Efficiently Mining Maximal Diverse Frequent Itemsets

Dingming Wu[1(✉)], Dexin Luo[1], Christian S. Jensen[2], and Joshua Zhexue Huang[1]

[1] College of Computer Science and Software Engineering, Shenzhen University, Shenzhen, China
{dingming,zx.huang}@szu.edu.cn, luodexin2016@email.szu.edu.cn
[2] Department of Computer Science, Aalborg University, Aalborg, Denmark
csj@cs.aau.dk

Abstract. Given a database of transactions, where each transaction is a set of items, maximal frequent itemset mining aims to find all itemsets that are frequent, meaning that they consist of items that co-occur in transactions more often than a given threshold, and that are maximal, meaning that they are not contained in other frequent itemsets. Such itemsets are the most interesting ones in a meaningful sense. We study the problem of efficiently finding such itemsets with the added constraint that only the top-k most diverse ones should be returned. An itemset is diverse if its items belong to many different categories according to a given hierarchy of item categories. We propose a solution that relies on a purposefully designed index structure called the FP*-tree and an accompanying bound-based algorithm. An extensive experimental study offers insight into the performance of the solution, indicating that it is capable of outperforming an existing method by orders of magnitude and of scaling to large databases of transactions.

Keywords: Frequent itemsets · Diversification · Algorithm

1 Introduction

Frequent itemset mining [2] is important data analysis functionality. The prototypical application is supermarket basket analysis that allows a retailer to learn which items are commonly bought together. For instance, it might be found that "bread" and "milk" are often bought together. A major issue in frequent itemset mining is the consideration of a huge number of itemsets, many of which are eventually found to be insignificant. Hence, researchers have made efforts to mine different constraint-based frequent itemsets, considering different kind of itemsets, including closed [13], maximal [4], periodic [19], top-k [14], cost (utility) [15], sequential [12], weighted [20], and diverse [17,18] itemsets.

Diverse frequent itemset mining [17,18] targets scenarios where it may be useful to give priority to frequent itemsets with items belonging to different

G. Li et al. (Eds.): DASFAA 2019, LNCS 11447, pp. 191–207, 2019.
https://doi.org/10.1007/978-3-030-18579-4_12

item categories. A measure called DiverseRank was introduced to quantify the extent to which items in a set belong to multiple categories. The existing algorithm [17] is inefficient for computing the diverse frequent itemsets on large data sets. The algorithm first extracts all frequent itemsets using the state-of-the-art algorithm [9] and then extracts the possibly very small subset of diverse itemsets from the frequent itemsets. It is a waste of time computing frequent itemsets that are later eliminated because they are not diverse.

We combine the maximality and diversity constraints and study the problem of efficiently finding the top-k maximal diverse frequent itemsets (MDFIs). Compared to just finding diverse frequent itemsets, the maximality constraint is able to reduce the number of discovered itemsets, since a frequent itemset is maximal if none of its supersets are frequent. To support MDFI mining efficiently on large data sets, we propose the FP*-tree, a variant of the FP-tree [9] that not only is able to store compactly the necessary information for MFI computation, but also contains a posting list for each item, which makes it possible to construct supersets for maximal frequent itemsets (MFIs). We show that these supersets can cover all the MFIs in the data set. However, the function for computing the diversity score is non-monotonous. Therefore, the diversity score of those supersets cannot be used as upper bounds on the diversity scores of the covered MFIs. Hence, we propose an algorithm that derives upper bounds on the diversity scores of the MFIs to be computed. Using the FP*-tree, we present a bound-based algorithm that is able to return the top-k MDFIs while computing only some of the MFIs in the data set. Unlike existing methods that mine the MFIs in descending order of item frequency, the proposed algorithm adopts a new ordering based on the upper bounds on the diversity score for the MDFI computation, so that the top-k result can be obtained by computing only a few candidate MFIs. The proposed algorithm is compared to a basic algorithm that extends two existing algorithms. The performance evaluation on a real data set shows that the proposed algorithm on the FP*-tree outperforms the basic algorithm by up to several orders of magnitude.

The rest of the paper is organized as follows. First, Sect. 2 presents preliminaries and defines the paper's problem formally. Then, Sect. 3 presents the FP*-tree and the accompanying bound-based algorithm. Experimental results are reported in Sect. 4, and Sect. 5 reviews related work. Finally, Sect. 6 concludes and offers research directions.

2 Preliminaries and Problem Definition

Let I be a finite set of items, $I = \{i_1, i_2, \cdots, i_m\}$. A database $\mathcal{D} = \{T_1, T_2, \cdots, T_n\}$ is a set of transactions, where each transaction $T_j \in \mathcal{D}$ $(1 \leq j \leq n)$ is a subset of I and is assigned an unique identifier j. An itemset $X = \{i_1, i_2, \cdots, i_l\}$ is a set of l items, where $i_j \in I$ $(1 \leq j \leq l)$ and l is the length of X. An itemset X is contained in a transaction T if $X \subseteq T$. The **support**[1] $s(X)$ of an itemset X is the

[1] The support of an itemset can be also defined as the fraction of transactions that contain it. For simplicity, we use the count of transactions, which is equivalent when database $\mathcal{D}$ is fixed.

number of transactions containing X in $\mathcal{D}$. Given $1 \leq \sigma \leq |\mathcal{D}|$, an itemset X is called **σ-frequent** in $\mathcal{D}$ if $s(X) \geq \sigma$. A σ-frequent itemset X in $\mathcal{D}$ is a **maximal σ-frequent itemset** (MFI) in $\mathcal{D}$ if no σ-frequent itemset X' exists in $\mathcal{D}$ such that $X' \supset X$ [4].

A category tree CT is a tree structure, where non-leaf nodes correspond to categories and leaf nodes are items in I. Each internal node is the sub-category of its parent node. Each item (leaf node) $i_j \in I$ $(1 \leq j \leq m)$ belongs to the category of its parent node. Nodes close to the root correspond to general categories, while nodes close to leaf nodes correspond to specialized categories. The height h of a category tree is the length of the longest path from the root to a leaf node. The height of the root is h, and the height of a leaf node is 0. The level of a node is the length of the path from the node to the root. The level of the root is then 0, and the level of a leaf node is h. We consider only balanced category trees, i.e., paths from the root to a leaf node have the same length. Including a category tree in maximal frequent itemset mining makes it possible to distinguish between itemsets with similar items and itemsets with dissimilar items.

Definition 1. *Let X be an itemset, let l be a level in the category tree, where $0 \leq l \leq h$. We define $GP(X, l)$ to be the **generalized pattern** [17] of X at level l as follows. $GP(X, h) = X$, and $GP(X, l), l < h$, is obtained by replacing each item in $GP(X, l+1)$ with its corresponding parent at level l with duplicates removed, if any.*

Definition 2. *The **Merging Factor** [17] $MF(X, l)$ of itemset X at a level l depends on the number of items getting merged when the pattern is moved from the immediate lower level $(l+1)$ to level l in the category tree*

$$MF(X, l) = \frac{|GP(X, l)| - 1}{|GP(X, l+1)| - 1}, \quad 0 \leq l \leq h - 1 \tag{1}$$

Definition 3. *The **Proportional Level Factor** [17] $PLF(l)$ of level l is defined as:*

$$PLF(l) = \frac{2(h-l)}{(h-1)h}, \quad 1 \leq l \leq h-1, \quad h > 1 \tag{2}$$

Definition 4. *The **diversity score** $div(X)$ of itemset X is defined as the DiverseRank [17] of X.*

$$div(X) = \sum_{l=h-1}^{s+1} PLF(l) MF(X, l), \tag{3}$$

where s is the level at which $|GP(X, s)| = 1$.

A generalized pattern GP of an itemset represents the itemset in a category space. The smaller the level of a category is, the more general the category is. The merging factor reflects how fast the size of the GP is reduced when moving upward in the category tree. The *PLF* assigns weights to levels. The contributions of the levels near the root should be larger than those of the levels

near the leaf nodes. The diversity score of a frequent itemset ranges from $[0, 1]$. If all the items in a frequent pattern have the same immediate parent, the score is 0. On the other hand, if all the items in a frequent itemset have only the root as the common ancestor, the score is 1. The higher the score is, the more diverse the itemset is.

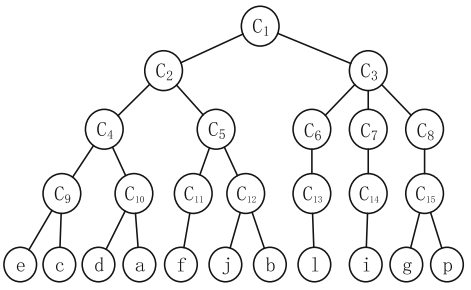
Example 1. Figure 1 shows an example of a category tree with height $h = 4$. Consider itemset $X = \{c, e, f\}$. The generalized pattern of X at level 3 is $GP(X, 3) = \{C_9, C_{11}\}$. Items c and e both have C_9 as parent at level 3 and because the parent of item f at level 3 is C_{11}. The merging factor $MF(X, 3)$ of itemset X at level 3 is $(|GP(X, 3)| - 1)/(|GP(X, 4)| - 1) = (2 - 1)/(3 - 1) = 0.5$. The proportional level factor (PLF) at each level is shown in Fig. 1. The diversity score of itemset X is $div(X) = PLF(3)MF(X, 3) + PLF(2)MF(X, 2) = (1/6) \cdot 0.5 + (1/3) \cdot 1 = 0.42$ because the generalized pattern of X at level 1 is $\{C_2\}$ and $|GP(X, 1)| = 1$, meaning that $s = 1$.

Definition 5. *An itemset X is called a* ***top-k maximal diversified σ-frequent itemset*** (kMDFI) *in $\mathcal{D}$ if it satisfies two conditions:*

1. *X is a maximal σ-frequent itemset in $\mathcal{D}$.*
2. *There are fewer than k maximal σ-frequent itemsets in $\mathcal{D}$ with diversity scores that exceed $div(X)$.*

Problem Statement. Given a database $\mathcal{D}$ of transactions, a category tree CT, a user-defined support threshold σ, and a desired number of itemsets k, the problem is to find efficiently a set of **top-k maximal diversified σ-frequent itemsets** (kMDFIs) in $\mathcal{D}$, i.e., to discover efficiently k maximal σ-frequent itemsets with the highest diversity scores in $\mathcal{D}$.

Example 2. Consider the transactional database $\mathcal{D}$ in Table 1 and the category tree CT in Fig. 1. Let $\sigma = 2$. The top-2 maximal diversified 2-frequent itemsets are $X_1 = \{a, c, e, g\}$ and $X_2 = \{a, b, c, e, f\}$ with diversity scores $div(X_1) = 0.78$ and $div(X_2) = 0.24$.



level	PLF
0	
1	1/2
2	1/3
3	1/6
4	0

Fig. 1. Example category tree

T_1	a b c e f o
T_2	a c g
T_3	e i
T_4	a c d e g
T_5	a c e g l
T_6	e j
T_7	a b c e f p
T_8	a c d
T_9	a c e g m
T_{10}	a c e g n

Table 1. Transactions

3 Bound-Based Mining Algorithm

We present a method that is able to efficiently compute the kMDFIs in large databases. Section 3.1 introduces the FP*-tree that stores information necessary to enable kMDFI mining. Section 3.2 presents the bound-based algorithm that uses the FP*-tree for mining kMDFIs. Section 3.3 derives bounds on the diversity scores of MDFIs.

3.1 The FP*-Tree

The FP-tree [9] is an index on transactions that enables frequent itemset mining. It consists of a tree structure and a header table. Each row in the header table stores a frequent item, its frequency, and a pointer to a tree node. The rows are sorted in non-increasing order of their frequencies. Ties are broken arbitrarily if multiple items have the same frequency. The header table for the transactions in Table 1 is shown in Fig. 2, given $\sigma = 2$. The tree structure stores all information necessary for mining frequent itemsets in a compact manner. It is built in the following way. Initially, the tree contains only one root node. Tree nodes are created and updated as transactions are scanned one by one. A branch in the FP-tree stores items that appear in the same transactions, and the nodes along a branch occur in the same order of the corresponding items in the header table. Overlapping itemsets are represented by the sharing of prefixes of the corresponding branches. For each transaction, the items included are sorted using the item order in the header table. Consider the transactions in Table 1. The scan of the first transaction leads to the construction of the first branch of the tree: $(e:1), (c:1), (a:1), (b:1), (f:1)$. Item o is removed, since it is not contained in the header table, meaning that o is infrequent. For the fourth transaction, since its (ordered) frequent item list e, c, a, g, d shares a common prefix e, c, a with the existing path, the count of each node along the prefix is incremented by 1, and a new node $(g:1)$ is created and linked as a child of $(a:2)$ and another new node $(d:1)$ is created and linked as the child of $(g:1)$. The tree-structure on the right side of Fig. 2 is the FP-tree built on the transactions in Table 1.

Before presenting the FP*-tree, we define the rank $r(i)$ of item i in the header table as the position of i in the table. The rank of the first item is 1, and if item i occurs before item i', $r(i) < r(i')$. Let $T(i)$ be the set of transactions that contain i. The FP*-tree extends the FP-tree [9] by adding a posting list $L(i)$ of (item, counter) pairs for each item i in the header table. A pair $(i', \mathit{counter})$ for item i indicates that i and i' co-occur *counter* times in transactions. An item i' must satisfy two conditions to be included in the posting list of item i: $r(i') < r(i)$ and $T(i) \cap T(i') \neq \emptyset$.

Example 3. Figure 2 shows the FP*-tree of the transactions in Table 1 for $\sigma = 2$. The header table and the tree structure are the same as in the FP-tree. The posting list of item e is empty, since it is the first item in the header table and no item has lower rank. The posting list of item a consists of pairs $(c, 8)$ and $(e, 6)$, meaning that (1) c and e have lower rank than a, (2) items c and a co-occur 8 times in transactions, and (3) items e and a co-occur 6 times in transactions.

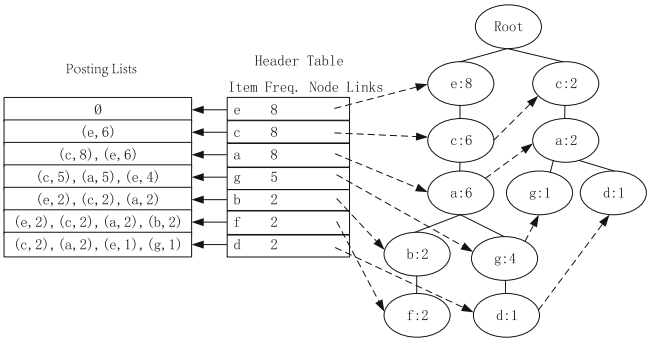


Fig. 2. Example FP*-Tree

An FP*-tree can be built by a procedure that is a minor modification of that for building the FP-tree. The construction of an FP-tree requires two scans of the transactional database. After the first scan, the header table is constructed. The posting list of each item can be built during the second scan when the tree structure is being constructed. For each transaction, the items are first sorted following the item order in the header table. Then the items are inserted into the tree structure one by one. In addition, each item i (except the first one) in the transaction is added to the posting lists of the items whose ranks are lower in the header table than that of i, and the corresponding counters are updated. The following example shows how to build posting lists during the second scan.

Example 4. Figure 3 shows how posting lists are updated as transactions are processed. So far, the header table has been constructed for $\sigma = 2$ (Fig. 2). Initially, the posting list of each item is empty. When transaction T_1 in Table 1 is processed, the items in T_1 are re-ordered as e, c, a, b, f to follow the item order in the header table. Item o is removed, since it does not occur in the header table, meaning that its frequency is less than $\sigma = 2$. The posting lists of item c, a, b, and f are updated as shown in Fig. 3. Take item a as an example. Items e and c are added to its posting list, since their ranks in the header table are higher than that of a. The corresponding counters are set to 1 because both e and c co-occur with a in T_1. Similarly, when T_2 is processed, the posting lists are updated as shown.

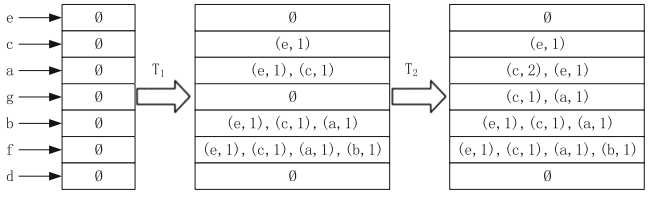


Fig. 3. Building posting lists

3.2 Algorithm

Given a transaction database $\mathcal{D}$ and a category tree CT, a user defined frequency threshold σ, and the desired number of MDFIs k, the state-of-the-art method FPMAX [8] can compute the result kMDFIs by first discovering all maximal frequent itemsets (MFIs) using the FP-tree and then computing the diversity score of each MFI. Finally, the kMDFIs are the k MFIs with the largest diversity scores. The limitation of FPMAX is that it has to compute many MFIs with small diversity scores that do not contribute to the result.

The proposed bound-based algorithm adopts the FP*-tree, making it possible to avoid computing MFIs with small diversity scores, thus saving substantial computational costs. Algorithm 1 shows the pseudo code. It first uses Algorithm 2 to construct a superset for each item and then uses Algorithm 3 to compute an upper bound on the diversity score for each item. Next, the items in the header table are sorted in non-increasing order of their bounds, and the algorithm then processes the items using this bound-based order. For each item i, the algorithm computes the MFIs containing i as does FPMAX. The discovered MFIs are added to the candidate set, and the diversity score of the current k^{th} MFI is recorded as τ. Next, when an item is to be processed, its bound is first compared with τ. If the bound is smaller than τ, the algorithm returns the current top-k MFIs; otherwise, the item is processed, and the MFIs containing the item are computed. Function **getNextItem**() follows the bound-based order in the header table and returns the next unprocessed item, and function **MFI**(i) is the sub-routine in algorithm FPMAX that computes the MFIs that contain item i.

input : Transactional database $\mathcal{D}$, category tree CT, frequency threshold σ, desired number of MFIs k
output: Top-k MDFIs

1 $\overline{X_i} \leftarrow$ call Algorithm 2 to construct a superset of the MFIs containing item i in the header table according to σ;
2 Call Algorithm 3 to compute the upper bound b_i on the diversity scores of the MFIs containing item i using $\overline{X_i}$;
3 Sort the items in the header table in descending order of their upper bounds;
4 $\tau \leftarrow -\infty$; ▷ `The diversity score of the current` k^{th} `MFI.`
5 **while** $i \leftarrow$ `getNextItem()` $\wedge b_i \geq \tau$ **do**
6 $\mathcal{X} \leftarrow$ `MFI(`i`)`;
7 **foreach** $X \in \mathcal{X}$ **do**
8 Compute the diversity score $div(X)$ of X;
9 Add X to the candidate set;
10 $\tau \leftarrow$ the diversity score of the current k^{th} MFI in the candidate set;
11 **end**
12 **end**
13 Return the top-k MFIs in the candidate set;

Algorithm 1. Bound-based Algorithm

3.3 Bounds on Diversity Scores

To define bounds on diversity scores, we need the concept of the tail of an itemset.

Definition 6. *The tail $t(X)$ of an itemset X is the item in X whose rank in the header table is lower than the ranks of all the other items in X.*

We derive a bound on the diversity score of an itemset (Lemma 2) and a bound on the diversity scores of the MFIs who have the same tail (Definition 6 and Lemma 4). Our bound-based algorithm efficiently computes the top-k MDFIs using these bounds.

Lemma 1. *Given an FP*-tree, σ, and item i, Algorithm 2 constructs a superset $\overline{X_i}$ of all possible MFIs whose tails are i. Note that there may exist multiple MFIs whose tails are i.*

Proof. Let M_i be the set containing all MFIs X such that $t(X) = i$. We now prove that $\overline{X_i}$ as constructed by Algorithm 2 is a superset of any X in M_i. Suppose an itemset X' exists in M_i that is not a subset of $\overline{X_i}$. Then there must be an item i' in X' that is not in $\overline{X_i}$. Since $X' \in M_i$ is an MFI and $t(X') = i$, items i' and i must co-occur no fewer than σ times in transactions, and the rank of i' must be higher than that of i in the header table. According to the method for constructing $L(i)$, item i' must be contained in $L(i)$. And based on Algorithm 2, item i' should be added to $\overline{X_i}$, which contradicts the assumption that i' is not in $\overline{X_i}$. Thus, we have $\forall X \in M_i\ (\overline{X_i} \supseteq X)$.

input : FP*-tree, support σ, item i
output: A superset of all possible MFIs whose tail is i

```
1 Get the posting list L(i) of item i from the FP*-tree;
2 Add i to X̄_i;
3 foreach item i' in L(i) do
4 |   if the count associated with i' ≥ σ then
5 |   |   Add i' to X̄_i;
6 |   end
7 end
```

Algorithm 2. Superset Construction

Lemma 2. *Given an itemset X, the set $X^w = \{i_L, i_R\}$ consists of the furthest apart pair of items in X according to the shortest path length in the category tree. Then, the diversity score of X^w is an upper bound on the diversity score of X, i.e., $div(X^w) \geq div(X)$.*

Proof. Let C be the lowest common ancestor of the items in X, and let the level of C in the category tree be s. Then C is also the lowest common ancestor of the items in X^w, since set $X^w = \{i_L, i_R\}$ consists of the furthest apart pair of items in X according to the shortest path length. Then, we have $|GP(X, s)| = |GP(X^w, s)| = 1$. For $s+1 \leq l \leq h-1$, $MF(X, l) \leq 1 = MF(X^w, l)$. This holds because (i) X^w contains only two items, and C is the lowest common ancestor of the two items at level s, so that the cardinality of the generalized pattern of X^w at level $s+1 \leq l \leq h-1$ is 1, and (ii) $|GF(X, l)| \leq |GF(X, l+1)|$. Hence, $div(X^w) \geq div(X)$.

Example 5. To exemplify Lemma 2, consider Fig. 4 where $X = \{a, c, e\}$. The furthest apart pair of items in X are a and e, so $X^w = \{a, e\}$. Both X and X^w have the same lowest common ancestor C_4 and its level is 2 in the category tree in Fig. 1. The diversity score of X^w is $(1/6) \cdot 1 = 0.17$, and the diversity score of X is $(1/6) \cdot (1/2) = 0.08$, so that $div(X^w) > div(X)$.

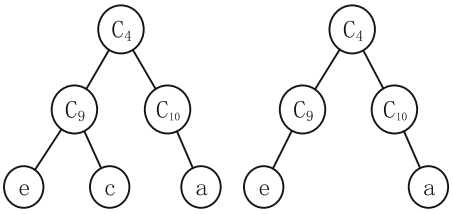


Fig. 4. Example of Lemma 2

Lemma 3. *Let itemsets X_1 and X_2 each consist of two items, and let $|GP(X_1, s_1)| = 1$ and $|GP(X_2, s_2)| = 1$. If $s_1 \leq s_2$, $div(X_1) \leq div(X_2)$.*

Proof. Since itemset X_1 consists of two items and $|GP(X_1, s_1)| = 1$, we have $div(X_1) = \sum_{l=h-1}^{s_1+1} PLF(l) \cdot 1$. Similarly, we obtain $div(X_2) = \sum_{l=h-1}^{s_2+1} PLF(l) \cdot 1$. If $s_1 \leq s_2$, we have $div(X_1) \leq div(X_2)$.

Lemma 4. *$div(\overline{X_i}^w)$ is an upper bound on the diversity score of any MFI whose tail is i.*

Proof. According to Lemma 1, set $\overline{X_i}$ is a superset of any MFI whose tail is i. Let X represent any MFI whose tail is i. Sets $\overline{X_i}^w$ and X^w consist of the furthest apart pair of items in $\overline{X_i}$ and X, respectively. Given a category tree, $|GP(\overline{X_i}^w, s_1)| = 1$ and $|GP(X^w, s_2)| = 1$. Since $X \subseteq \overline{X_i}$, it follows that $s_1 \geq s_2$. Based on Lemma 3, $div(\overline{X_i}^w) \geq div(X^w)$. According to Lemma 2, we have $div(X^w) \geq div(X)$. Hence, we derive $div(\overline{X_i}^w) \geq div(X)$, meaning that $div(\overline{X_i}^w)$ is an upper bound on the diversity score of any MFI with tail i.

According to Lemma 4, the bound on the diversity score of any MFI whose tail is i is the diversity score of $\overline{X_i}^w$ which consists of the furthest apart pair of items in $\overline{X_i}$ returned by Algorithm 2. Actually, the diversity score of $\overline{X_i}^w$ can be computed without explicitly finding the furthest apart pair of items in $\overline{X_i}$ using Algorithm 3. It takes the codes of the items in $\overline{X_i}$ as input. The length of each code is the height of the category tree. Each element in the code of an item corresponds to a node on the path from the root to the item. The diversity score bound is initialized to 0. The algorithm checks the elements at the same level in the codes of all the items from $level = h - 1$ to the level of the lowest common ancestor of all the items. At each level, the corresponding proportional level factor is added to the diversity score bound. It is because that for $\overline{X_i}^w$, before reaching the level where the lowest common ancestor is found, $MF(\overline{X_i}^w, l) = 1$. Finally, the bound on the diversity score of any MFI with tail i is returned.

Example 6. Table 2 shows itemsets $\overline{X_i}$ and the diversity score bounds $div(\overline{X_i}^w)$ computed by Algorithms 2 and 3, given the FP*-tree in Fig. 2. Then bound-based order of the items in the header table is g, b, f, a, d, c. Suppose the top-1 MFI is requested. The bound-based algorithm first computes the MFIs whose tail is item g, i.e., itemset $\{a, c, e, g\}$ with diversity score 0.78. Now, it is found that the diversity score of the current top-1 candidate exceeds the bound of the item to be processed next ($0.78 > 0.5$). The algorithm returns $\{a, c, e, g\}$ as the top-1 result and terminates. If using the FPMAX algorithm, following the order of the item frequency before finding the top-1 result, items c, a, g, b, and f have to be processed. Even when item d has been processed, FPMAX is still not aware of whether the MFIs that have not been found will have higher diversity scores.

input : Codes of items in set $\overline{X_i}$
output: Bound on the diversity score of the MFIs with tail i

```
1  div ← 0;
2  level ← h − 1;
3  while level > 0 do
4  |  if the elements at level in all the codes are the same then
5  |  |  Break;
6  |  end
7  |  else
8  |  |  div ← div + PLF(level);
9  |  |  level ← level − 1;
10 |  end
11 end
12 Return div;
```

Algorithm 3. Bound Computation

Table 2. Example diversity score bounds

Item i	$\overline{X_i}$	$div(\overline{X_i}^w)$
g	a, c, e, g	1
b	a, b, c, e	0.5
f	a, b, c, e, f	0.5
a	a, c, e	0.17
d	a, c, d	0.17
c	c, e	0

4 Empirical Study

The proposed algorithms are evaluated on a real commercial data set that consists of 3,040,715 transactions. The number of unique items is 37,984. The height of the category tree is 5. The number of non-leaf nodes in the category tree is 1,947. All algorithms have been implemented using Java and performed on a machine with Intel(R) Core(TM) i5-4590 CPU @ 3.30 GHz 3.30 GHz, 16 GB RAM and the Windows 10 Professional operating system.

4.1 Result Investigation

Maximal Diversified Frequent Itemsets (MDFIs). Table 3 shows example MDFIs and the number of MDFIs found from the real data set using various frequency threshold σ, where N is the number of transactions in the data set. The items in the discovered MDFIs belong to different categories. Take MDFI "yoghurt, pear" as an example. Item "yoghurt" belongs to category "drink" and item "pear" belongs to category "vegetable". The data set used only contains food-related categories. It is expected that more interesting MDFIs will be found if other types of items, such as clothes and home appliances, are included. When σ is small, e.g., $0.000005 \times N$, a large number $(980,241)$ of MDFIs are found.

Table 3. Maximal diversified frequent itemsets

$\sigma = 0.001 \times N$		$\sigma = 0.0005 \times N$		$\sigma = 0.0001 \times N$	
MDFIs: 71	*div*	MDFIs: 457	*div*	MDFIs: 11313	*div*
yoghurt, banana	1	yoghurt, salt	1	stationary, auto accessories	1
pork, tomato	1	pork, pumpkin	1	yoghurt, fish	1
rice, fish	1	tomato, shrimp	1	rice, chicken	1
$\sigma = 0.00005 \times N$		$\sigma = 0.00001 \times N$		$\sigma = 0.000005 \times N$	
MDFIs: 32990	*div*	MDFIs: 356204	*div*	MDFIs: 980241	*div*
yoghurt, hot dog	1	pistachio, tea	1	basket, chocolate, pistachio, sugar box, tea	1
yoghurt, dumpling	1	Coca Cola, nuts	1	yoghurt, shorts	1
potato, curry	1	oil, shampoo	1	sugar, crab	1

When σ is large, e.g., $0.001 \times N$, only 71 MDFIs are discovered. In this dataset, no MDFI is found when $\sigma \geq 0.005 \times N$.

Length of MDFIs. Figure 5 shows the length distribution of the MDFIs. When σ is large ($0.0005 \times N$), all MDFIs are of length 2. When $\sigma = 0.00005 \times N$, around 90% of the MDFIs are of length 2 and 10% of the MDFIs contain 3 items. When σ is small ($0.000005 \times N$), 50% of the MDFIs are of length 2, 40% contain 3 items, and 10% are of length larger than 3. It is expected that longer MDFIs are found when using small σ values, since more items are considered as frequent.

Diversity Score Distribution. Figure 6 shows the diversity score distribution of the discovered MDFIs. When σ is large ($0.0005 \times N$), 66% of the MDFIs have diversity scores from 0.75 to 1, and 10% have diversity scores from 0.5 to 0.75. When σ is small ($0.000005 \times N$), 42% of the MDFIs have diversity scores from 0.75 to 1, 19% have diversity scores from 0.25 to 0.5, and 40% have diversity scores from 0 to 0.25. As expected, when using small σ values, long MDFIs with high diversity scores are found.

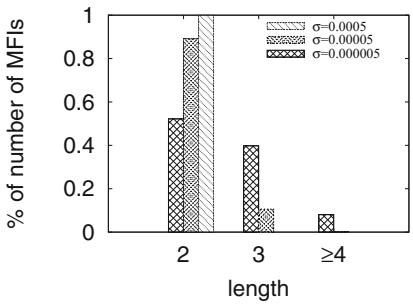


Fig. 5. Length distribution

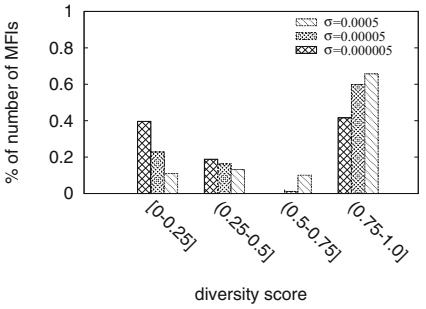


Fig. 6. Diversity distribution

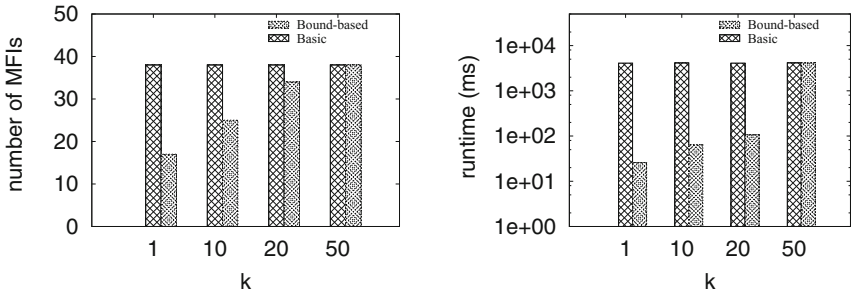
4.2 Efficiency

We proceed to evaluate the performance of the proposed algorithm and a basic algorithm under different parameter settings.

Basic Algorithm. No algorithm exists that targets top-k MDFIs. Instead, for comparison, we provide a basic algorithm that extends some existing techniques. The basic algorithm has two steps. It firstly finds all maximal frequent itemsets using the FPMAX [8] algorithm. Then, it computes the diversity score of each discovered maximal frequent itemset using the Item-Encoding algorithm [17]. Finally, the top-k MDFIs with the highest diversity scores are returned.

Varying the Number of Requested Sets k. Recall that to obtain the top-k MDFIs, the basic algorithm computes all the MFIs from the data set, while the bound-based algorithm only computes a few candidate MFIs. Figure 7 shows

the number of MFIs computed and the CPU time of the two algorithms when k is varied from 1 to 50. Parameter σ is set to 4000, which is roughly 0.13% of the transactions in the data set. Note that the number of MFIs computed and the CPU time of the basic algorithm do not change with k, since whatever k is, the basic algorithm always computes all the MFIs in the data set and ranks them. Nevertheless, the number of MFIs computed, and the CPU time of the bound-based algorithm increase as k increases. Because the more MDFIs that are requested, the more candidates are computed, yielding more computational cost. When $k = 1$, the number of MFIs computed using the bound-based algorithm is 55% less than that of the basic algorithm and the CPU time of the bound-based algorithm is only 0.6% of the CPU time of the basic algorithm. When k is set to 10 and 20, the bound-based algorithm also significantly outperforms the basic algorithm. When $k = 50$, the performance of the bound-based algorithm and the basic algorithm are the same. The reason is that there are 38 MFIs in the data set under the current parameter setting. Requesting top-50 MDFIs incurs the same computational cost for both algorithms.

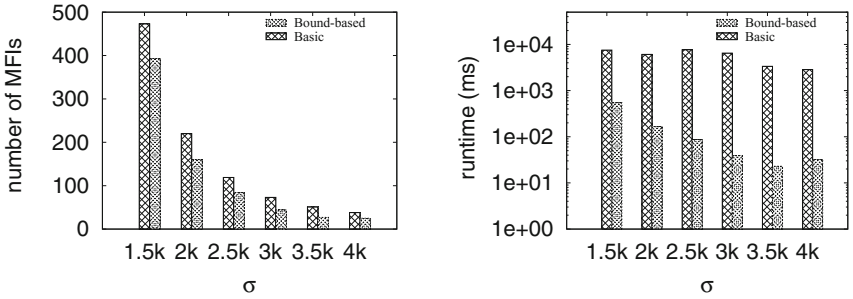


(a) # of MFIs (b) CPU time

Fig. 7. Varying k

Varying the Frequency Threshold σ. Figure 8 shows the number of MFIs computed and the CPU time of the two algorithms when σ is varied from 1500 (0.05% of the transactions in the data set) to 4000 (0.1% of the transactions in the data set). The number of requested MDFIs k is fixed at 10. When σ is small, many MFIs can be discovered. When σ is large, only few MFIs exist. Hence, as σ increases, the computational costs of both algorithms decrease. The bound-based algorithm beats the basic algorithm for all values of σ. The number of MFIs computed using the bound-based algorithm is 53%–83% of that using the basic algorithm. The CPU time of the bound-based algorithm is 0.6%–7.2% of the CPU time of the basic algorithm.

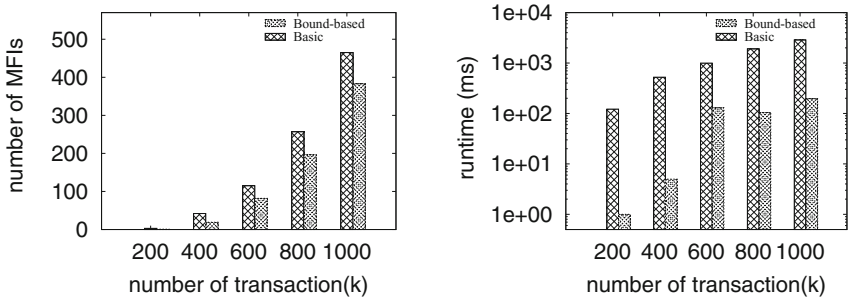
Scalability. To study how the computational cost of the proposed algorithm changes when varying the size of the data set, we have generated five data sets from the original data set by randomly selecting $200K$, $400K$, $600K$, $800K$, and



(a) # of MFIs (b) CPU time

Fig. 8. Varying σ

$1M$ transactions. Figure 9 shows the number of MFIs computed and the CPU time of the two algorithms when the size of the data set is varied. Parameter σ is set to 500, which is roughly 0.25%, 0.125%, 0.083%, 0.063%, and 0.05% of the number of transactions in the five data sets, respectively. The number of requested MDFIs k is fixed at 1. The computational costs of both the basic and the bound-based algorithms increase as the size of data set increases. On the five data sets, the bound-based algorithm outperforms the basic algorithm by orders of magnitude in terms of CPU time. The number of computed MFIs of the bound-based algorithm is 33.3%–82.3% of that of the basic algorithm.



(a) # of MFIs (b) CPU time

Fig. 9. Varying the number of transactions

5 Related Work

Frequent Itemsets with Various Constraints. Since mining frequent itemsets from transactional databases involves an exponential mining space and generates a huge number of itemsets, efficient discovery of constrained or user-interest based frequent itemsets is attractive. In many real-world scenarios, it is

often sufficient to mine a small and interesting representative of frequent itemsets. Hence, various constraints have been posed on the frequent itemsets in the literature. An itemset X is closed in a data set if there exists no superset that has the same frequency as X. Pasquier et al. [13] propose the A-Close algorithm that uses a closure mechanism to find frequent closed itemsets. A frequent itemset is maximal if none of its supersets are frequent. Burdick et al. [4] introduce the MAFIA algorithm for mining maximal frequent itemsets from a transactional database. A frequent itemset is periodic-frequent if it appears at a user-specified regular interval in the database. Tanbeer et al. [19] present the periodic-frequent pattern tree that captures the database contents in a highly compact manner and enables a pattern growth mining technique to generate the complete set of periodic-frequent itemsets in a database for user-given periodicity and support thresholds. Top-k frequent itemset mining finds interesting itemsets from the highest support to the k-th support. The CRM and CRMN algorithms [14] are proposed to mine top-k frequent itemsets efficiently. In utility mining, each item has external utility such as a profit or price and internal utility that refers to a non-binary value in a transaction. The importance of an itemset is measured by the concept of utility, which is the sum of the products of external and internal utilities of items in the itemset. An itemset is called a high utility itemset [15] when its utility is no less than a user-specified minimum utility threshold. Mallick et al. [12] consider the problem of the incremental mining of sequential patterns when new transactions or new customers are added to an original database. They present an algorithm for mining frequent sequences that uses information collected during an earlier mining process to cut down the cost of finding new sequential patterns in the updated database. Frequent weighted itemset mining considers a database where each item in a transaction may have a different significance. Vo et al. [20] propose a method for mining frequent weighted itemsets using WIT-trees. The diverse frequent itemset mining [17] uses the DiverseRank to rank the frequent itemsets based on the items' categories. Later, Swamy et al. [18] study the diverse frequent itemsets in the context where there concept hierarchies are unbalanced.

In this paper, we study the MDFI that extends the diverse frequent itemset by considering the maximality constraint. And an efficient bound-based algorithm is proposed for mining the top-k MDFIs.

Maximal Frequent Itemset Mining. MAFIA [5] mines maximal frequent itemsets (MFIs) using a depth-first traversal of the itemset lattice with pruning mechanisms and combining a vertical bitmap representation of the database. GenMax [6,7] is a backtrack search based algorithm for mining MFIs. It uses progressive focusing to perform maximality checking, and it uses diffset propagation to perform fast frequency computation. MinMax [21] is also based on depth-first traversal and iterations for mining MFIs. It removes all the non-maximal frequent itemsets without enumerating all the frequent itemsets from smaller ones. It backtracks to the proper ancestor directly, instead of level by level. Algorithms LFIMiner and LFIMiner-ALL [10] adopt a pattern fragment growth methodology based on the FP-tree for mining maximum length frequent

itemsets that is a subset of the MFIs. Yang [23] studied the complexity-theoretic aspects of MFI mining, from the perspective of counting the number of solutions. MaxDomino [16] uses the notions of dominance factor and collapsibility of transaction for efficiently mining MFIs. It employs a top-down strategy with selective bottom-up search. Pincer-Search [11] combines both bottom-up and top-down search for discovering MFIs. A restricted search is conducted in the top-down direction for maintaining and updating the maximum frequent candidate set, which is used for early pruning of candidates that would normally be encountered in the bottom-up search. CfpMfi [22] is a depth-first search algorithm based on CFP-tree for mining MFIs. It uses a variety pruning techniques and an item ordering policy to reduce the search space. DepthProject [1] finds long itemsets using a depth first search of a lexicographic tree of itemsets, and it uses a counting method based on transaction projections along its branches. MaxMiner [3] employs a breadth-first traversal of the search space and reduces database scanning by employing a look-ahead pruning strategy, i.e., if a node with all its extensions can be determined to be frequent, there is no need to further process that node. FPMAX [8] is an extension of the well know FP-growth [9] for mining MFIs. The maximal frequent itemset tree (MFI-tree) is used to keep track of all maximal frequent itemsets.

The basic algorithm for mining MDIFs proposed in this paper uses the state-of-the-art FPMAX as a component. And the proposed bound-based algorithm outperforms the basic algorithm significantly.

6 Conclusions

This work studies the problem of finding the top-k most diverse itemsets that are frequent. It tries to find long frequent itemsets of items belonging to different categories. Since no existing algorithm targets top-k MDFIs mining, we propose a basic algorithm that extends existing techniques. However, the basic algorithm fails to scale well to large data sets. We also propose the so-called FP*-tree along with a bound-based algorithm that is able to reduce the computational costs very significantly. Extensive experiments conducted on a large data set demonstrate that the proposed method consistently outperforms the basic algorithm.

References

1. Agarwal, R.C., Aggarwal, C.C., Prasad, V.V.V.: Depth first generation of long patterns. In: KDD, pp. 108–118 (2000)
2. Agrawal, R., Imielinski, T., Swami, A.N.: Mining association rules between sets of items in large databases. In: SIGMOD, pp. 207–216 (1993)
3. Bayardo Jr., R.J.: Efficiently mining long patterns from databases. SIGMOD Rec. **27**(2), 85–93 (1998)
4. Burdick, D., Calimlim, M., Flannick, J., Gehrke, J., Yiu, T.: MAFIA: a maximal frequent itemset algorithm. IEEE Trans. Knowl. Data Eng. **17**(11), 1490–1504 (2005)

5. Burdick, D., Calimlim, M., Gehrke, J.: MAFIA: a maximal frequent itemset algorithm for transactional databases. In: ICDE, pp. 443–452 (2001)
6. Gouda, K., Zaki, M.J.: GenMax: an efficient algorithm for mining maximal frequent itemsets. Data Min. Knowl. Discov. **11**(3), 223–242 (2005)
7. Gouda, K., Zaki, M.J.: Efficiently mining maximal frequent itemsets. In: ICDM, pp. 163–170 (2001)
8. Grahne, G., Zhu, J.: High performance mining of maximal frequent itemsets. In: HPDM (2003)
9. Han, J., Pei, J., Yin, Y.: Mining frequent patterns without candidate generation. In: SIGMOD, pp. 1–12 (2000)
10. Hu, T., Sung, S.Y., Xiong, H., Fu, Q.: Discovery of maximum length frequent itemsets. Inf. Sci. **178**(1), 69–87 (2008)
11. Lin, D.I., Kedem, Z.M.: Pincer-search: an efficient algorithm for discovering the maximum frequent set. IEEE Trans. Knowl. Data Eng. **14**(3), 553–566 (2002)
12. Mallick, B., Garg, D., Grover, P.S.: Incremental mining of sequential patterns: progress and challenges. Intell. Data Anal. **17**(3), 507–530 (2013)
13. Pasquier, N., Bastide, Y., Taouil, R., Lakhal, L.: Discovering frequent closed itemsets for association rules. In: Beeri, C., Buneman, P. (eds.) ICDT 1999. LNCS, vol. 1540, pp. 398–416. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-49257-7_25
14. Pyun, G., Yun, U.: Mining top-k frequent patterns with combination reducing techniques. Appl. Intell. **41**(1), 76–98 (2014)
15. Ryang, H., Yun, U., Ryu, K.H.: Fast algorithm for high utility pattern mining with the sum of item quantities. Intell. Data Anal. **20**(2), 395–415 (2016)
16. Srikumar, K., Bhasker, B.: Efficiently mining maximal frequent sets in dense databases for discovering association rules. Intell. Data Anal. **8**(2), 171–182 (2004)
17. Srivastava, S., Kiran, R.U., Reddy, P.K.: Discovering diverse-frequent patterns in transactional databases. In: COMAD, pp. 69–78 (2011)
18. Kumara Swamy, M., Reddy, P.K., Srivastava, S.: Extracting diverse patterns with unbalanced concept hierarchy. In: Tseng, V.S., Ho, T.B., Zhou, Z.-H., Chen, A.L.P., Kao, H.-Y. (eds.) PAKDD 2014. LNCS (LNAI), vol. 8443, pp. 15–27. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-06608-0_2
19. Tanbeer, S.K., Ahmed, C.F., Jeong, B.-S., Lee, Y.-K.: Discovering periodic-frequent patterns in transactional databases. In: Theeramunkong, T., Kijsirikul, B., Cercone, N., Ho, T.-B. (eds.) PAKDD 2009. LNCS (LNAI), vol. 5476, pp. 242–253. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-01307-2_24
20. Vo, B., Coenen, F., Le, B.: A new method for mining frequent weighted itemsets based on WIT-trees. Expert Syst. Appl. **40**(4), 1256–1264 (2013)
21. Wang, H., Li, Q., Ma, C., Li, K.: A maximal frequent itemset algorithm. In: RSFDGrC, pp. 484–490 (2003)
22. Yan, Y., Li, Z., Wang, T., Chen, Y., Chen, H.: Mining maximal frequent itemsets using combined FP-tree. In: Webb, G.I., Yu, X. (eds.) AI 2004. LNCS (LNAI), vol. 3339, pp. 475–487. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30549-1_42
23. Yang, G.: The complexity of mining maximal frequent itemsets and maximal frequent patterns. In: KDD, pp. 344–353 (2004)