2010 Third International Conference on Knowledge Discovery and Data Mining

Mining periodic-frequent itemsets with approximate periodicity using interval transaction-ids list tree

Komate Amphawan, Athatsit Surarerks
Chulalongkorn University, ELITE Laboratory
Bangkok, Thailand
g48kmp, athasit@cp.eng.chula.ac.th

Philippe Lenca
Institut Telecom, Telecom Bretagne,
UMR CNRS 3192 Lab-STICC
Brest, France
Université europèenne de Bretagne
philippe.lenca@telecom-bretagne.eu

***Abstract*—Temporal periodicity of itemset appearance can be regarded as an important criterion for measuring the interestingness of itemsets in several application. A frequent itemset can be said periodic-frequent in a database if it appears at a regular interval given by the user. In this paper, we propose a concept of the approximate periodicity of each itemset. Moreover, a new tree-based data structure, called ITL-tree (Interval Transaction-ids List tree), is proposed. Our tree structure maintains an approximation of the occurrence information in a highly compact manner for the periodic-frequent itemsets mining. A pattern-growth mining is used to generate all of periodic-frequent itemsets by a bottom-up traversal of the ITL-tree for user-given periodicity and support thresholds. The performance study shows that our data structure is very efficient for mining periodic-frequent itemsets with approximate periodicity results.**

***Keywords*-Data mining; knowledge discovery; frequent itemsets; periodic-frequent itemsets;**

I. Introduction

Mining frequent itemsets has become one of the fundamental data mining tasks. Depending on the type of databases, these mining approaches may be classified as working on transactional databases, temporal databases, relational databases and multimedia databases among others (see [1] for a review). Traditional frequent itemsets mining generates a huge number of itemsets and most of them might be found insignification for the user requirement. Moreover, the computational and space costs in finding such number of itemsets may not be trivial.

Recently, Tanbeer et al. [2] proposed the problem of discovering periodic-frequent itemsets from a transactional database. They defined a new periodicity measure for an itemset by the maximum interval at which the itemset occurs in the database. As pointed out in [2], there are several applications to apply periodic-frequent itemsets mining where the occurrence periodicity plays an important role. Thus, the authors proposed an efficient tree-based data structure, named *PF-tree (Periodic-Frequent pattern tree)*, which maintains the occurrence information *(tids list)* at the leaf node of each transaction and enables a pattern-growth mining technique to generate the complete set of periodic-frequent itemsets with exact periodicity for such itemsets. In [3], we extended this work and proposed a new algorithm for mining top-*k* periodic-frequent itemsets.

In this paper, we focus on the problem of mining the complete set of periodic-frequent itemsets with approximate periodicity of such itemsets. Our approximation on the periodicity will be compensated by an important reduction of the computational time. For that purpose, we propose a new tree structure, called *ITL-tree (Interval Transaction-ids List tree)*. It maintains the occurrence information in a highly compact manner by using interval trasnaction-ids list instead of *tids list*. By this technique, the approximate periodicity of each itemset can be discovered. A pattern growth mining is used to generate all of periodic-frequent itemsets by bottom-up traversal. The experimental evaluation shows that our data structure is very efficient for periodic-frequent itemsets mining with approximate periodicity.

II. Problem Statement and Definitions

Let $I = \{i_1, i_2, \ldots, i_n\}$ be a set of items. A set $X \subseteq I$ is called an itemset. A transactional database $TDB = \{t_1, t_2, \ldots, t_m\}$ over I is a set of $m = |TDB|$ transactions. Each transaction $t_j = (tid, Y)$ is a tuple where $tid = j$ represents the transaction-id and $Y \subseteq I$ is an itemset. If $X \subseteq Y$, it is said that t_j contains X and is denoted as t_j^X. Therefore, $T^X = \{t_j^X, \ldots, t_k^X\}$, where $j, k \in [1, m]$ and $j \leq k$, is the set of all *ordered tids* (*tids list*) where X occurs.

Let t_j^X and t_k^X be two consecutive transactions in T^X, i.e. where $j < k$ and there is no transaction t_i, $j < i < k$, such that t_i contains X. Thus, $pt^X = t_k^X - t_j^X$ is the periodicity between the two consecutive transactions t_j^X and t_k^X where X occurs. We denote by $PT^X = \{pt_1^X, pt_2^X, \ldots, pt_r^X\}$, the set of all periodicities between each pair of two consecutive transactions of X in TDB. Then, the periodicity of X can be denfined as $p^X = max(pt_1^X, pt_2^X, \ldots, pt_r^X)$. The support of X, denoted as $s^X = |T^X|$, can be defined as the number of transactions that contain X in TDB.

Therefore, an itemset X is called a *periodic-frequent itemset* if it satisfies both of following constraints [2]: *(i)*

978-0-7695-3923-2/10 $26.00 © 2010 IEEE
DOI 10.1109/WKDD.2010.126

its periodicity is no greater than a user-given maximum periodicity threshold σ_p *(ii)* its support is no less than a user-given minimum support threshold σ_s. Thus, the periodic-frequent itemsets mining problem is to discover the complete set of periodic-frequent itemsets from *TDB* having exact periodicity no more than σ_p and support no less than σ_s, for user-given σ_p and σ_s.

III. ITL-TREE: INTERVAL TRANSACTION-IDS LIST TREE

To increase efficiency of periodic-frequent itemsets mining, we propose a new technique to find the approximate periodicity (ap) of each itemset instead of the exact periodicity as it was done by [2]. By using this technique, our approach can reduce the time to collect *tids list* and the time to compute the periodicity of each itemset.

As proposed in [2], the *PF-tree* collects the occurrence information *(tids list)* at the leaf node of each transaction. In this paper, we propose a new tree-based data structure, named *ITL-tree*, which maintains the occurrence information in a compact manner using interval transaction-ids list. For the interval transaction-ids list, the transactional database is cut into a complete set of non-overlapping intervals. Each interval contains $\alpha = \sigma_p \times |TDB|$ transactions, except the last interval which may contain less transactions, where σ_p is the maximum periodicity threshold defined by the user. Each interval transaction-id consists of (i) interval number (i_{num}), (ii) minimum-tid (min) and (iii) maximum-tid (max) of interval. The interval number is the interval-id (calculated by $\lfloor (j-1) \div \alpha \rfloor$ for transaction t_j) while the minimum-tid and maximum-tid are the border of the interval which are used to find the approximate periodicity between the interval transaction-ids. In our approach, a list of interval transaction-ids is used to maintain the occurrence information of each itemset.

Example. Consider the *TDB* of Fig. 1(a) and let $\alpha = 4$, the set of transactions where itemset ab appears is $T^{ab} = \{1, 2, 4, 5, 7, 9, 11\}$. Therefore, the set of interval numbers where ab occurs is $\{0(\lfloor (1-1) \div 4 \rfloor), 0(\lfloor (2-1) \div 4 \rfloor), 0(\lfloor (4-1) \div 4 \rfloor), 1(\lfloor (5-1) \div 4 \rfloor), 1(\lfloor (7-1) \div 4 \rfloor), 2(\lfloor (9-1) \div 4 \rfloor), 2(\lfloor (11-1) \div 4 \rfloor)\}$

Definition 1 (Minimum-tid and maximum-tid of interval tid)**.** *Let $it_{i_{num}}$ be an interval transaction-id with the interval number i_{num} and let $i_{num,k}$ be the tid where t_k occurs in interval $it_{i_{num}}$. The minimum-tid (min) and maximum-tid (max) of $it_{i_{num}}$ can be defined as*

$$it_{i_{num}}.min = min(i_{num,1}, i_{num,2}, \ldots, i_{num,\alpha})$$

$$it_{i_{num}}.min = max(i_{num,1}, i_{num,2}, \ldots, i_{num,\alpha})$$

For example, in the TDB of Fig. 1(a), the itemset ab has 3 interval numbers $(= \{0, 1, 2\})$ when $\alpha = 4$. The minimum-tid $it_{i_{num}}.min$ and maximum-tid $it_{i_{num}}.max$ of interval transaction-ids of ab are $\{(it_0.min = 1, it_0.max = 4), (it_1.min = 5, it_1.max = 7), (it_2.min = 9, it_2.max = 11)\}$

Proposition 1. *The maximum periodicity of each interval transaction-id is no greater than $(\sigma_p \times |TDB|)$.*

proof. *The number of transactions of each interval transaction-id is set to be $\alpha = \sigma_p \times |TDB|$, then the maximum periodicity of each interval transaction-id is equal to $it_{i_{num}}.max - it_{i_{num}}.min$ where the $it_{i_{num}}.min$ and $it_{i_{num}}.max$ are the border of interval tid i_{num}.*

Definition 2 (Approximate periodicity of two consecutive interval tids of itemset X)**.** *Let it_j^X and it_k^X, where $j \leq k$, be two consecutive interval transaction-ids where X occurs. Then, we can define apt as the approximate periodicity between the two consecutive interval transaction-ids of X which is the different of the number of transactions between it_j^X and it_k^X.*

$$apt = it_k^X.min - it_j^X.max$$

apt is an approximation of the real maximum periodicity of two consecutive interval tids of itemset X.

Definition 3 (Approximate periodicity of itemset X)**.** *Let $APT^X = \{apt_1^X, \ldots, apt_r^X\}$ be the set of the all approximate periodicities between each pair of two consecutive interval transaction-ids, where X occurs. The approximate periodicity of X, denoted as ap^X, is defined as*

$$ap^X = max(apt_1^X, apt_2^X, \ldots, apt_r^X)$$

An itemset X is called a *periodic-frequent itemset* if it satisfies both of following two criteria: (i) its approximate periodicity (ap^X) is no greater than $\sigma_p \times |TDB|$ and (ii) its supports is no less than $\sigma_s \times |TDB|$. Thus, a complete set of periodic-frequent itemsets can be mined by using the concept of interval transaction-id to compute the approximate periodicity of each itemset for given σ_p and σ_s.

A. The ITL-tree structure

In *ITL-tree*, a header table is maintained to order the items by support descending and to use bottom-up mining technique. Each entry in a header table explicitly maintains item-id, support and node link (to link nodes that have the same item). For tree structure, nodes are separated into two types; usual-node and tail-node. The former contains only item-id meanwhile the latter is the last item in each transaction which maintains all of the needed information such as item-id, support, interval transaction-ids list. To enable bottom-up mining the node link (with the same item node) is only maintained in the leaf node of each transaction. By this technique, *ITL-tree* can reduce the memory consumption from *PF-tree* which contains node link in every nodes.

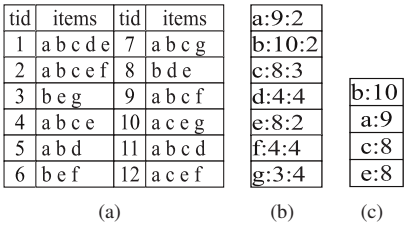

Figure 1. Transaction database and 1-item list construction

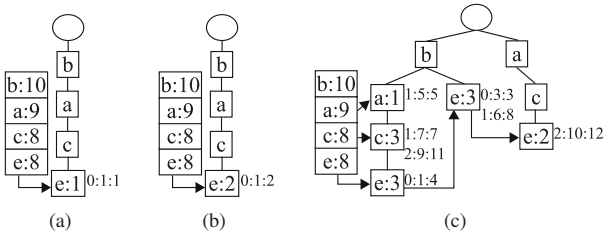

Figure 2. ITL-tree for TDB of Fig. 1(a)

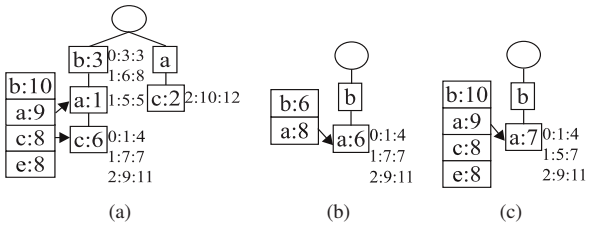

Figure 3. ITL-tree during mining process

B. The ITL-tree construction

By using the *PF-tree* construction technique, the *ITL-tree* is constructed in such a way that it contains only items in the header table. The occurrence information is explicitly maintained in a list called *interval transaction-ids list* at the tail-node of each transaction.

To create *ITL-tree*, the database is scanned to determine the list of items with their support and their periodicity. Items that are not periodic-frequent items are eliminated from the list. Then, the items list is sorted in support descending order and stored in the header table. After that, each transaction in TDB is scanned again and we keep only the items that belong to the header table. Then, these items are reordered according to the order of the header table. Finally, the ordered list of items is inserted into the *ITL-tree*. In addition, the interval-id, minimum-tid and maximum-tid are inserted at the tail-node if the tail-node does not have the the same interval-id as the interval-id of the transaction. Otherwise, the maximum-tid and the support of the tail-node are only updated.

Consider the TDB from Fig. 1(a). Figure 1 and 2 show the process of *ITL-tree* construction with $\sigma_p = 4$ and $\sigma_s = 5$ respectively. The list of items is created by scanning the database (Fig. 1(b)) and the periodic-frequent items are stored in the header table as shown in Fig. 1(c). The database is then scanned second time. After reading t_1 and according to the order of the header table, the itemset $\{b\,a\,c\,e\}$ is inserted into the *ITL-tree* and the interval-tids list and the node link are is created at the last node e as shown in Fig. 2(a). With the second transaction t_2, we get the itemset $\{b\,a\,c\,e\}$ from the transaction. A path for itemset $\{b\,a\,c\,e\}$ already exists in ITL-tree. We thus only update the support of node e to be 2 and the maximum-tid of it_0 to be 2 (Fig. 2(b)). After scanning all the transactions and interting them in same manner, the final *ITL-tree* with $\sigma_p = 4$ and $\sigma_s = 5$ is shown in Fig. 2(c)

C. Mining periodic-frequent itemsets from ITL-tree

To mine the periodic-frequent itemsets, the algorithm traverse the link from the header table to the tail-nodes. The basic operations of this proposed bottom-up approach for mining periodic-frequent itemsets from *ITL-tree* are *(i)* computing the approximate periodicity and support of prefix items which are in the same path of the tail-node, *(ii)* constructing the conditional tree for the periodic-frequent prefix items and *(iii)* pushing interval transaction-ids list of the tail-node to its parent after finishing to consider the tail-node.

The following example illustrates the periodic-frequent itemset mining process from *ITL-tree*. Consider the *ITL-tree* with the header table in Fig. 2(c), *ITL-tree* mining starts from considering item e which is the bottom item in the header table. The link from header to every nodes of e is traversed to compute approximate peridocity and support of prefix items of e . Hence, $a : 5 : 6 : [1 : 1 : 4][3 : 10 : 12]$ and $e : 5 : 6 : [1 : 1 : 4][3 : 10 : 12]$ (The numbers after each item indicate support, approximate periodicity and interval tids list) have approximate periodicity greater than $\sigma_p = 4$. Then, item $b : 6 : 4 : [1 : 1 : 4][2 : 6 : 8]$ is the only one to be periodic-frequent with e (itemset be is then an output of mining phase). After considering item e, the interval transaction-ids list in each node e is pushed-up to its parent node and all nodes of e are removed as shown in Fig. 3(a). Next, the link of item c is traversed to compute the support and approximate periodicity of each prefix item occurring with c, that are $\{(b : 6 : 3 : [1 : 1 : 4], [2 : 7 : 7], [3 : 9 : 11]), (a : 8 : 3 : [1 : 1 : 4], [2 : 7 : 7], [3 : 9 : 12])\}$. Since there are two periodic-frequent items occurring with item c, that are b and a, the condition tree is constructed by considering the path c that have prefix periodic-frequent items more than one as shown in Fig. 3(b). Next, the support $(s^b : 6)$ and the approximate periodicity $(ap^b : 3)$ of b (which is the prefix item of itemset ca) are computed and output with ac as periodic-frequent itemset because $s^b(6) > 5$ and $ap^b(3) < 4$. After that, the condition tree

of item c is removed and the interval transaction-ids lists of nodes of c in *ITL-tree* are pushed-up to their parent nodes. Finally, all nodes of c are removed as shown in Fig. 3(c). After considering all the items of the header table in the same manner, we can retrieve all of periodic-frequent itemsets with approximate perodicity from *ITL-tree*.

The above *ITL-tree* mining is efficient because it uses bottom-up mining technique on support descending *ITL-tree* which is a very compact tree. By this technique, the search space is shanked dramatically while mining periodic-frequent itemsets.

IV. Experimental study

To evaluate the performance of our approach and tree structure, *ITL-tree* is tested and compared with *PF-tree* on synthetic and real datasets. Due to the space constraint we only report the results on *T10I4D100K* and *Mushroom*. All reports of runtime of *ITL-tree* and *PF-tree* include the time to construct and mine the periodic-frequent itemsets. Both algorithms are implemented in C and run with a 1.6 GHz Intel Xeon with 4 GB main memory on Linux Platform. We evaluate the performance of the algorithms with minimum support from 5% to 25% and periodicity from 1% to 5%.

The performance of *ITL-tree* on *T10I4D100K* is shown in Fig. 4. From this figure, *ITL-tree* outperforms *PF-tree* for all minimum supports and maximum periodicities.The runtime of both algorithms increases slightly as the maximum periodicity increases and the minimum support decreases. For *T10I4D100K* which is sparse dataset, the tree has a large number of nodes and paths, then the pushing-up tids list into parent node of *PF-tree* takes a lot of time. It has to order the list of transaction-ids in order to find the exact periodicity of each itemset.

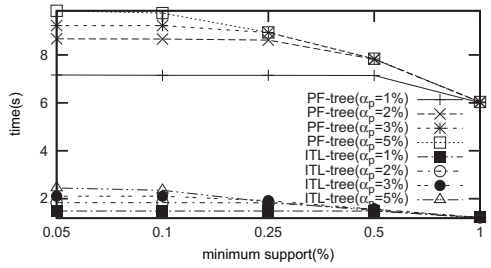

Figure 4. Runtime on T10I4D100K

Figure 5 demonstrates that *ITL-tree* still outperforms *PF-tree* on the dense dataset (*Mushroom*). As the maximum periodicity threshold increase and minimum support threshold decrease, the runtime of *PF-tree* is slightly increased meanwhile the runtime of *ITL-tree* is quite stable because our algorithm gets benefit of using interval transaction-ids list. *ITL-tree* can sum up several transaction-ids by using only one interval transaction-tids list. Then, it can reduce the time to collect transaction-ids and the time to find periodicity of each itemset by using periodicity approximation.

For *T10I4D100K* dataset, *ITL-tree* find over 70% of results that have the same periodicity as exact periodicity when the maximum periodicity is equal or less than 3% and when the minimum support is equal or less than 0.25%. For *Mushroom* dataset, *ITL-tree* can find the exact periodicity over 60% when the maximum periodicity is equal or less than 2% and the minimum support is equal or less than 15%. Obviously *ITL-tree* is very efficient for very small maximum periodicity to find the correct result over 60%.

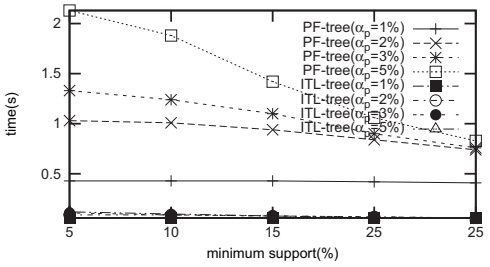

Figure 5. Runtime on Mushroom

V. Conclusion

In this paper, we proposed a new approach to improve the performance of periodic-frequent itemset mining. The periodicity approximation technique is proposed to reduce the time to store transaction-ids list and the time to compute the exact periodicity. Moreover, a new tree structure, named *ITL-tree* (Interval Transaction-ids List tree), is proposed. All of periodic-frequent itemsets can be mined from ITL-tree by using a pattern-growth approach. Based on ITL-tree, several new techniques are applied such as interval transaction-ids list and node link only at the last node of each transaction. By using such techniques, the performance of calculating periodicity and merging occurrence information process can be improved. Our empirical studies, on real and synthetic datasets, show that our approach is efficient.

References

[1] J. Han, H. Cheng, D. Xin, and X. Yan, "Frequent pattern mining: current status and future directions," *Data Min. Knowl. Discov.*, vol. 15, no. 1, pp. 55–86, 2007.

[2] S. K. Tanbeer, C. F. Ahmed, B.-S. Jeong, and Y.-K. Lee, "Discovering periodic-frequent patterns in transactional databases," in *PAKDD 2009*, ser. LNCS, vol. 5476. Springer, 2009, pp. 242–253.

[3] K. Amphawan, P. Lenca, and A. Surarerks, "Mining top-k peirodic-frequent patterns without support threshold," in *The 3rd International Conference on Advances in Information Technology, IAIT'09, Bangkok, Thailand, December 1-5*, ser. CCIS, vol. 55. Springer, 2009, pp. 18–29.