A Novel Incremental Algorithm for Frequent Itemsets Mining in Dynamic Datasets

Raudel Hernández-León[1,2], José Hernández-Palancar[1], J.A. Carrasco-Ochoa[2], and J. Fco. Martínez-Trinidad[2]

[1] Advanced Technologies Application Center (CENATAV), 7a ♯ 21812 e/ 218 and 222, Rpto. Siboney, Playa, C.P. 12200, La Habana, Cuba
[2] Computer Science Department National Institute of Astrophysics, Optics and Electronics
Luis Enrique Erro No. 1, Sta. María Tonantzintla, Puebla, CP:72840, Mexico
{rhernandez,jpalancar}@cenatav.co.cu, {ariel,fmartine}@ccc.inaoep.mx

Abstract. Frequent Itemsets (FI) Mining is one of the most researched areas of data mining. When some new transactions are appended, deleted or modified in a dataset, updating FI is a nontrivial task since such updates may invalidate existing FI or introduce new ones. In this paper a novel algorithm suitable for FI mining in dynamic datasets named Incremental Compressed Arrays is presented. In the experiments, our algorithm was compared against some algorithms as Eclat, PatriciaMine and FP-growth when new transactions are added or deleted.

Keywords: Data mining, Frequent itemsets, Dynamic datasets.

1 Introduction

Mining FI in transaction datasets is useful and technically feasible in several application areas, particularly in retail sales [1]. Traditional methods for data mining typically make the assumption that the dataset is static and a dataset update requires recomputing all the itemsets by scanning the updated dataset.

The use of the prior knowledge to find out new itemsets on the updated dataset produces three kinds of problems: (1) to discover FI under new support threshold without dataset updating; (2) to discover FI when the dataset is updated, but support threshold is unchanged; (3) to discover FI when the dataset is updated and the support threshold is changed.

In this paper, a novel algorithm for FI mining in dynamic datasets named Incremental Compressed Arrays (ICA) which solves the three kinds of problems above mentioned is presented. Our algorithm is based on a breadth first search of FI and the use of equivalence classes to group them. The use of equivalence classes combined with a compressed vertical binary representation of the dataset allows a very fast support count.

The paper is organized as follows: in section 2 the related work is exposed; in section 3 we give some formal definitions; section 4 contains the description of ICA; the experimental results are discussed in the section 5 and finally the conclusion are given in section 6.

J. Ruiz-Shulcloper and W.G. Kropatsch (Eds.): CIARP 2008, LNCS 5197, pp. 145–152, 2008.

2 Related Work

Most FI mining algorithms assume a static dataset, in this approach when the dataset is updated in order to generate the new set of FI the algorithm must be executed again.

Many proposals have been developed to facilitate the updating of FI [4,5,7,8,9,10,11]. Most of the developed algorithms are often referred to as incremental updating strategies when only additions to the transaction dataset are considered.

The first incremental updating strategy was called Fast Update (FUP) [4] and was introduced to deal with addition of new transaction data. In [5] the FUP2 algorithm was proposed, FUP2 can efficiently update discovered FI with addition of some new transactions and deletion of some obsolete ones. The FUP2 algorithm is efficient only when there are few changes in the dataset.

In [7] the ZIGZAG algorithm was presented which is able to find the FI in the update dataset. In this case the author used an incremental technique based on the MFI (Maximal Frequent Itemsets). In [8] the authors proposed two algorithms named Frequent Itemsets Incremental Updating (FIIU) and its distributed variant (DFIIU). They also introduced the concept of Interesting Support Threshold Itemsets on massive dataset, but the experiments showed that FIIU and DFIIU algorithms had better performance than traditional algorithms on massive datasets when the number of items is small.

In [9] a novel data structure, CATS Tree, is developed. CATS Tree extends the idea of FP-Tree to improve storage compression and it allows FI mining without candidate generation. In [10] another tree structure was proposed, CanTree, it captures the content of the transaction dataset and sorts tree nodes according to a canonical order. The CanTree allows addition, deletion and modification on the dataset.

In [11] an incremental updating pattern tree (INUP-Tree) structure and a frequent pattern mining algorithm (IUF-Mine) based on conditional matrix is presented. In IUF-Mine when the dataset is updated, only the new added transactions are scanned. Besides, original conditional matrix is used to speed up the new mining process.

All these incremental algorithms store the whole dataset in memory, into a tree structure where it is easy to add or delete transactions, but they recompute all the FI from the updated tree, without using the previous set of FI. The proposed algorithm does not need to store the whole dataset in memory, and reuses the previous computed FI for searching the new FI set.

3 Preliminaries

Let $I = \{i_1, i_2, \ldots, i_n\}$ be a set of items. Let D be a set of transactions, where a transaction T is a set of items, so that $T \subseteq I$. An itemset X is a subset of I. The support of an itemset X is the number of transactions in D containing X.

If the support of an itemset is greater than or equal to a given support threshold (minSup), the itemset is called a frequent itemset. The size of an itemset is defined as its cardinality, an itemset containing k items is called a k-itemset. In [6] the itemset space is partitioned into equivalence classes based on their common $k-1$ length prefix. The elements of equivalence classes with $k-1$ length prefix has size k.

Most of the algorithms for finding FI are based on the Apriori algorithm [1]. Apriori is a breadth first search algorithm that iteratively generates two kinds of sets: The set L_k containing frequent k-itemsets, and the set C_k containing candidate k-itemsets, $L_k \subseteq C_k$. The set L_k is obtained scanning the dataset and determining the support for each candidate k-itemset in C_k. The set C_k is generated from L_{k-1} as follows:

$$C_k = \{c | Join(c, L_{k-1}) \wedge Prune(c, L_{k-1})\} \tag{1}$$

where:

$$Join(\{i_1, i_2, \ldots, i_{k-2}, i_{k-1}, i_k\}, L_{k-1}) \equiv$$
$$\langle \{i_1, \ldots, i_{k-2}, i_{k-1}\} \in L_{k-1} \wedge \{i_1, \ldots, i_{k-2}, i_k\} \in L_{k-1} \rangle, \tag{2}$$

$$Prune(c, L_{k-1}) \equiv \langle \forall s[(s \subset c \wedge |s| = k-1) \rightarrow s \in L_{k-1}] \rangle \tag{3}$$

4 ICA Algorithm

Unlike most of the incremental algorithms for FI mining, ICA does not use tree structures to store the knowledge that has been discovered. ICA searches the itemset supports doing arithmetic intersections on integer arrays, where each integer represents 32 transactions (in case of a 32 bit architecture).

4.1 Storing FI

In order to store the FI the ICA algorithm creates separated binary files per level, a file for frequent 2-itemsets, another for frequent 3-itemsets and so on. Inside the files, FI are stored in equivalence classes [6] i.e., first the prefix class P and later the suffixes S_i followed by the support of $P \cup S_i$. Additionally, we store a binary file with all different items and its supports, these items are lexicographically ordered and the lexicographical value of an item does not change although the item becomes infrequent.

When ICA algorithm needs to load a binary file to search FI, a fast search is convenient. Therefore, instead of storing the item values in the binary files per level we store the lexicographical value of the items. Therefore, ICA can do the search in time $O(\log n)$, the FI additions in time $O(\log n)$, and the equivalence class addition in time $O(n)$, being n the number of equivalence classes in the current level.

4.2 ICA Algorithm

The datasets can be represented as a m x n binary matrix being m the number of transactions and n the number of frequent items. The binary value 1 or 0 denotes the presence or absence of an item in a transaction respectively. We propose to use an array of integers to store each frequent item, where each integer represents w (CPU word size) transactions. Let M be the binary representation of a dataset. Retrieving from M the columns associated to frequent items, we can represent the column associated to item j as an integer array I_j, as follows:

$$I_j = \{W_{1,j}, W_{2,j}, \ldots, W_{q,j}\},\ q = \lceil m/w \rceil \tag{4}$$

where each integer of the array can be defined as:

$$W_{k,j} = \sum_{r=1}^{w} 2^{w-r} * M_{((k-1)*w+r),j} \tag{5}$$

being $M_{i,j}$ the bit value of item j in transaction i, in case of $i > m$ then $M_{i,j} = 0$.

ICA is a breadth first search algorithm with a vertical binary organization that iteratively generates a list EC_k. The elements of this list represent the equivalence classes containing frequent k-itemsets and have the format:

$$\langle Prefix_{k-1}, IA_{Prefix_{k-1}}, Suffixes_{Prefix_{k-1}} \rangle, \tag{6}$$

where $Prefix_{k-1}$ is the $(k-1)$-itemset that is common to all the itemsets of the equivalence class, $Suffixes_{Prefix_{k-1}}$ is the set of all items j which extend to $Prefix_{k-1}$, where j is lexicographically greater than each item in the prefix, and $IA_{Prefix_{k-1}}$, is an array of non null integers that stores the accumulated intersection (through an *and* operation) of items that belong to $Prefix_{k-1}$. Note that if the amount of FI is huge, the array IA will have less elements due to

Algorithm 1. ICA

Input: Dataset in binary representation
Output: Frequent itemsets

```
1  EC_1 = {frequent 1-itemsets} //global 1-frequents
2  k = 2
3  while EC_{k-1} ≠ ∅ do
4  |   EC_k = ∅,   KFreq = ∅
5  |   LoadKFrequents(KFreq)
6  |   forall ec ∈ EC_{k-1} do
7  |   |   GenAndCount(ec, KFreq, EC_k)
8  |   end
9  |   WriteUpdatedKFrequents(KFreq) //KFreq is updated with EC_k
10 |   k = k + 1
11 end
```

Fig. 1. pseudo code of ICA algorithm

the downward closure property [1]. It allows to compute in a fast way (8). The procedure for obtaining IA is as follows: Let i and j two frequent items,

$$IA_{\{i\}\cup\{j\}} = \{(W_{k,i} \,\&\, W_{k,j}, k) \mid (W_{k,i} \,\&\, W_{k,j}) \neq 0, k \in [1,q]\} \tag{7}$$

and, let X be a FI and j a frequent item

$$IA_{X\cup\{j\}} = \{(b \,\&\, W_{k,j}, k) \mid (b,k) \in IA_X, (b \,\&\, W_{k,j}) \neq 0, k \in [1,q]\}. \tag{8}$$

This representation not only reduces the required memory space to store the integer arrays but also eliminates the Join step described in (2). The ICA algorithm pseudo code is shown in Algorithm 1.

In the step 1 the global frequent 1-itemsets are obtained, notice that only the using the history record of all different items, only global 1-itemsets can generate new FI or to promote infrequent itemsets. The function *GenAndCount* (Algorithm 2) takes as argument a set of frequent $(k-1)$-itemsets (EC_{k-1}),

Algorithm 2. GenAndCount

Input: An equivalence class in $\langle Prefix, IA_{Prefix}, Suffixes_{Prefix}\rangle$ format (EC_{k-1}), the set of frequent k-itemsets currently mined ($KFreq$) and a set of equivalence classes EC_k (initially empty)

Output: The equivalence-classes set generated (EC_k updated)

```
1  Answer = EC_k
2  forall i ∈ Suffixes_Prefix do
3  |  Prefix' = Prefix ∪ {i}
4  |  IA_Prefix' = IA_Prefix∪{i}
5  |  Suffixes'_Prefix' = ∅
6  |  forall (i' ∈ Suffixes_Prefix) and (i' lexicographically greater than i) do
7  |  |  Sup = Support(IA_Prefix'∪{i'})
8  |  |  if (Prefix' ∪ {i'} ∈ KFreq) and (let Sup' its old support) then
9  |  |  |  Sup = Sup + Sup'    //in case of deletion Sup = −Sup + Sup'
10 |  |  |  Sup' = Sup    //the Prefix' ∪ {i'} support is updated in KFreq
11 |  |  end
12 |  |  else
13 |  |  |  Sup = Sup + Rerun(Prefix' ∪ {i'})
14 |  |  end
15 |  |  if Sup > minSup then
16 |  |  |  Suffixes'_Prefix' = Suffixes'_Prefix'∪{i'}
17 |  |  end
18 |  end
19 |  if Suffixes'_Prefix' ≠ ∅ then
20 |  |  Answer = Answer ∪ {⟨Prefix', IA_Prefix', Suffixes'_Prefix'⟩}
21 |  end
22 end
23 return Answer
```

Fig. 2. pseudo code of GenAndCount function

the set of frequent k-itemsets currently mined ($KFreq$) and updates all the equivalence classes EC_k whose elements are frequent k-itemsets. The function *LoadKFrequents* loads the frequent k-itemsets currently mined and the function *WriteUpdatedKFrequents* writes into a binary file the updated FI ignoring the FI that are invalidated due to an increase of the support or due to a deletion or modification of transactions.

In algorithm 2 each new equivalence class is initialized (steps 3-5), later each itemset belonging to the current equivalence class is verified (steps 6-18). In step 7 the support of an itemset X from an integer array IA_X is computed by obtaining the number of bits having 1 in IA_X. The FI are updated (steps 8-11) and new candidates are verified along the datasets (step 13). The function *Rerun* allows us to calculate the itemset support retrieving the integer arrays of each item belonging to the itemset and doing *AND* operations with them. If a verified itemset is frequent then it is added to the current equivalence class (steps 15-18).

For eliminating transactions the algorithm receives as input the *id*-list of transactions to be deleted and, the pseudo code of algorithm 2 only changes in step 9 (see Fig. 2). A modification is a deletion followed by an addition.

5 Experimental Results

We did two experiments in order to evaluate the performance of ICA algorithm because it was not possible to obtain another incremental algorithm to compare our algorithm.

In the first exeriment we divided the dataset, randomly, in ten subsets of transactions, the first subset was mined and the FI were stored in binary files. Later, ICA used that information to mine the other nine subsets and update the FI. The execution times were compared (Fig. 3 and Fig. 4) against the execution time of algorithms as Fp-growth and Eclat [3], and PatriciaMine [2] over the whole dataset after the addition of each subsect of transactions. The second experiment is similar but it begin with the whole dataset and each time one of the subsets is eliminated, ICA is used to update the FIs. The execution times were compared against the execution time of applying algorithms over the reduced datasets. (Fig. 5).

Experiments were performed with two sparse datasets as "T40I10D100K" and "TDT" and two very sparse datasets as "Kosarak" and "Webdocs". Some characteristic of the datasets are shown in Table 1.

Our tests were performed on a PC with an Intel P4 at 3 GHz CPU and 1 GB DDR2 RAM. The operating system was Windows XP SP2. We considered

Table 1. Summary of the main dataset characteristics

	Transactions	Items Count	Avg. Length
TDT	8169	55532	133.5
T40I10D100K	100000	942	39.54
Kosarak	990002	41935	8.1
Webdocs	1704140	5266562	175.98

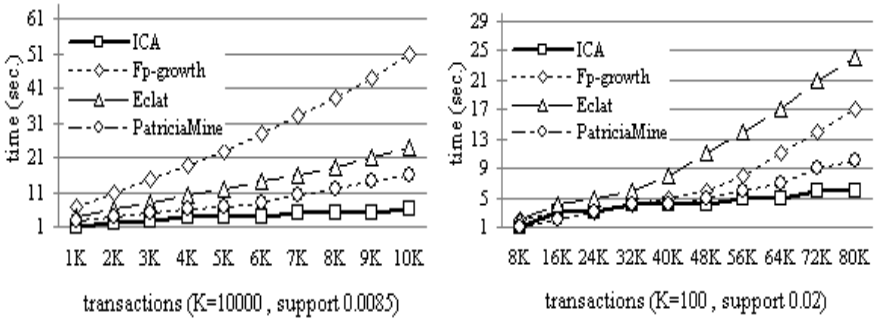

Fig. 3. Adding transactions in "T40I10D100K" and "TDT" datasets

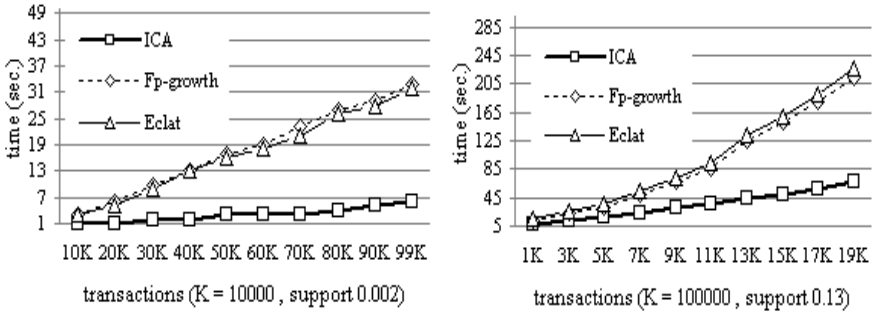

Fig. 4. Adding transactions in "Kosarak" and "Webdocs" datasets

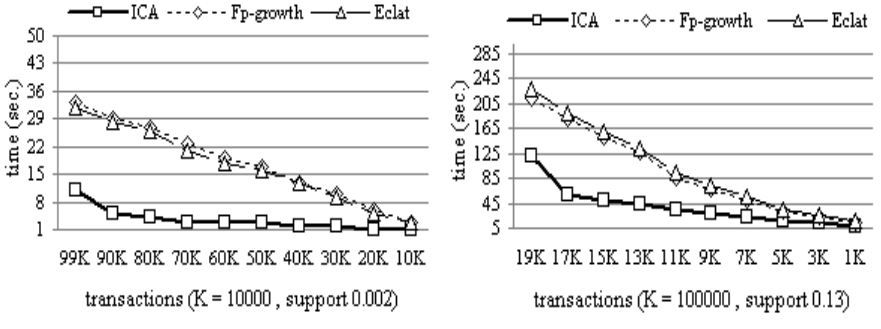

Fig. 5. Deleting transactions in "Kosarak" and "Webdocs" datasets

CPU+IO time (in seconds) at execution time for all algorithms. In the experiments it is shown that ICA algorithm had the best performance with respect to the others algorithms when new transactions are added or deleted. PatriciaMine algorithm had a good performance in the shortest datasets but in the big ones as Kosarak and Webdocs the execution time is over an hour, in these cases the results were not plotted (Fig. 4 and Fig. 5).

6 Conclusions

In this paper a novel algorithm for mining FI in dynamic datasets was presented. In our experiments our ICA algorithm had better performance than the other algorithms and also it was very scalable. The performance improvement of ICA algorithm when working with very huge datasets is an interesting research topic for us. Also, in the incremental process there are many task that can be parallelized thus improving the scalability and performance of our algorithm. These issues will be studied and presented in future works.

References

1. Agrawal, R., Shrikant, R.: Fast Algorithms for Mining Association Rules. In: Proceedings of the 20th VLDB Conference, pp. 487–499 (1994)
2. Pietracaprina, A., Zandolin, D.: Mining Frequent Itemsets using Patricia Tries. In: Proceedings of the ICDM, Workshop on Frequent Itemset Mining Implementations, Melbourne, Florida, USA (2003)
3. A C++ Frequent Itemset Mining Template Library, `http://www.cs.bme.hu/~bodon/en/index.html`
4. Cheung, D., Han, J., Ng, V., Wong, C.Y.: Maintenance of Discovered Association Rules in Large Databases: An Incremental Updating Technique. In: Proceedings of the 12th Intl. Conf. on Data Engineering (1996)
5. Cheung, D., Lee, S., Kao, B.: A General Incremental Technique for Maintaining Discovered Association Rules. In: Proceedings of the 15th Intl. Conf. on Database Systems for Advanced Applications (1997)
6. Zaki, M., Parthasarathy, S., Ogihara, M., Li, W.: New Algorithms for Fast Discovery of Association Rules. Technical Report 651, The University of Rochester, New York, USA (1997)
7. Veloso, A., Meira Jr., W., de Carvalho, M.B., Possas, B., Parthasarathy, S., Zaki, M.: Mining Frequent Itemsets in Evolving Databases. In: Proceedings of the 2nd SIAM Intl. Conf. on Data Mining, Arlington, USA (2002)
8. Veloso, A., Gusmão, W., Meira Jr., W., de Carvalho, M.B., Parthasarathy, S., Zaki, M.: Parallel, Incremental and Interactive Mining for Frequent Itemsets in Evolving Databases. In: Intl. Workshop on High Performance Data Mining: Pervasive and Data Stream Mining (2003)
9. Cheung, W., Zaiane, O.R.: Incremental mining of frequent patterns without candidate generation or support constraint. In: Proceedings of the Seventh IEEE International Database Engineering and Applications Symposium, pp. 111–116 (2003)
10. Leung, C.K., Quamrul, I.K., Hoque, T.: CanTree: A Tree Structure for Efficient Incremental Mining of Frequent Patterns. In: Proceedings of the Fifth IEEE International Conference on Data Mining (2005)
11. Hai, T.H., Shi, L.Z.: A New Method for Incremental Updating Frequent Patterns Mining. In: Proceedings of the Second International Conference on Innovative Computing, Informatio and Control, p. 561 (2007)