Appl Intell (2017) 47:809–827
DOI 10.1007/s10489-017-0932-1

A hybrid framework for mining high-utility itemsets in a sparse transaction database

Siddharth Dawar[1] · **Vikram Goyal**[1] · **Debajyoti Bera**[1]

Published online: 25 April 2017
© Springer Science+Business Media New York 2017

Abstract High-utility itemset mining aims to find the set of items with utility no less than a user-defined threshold in a transaction database. High-utility itemset mining is an emerging research area in the field of data mining and has important applications in inventory management, query recommendation, systems operation research, bio-medical analysis, etc. Currently, known algorithms for this problem can be classified as either 1-phase or 2-phase algorithms. The 2-phase algorithms typically consist of tree-based algorithms which generate candidate high-utility itemsets and verify them later. A tree data structure generates candidate high-utility itemsets quickly by storing some upper bound utility estimate at each node. The 1-phase algorithms typically consist of inverted-list based and transaction projection based algorithms which avoid the generation of candidate high-utility itemsets. The inverted list and transaction projection allows computation of exact utility estimates. We propose a novel hybrid framework that combines a tree-based and an inverted-list based algorithm to efficiently mine high-utility itemsets. Algorithms based on the framework can harness benefits of both types of algorithms. We report experiment results on real and synthetic datasets to demonstrate the effectiveness of our framework.

✉ Vikram Goyal
vikram@iiitd.ac.in

Siddharth Dawar
siddharthd@iiitd.ac.in

Debajyoti Bera
dbera@iiitd.ac.in

[1] Department of Computer Science, Indraprastha Institute of Information Technology, Delhi, India

Keywords Data mining · Mining methods and algorithms · Pattern growth mining · Frequent pattern mining · Utility mining

1 Introduction

Frequent itemset mining [1, 13, 15] finds the set of items in a transaction database with a frequency no less than a user defined frequency threshold. It finds applications in mining association rules, supermarket shelf management, mining frequent itemsets from web logs and a part of many important data mining tasks like clustering, classification, etc. *Frequent itemset mining* only considers the presence or absence of items in the database and assumes that all items have equal importance. However, in real life, items in a transaction can have different quantity (often known as the internal utility of the items) and generate different profit (often known as the external utility of the items). For example, someone may buy six boxes of DVDs, one video player from a store and furthermore, the store will not make same profit with each item.

High-utility itemset mining [20, 21, 25, 29] has emerged as a research area to address these issues. A conventional approach is to define the utility of an item in a transaction as the product of its quantity and its associated profit. The utility of an itemset in a transaction is defined as the sum of the utility of individual items. However, the utility can be defined according to a given application domain. For example, consider the problem of query expansion where an objective is to recommend query words to a user to improve her search results from a documents collection. Here, each document can be modeled as a transaction consisting of words as its items, and frequency of a word as its quantity. The relative importance of each word can be modeled by

its inverse document frequency (IDF) over the corpus of documents.

Utility-mining also finds its applications in cross-marketing in retail stores [8, 17], web click stream analysis [16, 25], bio-medical data analysis [6], and mobile commerce environment planning [26]. Utility-mining has also been applied with other mining techniques like sequential-pattern mining [32, 33], episode-pattern mining [24, 31], stream mining [4, 16], maximal high-utility pattern mining [18, 27], and high-utility rare pattern mining [12]. We use the terms "itemset" and "pattern" interchangeably in this paper.

Existing algorithms for high-utility itemset mining can be classified into two different paradigms: 1-phase and 2-phase algorithms. The tree-based algorithms [28] are 2-phase algorithms which find candidate high-utility itemsets in the first phase and verify them in the second phase. The bottleneck of tree-based algorithms lies in verifying the candidates to find high-utility itemsets among the candidates. However, they are generally fast in computing estimates of utility for an itemset. Inverted-list is another data structure for storing the database information. Inverted-list associated with an item, stores a list of transactions which contain the associated item. The vertical mining algorithms [19] use an inverted-list data structure for its working and are 1-phase. Liu et al. [19] proposed a utility-list data structure based on inverted-list and an algorithm which finds $\{k\}$-length high-utility itemsets by intersecting the utility-lists of $\{k-1\}$-length itemsets. Viger et al. [10] proposed a strategy to improve the performance of HUI-Miner by reducing the number of intersection operations. These vertical mining algorithms use simple intersection operation on inverted-lists of $\{k-1\}$-length itemsets to calculate the support of an $\{k\}$-length itemset. Solutions based on the vertical mining have been shown to perform better as compared to the tree-based approaches as they compute the estimate of utility correctly. However, the cost of intersection operations is higher for smaller itemsets as compared to that for larger ones. The cost is higher due to the larger size of lists usually associated with smaller itemsets.

To validate our hypothesis that algorithms based on our hybrid framework will perform better due to avoidance of costly intersection operations and candidate verification, we conduct an experiment on a real life dataset called Accidents available from the FIMI repository [11]. The dataset is augmented with quantity and utility values for items as per previous works in the literature [19, 29, 36]. The results of the experiment are shown in Fig. 1. We observe that there are few high-utility itemsets of small length, where the length of an itemset is the number of individual items contained in it. Smaller itemsets have large utility-lists as they occur in a large number of transactions. The size of utility-lists decreases as the number of transactions containing an itemset decreases with the increase in itemset length. Therefore, a tree-based algorithm can be used in the beginning to avoid cost due to joining of long lists associated with short itemsets.

Transaction databases can be categorized by the average transaction length and the number of items. Datasets with small average transaction length and a large number of items are known as sparse datasets. Transaction databases with large average transaction length and a small number of items are known as dense datasets. Dense datasets are known to generate a large number of long high-utility itemsets. Ahmed et al. [3] proposed a measure R to characterize the nature of a dataset. They used the ratio of average transaction length to the number of items to characterize sparsity. If the ratio is less than 1%, the dataset can be sparse, otherwise dense. Sparse datasets are generated by retail stores, web click stream data, etc. Retail stores like Walmart usually sell millions or billions of products. On the other hand, customers purchase very few items at a time. So, transactions generated by such retail stores are usually sparse with small average transaction length and a large number of items to chose from. Similarly, E-commerce vendors like Amazon, Snapdeal, etc. sell million of products online but

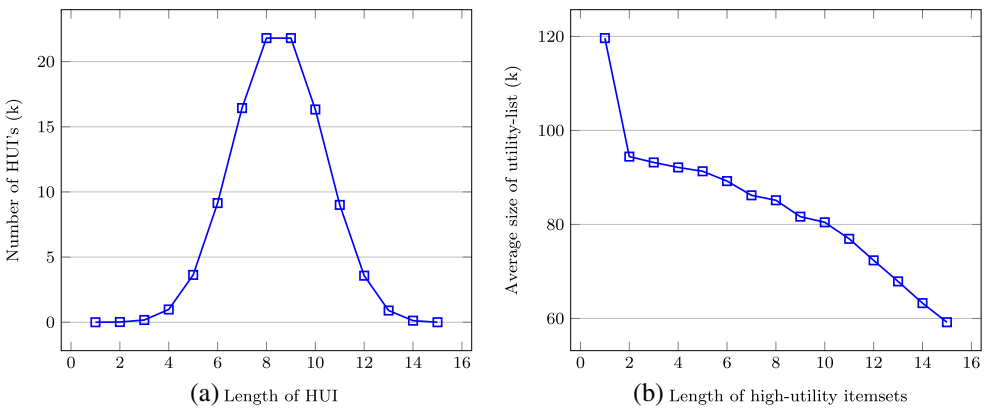


Fig. 1 Effect of pattern size on the number of high-utility itemsets and utility-list size on Accidents dataset

Table 1 Characteristics of real datasets

Dataset	#Tx	Avg. length (A)	#Items (D)	Density score R (%) R=(A/D)x100	Type
Retail	88,162	10.3	16,470	0.0625	Sparse
Chainstore	11,12,949	7.2	46,086	0.0156	Sparse
Kosarak	9,90,002	8.1	41,270	0.0196	Sparse
OnlineRetail	5,40,555	4.37	2,603	0.167	Sparse
BMS WebView 1	59,602	2.51	497	0.50	Sparse
BMS WebView 2	77,512	4.62	3,340	0.13	Sparse
PowerC	10,40,000	7	125	5.6	Dense
KDDCup99	10,00,000	16	135	11.8	Dense
Mushroom	8,614	23	119	19.32	Dense
Connect	67,557	43	129	33.33	Dense
Accidents	3,40,183	33.8	468	7.2	Dense

have small transaction length. In real life, sparse datasets are abundant, and many experimental sparse datasets are available like Retail, ChainStore, Kosarak, etc. as shown in Table 1.

Recently, Zida et al. [36] proposed a depth-first search based recursive algorithm *EFIM*, which is a 1-phase algorithm. It generates projected databases to mine high-utility itemsets. The algorithm uses transaction merging at each step to reduce the size of the projected databases during recursive invocations. *EFIM* was shown to have 10 to 100 times more efficiency compared to *FHM* on dense datasets. Like other 1-phase algorithms, it computes the utility estimates correctly. The key to superior performance of *EFIM* lies in transaction merging which reduces the size of the projected database significantly during recursive invocations. We also observed that transaction merging reduces size of the database by 10 to 500 times for dense datasets.

In this paper, we propose a novel framework, which combines the advantages of tree-based and vertical mining algorithms for generating high-utility itemsets. The framework exploits the benefit of both approaches, i.e. efficiently compute utility estimates like tree-based algorithms and prune non-candidates using correct estimates of utility like inverted-list algorithms. We demonstrate an application of the framework by combining UP-Growth+ and FHM algorithms. Our contributions can be summarized as follows:

1. We identify the issues with the tree-based and inverted-list based approaches and propose a novel hybrid framework which obtains the benefits of these approaches. Our framework can combine any tree-based algorithm like UP-Growth+ [29], MU-Growth [34], UP-Hist Growth [7] etc. with inverted-list based algorithms like HUI-Miner [19], FHM [10].
2. We design an efficient hybrid algorithm called *UFH* based on the integration of UPGrowth+ and FHM. We also discuss several optimization techniques like memoization, early termination and transaction merging to enhance the performance of our hybrid algorithm.[1]
3. We demonstrate the benefit of hybridization for sparse datasets by empirically comparing *UFH* to the state-of-the-art algorithms, FHM and EFIM. FHM has been previously shown to be better compared to the state-of-the-art tree-based algorithms.

The paper is organized as follows. Section 2 reviews the related work and problem statement is defined in Section 3. We describe our framework in Section 4 which shows how to combine the ideas of tree-based and inverted-list based approach. We study the integration of UP-Growth+ and FHM algorithm in Section 5. The experimental results are presented in Section 6 and Section 7 concludes the paper.

2 Related work

Frequent itemset mining [1, 13, 15] has been studied extensively in the literature. Agrawal et al. [1] proposed two algorithms called Apriori and Apriori-TID for mining association rules from market-basket data. Their algorithms were based on the downward closure property [1]: Every subset of a frequent itemset is also frequent. The algorithms explore the search space in a level-wise manner. Candidate itemsets having k items is generated by joining itemsets having $k-1$ items. The Apriori algorithm uses the database

[1] In an earlier work [5], we designed a similar hybrid algorithm for solving a similar problem of mining high-utility itemsets with discounts where UP-Hist Growth [7] and FHM [10] were combined.

to count the support of candidate itemsets. Apriori-TID algorithm encodes the candidate itemsets and uses them for counting purpose to reduce the effort in reading the database. Park et al. [22] proposed a hash-based algorithm which generates less number of candidates compared to Apriori algorithm. Zaki et al. [35] proposed an algorithm called ECLAT, which uses inverted-list of transactions associated with each itemset to find a set of potentially maximal frequent itemsets. Han et al. [13] proposed a pattern-growth algorithm to find frequent itemsets by using a data structure known as FP-tree.

Some hybrid algorithms were also proposed to find frequent itemsets from sequence and transaction databases. Agrawal et al. [1] proposed an algorithm called Apriori-Hybrid which combines the best features of Apriori and Apriori-TID algorithms. The algorithm uses Apriori in the initial passes and switches to Apriori-TID when the set of generated candidates at the end of a pass is expected to fit into main memory. Vu et al. [30] proposed a hybrid algorithm which combined FP-Growth and Eclat algorithms to mine frequent itemsets from a transaction database. However, frequent itemset mining algorithms cannot be used to find high-utility itemsets as it is not necessarily true that a frequent itemset is also a high-utility itemset in the database. Moreover, an infrequent itemset can also have high-utility and will be missed by a frequent itemset mining algorithm. Mining high-utility itemsets is even more challenging compared to the frequent "itemsets" mining, as there is no downward closure property.

High-utility itemset mining is a natural extension to frequent itemset mining and has also received significant research attention. Several tree-based algorithms have been proposed to find high-utility itemsets of which an important one is a two-phase algorithm given by Liu et al. [21]. The candidate high-utility itemsets are generated in the first phase and verified in the second phase. Ahmed et al. [2] proposed a data structure called IHUP-tree and another two-phase algorithm to mine high-utility patterns incrementally from dynamic databases. However, the above algorithms generate a lot of candidate itemsets in the first phase. In order to reduce the number of candidates, Tseng et al. [28] proposed a new data structure called UP-Tree and proposed two algorithms namely, UP-Growth [28] and UP-Growth+ [29]. The authors also presented some effective strategies to reduce the overestimated utilities. Yun et al. [34] proposed a tree data structure called MIQ-tree, which is similar to UP-tree and stores the maximum utility of an item in each node of the tree. However, the maximum utility information is stored only in the global MIQ tree. Yun et al. [34] proposed a recursive two-phase algorithm called MU-Growth which is similar to UP-Growth+. Dawar et al. [7] proposed another data structure called UP-Hist tree which stores a histogram of quantity information with each node of the tree. The idea behind associating more information with the nodes of a tree is that it helps in reducing the number of candidates generated by using better utility estimates. The problem with the tree-based algorithms is that the generation and verification of a vast number of candidate itemsets often result in poor performance.

In the category of inverted-list based approaches to mine high-utility itemsets, there are mainly two algorithms, HUI-Miner [19] and FHM [10]. Liu et al. [19] proposed a new data structure called utility-lists and an algorithm HUI-Miner for mining high-utility itemsets. The algorithm intersects the utility-list of itemset X with the utility-list of each item in the external list to generate the utility-lists of supersets of X. The algorithm avoids the costly generation and verification of candidates by storing the exact utility of the itemset and expected utility values of its supersets in the utility-lists. However, the joining of utility-lists of an itemset to produce a new itemset is a costly operation. In order to reduce the number of join operations, Viger et al. [10] proposed a novel data structure EUCS(Estimated Utility Co-occurrence Structure) to prune itemsets without performing the join operation.

Recently, a couple algorithms have been proposed which use projection techniques to mine high-utility itemsets. Lan et al. [14] proposed an efficient projection-based indexing approach for mining high-utility itemsets. The algorithm uses an indexing structure to find candidate high-utility itemsets and utilizes the concept of projection to find the transactions containing the itemsets. Zida et al. [36] proposed an algorithm called EFIM which uses projection techniques along with transaction merging to find high-utility itemsets without generating any candidate high-utility itemsets.

3 Problem statement

We have a set of m items $I = \{i_1, i_2, ..., i_m\}$, where each item i has a positive external utility $eu(i)$ associated with it. Every item i in a transaction T has an internal utility $iu(i, T)$ associated with it. An itemset X of length k is a set of k distinct items $\{i_1, i_2, ..., i_k\} \subseteq I$. A transaction database $D = \{T_1, T_2, ..., T_n\}$ is a set of n transactions, where every transaction has an associated itemset.

Definition 1 (Utility of an item in a transaction) The utility of an item i in a transaction T is denoted as $u(i, T)$

Fig. 2 Example database

TID	Transaction	TU
T_1	$(A:1)\,(C:10)\,(D:1)$	17
T_2	$(A:2)\,(C:6)\,(E:2)\,(G:5)$	27
T_3	$(A:2)\,(B:2)\,(D:6)\,(E:2)\,(F:1)$	37
T_4	$(B:4)\,(C:13)\,(D:3)\,(E:1)$	30
T_5	$(B:2)\,(C:4)\,(E:1)\,(G:2)$	13
T_6	$(A:6)\,(B:1)\,(C:1)\,(D:4)\,(H:2)$	43

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

and defined as the product of its internal utility and external utility i.e., $u(i, T) = iu(i, T) * eu(i)$.

Definition 2 (Utility of an itemset in a transaction) The utility of an itemset X in a transaction T is denoted as $u(X, T)$ and defined by $\sum_{\substack{X \subseteq T \\ i \in X}} u(i, T)$.

Consider our example database shown in Fig. 2. The utility of item $\{A\}$ in T_3 is $u(\{A\}, T_3) = 2 \times 5 = 10$ and $u(\{A, B\}, T_3) = u(A, T_3) + u(B, T_3) = 10 + 4 = 14$.

The utility of an itemset over the database is computed by adding the utility value of the itemset in each transaction of the database.

Definition 3 (Utility of an itemset in database) The utility of an itemset X in database D is denoted as $u(X)$ and defined as $\sum_{\substack{X \subseteq T \\ T \in D}} u(X, T)$.

For example, $u(\{A, B\}) = u(\{A, B\}, T_3) + u(\{A, B\}, T_6) = 14 + 32 = 46$.

Definition 4 (High-utility itemset) An itemset is called a high-utility itemset if its utility is no less than a given minimum user-defined threshold denoted by min_util.

For example, $u(\{A, C\}) = u(\{A, C\}, T_1) + u(\{A, C\}, T_2) + u(\{A, C\}, T_6) = 15 + 16 + 31 = 62$. If $min\_util = 50$, then $\{A, C\}$ is a high-utility itemset. However, if $min\_util = 75$, then $\{A, C\}$ is a low utility itemset.

Problem statement Given a transaction database D and a minimum user-defined threshold min_util, the aim is to enumerate all high-utility itemsets.

In frequent itemset mining, all subsets of a frequent itemset are frequent. All supersets of an infrequent itemset are infrequent. This property is known as the downward closure property [1]. However, high-utility itemset mining does not satisfy this property. The subset of a low utility itemset can have high-utility as well as vice versa. The concept of transaction weighted utility was coined by researchers, which satisfies the downward closure property.

Definition 5 (Transaction utility) The transaction utility of a transaction T is denoted by $TU(T)$ and defined as $u(T, T)$.

For example, the transaction utility of every transaction in our example database is shown in Figure 2. Using TU value of each transaction, we can now define an upper bound utility estimate of an itemset called as Transaction-Weighted utility (TWU).

Definition 6 (TWU of an itemset) TWU of an itemset X is the sum of the transaction utilities of all the transactions containing X, which is denoted as $TWU(X)$ and defined as $\sum_{\substack{X \subseteq T \\ T \in D}} TU(T)$.

Definition 7 (High TWU itemset) An itemset X is called a high transaction weighted utility itemset if $TWU(X)$ is no less than min_util.

For any itemset X, if X is not a high TWU itemset, any superset of X can not be a high-utility itemset. In our example, $TU(T_1) = u(\{ACD\}, T_1) = 17$; $TWU(\{A\}) = TU(T_1) + TU(T_2) + TU(T_3) + TU(T_6) = 124$. If $min\_util = 60$, $\{A\}$ is a high TWU itemset. However, if $min\_util = 130$, $\{A\}$ and none of its supersets is a high TWU itemset.

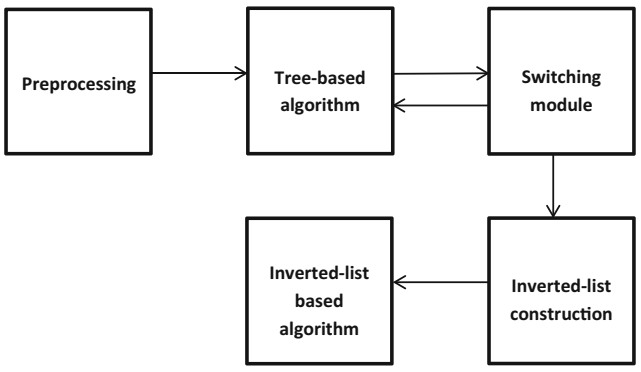


Fig. 3 Hybrid framework

4 Hybrid framework

In this section, we propose a hybrid framework for combining any tree-based and inverted-list based algorithms for mining high-utility itemsets. Our proposed framework consists of five steps and is illustrated in Fig. 3. In the pre-processing step, the transactions in the database are scanned to construct the inverted-lists of distinct items present in the database. Every transaction is inserted to construct a tree data structure. Each node of a tree consists of an item name, support count, overestimated utility and a pointer to its child node. The root of a tree is a special empty node which represents null item and points to its child nodes. A path from the root to a particular node in the tree, is the set of transactions, which contain the itemset consisting of nodes along that path. The support count of a node along a path is the number of transactions in the database that contain the itemset consisting of items on the path from the root to that node. One optimization can be applied to reduce the number of nodes of a tree. The nodes representing items whose superset itemsets can not have high-utility can be removed from the transactions before tree construction. Such items are called unpromising items. In order to apply this optimization during the construction of a tree, two scans of the database are needed. In the first scan, the overestimated utility of each item is computed by using some overestimated utility measure like *TWU*. Items with overestimated utility less than the minimum utility threshold are unpromising items and must be removed from the transaction database. The database is scanned again to remove the unpromising items from each transaction.

After construction of a tree data structure, a tree-based algorithm is called. Tree-based algorithms are recursive two-phase algorithms. The algorithm starts with an empty prefix and extends it with each item i present in the tree in a bottom-up manner. An overestimated utility of itemset is computed, and the itemset is added to the set of candidate high-utility itemsets if overestimated utility is greater than the minimum utility threshold. A local tree of the currently prefix is constructed and the algorithm is called recursively. At the end of the first phase, the complete set of candidate high-utility itemsets is generated. In the second phase, the exact utility of candidate itemsets is computed to find the complete set of high-utility itemsets.

One possibility can be to run only the tree-based algorithm and do not switch to the inverted-list based algorithm. Another possibility can be to invoke the inverted-list based algorithm directly after the pre-processing step and mine the high-utility itemsets. The above approaches have their own merits and demerits. In order to gain benefits of both the paradigms, the third possibility is to start with a tree-based algorithm and switch to an inverted-list based algorithm during some point of the execution. The switching module helps to decide a point where the execution is switched from a tree-based algorithm to an inverted-list based algorithm. There can be different criteria's which can be defined for the switching module. One possibility, can be to decide the switching criteria by observing the data distribution. Another possibility is to generate a candidate high-utility itemset from a tree-based algorithm and switch the execution to an inverted-list based algorithm. We evaluate the performance of the latter possibility in this paper.

An inverted-list based algorithm takes as input; the inverted-list of itemset I on which it will be invoked and the set of inverted-lists of $I's$ extensions. In order to ensure correctness, one can consider all the items in the set of items present in the database. However, this approach will be inefficient as it will result in the creation of inverted-lists of itemsets which either do not exist in the transaction database or are a low-utility itemset. The tree structure can be used to make this process efficient. The items which have been explored by the tree-based algorithm or items whose supersets can not be of high-utility can be removed from the set of I 1-extensions.

After creation of the inverted-lists, an inverted-list based algorithm can be called. The inverted-list based algorithm explores the search space in a level-wise manner similar to the Apriori algorithm [1]. An inverted-list based algorithm generates the inverted-list of $\{k\}$-length itemset by combining inverted-lists of $\{k-1\}$-length itemsets. After processing all the supersets of itemset I, the inverted-list based algorithm terminates, and the execution returns to the tree-based algorithm.

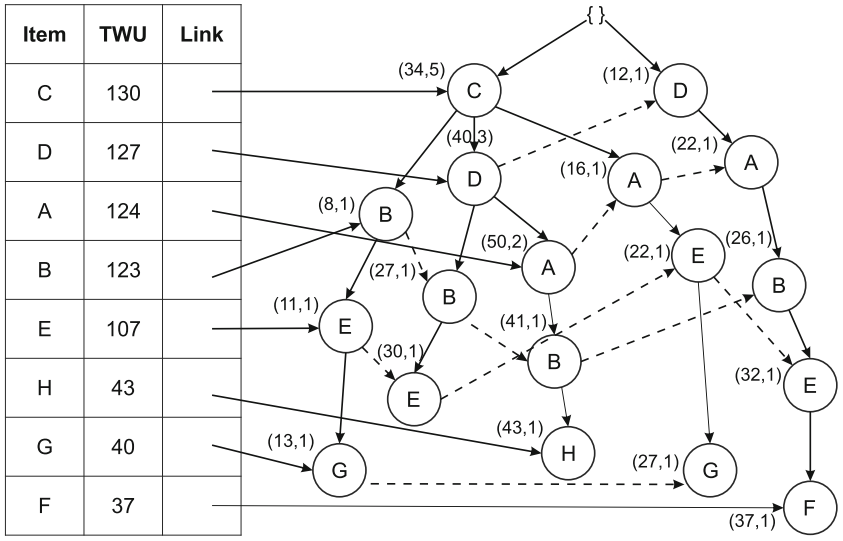
5 A hybrid algorithm for high-utility itemset mining

In this section, we will perform a case study where we will integrate UP-Growth+ and FHM algorithms into our proposed framework. We briefly discuss the state-of-the-art tree-based algorithm in Section 5.1 and inverted-list based algorithm in Section 5.2. We propose a hybrid algorithm called *UFH* in Section 5.3. We will discuss several optimization techniques to improve the performance of our hybrid algorithm in Section 5.4.

5.1 UP-tree based algorithm

UP-Growth+ [29] is a two-phase recursive algorithm based on UP-tree data structure for mining high-utility itemsets

Fig. 4 Global UP-tree



from a transaction database. Each node N in a UP-tree consists of a name $N.item$, overestimated utility $N.nu$, support count $N.count$, a pointer to the parent node $N.parent$ and a pointer $N.hlink$ to another node, which has the same name as $N.name$. The root of the tree is a special empty node which points to its child nodes. A path from the root node to any other node of the tree creates an itemset, which we call as prefix-itemset of the node. The support count of a node N defines the number of transactions that contain the prefix-itemset of the node N. $N.nu$ is the overestimated utility of the prefix-itemset. A header table is maintained to facilitate efficient traversal of the tree. The header table has three columns, *Item*, *TWU* and *Link*. The nodes in a UP-Tree along a path are maintained in descending order of their *TWU* values. All nodes with the same label are stored in a linked list, and the link pointer in the header table points to the head of this list.

In the first scan, the UP-Growth+ algorithm computes the transaction weighted utility (*TWU*) of all items. Items which have their *TWU* values less than the minimum threshold are removed from every transaction in the database as well as items in each transaction are sorted in decreasing order of *TWU* values. These reorganized transactions are then used to construct a global UP-tree. The global UP-tree for our example database is shown in Fig. 4. The algorithm extends an empty prefix by item i from the header table in a bottom-up manner. It computes the overestimated utility of the pattern by traversing the linked list associated with the item i of the pattern. If the overestimated node utility is greater than the minimum threshold, the prefix pattern is added to the set of candidate high-utility itemsets. After extending the prefix by an item i, a conditional-pattern-base is created. The conditional pattern base is the set of paths from the tree which contains the prefix. The local UP-Tree of the prefix is constructed considering the paths in the conditional-pattern-base as transactions and the algorithm mines candidate high-utility patterns recursively. After the first phase of the algorithm is complete, it scans the database once more to verify candidates for high-utility.

5.2 Inverted-list based algorithm

FHM [10] algorithm is a vertical data mining algorithm which uses a utility-list data structure for mining high-utility

Fig. 5 Reorganized transactions and TWU of items

TID	Transaction
T'_1	$(A:1)\,(D:1)\,(C:10)$
T'_2	$(G:5)\,(E:2)\,(A:2)\,(C:6)$
T'_3	$(F:1)\,(E:2)\,(B:2)\,(A:2)\,(D:6)$
T'_4	$(E:1)\,(B:4)\,(D:3)\,(C:13)$
T'_5	$(G:2)\,(E:1)\,(B:2)\,(C:4)$
T'_6	$(H:2)\,(B:1)\,(A:6)\,(D:4)\,(C:1)$

Item	A	B	C	D	E	F	G	H
TWU	124	123	130	127	107	37	40	43

TID	Iutils	Rutils
1	5	12
2	10	6
3	10	12
6	30	9

TID	Iutils	Rutils
3	4	22
4	8	19
5	4	4
6	2	39

Fig. 6 Utility-list of item $\{A\}$ and $\{B\}$

itemsets. Utility-list is a compact data structure for storing information about the transactions in which the itemset appears along with its utility value. A utility-list associated with an itemset I is a list of triples storing three columns of information: TID, $Iutils$ and $Rutils$. TID is the transaction identifier associated with each transaction in the database and $Iutils(I, T_i)$ is the exact utility of itemset I in the transaction T_i. FHM assumes that the items in a transaction are sorted in ascending order of their *TWU* values. A transaction is called as a reorganized transaction, if the items present in it are sorted in ascending order of their *TWU* values. The reorganized transactions for our example database is shown in Fig. 5. The *TWU* of items is also shown in Fig. 5. $Rutils(I, T_i)$ is the utility of items which occur after itemset I in transaction T_i according to a globally defined ordering (e.g. ascending order of TWU values). For example, the utility-list of items $\{A\}$ and $\{B\}$ is shown in Fig. 6.

In the first scan of the database, the algorithm finds out the high *TWU* items. The utility-lists of high *TWU* items are constructed in the next scan of the database. The algorithm generates the utility-list of an $\{k\}$-length itemset by combining the utility-lists of $\{k-1\}$-length itemsets. For example, the utility-list of itemset $\{AB\}$ constructed from the intersection of utility-list of item $\{A\}$ and $\{B\}$ is shown in Fig. 7.

The algorithm generates all distinct high-utility items and then proceeds to the generation of pairs, triplets, etc. The FHM algorithm doesn't generate any candidate high-utility itemsets, which need to be verified later. However, the joining of utility-lists is a costly operation. In order to reduce the number of join operations, FHM uses a pruning strategy called Estimated Utility Co-occurrence Pruning (EUCP). The pruning strategy relies on a novel data structure called *EUCS*. The *EUCS* data structure is built during the second scan of the database. It is defined as a set of triples (a, b, c) such that $TWU(a, b)$=c. The *EUCS* structure for our example database is shown in Fig. 8. Suppose we have an itemset Px whose supersets can be explored to find high-utility itemsets i.e. $sumIutils + sumRutils \geq threshold$. Before extending the itemset Px with item y, which comes after item x in the ordering of items, the algorithm will check if there exists an entry in the *EUCS* structure, where *TWU(x,y)* $\geq$ threshold. If no such entry exists, no super-set of Pxy will be explored further.

TID	Iutils	Rutils
3	14	12
6	32	9

Fig. 7 Utility-list of itemset $\{AB\}$

5.3 UFH: a hybrid algorithm by integrating UP-Growth+ and FHM

UFH is a recursive algorithm, which requires two scans of the database. In the first scan, the *TWU* of distinct items present in the database is computed. Items which have their *TWU* less than the minimum utility threshold are removed from transactions in the next scan of the database. The items in each transaction are sorted in decreasing order of *TWU* values and inserted to form a global UP-Tree. The utility-list of items which have their *TWU* no less than the minimum threshold is also created. After the completion of Pre-processing phase, UP-Growth+ algorithm is called. UP-Growth+ algorithm picks the first entry from the bottom of the header table and constructs a new itemset I, by appending the item picked from the header table to the current prefix. The transaction weighted utility(*TWU*) of itemset I is computed. If the *TWU* is greater than or equal to the minimum threshold, we compute an upper bound utility estimate of the currently processed itemset. If this estimated utility value satisfies the minimum threshold condition, there is a possibility of high-utility itemsets being generated from the currently processed itemset. The set of prefix $\{1\}$-length extensions of itemset I is constructed from the conditional pattern base of itemset I. After construction of the utility-lists of prefix $\{1\}$-length extensions, FHM algorithm is invoked. Else, the local tree is generated, and a tree-based algorithm is called recursively. We discuss the process of creating utility-list of any itemset in Section 5.4. The upper bound utility estimate is defined below.

Item	A	B	C	D	E	F	G
B	80						
C	87	56					
D	97	110	90				
E	64	67	70	67			
F	37	37	0	37	37		
G	27	13	40	0	40	0	
H	43	43	43	43	0	0	0

Fig. 8 EUCS data structure

Algorithm 1 UFH algorithm

Input: Transaction database D and minimum utility threshold min_util.
Output: Complete set of high-utility itemsets.

Scan D once to find the unpromising items.
Scan the database again to remove the unpromising items from each transaction.
Perform transaction merging.
Sort the items in the transactions in descending order of *TWU* values.
Insert all the reorganized transactions to form a UP-tree T with header table H.
Construct the utility-list of the promising items.
Call UP-Growth+(T, H, {}, min_util).

function UP-GROWTH+(T, H, Y, min_util) ▷ UP-Growth+ [29] with switching module.
  for each entry $\{i\}$ in H **do**
    Compute $node.nu$ by following the links from the header table for $\{i\}$.
    Itemset $I = Y \cup i$. ▷ Append the extension i to the current prefix Y.
    Compute the upper bound utility value for itemset I ($ub(\{I\})$).
    if $node.nu(i) \geq min\_util$ **then**
      if $ub(I) \geq min\_util$ **then** ▷ Switching criteria in UFH algorithm.
        Construct the utility-list of I and call FHM(I, extensions of I, min_util, EUCS).
        return.
      else
        Construct the conditional pattern base of itemset I.
      end if
      Put local promising items in the conditional pattern base of I and apply DLU strategy.
      Construct the local UP-tree (T_I) with header table (H_I) .
      if $T_I \neq null$ **then**
        Call UP-Growth+(T_I, H_I, I)
      end if
    end if
  end for
end function

function FHM(I,Extensions of I (Ext_I),min_util, EUCS) ▷ FHM algorithm [10].
  for each itemset Ix in Ext_I **do**
    if (**then** $Ix.utilitylist.sumIutils \geq min\_util$))
      Output Ix as a high-utility itemset.
    end if
    if (**then** $Ix.utilitylist.sumIutils + Ix.utilitylist.sumRutils \geq min\_util$))
      Ext_Ix={}.
      for each itemset Iy in Ext_I such that y comes after x **do**
        if $\exists(x, y, c)$ in EUCS such that $c \geq min\_util$ **then**
          Ixy=$Ix \cup Iy$.
          $Ixy.utilitylist$=Construct(I, Ix, Iy).
          Ext_Ix=Extensions of the itemset $\{Ix \cup Ixy\}$.
        end if
      end for
      FHM (Ix, Ext_Ix, min_util, EUCS).
    end if
  end for
end function

Definition 8 (Upper bound utility value) For a given support count s and for any path in the tree having itemset $I = \langle a_1, a_2, ..., a_k \rangle$, we define *Upper bound utility value* of I, denoted by $ub(I)$, as

$$ub(I) = \sum_{i=1}^{k} \max_{j} \{u(a_i, T_j) \mid \forall T_j \in D\} \times s$$

We used the implementation of UP-Growth+ algorithm available in SPMF library [9]. The implementation available in the library sorts the transactions in the local tree according to decreasing order of the path utility of items. Items which have the same path utility are sorted lexicographically. This ordering at a local level can cause some 1-length extensions of an itemset to be missed during the construction of utility-lists before invoking FHM. In order to resolve the issue, we keep the ordering of items intact along with a path in the local tree. For example, consider the database shown in Fig. 2 and let the minimum threshold be 30. Let us process item $\{H\}$ from the global UP-tree shown in Fig. 4. The conditional pattern base of item H consists of a single path $\langle CDAB : 43 \rangle$ with path utility 43. Since the path utility of all the items is same i.e. 43, the SPMF implementation of the UP-Growth+ algorithm sorts the items lexicographically. The reorganized path becomes $\langle ABCD \rangle : 43$ and inserted to form the local tree of $\{H\}$. Let us now observe the processing of itemset $\{HA\}$. The upper bound utility of itemset HA is 32, which is greater than the minimum utility threshold. Therefore, the algorithm computes the {1}-length extensions of $\{HA\}$ from its conditional pattern base. Since the conditional pattern base of $\{HA\}$ is empty, no prefix extensions are considered. However, $\{HAC\}$ is a high-utility itemset with utility 33. If we keep the ordering of the global tree intact, conditional pattern base of $\{HA\}$ would have path $\langle CD \rangle$ in it, and no prefix extensions would have been missed.

5.4 Optimization techniques for implementing UFH algorithm

Suppose, we have an itemset $I = \langle a_1, a_2, ..., a_k \rangle$ and the upper bound utility value of I is greater than the threshold. There are several ways to construct the utility-list of itemset I before calling the inverted-list based algorithm. A naive approach, is to construct the utility-list of I from scratch by joining the utility-lists of all the items present in this itemset. The problem with this approach is that constructing utility-lists from scratch is inefficient. We use memoization technique to make this construction process efficient. After constructing the utility-list of an itemset I, we store it for later reuse. Before the construction of the utility-list of any itemset I, we will check if the utility-list of any subset of I is stored in memory. If subsets exist, we will reuse the utility-list of the subset of maximum length for constructing the utility-list of itemset I.

We use another optimization while constructing the utility-list of an itemset I for the first time. We fetch the utility-list of items a_1, a_2 and join them to construct the utility-list of itemset $\{a_1a_2\}$. A check is performed to find out if the supersets of this intermediate itemset will be of high-utility or not by using the pruning strategy proposed by HUI-Miner [19]. If yes, we proceed with the construction process. Else, we add this intermediate itemset into a list which we call the black-list. Before invoking our algorithm on any prefix, we check if the current prefix or its subset is in the black-list or not. If the prefix is in the black-list, we know that it is not worthy to extend this prefix further. If the current prefix is in the black-list, we don't invoke the tree-based algorithm on that prefix and proceed with the next entry from the header table. We will refer to this strategy as the early termination strategy.

EFIM [36] uses the concept of transaction merging to reduce the cost of database scans. After the removal of unpromising items, it is possible that some transactions become identical i.e. containing the same set of items. Such transactions can be merged to create a single transaction. The internal utility value of each item in the new transaction is equal to the sum of its internal utility in the different identical transactions. We perform transaction merging once before the starting the mining process.

6 Experiments and results

In this section, we compare the performance of our proposed hybrid algorithm against the state-of-the-art algorithms FHM [10] and EFIM [36]. We obtained the Java source code of UP-Growth+, FHM and EFIM algorithm from the SPMF library [9]. The experiments were performed on an Intel Xeon(R) CPU=26500@2.00 GHz with 16 GB free RAM and Windows 8 operating system. We compared the performance of the algorithms by total execution time as well as the number of intermediate itemsets generated by algorithms. The number of intermediate generated itemsets gives an idea about the search space explored by the algorithm. We call the intermediate generated itemsets candidates, as it contains the set of high-utility itemsets. We also observe the memory consumed by different algorithms. We use the JVisual VM coupled with Java Development Kit for observing the memory consumed by algorithms. In our experiments, the utility values are expressed in terms of percentage of the total transaction utility of the database. Every algorithm is executed five times and an average is taken for total time and memory.

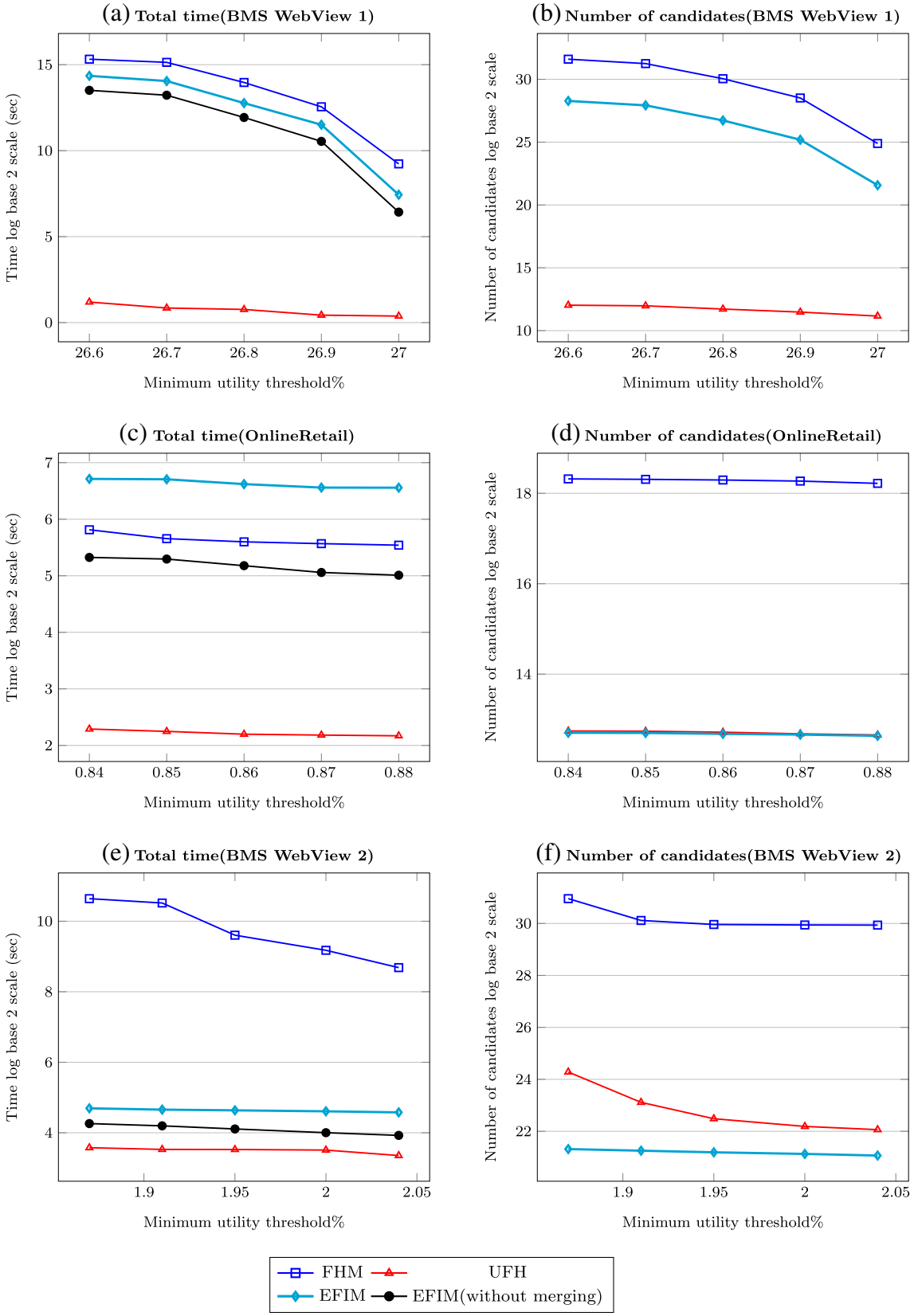


Fig. 9 Performance evaluation on real sparse datasets

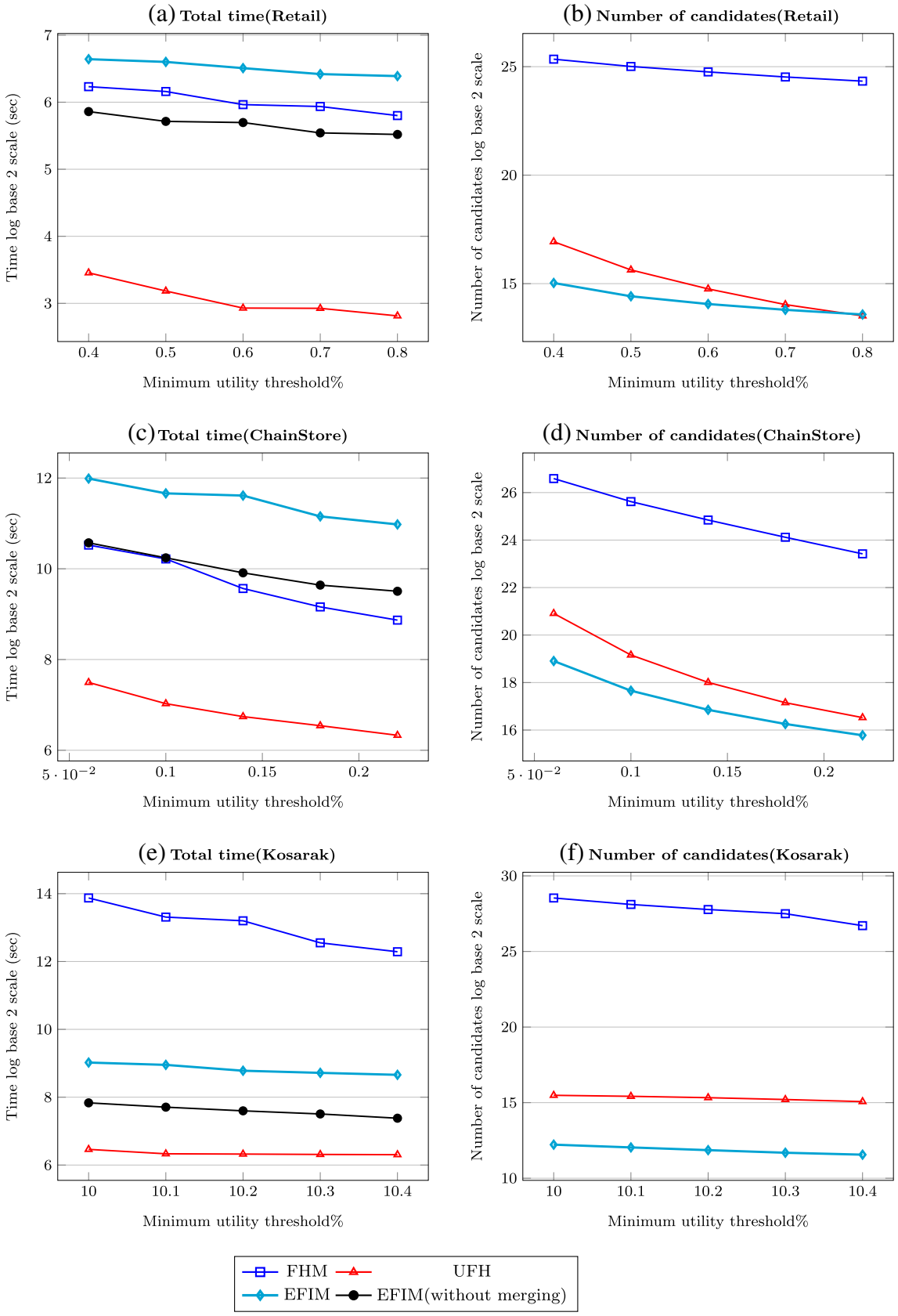


Fig. 10 Performance evaluation on real sparse datasets

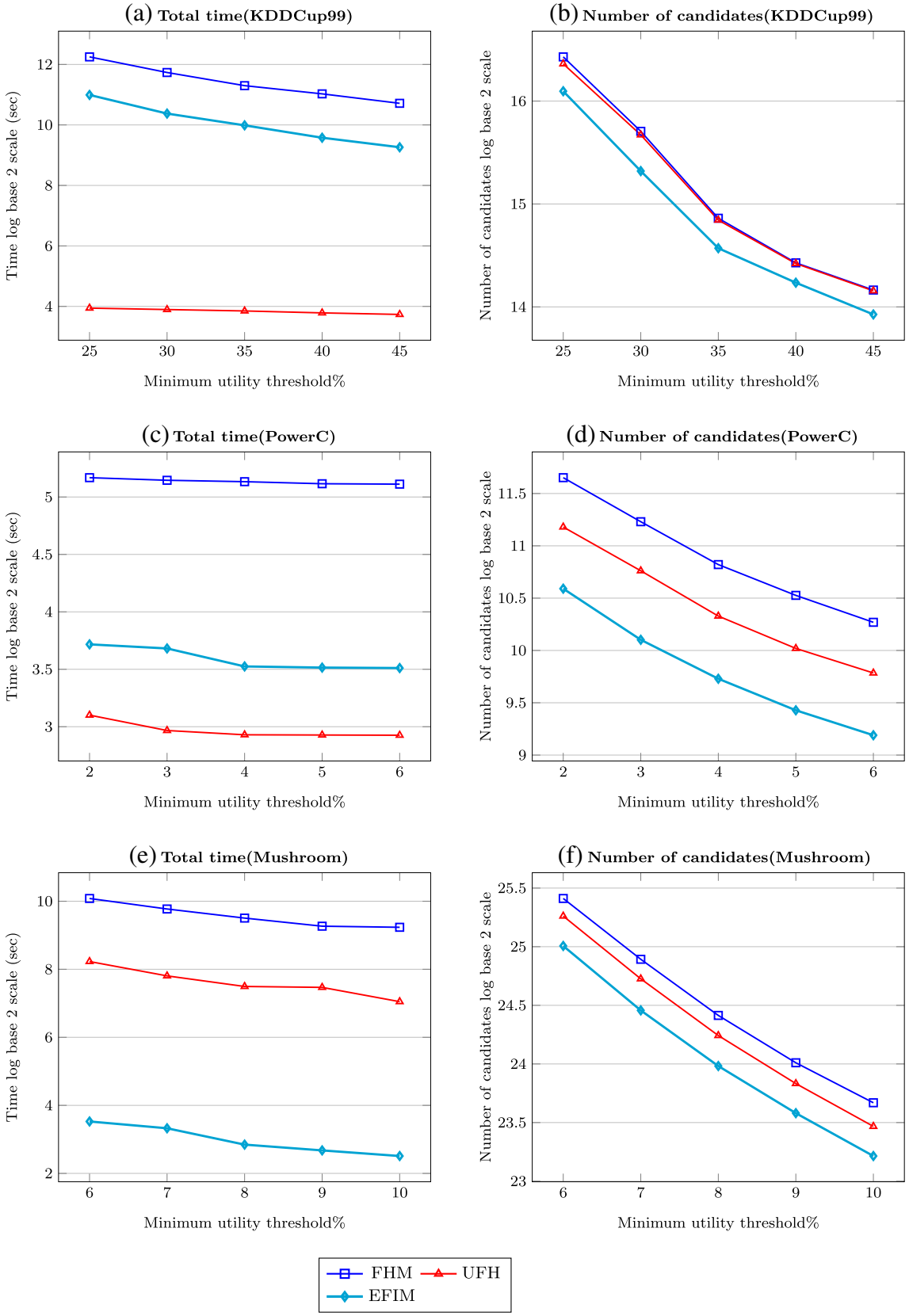


Fig. 11 Performance evaluation on real dense datasets

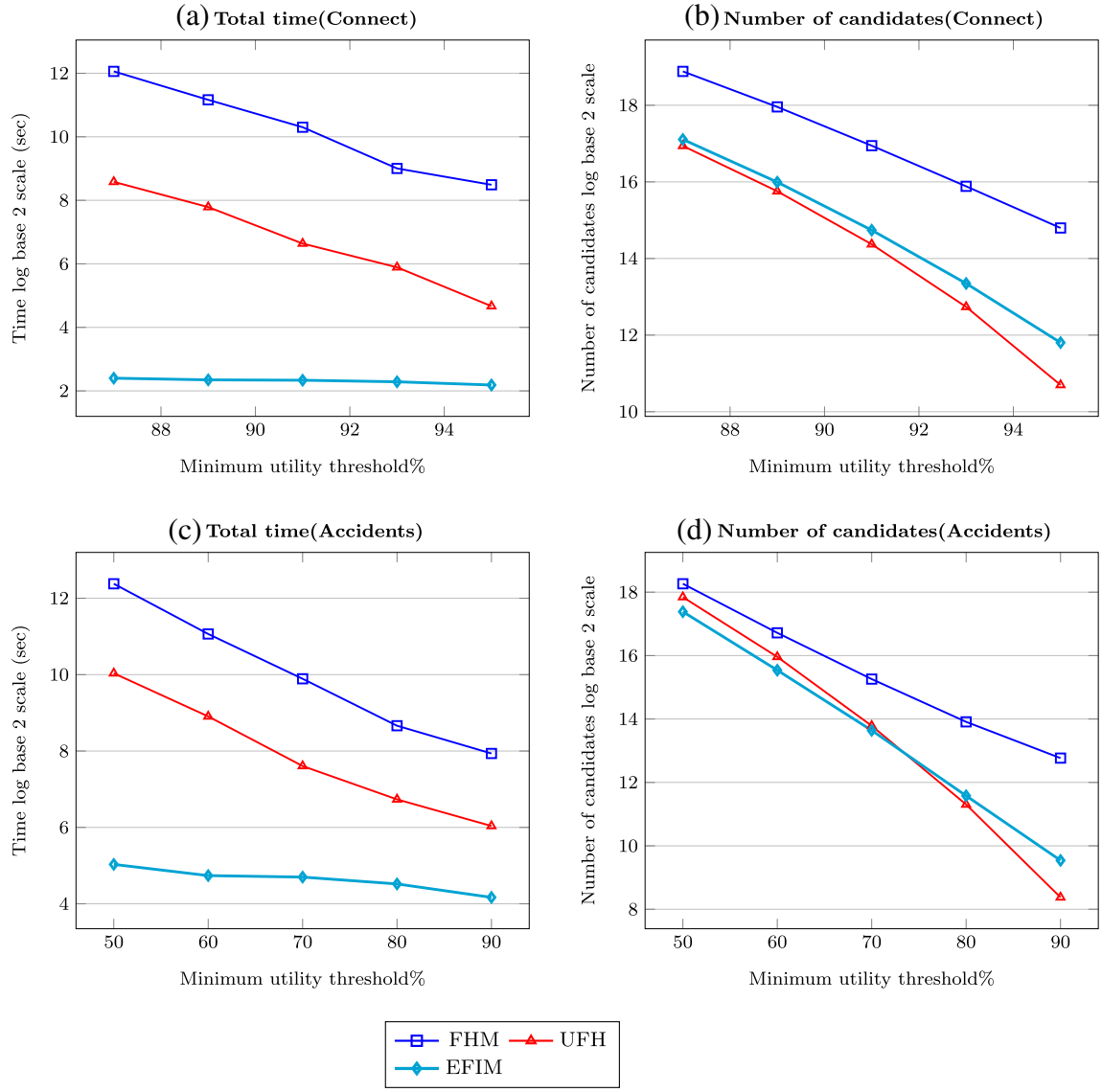


Fig. 12 Performance evaluation on real dense datasets

We conducted experiments on various real and synthetic dataset. The description of the real datasets is shown in Table 1. The datasets vary in the number of transactions (#Tx), average transaction length and the number of items (#Items). All datasets, except Chainstore, were obtained from SPMF library [9]. The ChainStore dataset was obtained from NU-Minebench 2.0 repository [23]. BMS WebView 1 and BMS WebView 2 contain sequences of click-stream data. Kosarak also contains sequences of click-stream data obtained from a Hungarian news portal. Retail is a market basket data obtained from an anonymous Belgian retail store. Chainstore and OnlineRetail are also obtained from a retail store. KDDCup99 is a transformed dataset from KDD Cup 1999. PowerC is a dataset about household electric power consumption. The Accidents dataset is prepared from an anonymized traffic accident data. The Connect dataset contains the positions in the game of connect-4. Only ChainStore had the quantity and external utility associated with each item in the database. The quantity information for items in the other datasets was chosen randomly from 1 to 5. The external utility values were generated between 1 to 1000 using log-normal distribution.

In the first set of experiments, we observe the total time as well as the number of candidates for different minimum utility threshold. We will first discuss the results for sparse datasets. We compare against two variants of EFIM algorithm; with and without transaction merging. For sparse datasets, we observe that EFIM without transaction merging performs better than EFIM for all sparse datasets. Sparse

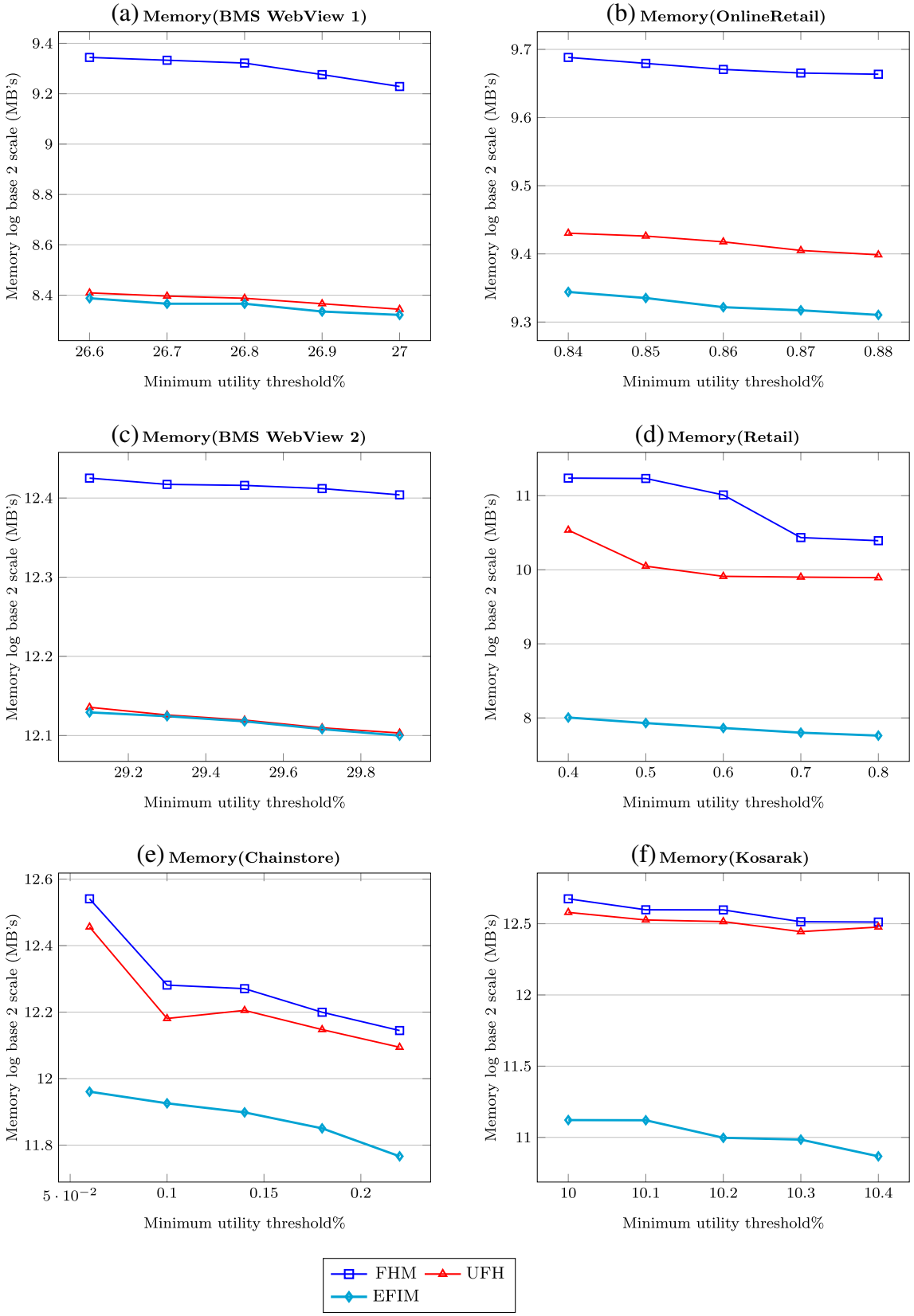


Fig. 13 Memory consumption on real sparse datasets

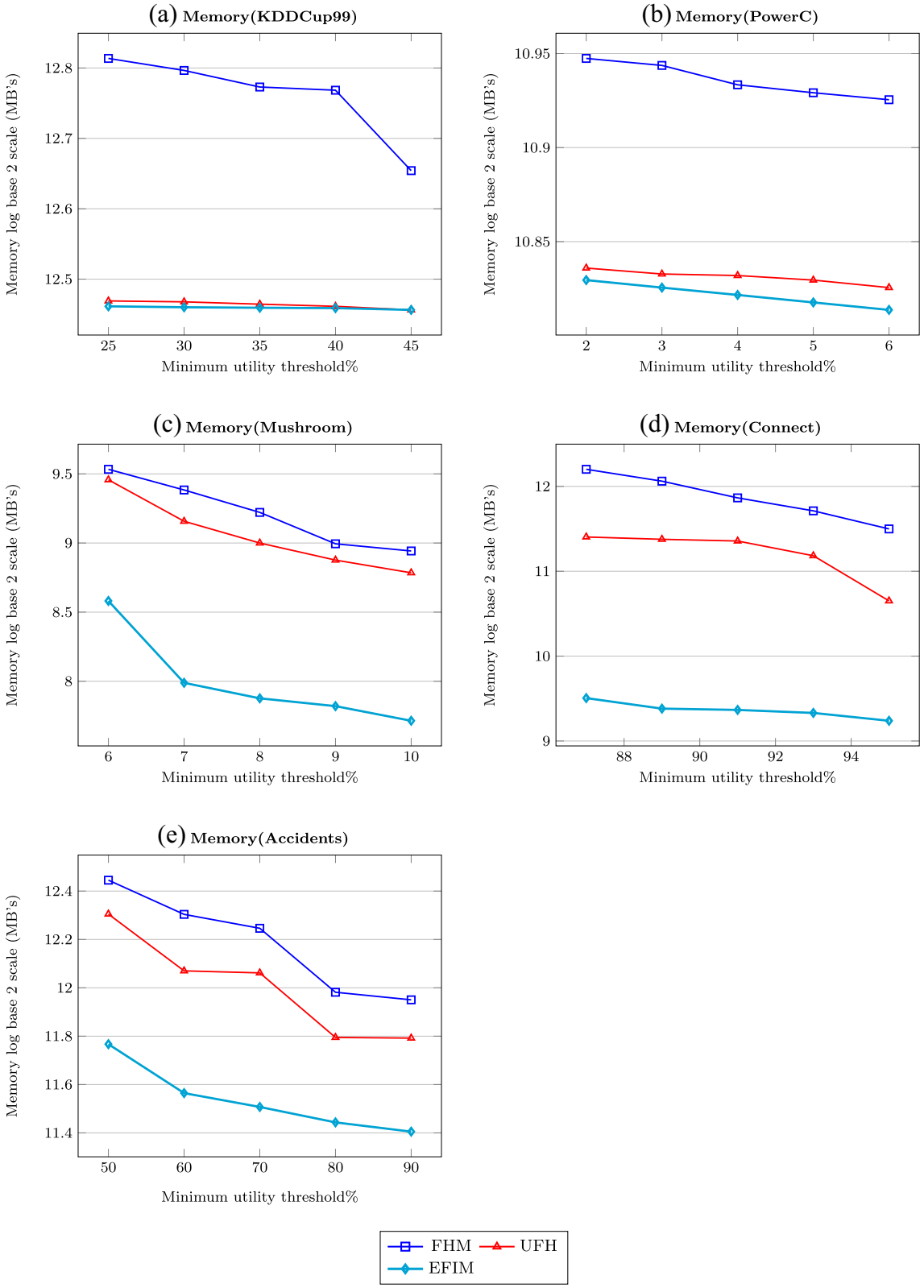


Fig. 14 Memory consumption on real dense datasets

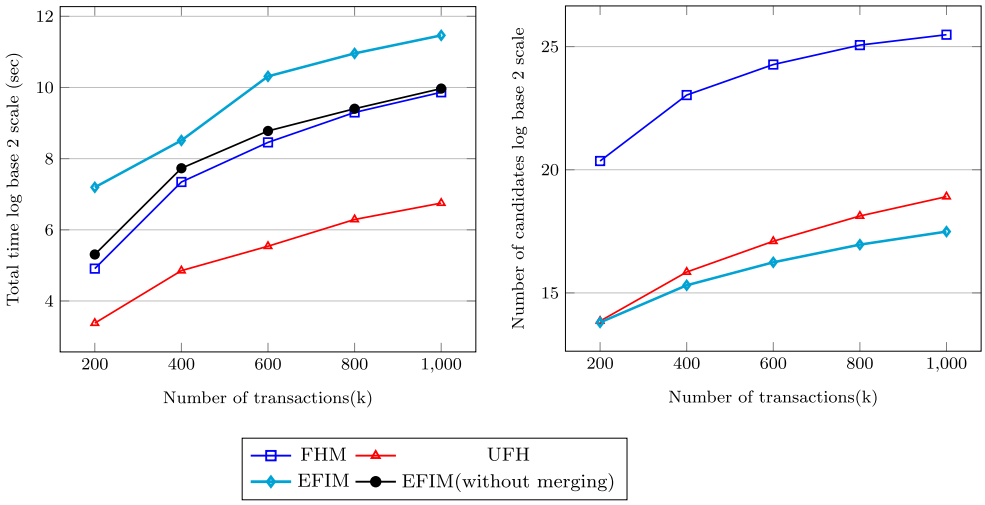


Fig. 15 Scalability on Chainstore dataset

datasets consist of transactions of small average length and a large number of items. The probability of an item presence in a transaction is less, compared to dense datasets. We believe that transaction merging does not perform well for sparse datasets. Our experimental results validate our hypothesis.

The results for BMS WebView 1, BMS WebView 2 and OnlineRetail is shown in Fig. 9. For BMS WebView 1, our UFH algorithm performs better than FHM by at least 400 times and EFIM by at least 66 times in terms of total execution time. UFH also generates fewer candidates compared to FHM and EFIM. For BMS WebView 2, UFH is better than FHM by at least 40 times and EFIM by 1.5 times. For BMS WebView 2, EFIM generates fewer candidates compared to UFH. However, the cost of creating projected database recursively is a key factor in the performance of EFIM. For OnlineRetail, UFH performs better than FHM by at least 10 times and EFIM by 7 times. UFH and EFIM produce a similar number of candidates for the OnlineRetail dataset.

The results for Retail, ChainStore, and Kosarak is shown in Fig. 10. For Retail, UFH performs better than FHM by at least 7 times and EFIM by 6.5 times. For Kosarak dataset, UFH beats FHM by 65 times and EFIM by 2 times. For ChainStore, FHM beats EFIM and EFIM with merging in terms of total execution time. Our algorithm UFH beats FHM by 5 times and EFIM by 9 times in terms of total execution time. The results show that FHM beats EFIM on a real sparse dataset like ChainStore. EFIM without transaction merging performs better than EFIM on sparse datasets. The total execution time and the number of candidates reduce with an increase in the minimum utility threshold for all datasets.

We will now discuss the performance of different algorithms on dense datasets. The results for KDDCup99, PowerC and Mushroom is shown in Fig. 11. UFH performs well on PowerC and KDDCup99 datasets as our algorithm performs an initial transaction merging before starting the mining process. The initial transaction merging reduces the size of the database by 99% for these datasets. The results for Connect and Accidents is shown in Fig. 12. We observe that EFIM is the state-of-the-art algorithm for datasets like Mushroom, Connect, and Accidents where initial transaction merging is not quite effective. Transaction merging works quite well in these datasets during the mining process.

We also observed the memory consumed by the algorithms on real datasets. The results are shown in Figs. 13 and 14. Our proposed algorithms consume less memory than FHM on different datasets. Our algorithm consumes memory comparable to EFIM on BMS WebView 1 and PowerC dataset. EFIM consumes less memory on other datasets as it computes the utility and heuristic information for an itemset from projected database only without creating utility-lists.

In order to assess the impact of scalability on the performance of our algorithm, we conduct another experiment on ChainStore dataset. The minimum utility threshold is fixed to 0.1%. The total time and number of candidates are observed for varying number of transactions in the database. The number of transactions is varied from 200k to 1000k. The results are shown in Fig. 15. The results show that our algorithm is scalable and perform well with an increase in the number of transactions.

7 Conclusions

In this paper, we proposed a hybrid framework for mining high-utility itemsets from transaction databases. Our proposed framework can combine any tree-based recursive algorithm with an inverted-list based algorithm. We perform a case study of the integration of the UP-Growth+ and FHM algorithm. Experimental results on real and synthetic

datasets demonstrate that the hybrid algorithm built on our framework performs better than the state-of-the-art algorithms on sparse datasets. We observed, that our hybrid algorithm beats FHM and EFIM on few dense datasets. As a part of our future work, we will try to identify characteristics of dense datasets on which algorithms based on our framework will become more efficient compared to EFIM. We will further study the impact of different switching criteria on the performance of hybrid algorithms.

Acknowledgments This work was supported in parts by Infosys Centre for Artificial Intelligence, IIIT-Delhi and Visvesvaraya Ph.D scheme for Electronics and IT.

Compliance with Ethical Standards

Conflict of interests The authors declare that they have no conflict of interest.

References

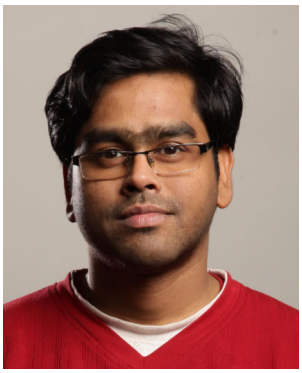
1. Agrawal R, Srikant R et al (1994) Fast algorithms for mining association rules. In: Proceeding 20th international conference on very large data bases, VLDB, vol 1215, pp 487–499
2. Ahmed CF, Tanbeer SK, Jeong BS, Lee YK (2009) Efficient tree structures for high utility pattern mining in incremental databases. IEEE Trans Knowl Data Eng 21(12):1708–1721. doi:10.1109/TKDE.2009.46
3. Ahmed CF, Tanbeer SK, Jeong BS, Lee YK (2011) Huc-prune: an efficient candidate pruning technique tomine high utility patterns. Appl Intell 34(2):181–198. doi:10.1007/s10489-009-0188-5
4. Ahmed CF, Tanbeer SK, Jeong BS, Choi HJ (2012) Interactive mining of high utility patterns over data streams. Expert Syst Appl 39(15):11,979–11,991. doi:10.1016/j.eswa.2012.03.062. http://www.sciencedirect.com/science/article/pii/S0957417412005854
5. Bansal R, Dawar S, Goyal V (2015) An efficient algorithm for mining high-utility itemsets with discount notion. Springer International Publishing, Cham, pp 84–98. doi:10.1007/978-3-319-27057-9_6
6. Chan R, Yang Q, Shen YD (2003) Mining high utility itemsets. In: Third IEEE international conference on data mining, 2003. ICDM 2003, pp 19–26. doi:10.1109/ICDM.2003.1250893
7. Dawar S, Goyal V (2014) Up-hist tree: an efficient data structure for mining high utility patterns from transaction databases. In: Proceedings of the 19th international database engineering & applications symposium, ACM, New York, NY, USA, IDEAS '15, pp 56–61. doi:10.1145/2790755.2790771
8. Erwin A, Gopalan RP, Achuthan NR (2008) Efficient mining of high utility itemsets from large datasets. Springer, Berlin, pp 554–561. doi:10.1007/978-3-540-68125-0_50
9. Fournier-Viger P, Gomariz A, Gueniche T, Soltani A, Wu CW, Tseng VS (2014) Spmf: a java open-source pattern mining library. J Mach Learn Res 15(1):3389–3393
10. Fournier-Viger P, Wu CW, Zida S, Tseng VS (2014) FHM: Faster High-utility itemset mining using estimated utility co-occurrence pruning. Springer International Publishing, Cham, pp 83–92. doi:10.1007/978-3-319-08326-1_9
11. Goethals B, Zaki M (2003) The frequent itemset mining implementations repository. http://fimi.ua.ac.be/
12. Goyal V, Dawar S, Sureka A (2015) High utility rare itemset mining over transaction databases. Springer International Publishing, Cham, pp 27–40. doi:10.1007/978-3-319-16313-0_3
13. Han J, Pei J, Yin Y (2000) Mining frequent patterns without candidate generation. In: Proceedings of the 2000 ACM SIGMOD international conference on management of data, ACM, New York, NY, USA, SIGMOD '00, pp 1–12. doi:10.1145/342009.335372
14. Lan GC, Hong TP, Tseng VS (2014) An efficient projection-based indexing approach for mining high utility itemsets. Knowl Inf Syst 38(1):85–107. doi:10.1007/s10115-012-0492-y
15. Leung CKS, Khan QI, Li Z, Hoque T (2007) Cantree: a canonical-order tree for incremental frequent-pattern mining. Knowl Inf Syst 11(3):287–311. doi:10.1007/s10115-006-0032-8
16. Li HF, Huang HY, Chen YC, Liu YJ, Lee SY (2008) Fast and memory efficient mining of high utility itemsets in data streams. In: 2008 8th IEEE international conference on data mining, pp 881–886. doi:10.1109/ICDM.2008.107
17. Li YC, Yeh JS, Chang CC (2008) Isolated items discarding strategy for discovering high utility itemsets. Data & Knowledge Engineering 64(1):198–217. doi:10.1016/j.datak.2007.06.009. http://www.sciencedirect.com/science/article/pii/S0169023X07001218
18. Li YC, Yeh JS, Chang CC (2008) Isolated items discarding strategy for discovering high utility itemsets. Data Knowl Eng 64(1):198–217. doi:10.1016/j.datak.2007.06.009. http://www.sciencedirect.com/science/article/pii/S0169023X07001218
19. Liu M, Qu J (2012) Mining high utility itemsets without candidate generation. In: Proceedings of the 21st ACM international conference on information and knowledge management, ACM, New York, NY, USA, CIKM '12, pp 55–64. doi:10.1145/2396761.2396773
20. Liu Y, Liao Wk, Choudhary A (2005) A fast high utility itemsets mining algorithm. In: Proceedings of the 1st international workshop on utility-based data mining, ACM, New York, NY, USA, UBDM '05, pp 90–99. doi:10.1145/1089827.1089839
21. Liu Y, Liao Wk, Choudhary A (2005) A two-phase algorithm for fast discovery of high utility itemsets. Springer, Berlin, pp 689–695. doi:10.1007/11430919_79
22. Park JS, Chen MS, Yu PS (1995) An effective hash-based algorithm for mining association rules. In: Proceedings of the 1995 ACM SIGMOD international conference on management of data, ACM, New York, NY, USA, SIGMOD '95, pp 175–186. doi:10.1145/223784.223813
23. Pisharath J, Liu Y, Wk Liao, Choudhary A, Memik G, Parhi J (2005) Nu-minebench 2.0. Department of Electrical and Computer Engineering, Northwestern University, Tech Rep
24. Rathore S, Dawar S, Goyal V, Patel D (2016) Top-k high utility episode mining from a complex event sequence. In: 21St international conference on management of data, COMAD 2016, Pune, India, March 11–13, 2016, pp 56–63. http://comad.in/comad2016/proceedings/paper_19.pdf
25. Shie BE, Tseng VS, Yu PS (2010) Online mining of temporal maximal utility itemsets from data streams. In: Proceedings of the 2010 ACM symposium on applied computing, ACM, New York, NY, USA, SAC '10, pp 1622–1626. doi:10.1145/1774088.1774436
26. Shie BE, Hsiao HF, Tseng VS, Yu PS (2011) Mining high utility mobile sequential patterns in mobile commerce environments. Springer, Berlin, pp 224–238. doi:10.1007/978-3-642-20149-3_18
27. Shie BE, Yu PS, Tseng VS (2012) Efficient algorithms for mining maximal high utility itemsets from data streams with different models. Expert Syst Appl 39(17):12,947–12,960. doi:10.1016/j.eswa.2012.05.035. http://www.sciencedirect.com/science/article/pii/S095741741200749X
28. Tseng VS, Wu CW, Shie BE, Yu PS (2010) 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, ACM, New York, NY, USA, KDD '10, pp 253–262. doi:10.1145/1835804.1835839
29. Tseng VS, Shie BE, Wu CW, Yu PS (2013) Efficient algorithms for mining high utility itemsets from transactional databases. IEEE Trans Knowl Data Eng 25(8):1772–1786. doi:10.1109/TKDE.2012.59
30. Vu L, Alaghband G (2011) A fast algorithm combining fp-tree and tid-list for frequent pattern mining. In: Proceedings of information and knowledge engineering, pp 472–477
31. Wu CW, Lin YF, Yu PS, Tseng VS (2013) Mining high utility episodes in complex event sequences. In: Proceedings of the 19th ACM SIGKDD international conference on knowledge discovery and data mining, ACM, New York, NY, USA, KDD '13, pp 536–544. doi:10.1145/2487575.2487654
32. Yin J, Zheng Z, Cao L (2012) Uspan: an efficient algorithm for mining high utility sequential patterns. In: Proceedings of the 18th ACM SIGKDD international conference on knowledge discovery and data mining, ACM, New York, NY, USA, KDD '12, pp 660–668. doi:10.1145/2339530.2339636
33. Yin J, Zheng Z, Cao L, Song Y, Wei W (2013) Efficiently mining top-k high utility sequential patterns. In: 2013 IEEE 13th international conference on data mining, pp 1259–1264. doi:10.1109/ICDM.2013.148
34. Yun U, Ryang H, Ryu KH (2014) High utility itemset mining with techniques for reducing overestimated utilities and pruning candidates. Expert Syst Appl 41(8):3861–3878. doi:10.1016/j.eswa.2013.11.038. http://www.sciencedirect.com/science/article/pii/S0957417413009585
35. Zaki MJ, Parthasarathy S, Ogihara M, Li W et al. (1997) New algorithms for fast discovery of association rules. In: KDD, vol 97, pp 283-286
36. Zida S, Fournier-Viger P, Lin JCW, Wu CW, Tseng VS (2015) EFIM: A highly efficient algorithm for high-utility itemset mining. Springer International Publishing, Cham, pp 530–546. doi:10.1007/978-3-319-27060-9_44

Siddharth Dawar received his B.Tech degree in Information Technology in 2012 from Guru Gobind Singh Indraprastha University, Delhi, India and his M.Tech degree in Information Security from Indraprastha Institute of Information Technology, Delhi, India in 2014. Since 2014 he is a Ph.D scholar at Indraprastha Institute of Information Technology, (IIIT-Delhi), Delhi, India. His research interests include data mining, machine learning and information security.

Vikram Goyal received his M.Tech. in Information Systems in 2003 at Netaji Subhash Institute of Technology, Delhi, India and his Ph.D degree in Computer Science from the Department of Computer Science and Engineering at IIT Delhi, India in 2009. Since 2009 he is a faculty at Indraprastha Institute of Information Technology, (IIIT-Delhi), New Delhi, India. His research interests include Knowledge Engineering and Data Privacy. Other areas of interest are data mining, big data analytics & information security.

He has completed a couple of projects with DST India and Deity, India on the problems related to Privacy in Location-based services and Digitized Document Fraud Detection, respectively.



Debajyoti Bera Debajyoti Bera received his B.Tech. in Computer Science and Engineering in 2002 at Indian Institute of Technology (IIT), Kanpur, India and his Ph.D. degree in Computer Science from Boston University, Massachusetts, USA in 2010. Since 2010 he is an assistant professor at Indraprastha Institute of Information Technology, (IIIT-Delhi), New Delhi, India.

His research interests include computational complexity theory and quantum complexity. Other areas of interest are application of algorithmic techniques in data mining, network analysis and information security.