An Algorithm for Mining Frequent Itemsets

Raudel Hernández León[1], Airel Pérez Suárez[1],
Claudia Feregrino Uribe[2], Zobeida Jezabel Guzmán Zavaleta[2]
[1] Advanced Technologies Application Center, CENATAV, CUBA
E-mail: {rhernandez,asuarez}@cenatav.co.cu
[2] National Institute for Astrophysics, Optics and Electronics, INAOE, MEXICO
E-mail: {cferegrino,zguzman}@inaoep.mx

***Abstract*— In this paper, we propose a new algorithm for mining frequent itemsets. This algorithm is named *AMFI* (*Algorithm for Mining Frequent Itemsets*). This algorithm compresses the data while maintaining the necessary semantics for the frequent itemsets mining problem and it is more efficient that traditional compression algorithms. The *AMFI* efficiency is based on a compressed vertical binary representation of the data and on a very fast support count. *AMFI* performs a breadth first search through equivalence classes. We compare our proposal with an implementation using *PackBits* algorithm.**

Keywords: data mining, frequent patterns, compression algorithms

I. Introduction

Mining association rules in transaction datasets has been demonstrated to be useful and technically feasible in several application areas, particularly in retail sales [1] and document datasets applications [2]. The management and storage of large datasets have always been a problem to solve. An interesting solution is to use compression algorithms on the data because the presence or absence of an item in a transaction can be stored in a bit. To find an algorithm that compresses the data while maintaining the necessary semantics for the frequent itemsets mining problem is the goal of this work.

The form in which itemsets are represented is decisive to compute their supports. Conceptually, a dataset is a two-dimensional matrix where the rows represent the transactions and the columns represent the items. This matrix can be implemented in the following four different formats [3]: Horizontal Item-List (*HIL*), Horizontal Item-Vector (*HIV*), Vertical Tid-List (*VTL*), and Vertical Tid-Vector (*VTV*). Many algorithms have been proposed using vertical binary representations (*VTV*) in order to improve the obtaining process of frequent itemsets [3], [4], [5], [6].

We proposed an algorithm based on a breadth first search through equivalence classes [7] combined with a compressed vertical binary representation of the dataset. This compressed representation, in conjunction with the equivalence class processing, produces a very fast support count and it produces a less expensive representation, specially in large sparse datasets.

This paper is organized as follows: the next section is dedicated to give some formal definitions; the third section describes some compression algorithms including *PackBits* method, the fourth section contains the description of *AMFI* and the pseudo code of the algorithm; the experimental results are discussed in the fifth section, and the paper finalizes with the conclusion.

II. Preliminaries

Let $I = \{i_1, i_2, \ldots, i_n\}$ be a set of items. Let D be a set of transactions, where each 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 to 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 (*FI*). The size of an itemset is defined as its cardinality; an itemset containing k items is called a k-itemset.

For example, in Figure 1, if we have a support threshold equal to three, the FI obtained are: {coke}, {diaper}, {beer} and {diaper, beer}.

id	ítems
1	coke, milk
2	bread, diaper, beer
3	coke, diaper, beer
4	pan, diaper, beer
5	coke, milk, diaper

Fig. 1: Transactional datasets

The itemset space can be partitioned into equivalence classes based on their common $k-1$ length prefix [7]. The elements of equivalence classes with $k-1$ length prefix have size k (see Fig. 2).

Each equivalence class of level $k-1$ generates several equivalence classes at level k.

Most of the algorithms for finding *FI* are based on the *Apriori* algorithm [8]. To achieve an efficient frequent patterns mining, an anti-monotonic property of frequent itemsets, called the *Apriori* heuristic, was formulated [8]. The basic intuition of this property is that any subset of a frequent itemset must be frequent. *Apriori* is a breadth-first search algorithm, with an *HIL* organization, that iteratively generates two kinds of sets: C_k and L_k. The set L_k contains the frequent k-itemsets. Meanwhile, C_k is

IEEE Catalog Number: CFP08827-CDR
ISBN: 978-1-4244-2499-3
Library of Congress: 2008903800
978-1-4244-2499-3/08/$25.00 ©2008 IEEE

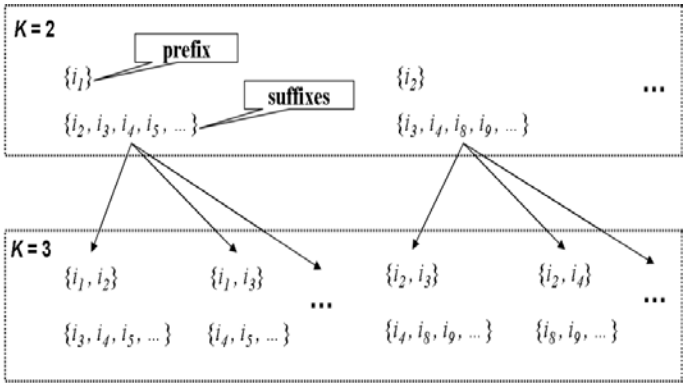

Fig. 2: Equivalence classes

the set of candidate k-itemsets, representing a superset of L_k. This process continues until a null set L_k is generated.

The set L_k is obtained by 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} following the next procedure.

$$C_k = \{c \mid \text{Join}(c, L_{k-1}) \wedge \text{Prune}(c, L_{k-1})\} \quad (1)$$

where:

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

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

The main problem about the computation of FI is the support counting, that is, computing the number of times that an itemset appears in the dataset. To choose a compression algorithm suitable for data compacting and later on to compute the FI is not an easy task.

III. Compression Algorithms

The compression of a transactional dataset can be performed horizontally or vertically. Taking into account the characteristics of the problem, horizontal compaction can be a throwaway due to transactions being defined as sets of items: the sets do not have repeated elements, for which no redundancy is present for a compression algorithm be able to work properly. When data are vertically represented, a transactional identifier list or vector can be obtained per every item. Then, to compute the support for an itemset under this representation, it is required to intersect a transactional identifier list from transactions associated to each item.

Since the last mid-century, many compression algorithms have been developed [9], [10], [11], [12], [13]. In all lossless compression implementations, there is a trade-off between computational resources and the compression ratio. Often, in both statistical and dictionary-based methods, the best compression ratios are obtained at expenses of long execution time and high memory requirements. Statistical compressors are characterized by consuming higher resources than dictionary based when they are implemented in both software and hardware, however they can achieve compression ratios near to the source entropy. The most demanding task in this kind of algorithms is the implementation of the model to get the statistics of the symbols and to assign the bit string. Perhaps, the most representative statistical method is the proposed by Huffman [9] in 1952. In this algorithm a tree is built according to the frequency of the symbols. All symbols are placed at the leaves of the tree. The Huffman method achieves compression by replacing every symbol by a variable bit string. The bit string assigned to every symbol is determined by visiting every internal node from the root up to the leaf corresponding to the symbol. Initially the bit string is the null string. For every internal node visited, one bit is concatenated to the bit string, 1 or 0, depending on the current visited node whether it is a right or left child of its father. Symbols at longer branches will be assigned larger bit strings.

In the dictionary-based methods, the most time-consuming task is searching for strings in a dictionary, which usually has hundreds of locations. Dictionary-based algorithms are considered simpler to implement than statistical ones but the compression ratio is lower. Another kind of compression algorithms, *ad-hoc*, that were developed in early days of data compression are Run Length Encoding-like (RLE) algorithms [14]. RLE takes advantage of the presence of consecutive identical single symbols often found in data streams. It replaces long runs of repeated symbols with a special token and the length of the run. This method is particularly useful for small alphabets and provides better compression ratios when symbols are correlated with their predecessors.

Selecting a compression method among the existent ones is non-trivial. While one method may be faster, other may achieve better compression ratio and yet another may require less computational resources. Furthermore, due to the nature of mining frequent itemsets, using these algorithms for the transactional identifier list compression, the semantics required for the intersection are lost, bringing as a consequence the necessity of decompressing before intersecting.

After a careful analysis of existing compression algorithms, we concluded that RLE type of algorithms are more suitable for compressing our data. In [15] several variants of RLE algorithm are described, however, [16] describes a variant that in our opinion, can adjust better to the type of data managed here and it may compress with higher compression rates besides allowing intersecting without requiring decompression.

A. PackBits Algorithm

PackBits algorithm is a fast and simple compression scheme for run-length encoding of data. A *PackBits* data

IEEE Catalog Number: CFP08827-CDR
ISBN: 978-1-4244-2499-3
Library of Congress: 2008903800

978-1-4244-2499-3/08/$25.00 ©2008 IEEE

stream consists of packets of one byte of header followed by data. The header is a signed byte; the data can be signed or unsigned.

In the following table, let n be the value of the header byte as a signed integer.

Header byte	Data following the header byte
0 to 127	$(1+n)$ literal bytes of data
0 to -127	One byte of data, repeated $(1+n)$ times in the decompressed output

TABLE I: Data stream of PackBits

Note that interpreting 0 as positive or negative makes no difference in the output. Runs of two bytes adjacent to non-runs are typically written as literal data. It should also be noted that there is no way, based on the *PackBits*, data to determine the end of the data stream; that is to say, one must already know the size of the uncompressed data before reading a *PackBits* data stream to know where it ends.

IV. Characteristic of *AMFI* Algorithm

A new algorithm for frequent itemsets mining is proposed in this section. The efficiency of this algorithm is based on a compressed vertical binary representation of the data and on a very fast support count.

A. Storing the transactions

In our algorithm the transactions are represented as an m x n matrix where m is the number of transactions and n is the number of frequent items. We can denote the presence or absence of an item in each transaction by a binary value (1 if it is present, otherwise 0).

If the maximum number of transactions is not greater than the *CPU* word size w (32 or 64 bits), the dataset can be stored as a simple set of integers. However, a dataset is normally much greater than the *CPU* word size. For that reason, we propose to use an array of integers to store the presence or not of each frequent item along the transactions. It will be explained later on how to extend these integer arrays to a frequent itemset.

Let M be the binary representation of a dataset, with n items and m transactions. Retrieving from M the columns associated to frequent items, we can represent each item j as an integer array I_j where each integer has size w, 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 $M_{i,j} = 0$.

B. Reordering of Frequent 1-itemsets

As other authors, in *AMFI*, we have used the heuristic of reordering the frequent 1-itemsets in increasing support order. This will cause a reduction of candidate sets in the next level. This heuristic was first used in MaxMiner [17], and has been used in other methods since then [4], [18], [19], [20], [21], [22], [23]. In the case of our algorithm, reordering frequent 1-itemsets contributes to a faster convergence, as well as saving memory.

C. AMFI Algorithm

AMFI is a breadth-first search algorithm through equivalence classes with a compressed vertical binary representation. This algorithm iteratively generates a list EC_k. The elements of this list represent the equivalence classes of size k and have the format:

$$\langle \text{Prefix}_{k-1}, \text{IA}_{\text{Prefix}_{k-1}}, \text{Suffixes}_{\text{Prefix}_{k-1}} \rangle, \tag{6}$$

where Prefix_{k-1} is the $(k-1)$-itemset that is common to all the itemsets of the equivalence class, $\text{Suffixes}_{\text{Prefix}_{k-1}}$ is the set of all items j which extend to Prefix_{k-1}, where j is lexicographically greater than every item in the prefix, and $\text{IA}_{\text{Prefix}_{k-1}}$, is an array of non null integers that stores the accumulated intersection (AND operation) of items that belong to Prefix_{k-1}. As frequent itemsets are larger, the array IA will have lesser elements. The procedure for obtaining IA is: Let i and j be two frequent items,

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

similarly, let the frequent itemset X and the frequent item j

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

This representation not only reduces the required memory space to store the integer arrays but also eliminates the *Join* step described in (2).

In order to compute the support of an itemset X with an integer-array IA_X, the following expression is considered:

$$\text{Support}(IA_X) = \sum_{(b,k)\in IA_X} \text{BitCount}(b) \tag{9}$$

where $\text{BitCount}(b)$ is a function that calculates the Hamming Weight of b. The IA cardinality is reduced with the increment of the itemsets size due to the downward closure property. It allows for improvement of the processes (8) and (9).The *AMFI* algorithm pseudo code is shown in Algorithm 1.

The ECGenAndCount function takes an equivalence class of length $k-1$ as argument and generates all the equivalence classes of length k (Algorithm 2).

IEEE Catalog Number: CFP08827-CDR
ISBN: 978-1-4244-2499-3
Library of Congress: 2008903800

Input: Dataset in binary representation
Output: Frequent itemsets

1 Answer = $\emptyset$
2 L = {frequent 1-itemsets}
3 **forall** $i \in L$ **do**
4 `ECGenAndCount`($\langle\{i\}, I_i, Suffixes_{\{i\}}\rangle, EC_2$)
5 $k = 3$
6 **while** $EC_{k-1} \neq \emptyset$ **do**
7 **forall** $ec \in EC_{k-1}$ **do**
8 `ECGenAndCount`(ec, EC_k)
9 **end**
10 Answer = Answer $\cup\, EC_k$
11 $k = k + 1$
12 **end**
13 **end**
14 **return** *Answer*

Algorithm 1: AMFI

Input: An equivalence class in $\langle \text{Prefix}, \text{IA}_{\text{Prefix}}, \text{Suffixes}_{\text{Prefix}}\rangle$ format
Output: The equivalence classes set generated

1 Answer = $\emptyset$
2 **forall** $i \in Suffixes_{Prefix}$ **do**
3 Prefix$' = \text{Prefix} \cup \{i\}$
4 $\text{IA}_{\text{Prefix}'} = \text{IA}_{\text{Prefix} \cup \{i\}}$
5 $\text{Suffixes}'_{\text{Prefix}'} = \emptyset$
6 **forall** $(i' \in Suffixes_{Prefix})$ *and* $(i' > i)$ **do**
7 **if** `Support`($IA_{Prefix' \cup \{i'\}}$) **then**
8 $\text{Suffixes}'_{\text{Prefix}'} = \text{Suffixes}'_{\text{Prefix}' \cup \{i'\}}$
9 **end**
10 **end**
11 **if** $Suffixes'_{Prefix'} \neq \emptyset$ **then**
12 Answer = Answer $\cup\, \{\langle \text{Prefix}', \text{IA}_{\text{Prefix}'}, \text{Suffixes}'_{\text{Prefix}'}\rangle\}$
13 **end**
14 **end**
15 **return** *Answer*

Algorithm 2: ECGenAndCount

In line 2 of algorithm ECGenAndCount, all the items i that form the suffix of the input equivalence class (EC_{k-1}) are crossed. In line 3 the prefixes Prefix$'$ of the equivalence classes of level k are built by adding each suffix i to the prefix of EC_{k-1}. In line 4, the IA array associated to each Prefix$'$ is calculated by means of *AND* operation between the IA of EC_{k-1} and the I_i associated to the item i (8). From lines 6 to 13, the suffix items j of EC_{k-1}, lexicographically greater than i, are crossed and the support of the sets Prefix$' \cup j$ is calculated.

D. Memory Considerations

There are four ways in which a dataset can be represented, two horizontal (*HIL* and *HIV*) and two vertical (*VTL* and *VTV*). Most of the authors agree in the advantages of the vertical storage over the horizontal because of vertical storage allows to calculate the itemset support by intersecting lists or arrays according to the case. Making a decision from *VTL* and *VTV* representations is a non-trivial task. Burdick, Calimlim and Gehrke [4] analyzed these two vertical formats. They pointed out that the memory efficiency of these representations depends on the density of the dataset. Particularly, on 32-bit machines the *VTL* format is guaranteed to be a more expensive representation in terms of space if the support of an item (or itemset) is greater than 1/32 or about 3%. In the *VTL* representation, we need an entire word to represent the presence of an item versus the single bit of the *VTV* approach.

In the compressed IA array of the *AMFI* algorithm, a pair of integers for each one of the simple (uncompressed) *VTV* format is required. As the *AMFI* representation includes pairs of words only for non-null integers, the memory overhead for this representation is higher than the simple *VTV* if the support of an item (or itemset) is greater than 1/2. Furthermore, considering a dataset of m transactions on 32 bit machines and an item (or itemset) with a support *sup*, a simple *VTV* requires $m/8$ bytes of memory while *AMFI*, in its worst case when the item (or itemset) transactions are sparsely distributed, requires $8 * \min(m * sup, m/32)$ bytes. If the *sup* decreases then the memory consumption decreases too.

V. Experimental Results

Several experiments were carried out where our proposed algorithm, AMFI, was compared against a version that compresses the data using PackBits. Time consumption and memory requirements were considered as measurements of efficiency.

Experiments were developed with two newsitem datasets and two synthetic datasets (Table II).

	Transactions	Items Count	Avg. Length
El Pais	550	14489	173.1
TDT	8169	55532	133.5
Kosarak	990002	41935	8.1
Webdocs	1704140	5266562	175.98

TABLE II: Summary of the main datasets characteristics

Some of these datasets are sparse, such as *El Pais*, and some, very sparse, such as *Webdocs*. Newsitems datasets were lemmatized using the *Treetagger* program [24], and the *stopwords* were eliminated.

The *Kosarak* dataset was provided by Ferenc Bodon to *FIMI* repository [25] and contains (anonymized) click-stream data of a Hungarian on-line news portal. The *Webdocs* dataset was built from a spidered collection of web html documents and was donated to *FIMI* repository by Claudio Lucchese *et al.* *TDT* dataset contains news (newsitems) data collected daily from six news sources in American English, over a period of six months (January

IEEE Catalog Number: CFP08827-CDR
ISBN: 978-1-4244-2499-3
Library of Congress: 2008903800

- June, 1998). The *El Pais* dataset contains 550 news, published at El Pais (Spain) newspaper in June in 1999.

Our tests were performed on a PC with an Intel Core 2 Duo at 1.86 GHz CPU and 1 GB DDR2 RAM. The operating system was Windows XP SP2. We considered CPU+IO time (in seconds) at execution time for all algorithms included in this paper.

In Figures 3, 4, 5 and 6, a comparison of memory consumption by level is shown, meaning that the comparison is done for frequent 1-itemsets, frequent 2-itemsets and so on until frequent 6-itemset. We plot the values until level 6 in order to not overload the graphics, but the performance is the same.

As it can be seen from the figures, the *AMFI* algorithm requires less memory than the variant of *PackBits* as the size of the *FI* increases. In level 1, *AMFI* consumes more memory since it stores all the bytes while *PackBits* compresses the bytes with equal value (as the datasets are very sparse, see Table II, many bytes are equal to 0).

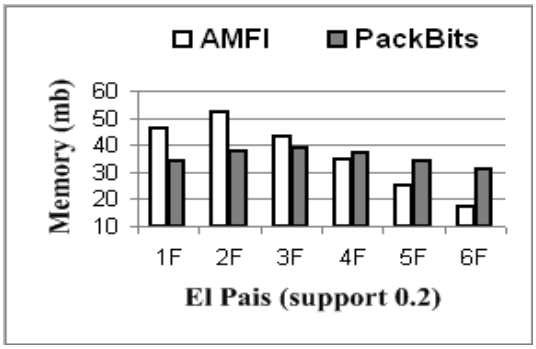

Fig. 3: Memory consumption (El Pais dataset)

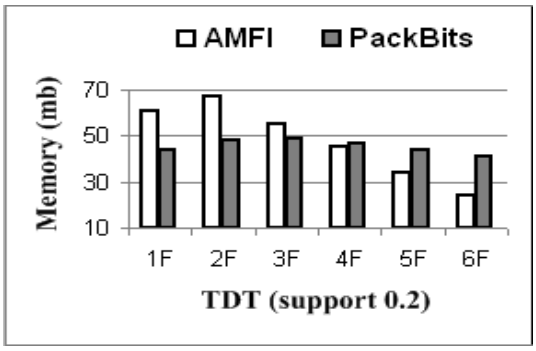

Fig. 4: Memory consumption (TDT dataset)

As the levels increase, *PackBits* requires always to compress a constant amount of bytes, while *AMFI* will store only the bytes that are different from 0, which diminish fast due to the intersection operations.

In Figures 7, 8, 9 and 10 a comparison of execution time with different supports is shown. As it can be seen, *AMFI* algorithm not only requires less memory but also is more efficient. This is mainly due to intersecting only with the blocks different from 0 is faster than iterating two compressed byte flows intersecting all the blocks.

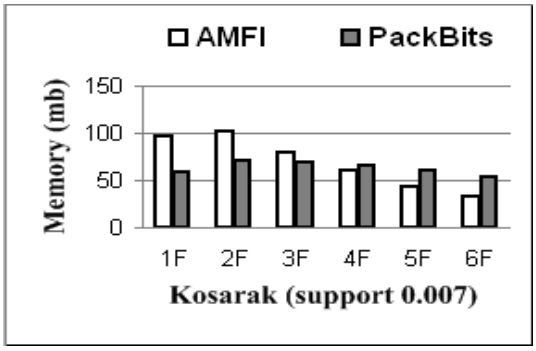

Fig. 5: Memory consumption (Kosarak dataset)

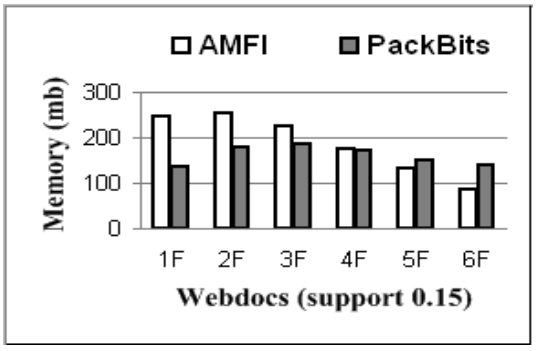

Fig. 6: Memory consumption (Webdocs dataset)

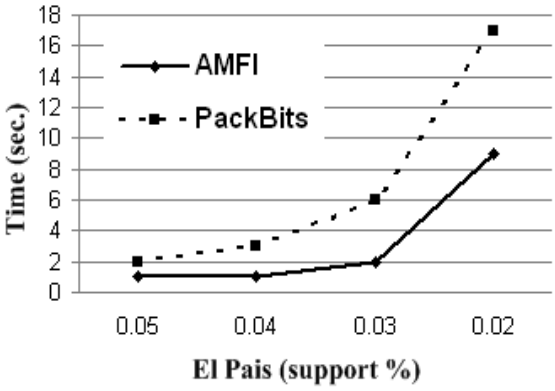

Fig. 7: Time consumption (El Pais dataset)

VI. Conclusions

In this paper we have presented a compressed vertical binary approach for mining *FI*. Our algorithm achieves better performance than *PackBits* as much in consumption of memory as in run time. It can be concluded that although existing compression methods are good, they are not always suitable for certain problems due to when compressing the required semantic is lost.

IEEE Catalog Number: CFP08827-CDR
ISBN: 978-1-4244-2499-3
Library of Congress: 2008903800

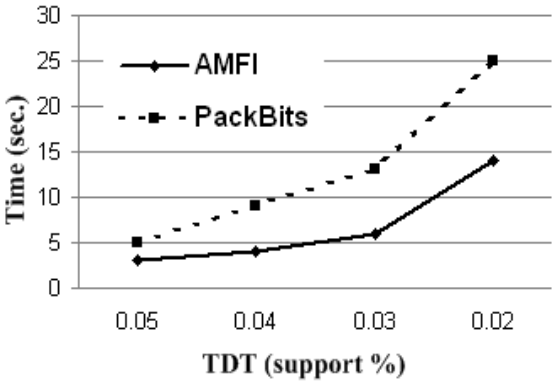

Fig. 8: Time consumption (TDT dataset)

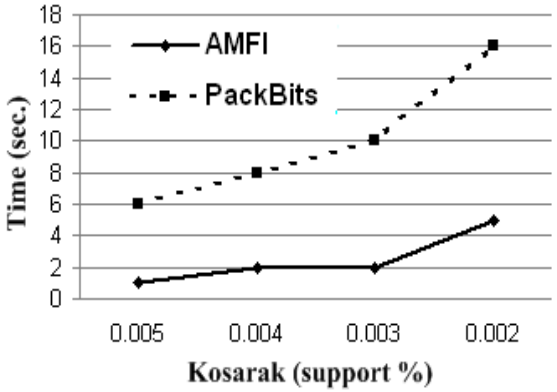

Fig. 9: Time consumption (Kosarak dataset)

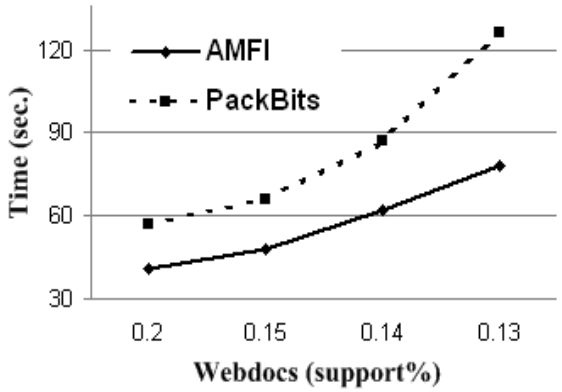

Fig. 10: Time consumption (Webdocs dataset)

References

[1] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. *In Proceedings of the ACM SIGMOD International Conference on Management of Data, Washington, D.C.*, pages 207–216, 1993.

[2] R. Feldman and I. Dagan. Kdt-knowledge discovery in texts. *In Proceedings of the First International Conference on Knowledge Discovery (KDD)*, pages 112–117, 1995.

[3] P. Shenoy, J. Haritsa, S. Sudarshan, G. Bhalotia, M. Bawa, and D. Shah. Turbo-charging vertical mining of large databases. *In Proceedings of the ACM SIGMOD International Conference on Management of Data, Dallas, USA*, 2000.

[4] D. Burdick, M. Calimlim, and J. Gehrke. Mafia: A maximal frequent itemset algorithm for transactional databases. *In Proceedings of the International Conference on Data Engineering (ICDE), Heidelberg, Germany*, 2001.

[5] G. Grahne and J. Zhu. Fast algorithms for frequent itemset mining using fp-trees. *IEEE Transactions on Knowledge and Data Engineering, Vol. 17, No. 10*, pages 1347–1362, 2005.

[6] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. *In Proceedings ACM-SIGMOD International Conference on Management of Data, New York, NY, USA*, 2000.

[7] M. J. Zaki, S. Parthasarathy, M. Ogihara, and W. Li. New algorithms for fast discovery of association rules. *In Proceedings of the 3rd International Conference on KDD and Data Mining, EU*, 1997.

[8] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. *In Proceedings of the 20*th *International Conference on Very Large Data Bases, VLDB'94, Santiago de Chile, Chile*, pages 487–499, 1994.

[9] D. Huffman. A method for the construction of minimum redundancy codes. *In Proceedings of the IRE 40(9)*, pages 1098–1101, 1952.

[10] J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory IT-23(3)*, pages 337–343, 1977.

[11] J. Ziv and A. Lempel. Compression of individual sequences via variable-rate coding. *IEEE Transactions on Information Theory IT-24(5)*, pages 530–536, 1978.

[12] E. R. Fiala and D. H. Greene. Data compression with finite windows. *Communications of the ACM 32(4)*, pages 490–505, 1989.

[13] D. Phillips. Lzw data compression. *The Computer Application Journal Circuit Cellar Inc., 27*, pages 36–48, 1992.

[14] Gollomb S. W. Run-length encodings. *IEEE Transactions on Information Theory, 12*, pages 399–401, 1966.

[15] D. Salomon. Data compression: The complete reference. *3rd Edition, Published by Springer. ISBN 0-387-40697-2. LCCN QA76.9 D33S25, 899 pages*, 2004.

[16] *http://www.fileformat.info/format/tiff/corion-packbits.htm.*

[17] R. J. Bayardo. Efficiently mining long patterns from databases. *In ACM SIGMOD Conf. on Management of Data*, pages 85–93, 1998.

[18] R. Agrawal, C. Aggarwal, and V. Prasad. Depth first generation of long patterns. *In 7th Int'l Conference on Knowledge Discovery and Data Mining*, pages 108–118, 2000.

[19] K. Gouda and M. J. Zaki. Genmax: An efficient algorithm for mining maximal frequent itemsets. *Data Mining and Knowledge Discovery, 11*, pages 1–20, 2005.

[20] M. J. Zaki and C. J. Hsiao. Charm: An efficient algorithm for closed itemset mining. *In 2nd SIAM International Conference on Data Mining*, pages 457–473, 2002.

[21] T. Calders, N. Dexters, and B. Goethals. Mining frequent itemsets in a stream. *Proceedings of the IEEE International Conference on Data Mining*, pages 83–92, 2007.

[22] T. Calders, N. Dexters, and B. Goethals. Mining frequent items in a stream using flexible windows. *Intelligent Data Analysis; Vol. 12, nr. 3*, 2008.

[23] B. Kalpana and R. Nadarajan. Incorporating heuristics for efficient search space pruning in frequent itemset mining strategies. *CURRENT SCIENCE 94*, pages 97–101, 2008.

[24] H. Schmid. Probabilistic part-of-speech tagging using decision trees. In *International Conference on New Methods in Language Processing, (Software in: www.ims.uni-stuttgart.de/ftp/pub/corpora/tree-tagger1.ps.gz)*, Manchester, UK, 1994.

[25] *FIMI-Frequent Itemset Mining Implementations repository, URL: http://fimi.cs.helsinki.fi/src.*

IEEE Catalog Number: CFP08827-CDR
ISBN: 978-1-4244-2499-3
Library of Congress: 2008903800

978-1-4244-2499-3/08/$25.00 ©2008 IEEE