right) \geq$$
$$Pr\left(\left|\frac{u_S(X)}{U_S} - \frac{u_D(X)}{U_D}\right| \leq \epsilon\right) \geq 1 - \delta \geq 1 - \delta - \frac{1}{k^2}, \forall\epsilon > 0, k > 1, \tag{5.42}$$

under the condition that $\sigma_D = 0$.

To sum up,

$$Pr\left(\left|\frac{u_S(X)}{U_S} - \frac{u_D(X)}{U_D}\right| \leq \epsilon + k\frac{\sigma_D}{\sqrt{m}\cdot avg_D}\right) \geq 1 - \delta - \frac{1}{k^2}, \forall\epsilon > 0, k > 1, \tag{5.43}$$

under all conditions, which concludes that for any itemset X, if $m \geq \frac{1}{2\epsilon^2}\left(\frac{max_D}{avg_D}\right)^2 \ln\frac{2}{\delta}$, then with probability at least $1-\delta-\frac{1}{k^2}$, $\frac{u_S(X)}{U_S}$ is within the interval $\left[\frac{u_D(X)}{U_D}-\left(\epsilon+k\frac{\sigma_D}{\sqrt{m}\cdot avg_D}\right), \frac{u_D(X)}{U_D}+\left(\epsilon+k\frac{\sigma_D}{\sqrt{m}\cdot avg_D}\right)\right]$. □

For simplicity, we denote

$$\omega(\epsilon,\delta,D)=\frac{1}{2\epsilon^2}\left(\frac{max_D}{avg_D}\right)^2\ln\frac{2}{\delta} \tag{5.44}$$

and

$$\epsilon'=\epsilon+k\frac{\sigma_D}{\sqrt{m}\cdot avg_D} \tag{5.45}$$

in the rest of this thesis.

In the final results, if the minimum threshold is θ, we output all the itemsets with utility at least $(\theta-\epsilon')U_S$ in sample S of size at least $\omega(\epsilon,\delta,D)$.

It is worth to mention that $\omega(\epsilon,\delta,D)$ is independent of the size of the original dataset. The only factor from the dataset comes from the statistics of it, i.e. max_D and avg_D. When the dataset becomes larger, if $\frac{max_D}{avg_D}$ does not change much, the required sample size will also stays similar. In general, we can expect that avg_D does not change much, and max_D grows slowly as the dataset size becomes bigger. If some transactions make max_D much higher than the majority of the transactions, we can also remove these transactions, since they are usually considered outliers.

We hereby prove that the accuracy of our results is guaranteed.

Theorem 2. *With minimum threshold θ, if we output all the itemsets with utility at least $(\theta - \epsilon')U_S$ in sample S of size at least $\omega(\epsilon, \delta, D)$, itemset X is output with probability at least $1 - \delta - \frac{1}{k^2}$, if X is an HUI in dataset D.*

Proof. Since itemset X is an HUI, according to the definition of HUI, we have

$$u_D(X) \geq \theta U_D. \tag{5.46}$$

Hence,

$$\frac{u_D(X)}{U_D} - \epsilon' \geq \theta - \epsilon'. \tag{5.47}$$

According to Theorem 1,

$$\frac{u_S(X)}{U_S} \geq \frac{u_D(X)}{U_D} - \epsilon' \tag{5.48}$$

with probability at least $1 - \delta - \frac{1}{k^2}$.

From 5.47 and 5.48, we have

$$\frac{u_S(X)}{U_S} \geq \theta - \epsilon' \tag{5.49}$$

with probability at least $1 - \delta - \frac{1}{k^2}$.

So X is output with probability at least $1 - \delta - \frac{1}{k^2}$. □

Theorem 3. *With minimum threshold θ, if we output all the itemsets with utility at least $(\theta - \epsilon')U_S$ in sample S of sample size at least $\omega(\epsilon, \delta, D)$, any itemsets in the output are guaranteed to have a utility at least $(\theta - 2\epsilon')U_D$ in dataset D with probability at least $1 - \delta - \frac{1}{k^2}$.*

Proof. Suppose itemset X is output, we have

$$u_S(X) \geq (\theta - \epsilon')U_S. \tag{5.50}$$

Hence,

$$\frac{u_S(X)}{U_S} \geq \theta - \epsilon'. \tag{5.51}$$

According to Theorem 1,

$$\frac{u_S(X)}{U_S} \leq \frac{u_D(X)}{U_D} + \epsilon' \tag{5.52}$$

with probability at least $1 - \delta - \frac{1}{k^2}$.

From 5.51 and 5.52, we have

$$\theta - \epsilon' \leq \frac{u_D(X)}{U_D} + \epsilon' \tag{5.53}$$

with probability at least $1 - \delta - \frac{1}{k^2}$.

Hence,

$$\frac{u_D(X)}{U_D} \geq \theta - 2\epsilon' \tag{5.54}$$

with probability at least $1 - \delta - \frac{1}{k^2}$.

So X has a utility at least $(\theta - 2\epsilon')U_D$ in dataset D with probability at least $1 - \delta - \frac{1}{k^2}$. □

There are 3 parameters used in this sampling strategy, namely ϵ, δ and k. The value of k determines a base value for the probability bound. k will be set to 2

in our experiments. The reason for choosing this value for k will be explained in Section 6.2. The value of δ, in the range of $(0, 1)$, determines the probability bound together with k. If users want to have more confidence in the results, δ could be set to a lower value, and the required sample size will be bigger. ϵ, in the range of $(0, 1)$, is the extra error bound for the relative utility, i.e. error in addition to $k\frac{\sigma_D}{\sqrt{m}\cdot avg_D}$. Choosing a smaller value for ϵ will lead to more accurate results, but a bigger required sample size. So choosing the values for δ and ϵ is very important in the sampling strategy. We need to make sure that the values for parameters will not result in a sample size even bigger than the total size of the dataset. Also, we want the error bound to be low and the probability bound to be high, so that the results could have a relatively high accuracy and confidence. In my experiments, we usually choose the value of ϵ to be $\frac{1}{10}$ of the relative utility threshold. If the required sample size is too big compared to the total size of the dataset, the value of ϵ will be increased a little bit for looser error bound and smaller sample size.

5.3 PHUI-Miner with Sampling

The sampling method mines an approximate set of HUIs, but reduces the size of the dataset to $\omega(\epsilon, \delta, D)$ as shown in this chapter. However, in most cases, a sample with size $\omega(\epsilon, \delta, D)$ is still too large to mine for a single machine. Besides, the parallel distributed algorithm proposed in the previous chapter also has the

running time issue, when the dataset gets bigger. Our solution to this problem is to combine the two methods, so that the dataset is sampled before being processed by *PHUI-Miner*.

In this section, an approach combining sampling and *PHUI-Miner*, referred to as, *PHUI-Miner with Sampling*, is proposed.

Given a dataset D, a minimum relative utility threshold θ, user provided parameters δ, ϵ and k. Theorem 1 guarantees that if we only use a sample of D with size $\omega(\epsilon, \delta, D)$, and mine all the itemsets with relative utility no less than $\theta - \epsilon'$, any high utility itemset X will appear in the output with probability $1 - \delta - \frac{1}{k^2}$.

Our approach first draws a sample with size $\omega(\epsilon, \delta, D)$ from the whole dataset, and then mines HUIs with threshold $\theta - \epsilon'$ parallelly with *PHUI-Miner*. In this way, *PHUI-Miner with Sampling* is able to achieve the same accuracy as using a single machine to mine a sample of the dataset, and have better time and memory performance than using *PHUI-Miner* alone.

6 Experimental Results

The parallel distributed algorithm *PHUI-Miner* and the sampling method, proposed in this thesis, are evaluated in this chapter. The evaluations are focused on the time performance of *PHUI-Miner*, and the accuracy of the sampling method. Experiments are also conducted to evaluate the time performance of *PHUI-Miner with Sampling* in Section 6.1.

In our experiments, parallel distributed algorithms are run on Amazon Web Services (AWS). We used twenty r3.xlarge instances to run HDFS and Spark on them. One of the instances is used as *Master*, while the others are *Workers*. The non-parallel algorithms are conducted on a single Intel(R) Xeon(R) X5660 computer with 50 GB of RAM.

The experiments are conducted on different datasets, *kosarak* [19], *accidents* [19], *chess* [19], *twitter* [26], *T5000L10I1P10PL6*, *ta-feng* [4] and *globe*. *T5000L10I1P10PL6* is a synthetic dataset, while the other six are real-world datasets. *T5000L10I1P10PL6* is generated using the IBM Quest Data Generator [21]. The IBM Quest Data

number of transactions	5,000,000
average items per transaction	10
number of different items	1,000
number of patterns	10
average length of maximal pattern	6

Table 6.1: Parameters of the Synthetic Dataset

Generator is often used in studies of frequent pattern mining and association rule mining, and can generate datasets without utilities according to input parameters. The parameters for the synthetic dataset used in this paper are shown in Table 6.1. For *T5000L10I1P10PL6*, the internal utilities are generated using a uniform distribution in $[1, 10]$, while the external utilities are generated using a log-normal distribution, with $\mu = 1$ and $\sigma = 0.5$. The *kosarak* dataset contains anonymized click-stream data of a Hungarian online news portal, which is a sparse dataset. The *accidents* dataset consists of a collection of traffic accident records. Each record contains a description of an accident such as gender of the driver, speed limit of the road, and whether alcohol is involved. The *chess* dataset is derived from chess game steps, which is very dense. The utility values for *kosarak*, *accidents* and *chess* are taken from [19], where the internal utility values are generated using a uniform distribution in $[1, 10]$ and the external utility values are generated

using a normal distribution. The *twitter* dataset describes the followers of Twitter users, in which each transaction corresponds to a user and contains the list of followers of the user. For the *twitter* dataset, the internal utilities are generated using a log-normal distribution, with $\mu = 2.22$ and $\sigma = 0.6$. The external utilities are generated using a log-normal distribution, with $\mu = 0$ and $\sigma = 0.1$. The *ta-feng* dataset is from a supermarket in Taiwan and describes the transactions collected within a time span of four months, from November, 2000 to February, 2001. There are a total of 119,578 transactions involving 24,069 products and 32,266 customers in the dataset [33], which contains profits of each merchandise. The *globe* dataset comes from The Globe and Mail [5], which is a news company in Canada. Each transaction in the *globe* dataset represents page views of a visitor in one time. The items are the titles of articles, while the internal utilities are the times spent on the articles and the external utilities are all 1's. The reason for choosing the values in *T5000L10I1P10PL6* and *twitter* is that, in the sampling method, the resulting sample size $\omega(\epsilon, \phi, D)$ should be smaller than the original dataset. Otherwise, we should mine the HUIs using the original dataset without sampling. We also trimmed the lengths of transactions in *twitter* to at most 15 for simplicity.

Below, we first present the performance of *PHUI-Miner* and *PHUI-Miner with Sampling*. Then, the accuracy of the sampling strategy will be evaluated. Finally, the application of high utility itemset mining will be discussed.

6.1 PHUI-Miner and PHUI-Miner with Sampling

Since *PHUI-Miner* is an exact approach, there is no need for the accuracy evaluation for it. The evaluation of *PHUI-Miner* is conducted in terms of time performance and speedup. We also evaluate the time performance of *PHUI-Miner with Sampling* in this section.

To better evaluate our algorithm, we designed an algorithm, called *PHUI-Miner Rnd*, for comparison purpose. The *PHUI-Miner Rnd* algorithm is the same as *PHUI-Miner* except for the *Split Search Space* phase. In *PHUI-Miner Rnd*, the items are split randomly into different nodes instead of using the procedure shown in Algorithm 3. Comparing *PHUI-Miner Rnd* with *PHUI-Miner* shows that the choice of the way of splitting the search space in our designing of *PHUI-Miner* is reasonable. Experiments on *PHUI-Miner Rnd* are conducted at least 5 times for each experiment to get average results.

Below, the time performance of *PHUI-Miner* and *PHUI-Miner with Sampling* is first presented. And then, the speedup of *PHUI-Miner* is evaluated.

6.1.1 Time Performance

To the best of our knowledge, very few studies have been proposed to use distributed computing technique to mine high utility itemsets. Thus, the time performance of

PHUI-Miner is evaluated against *HUI-Miner* and *PHUI-Miner Rnd*, while *PHUI-Miner with Sampling* is compared with *PHUI-Miner*.

Since the datasets *twitter* and *T5000L10I1P10PL6* consume too much memory as well as running time, *HUI-Miner* is not able to mine these two datasets. So the comparison of *PHUI-Miner* and *HUI-Miner* is only conducted on *kosarak*, *accidents*, *chess*, *ta-feng* and *globe*. Figures 6.1a, 6.1b, 6.1c, 6.1f and 6.1g show the results for comparing *PHUI-Miner* with *HUI-Miner*. Our method outperforms *HUI-Miner* in all the cases. However, when the relative utility threshold is big, the time performance of *HUI-Miner* approaches *PHUI-Miner*. This is because of the network latency the cluster introduces. *PHUI-Miner* will need at least some time to read the data from the HDFS, repartion the data, etc. When the threshold is small, and a large amount time is needed in the mining process. *PHUI-Miner* works much better than *HUI-Miner*.

Figure 6.1 also shows the time performance of *PHUI-Miner* and *PHUI-Miner Rnd*. In *kosarak*, *accidents*, *chess*, *ta-feng* and *globe*, the time performances of the two methods are similar. However, *PHUI-Miner* is slightly faster than *PHUI-Miner Rnd* in these datasets. For the *twitter* dataset, *PHUI-Miner* works much better than *PHUI-Miner Rnd*. For the *T5000L10I1P10PL6* dataset, *PHUI-Miner* is slightly slower than *PHUI-Miner Rnd* though. These are normal behaviours, since the distributions of different datasets are different. Some datasets may have

very special distribution, that our approach is not the optimal solution to them. However, it is shown that in most cases, *PHUI-Miner* works better than *PHUI-Miner Rnd*. In the cases that *PHUI-Miner* works slower, the difference of them is very small. As a result, our way of splitting the search space is demonstrated to be a good choice.

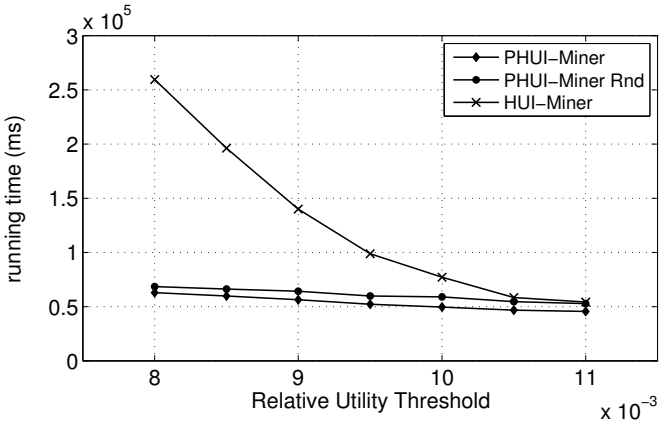
For the comparison of *PHUI-Miner* with *PHUI-Miner with Sampling*, it is only conducted on datasets *accidents*, *twitter* and *T5000L10I1P10PL6*. The other four datasets are not suitable for the sample technique, since their minimum required sample sizes exceed the sizes of the whole datasets. Figures 6.1b, 6.1d and 6.1e show the experimental results of the time performance for *PHUI-Miner with Sampling* and *PHUI-Miner*. The values of other parameters used in *PHUI-Miner with Sampling* in this section are provided in Table 6.2. It's demonstrated that *PHUI-Miner with Sampling* has better time performances in all three datasets. In datasets *twitter* and *T5000L10I1P10PL6*, which have millions of transactions, *PHUI-Miner with Sampling* is much faster than *PHUI-Miner*.

6.1.2 Speedup

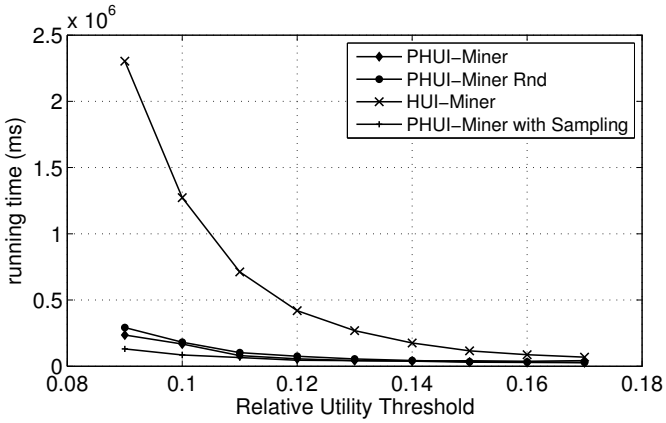
The speedup for *PHUI-Miner* is evaluated on the *kosarak* dataset. We used different number of nodes to run the experiments, regarding the speed using two nodes as 1.

The results are in Figure 6.2. The relative utility threshold used for *kosarak*

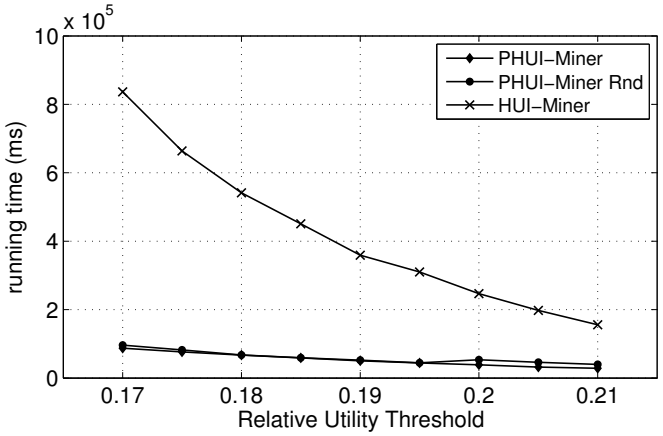
(a)


x 10^5
3
2.5
2
1.5
1
0.5
0
8
9
10
11
x 10^−3
running time (ms)
Relative Utility Threshold
PHUI−Miner
PHUI−Miner Rnd
HUI−Miner

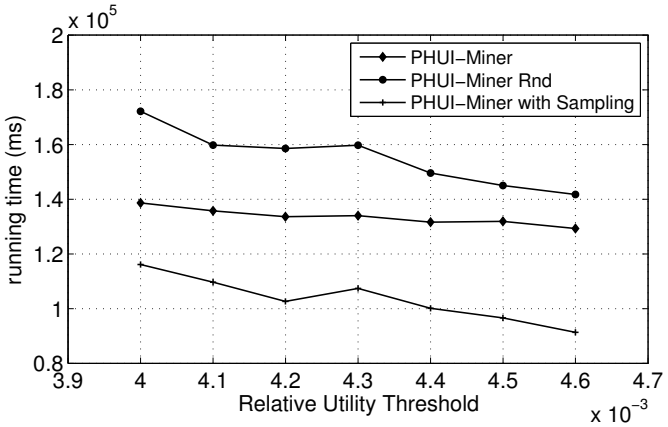
(b)


x 10^6
2.5
2
1.5
1
0.5
0
0.08
0.1
0.12
0.14
0.16
0.18
running time (ms)
Relative Utility Threshold
PHUI−Miner
PHUI−Miner Rnd
HUI−Miner
PHUI−Miner with Sampling

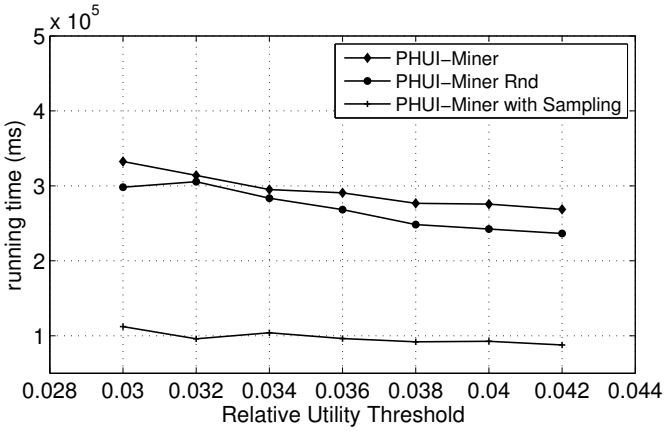
(c)


x 10^5
10
8
6
4
2
0
0.17
0.18
0.19
0.2
0.21
running time (ms)
Relative Utility Threshold
PHUI−Miner
PHUI−Miner Rnd
HUI−Miner

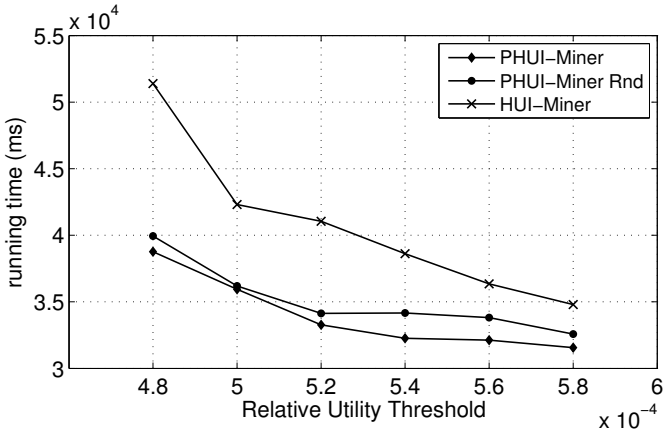
(d)


x 10^5
2
1.8
1.6
1.4
1.2
1
0.8
running time (ms)
PHUI−Miner
PHUI−Miner Rnd
PHUI−Miner with Sampling
3.9
4
4.1
4.2
4.3
4.4
4.5
4.6
4.7
Relative Utility Threshold
x 10^−3

(e)


x 10^5
5
4
3
2
1
running time (ms)
PHUI−Miner
PHUI−Miner Rnd
PHUI−Miner with Sampling
0.028
0.03
0.032
0.034
0.036
0.038
0.04
0.042
0.044
Relative Utility Threshold

(f)


x 10^4
5.5
5
4.5
4
3.5
3
running time (ms)
PHUI−Miner
PHUI−Miner Rnd
HUI−Miner
4.8
5
5.2
5.4
5.6
5.8
6
Relative Utility Threshold
x 10^−4

Dataset	ϵ	δ	k
accidents	0.005	0.4	2
T5000L10I1P10PL6	0.005	0.1	2
twitter	0.001	0.7	2

Table 6.2: Values of Parameters

(g)

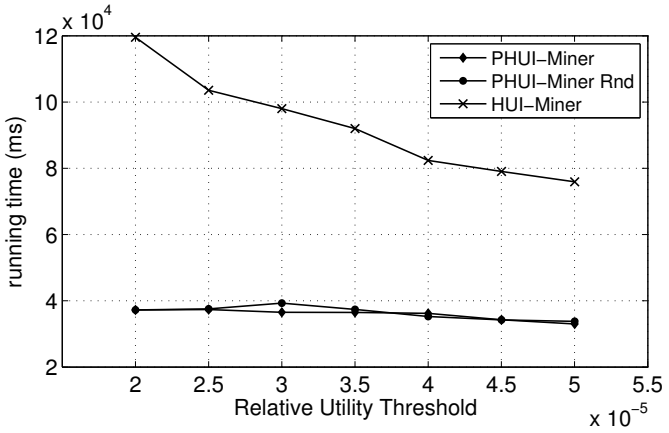


Figure 6.1: Running Time of PHUI-Miner on (a) *kosarak*, (b) *accidents*, (c) *chess*, (d) *twitter*, (e) *T5000L10I1P10PL6*, (f) *ta-feng* and (g) *globe*

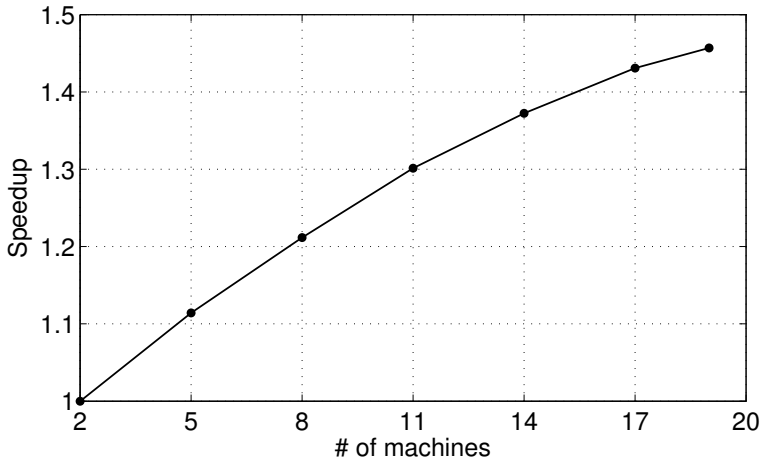


Figure 6.2: Speedup of PHUI-Miner

is 0.01. It's shown that the speedup of *PHUI-Miner* is near linear, which means our approach could scale well when we have more and more nodes. However, it is notable that our algorithm could not have a linear speedup when we have a large number of nodes. If there are too many nodes, the communication cost will be dominating the total running time. But if the number of nodes is not very big, the computation cost is the main cost.

6.2 Accuracy of Sampling Strategy

In order to evaluate the accuracy of our sampling strategy, we have performed several experiments on *accidents*, *T5000L10I1P10PL6* and *twitter*.

The statistics for the three datasets are in Table 6.3.

In this section, the effectiveness of Theorem 1 is demonstrated by experiments

Dataset	*twitter*	*T5000L10I1P10PL6*	*accidents*
# of transactions	19,265,416	4,947,263	340,183
maximum utility	456	788	1034
average utility	121.15	196.04	576.58

Table 6.3: Statistics of Datasets in the Sampling Strategy

Dataset	θ	ϵ	k
accidents	0.085	0.005	2
T5000L10I1P10PL6	0.03	0.005	2
twitter	0.004	0.001	2

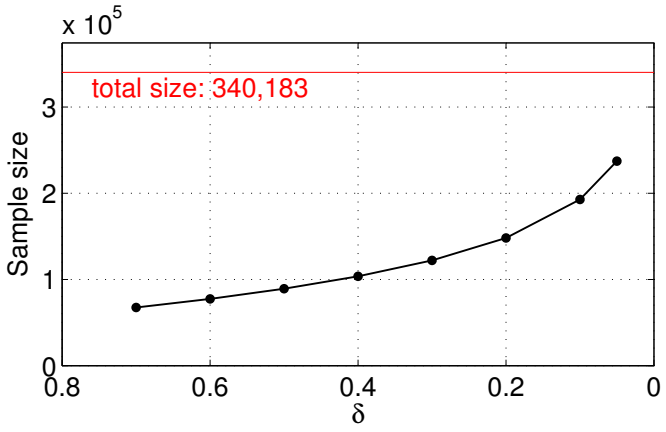
Table 6.4: Values of Parameters

on mining samples drawn from different datasets. We evaluate the effectiveness of the proposed sampling method against the exact results and provide the precision, recall, f-measure and relative utility error of our method on the three datasets. The relative utility error is computed as the utility error compared with the exact utility results:

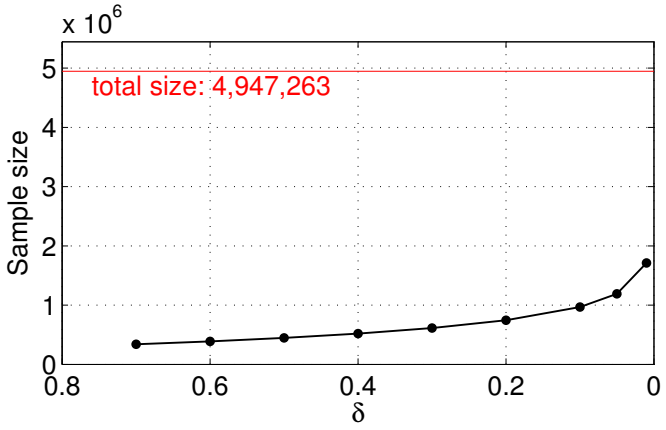
$$relative\ utility\ error = \left| \frac{approximate\ utility - exact\ utility}{exact\ utility} \right|. \qquad (6.1)$$

Since $\omega(\epsilon, \delta, D)$ is independent of the total size of the dataset, even when the dataset is very small, it's still possible that a huge dataset size is required for a

(a)



(b)



(c)

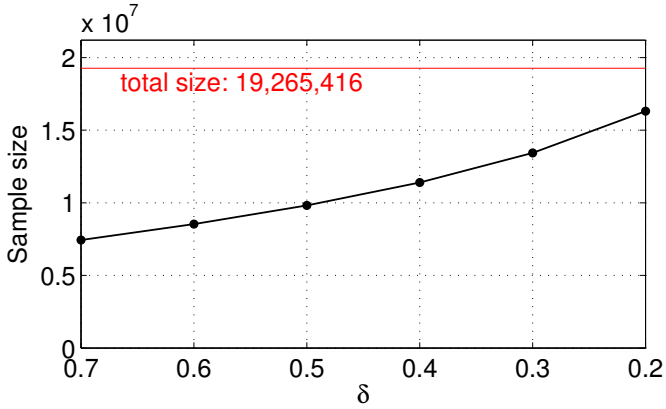


Figure 6.3: Sample Size of (a) *accidents*, (b) *T5000L10I1P10PL6* and (c) *twitter*

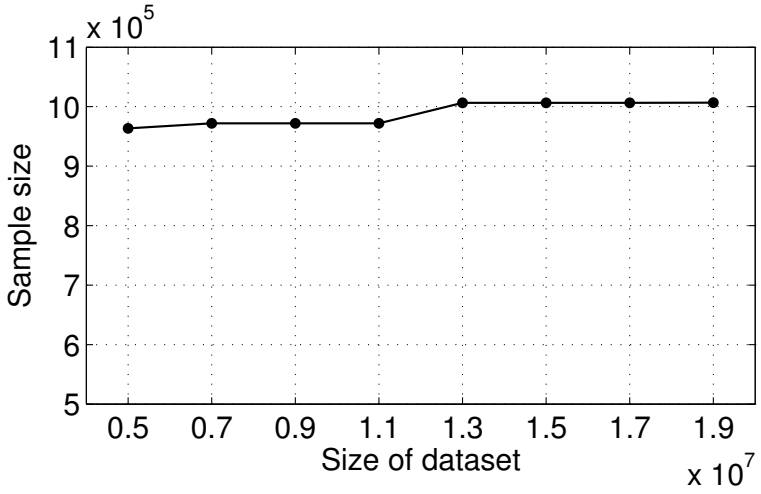


Figure 6.4: Sample Sizes for Different Sizes of Datasets

given accuracy according to our theorem, depending on the data distribution of the dataset, while for some datasets, only a small fraction of the whole dataset is required. Figure 6.3 presents the required sample size of the three datasets for different δ's. The values for other parameters used in this section are shown in Table 6.4. The sample sizes grow exponentially as δ decreases. However, if a reasonable δ is chosen, the sample size is usually much smaller than real-world big datasets, that can have billions of transactions. Figure 6.4 shows the sample size for different sizes of datasets. The datasets used in this figure are all generated using the IBM Quest Data Generator with the same parameters as used in generating *T5000L10I1P10PL6* except the dataset size. The sample size in this figure is shown to be not very related to the total size of the dataset. The sample size gets slightly

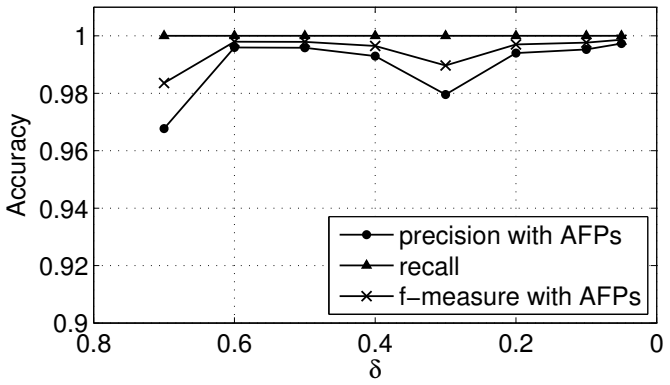
bigger as the total size gets bigger because the value of max_D is more likely to get bigger.

It is worth to mention that we chose the values for ϵ's in Table 6.4 according to the characteristics of the datasets. ϵ's are chosen so that ϵ is much smaller than θ so as to have a good accuracy. In the meanwhile, ϵ's cannot be too small, since smaller ϵ values result in bigger sample sizes. We chose 2 for k's in all the datasets, since the probability $1-\delta-\frac{1}{k^2}$ would be $0.75-\delta$, which is acceptable in our experiments. If k is bigger, the threshold $\theta-\epsilon'$ used for mining would be smaller, which will affect the accuracy and running performance of the sampling strategy. If k is smaller, the probability guarantee of $1-\delta-\frac{1}{k^2}$ would be smaller. For the same reason, we choose the same value for k's in Section 6.1.

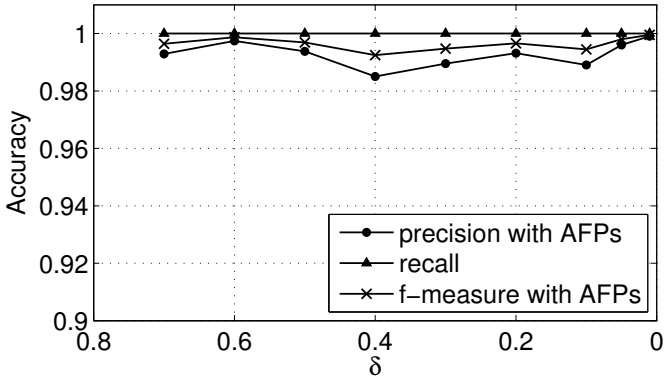
In our experiments, in order to better measure the accuracy of our algorithms, we used a measure called precision with AFPs. Since our approach is to find all HUIs with relative utility at least $\theta-\epsilon'$, the HUIs with relative utility in the range of $[\theta-\epsilon', \theta)$, called *Acceptable False Positives* (AFPs), are considered as true positives in our results when computing the precision with AFPs. Correspondingly, we use f-measure with AFPs to replace the commonly used f-measure.

Figure 6.5 shows the precision with AFPs, recall, and f-measure with AFPs for different δ values. The recall is constantly 1, which means all the exact high utility itemsets are found, although our theorem only guarantees that a high utility

(a)



(b)



(c)

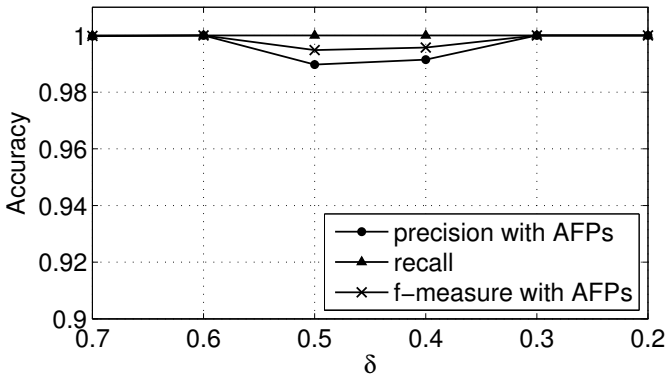
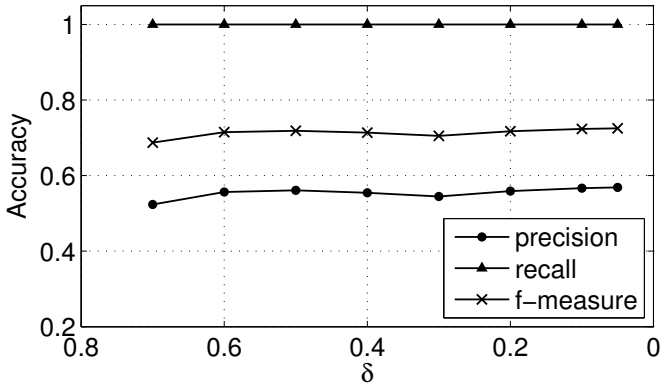
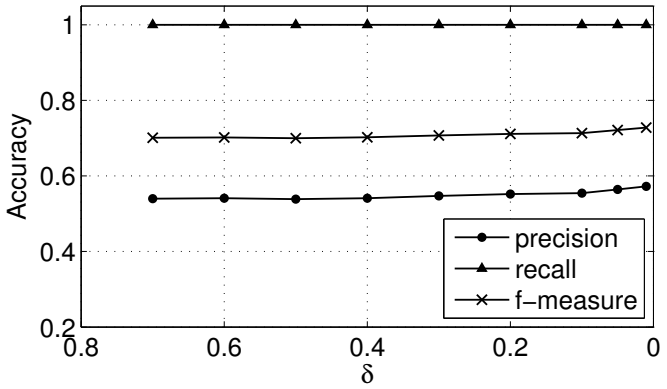


Figure 6.5: Accuracy of Sampling on (a) *accidents*, (b) *T5000L10I1P10PL6* and (c) *twitter*

(a)



(b)



(c)

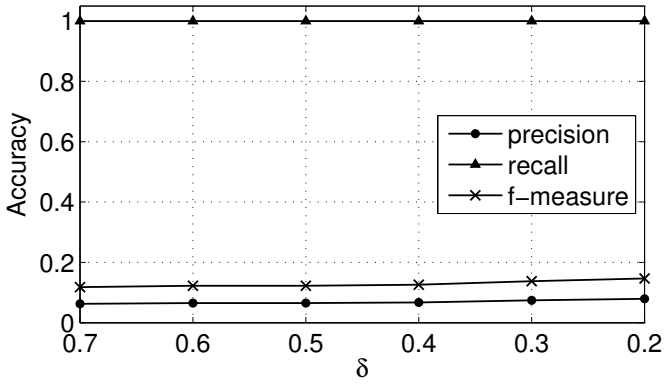
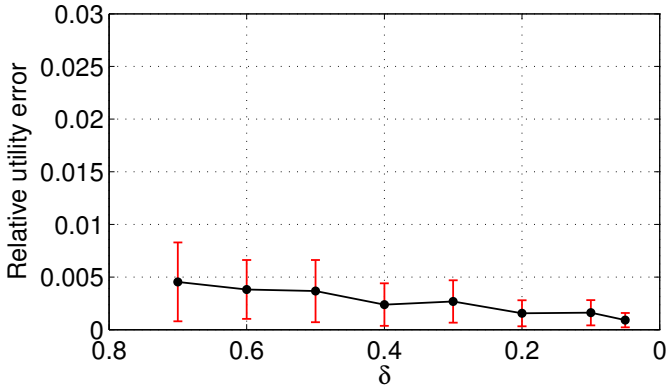
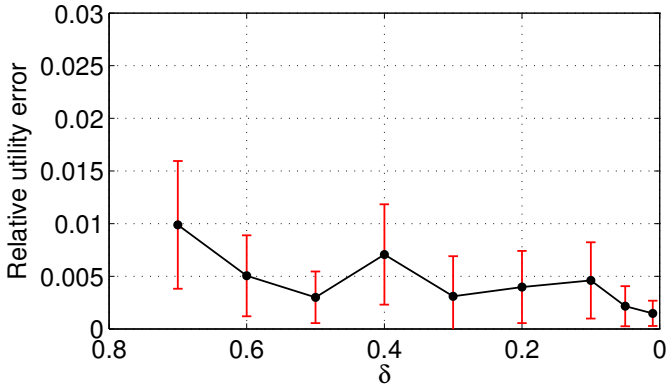


Figure 6.6: Accuracy of Sampling without AFPs on (a) *accidents*, (b) *T5000L10I1P10PL6* and (c) *twitter*

(a)



(b)



(c)

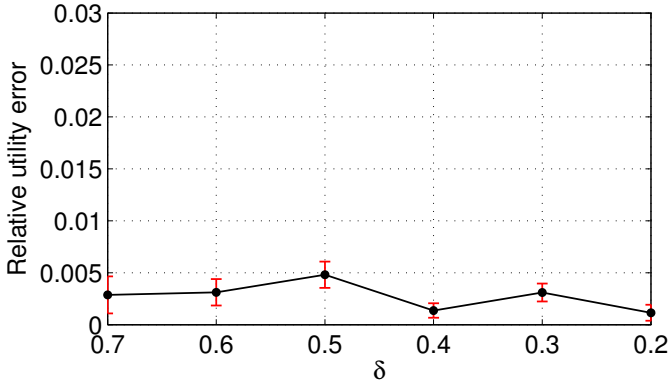


Figure 6.7: Average Value and Standard Deviation of Relative Utility Error on (a) *accidents*, (b) *T5000L10I1P10PL6* and (c) *twitter*

itemset can be found with probability $1 - \delta - \frac{1}{k^2}$. This behaviour is reasonable. We have a relatively loose bound for the sample size, which leads to sets of results having recall higher than expected in our theorem. According to our experiments, the precision with AFPs is always above 0.96, and approaches 1 as δ decreases. The f-measure with AFPs has higher values than precision with AFPs as expected, and also grows as δ decreases.

Sometimes, AFPs are even more than the true positives we have. However, it is acceptable, since they all have relative utilities very close to θ. The number of AFPs in the results depends on the distribution of datasets, as well as the parameters provided. We also provide the experimental results of precision, recall, and f-measure without considering AFPs in Figure 6.6.

The average relative utility errors and its standard deviations of itemsets in the results are provided for different δ's and different datasets, in Figure 6.7. The relative utility errors and their standard deviations are trivial. Their values decrease as δ gets smaller.

The experimental results show that our method works well and provides results even better than what is expected in our theorem.

6.3 Usefulness of High Utility Itemset Mining

High utility itemset mining has been studied by many researchers. In this section, we use experimental results of the real-world dataset, *globe*, to present a real-world example of the application of high utility itemset mining, for showing the usefulness of high utility itemset mining.

As mentioned in the beginning of this chapter, the *globe* dataset contains news articles and the time spent on them for each user. Our objective of this application is to make recommendations of news articles to readers. When a user reads an article on *The Globe and Mail*, we want to recommend some other related articles to this reader.

Frequent itemset mining and high utility itemset mining can both be used in this application. Given an high utility itemset or frequent itemset, if a user reads any item in the itemset, an simple way to do recommendation is to recommend the remaining items in the itemset to the user. Frequent pattern mining is used a lot in this type of applications. However, frequent pattern mining does not consider the time spent on the articles. It is possible that the users browse articles very quickly without interests in them. High utility itemset mining solves this problem by only getting the itemsets with high utilities, which, in this case, is the total time spent on these items. So if we recommend articles according to high utility itemsets, the

users are expected to spend more time on the website, than recommending articles based on frequent itemsets.

In this section, we use *FP-Growth* [22] and *HUI-Miner* [35] to get the top 10 frequent patterns and high utility itemsets from the *globe* dataset, and compare the differences between them. Our objective is to find itemsets, that contain two or more items so that we can make recommendations according to them. Thus, the itemsets with only 1 item are not included in the results.

Rank	**Frequent Pattern**	**Support**	**Utility**
1	[Vigil held for daughter of Conservative Party president] [MH17: Disaster ratchets up Russia-Ukraine tensions]	367	163523
2	[Canadian professor was killed in targeted attack, Florida police say] [La Prairie, Quebec mayor dies from wasp stings]	271	58493
3	[Target faces calls to withdraw from Canada] [Mike Duffy facing 31 charges from Senate expenses scandal, RCMP says]	231	77388

4	[MH17: Disaster ratchets up Russia-Ukraine tensions] [Rob Ford to undergo foot surgery; sobriety coach no longer working full-time]	211	85182
5	[CBC lays off veteran sportscasters amid budget cuts] [Celine Dion takes indefinite break to focus on health, family]	201	46191
6	[Robin Williams warp-speed improvisation was almost too fast to be human] [CBC lays off veteran sportscasters amid budget cuts]	200	70219
7	[Wednesday's analyst upgrades and downgrades] [One of the few quality dividend stocks left that pays a 5% yield]	199	54302
8	[We need to talk about masturbation, the last great sexual taboo] [Retiree, 60, wonders how long her money will last]	191	123149
9	[Controversial First Nation chiefs salary raises concern] [Harper sticks to hard line on Hamas; U.S. condemns Israel's deadly shelling of UN school]	186	68096

10	['Massive explosive decompression' downed MH17: Kiev] [Canada should learn from Ireland's housing crash]	182	60293

Table 6.5: Top 10 Frequent Patterns of *globe*

Rank	**High Utility Itemset**	**Utility**	**Support**
1	[Vigil held for daughter of Conservative Party president] [MH17: Disaster ratchets up Russia-Ukraine tensions]	163523	367
2	[Retiree, 60, wonders how long her money will last] [We need to talk about masturbation, the last great sexual taboo]	123149	191
3	[Retiree, 60, wonders how long her money will last] [Which is better, a RRIF or an annuity? You may be surprised]	104080	167
4	[Rob Ford to undergo foot surgery; sobriety coach no longer working full-time] [MH17: Disaster ratchets up Russia-Ukraine tensions]	85182	211

5	[Israel prepares to 'significantly' expand campaign as UN chief heads for Middle East as mediator] [MH17: Disaster ratchets up Russia-Ukraine tensions]	78104	138
6	[Exercise both body and mind with a different kind of cross-training] [How used Google smartphones cough up former owners personal data] [Christy Clark goes public with support for Israel] [Say goodbye to the family cottage before it's too late]	77994	1
7	[Exercise both body and mind with a different kind of cross-training] [How used Google smartphones cough up former owners personal data] [Say goodbye to the family cottage before it's too late]	77888	1
8	[Exercise both body and mind with a different kind of cross-training] [Christy Clark goes public with support for Israel] [Say goodbye to the family cottage before it's too late]	77831	1

9	[Exercise both body and mind with a different kind of cross-training] [How used Google smartphones cough up former owners personal data] [Christy Clark goes public with support for Israel]	77727	1
10	[Exercise both body and mind with a different kind of cross-training] [Say goodbye to the family cottage before it's too late]	77725	1

Table 6.6: Top 10 High Utility Itemsets of *globe*

Table 6.5 and 6.6 show the top 10 results from frequent pattern mining and high utility itemset mining respectively. The patterns and itemsets in the tables are titles of news articles. The titles in the same pattern or itemset usually belong to the related categories of topics, as shown in the tables. However, the rank of some patterns and itemsets is very different in the results of frequent pattern mining and high utility itemset mining. The 3rd and 5th HUIs are not within the top 10 FPs. If we only use the top 10 results for recommendation, we would miss such patterns if frequent pattern mining is used. In addition, the last 4 HUIs in the results have very low supports. However, users spent a long time reading these articles together. This pattern will not be discovered by frequent itemset mining.

In the results of frequent pattern mining, the 2nd and 3rd FPs do not have as high utility as the itemsets in the top 10 HUIs. Using the top FPs for recommendation may not lead to a longer reading time of the user, than using the top HUIs, since the HUIs have higher utility values, which in this case represents the total reading time of the itemset.

Apart from the simple recommendation method we showed above, there are quite a few other ways to do recommendation, which could have improvements over our method. However, The discuss of these improvements is out of the scope of this thesis.

7 Conclusion

7.1 Summary of Contributions

The contributions of this thesis are summarized as follows.

- A distributed algorithm, *PHUI-Miner*, is proposed, which parallelizes the state-of-the-art algorithm *HUI-Miner*, for mining exact set of HUIs. The algorithm employs the memory-based distributed computing framework, Apache Spark, which enables *PHUI-Miner* to have a good performance and be fault-tolerant.

- We proposed and proved a new theorem, which provides us with the required sample size to achieve a given accuracy for approximately mining HUIs. The theorem leads to a sampling method with theoretical guarantees on the probability that an HUI can be returned and on the utility of a returned itemset. A feature of this sampling method is that the sample size required to achieve the theoretical guarantees is independent of the size of the original data, and

is thus not necessarily going up as the data set grows.

- We proposed an approach, *PHUI-Miner with Sampling*, combining *PHUI-Miner* with the sampling technique, which mines an approximate set of high utility itemsets, but achieves better time performances.

- Extensive experiments are conducted to evaluate our proposed algorithms. Our experimental results on different datasets show that our sampling method achieves highly accurate results, much better than what the theory guarantees. Empirically, we demonstrated that the sampling strategy could achieve very high accuracy. We also demonstrated that *PHUI-Miner* has a good time performance, and outperforms the state-of-the-art non-parallel algorithm *HUI-Miner*. Finally, *PHUI-Miner with Sampling* is shown to have better time performance than *PHUI-Miner*.

7.2 Future Research

There are several possible improvements, which could be done in the future:

- The search space division in *PHUI-Miner* is not even and has rooms for further improvements. In most cases, the mining processes in different nodes end at very differentiated times. It is possible that there are other ways to split the search space so that the workload of each node in the cluster is more even.

- The required sample size in the sampling strategy is relatively a loose bound, which means it is possible to find another better bound for the required sample size. Future research could focus on improving the bound for better time performance.

Bibliography

[1] http://hadoop.apache.org/.

[2] http://cassandra.apache.org/.

[3] http://www.openstack.org/.

[4] http://recsyswiki.com/wiki/Grocery_shopping_datasets.

[5] http://www.theglobeandmail.com/.

[6] R. Agrawal, T. Imieliński, and A. Swami. Mining association rules between sets of items in large databases. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, SIGMOD '93, pages 207–216, New York, NY, USA, 1993. ACM.

[7] R. Agrawal and R. 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.

[8] R. Agrawal, R. Srikant, et al. Fast algorithms for mining association rules. In *Proc. 20th int. conf. very large data bases, VLDB*, volume 1215, pages 487–499, 1994.

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

[10] C. F. Ahmed, S. K. Tanbeer, B.-S. Jeong, and Y.-K. Lee. Huc-prune: an efficient candidate pruning technique to mine high utility patterns. *Applied Intelligence*, 34(2):181–198, 2011.

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

[12] R. J. Bayardo Jr. Efficiently mining long patterns from databases. In *ACM Sigmod Record*, volume 27, pages 85–93. ACM, 1998.

[13] T. Calders and B. Goethals. Mining all non-derivable frequent itemsets. In *Principles of Data Mining and Knowledge Discovery*, pages 74–86. Springer, 2002.

[14] V. T. Chakaravarthy, V. Pandit, and Y. Sabharwal. Analysis of sampling techniques for association rule mining. In *Proceedings of the 12th international conference on database theory*, pages 276–283. ACM, 2009.

[15] R. Chan, Q. Yang, and Y.-D. Shen. Mining high utility itemsets. In *Data Mining, 2003. ICDM 2003. Third IEEE International Conference on*, pages 19–26. IEEE, 2003.

[16] K.-T. Chuang, J.-L. Huang, and M.-S. Chen. Power-law relationship and self-similarity in the itemset support distribution: analysis and applications. *The VLDB Journal*, 17(5):1121–1141, 2008.

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

[18] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, Jan. 2008.

[19] P. Fournier-Viger. `http://www.philippe-fournier-viger.com/spmf/index.php?link=datasets.php`.

[20] A. W.-c. Fu, R. W.-w. Kwong, and J. Tang. Mining n-most interesting itemsets. In *Foundations of Intelligent Systems*, pages 59–67. Springer, 2000.

[21] C. Giannella. `http://www.cs.loyola.edu/~cgiannel/assoc_gen.html`.

[22] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. *SIGMOD Rec.*, 29(2):1–12, May 2000.

[23] S. Har-Peled and M. Sharir. Relative (p, ε)-approximations in geometry. *Discrete & Computational Geometry*, 45(3):462–496, 2011.

[24] W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):pp. 13–30, 1963.

[25] C. Jia and R. Lu. Sampling ensembles for frequent patterns. In *Fuzzy Systems and Knowledge Discovery*, pages 1197–1206. Springer, 2005.

[26] H. Kwak, C. Lee, H. Park, and S. Moon. `http://an.kaist.ac.kr/traces/WWW2010.html`.

[27] H. Li, Y. Wang, D. Zhang, M. Zhang, and E. Y. Chang. Pfp: parallel fp-growth for query recommendation. In *Proceedings of the 2008 ACM conference on Recommender systems*, pages 107–114. ACM, 2008.

[28] L. Li and M. Zhang. The strategy of mining association rule based on cloud computing. In *Business Computing and Global Informatization (BCGIN), 2011 International Conference on*, pages 475–478. IEEE, 2011.

[29] Y. Li and R. P. Gopalan. Effective sampling for mining association rules. In *AI 2004: Advances in Artificial Intelligence*, pages 391–401. Springer, 2005.

[30] Y.-C. Li, J.-S. Yeh, and C.-C. Chang. Direct candidates generation: a novel algorithm for discovering complete share-frequent itemsets. In *Fuzzy Systems and Knowledge Discovery*, pages 551–560. Springer, 2005.

[31] Y.-C. Li, J.-S. Yeh, and C.-C. Chang. A fast algorithm for mining share-frequent itemsets. In *Web Technologies Research and Development-APWeb 2005*, pages 417–428. Springer, 2005.

[32] Y.-C. Li, J.-S. Yeh, and C.-C. Chang. Isolated items discarding strategy for discovering high utility itemsets. *Data & Knowledge Engineering*, 64(1):198–217, 2008.

[33] K.-L. Lin, C.-N. Hsu, H.-S. Huang, and C.-N. Hsu. A recommender for targeted advertisement of unsought products in e-commerce. In *E-Commerce Technology, 2005. CEC 2005. Seventh IEEE International Conference on*, pages 101–108. IEEE, 2005.

[34] G. Liu, J. Li, and L. Wong. A new concise representation of frequent itemsets using generators and a positive border. *Knowledge and Information Systems*, 17(1):35–56, 2008.

[35] M. Liu and J. 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.

[36] Y. Liu, W.-k. Liao, and A. Choudhary. A fast high utility itemsets mining algorithm. In *Proceedings of the 1st International Workshop on Utility-based Data Mining*, UBDM '05, pages 90–99, New York, NY, USA, 2005. ACM.

[37] H. Mannila, H. Toivonen, and A. I. Verkamo. Efficient algorithms for discovering association rules. In *KDD-94: AAAI workshop on Knowledge Discovery in Databases*, pages 181–192, 1994.

[38] H. Mannila, H. Toivonen, and A. I. Verkamo. Efficient algorithms for discovering association rules. In *KDD-94: AAAI workshop on Knowledge Discovery in Databases*, pages 181–192, 1994.

[39] R. Rymon. Search through systematic set enumeration. *Technical Reports (CIS)*, page 297, 1992.

[40] H. Toivonen. Sampling large databases for association rules. In *VLDB*, volume 96, pages 134–145, 1996.

[41] 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*, 25:1772–1786, 2013.

[42] V. S. Tseng, C.-W. Wu, B.-E. Shie, and P. 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.

[43] J. Wang, J. Han, Y. Lu, and P. Tzvetkov. Tfp: An efficient algorithm for mining top-k frequent closed itemsets. *Knowledge and Data Engineering, IEEE Transactions on*, 17(5):652–663, 2005.

[44] R. S. Xin, J. Rosen, M. Zaharia, M. J. Franklin, S. Shenker, and I. Stoica. Shark: Sql and rich analytics at scale. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, SIGMOD '13, pages 13–24, New York, NY, USA, 2013. ACM.

[45] X. Y. Yang, Z. Liu, and Y. Fu. Mapreduce as a programming model for association rules algorithm on hadoop. In *Information Sciences and Interaction*

Sciences (ICIS), 2010 3rd International Conference on, pages 99–102. IEEE, 2010.

[46] H. Yao, H. J. Hamilton, and C. J. Butz. A foundational approach to mining itemset utilities from databases. In *SDM*, volume 4, pages 215–221. SIAM, 2004.

[47] J. X. Yu, Z. Chong, H. Lu, and A. Zhou. False positive or false negative: Mining frequent itemsets from high speed transactional data streams. In *Proceedings of the Thirtieth International Conference on Very Large Data Bases - Volume 30*, VLDB '04, pages 204–215. VLDB Endowment, 2004.

[48] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, pages 2–2. USENIX Association, 2012.

[49] M. J. Zaki, S. Parthasarathy, W. Li, and M. Ogihara. Evaluation of sampling for data mining of association rules. In *Research Issues in Data Engineering, 1997. Proceedings. Seventh International Workshop on*, pages 42–50. IEEE, 1997.

[50] C. Zhang, S. Zhang, and G. I. Webb. Identifying approximate itemsets of interest in large databases. *Applied Intelligence*, 18(1):91–104, 2003.

[51] Y. Zhao, C. Zhang, and S. Zhang. Efficient frequent itemsets mining by sampling. In *AMT*, pages 112–117, 2006.