Mining Frequent Item Sets Efficiently by Using Compression Techniques

Selim Mimaroglu, Cagri Cubukcu, Emin Aksehirli, and Ertunc Erdil
Department of Computer Engineering, Bahcesehir University, Ciragan Cad. 34353 Besiktas, Istanbul, Turkey

Abstract— *Mining frequent item sets in a data set is a significant problem of data mining that can only be solved in exponential time. For especially very large data sets, finding frequent item sets in practical run times is extremely important. A market basket data set can be represented by a set of bit vectors, which enables fast computation and low memory requirements. Space requirements can be reduced and better run time results can be obtained if bit vectors can be compressed, and binary operations can be applied on compressed bit vectors. In this paper, we study the advantages and disadvantages of using compression techniques for finding frequent item sets. Experimental evaluations clearly show that applying special purpose compression techniques has many benefits on a wide range of data sets.*

1. Introduction

Frequent item sets can be defined as subsets of items that appear together frequently in a data set, where frequency value is set by the user and is determined with regards to business, type of items, location and season.

Although identification of frequent item sets has been studied widely, it is still a major research topic because of its computationally expensive nature. Finding frequent item sets plays an essential role to disclose the hidden relations between items. Furthermore, frequent item set detection can be used in other data mining tasks like classification and clustering [1].

Apriori [2] is a seminal frequent item set detection algorithm which starts by scanning the data set to count each item and regards the items existing frequently enough as frequent 1-item sets. Like all the level-wise algorithms, Apriori uses frequent 1-item sets and rescans the data set for discovering frequent 2-item sets. Larger frequent item sets are discovered in level-wise manner, and the algorithm halts when all the frequent item sets are discovered. Scanning the data set at each iteration and generating the candidate item sets are the most important shortcoming of Apriori algorithm. Therefore, using Apriori on very large data sets is impractical.

FP-growth [3] algorithm mines frequent item sets without candidate generation. It also compresses the data set into a data structure called FP-tree, which generally fits into the main memory, for efficient scanning. FP-growth finds all the frequent item sets by searching the FP-tree, recursively.

ECLAT, which is another efficient frequent item set detection technique using vertical data set format, is introduced in [4]. Advantages of vertical representation of a data set is presented in this work as well.

Since finding all the frequent item sets is computationally challenging, there has been studies for finding a preponderant portion of the actual frequent item sets. A very good method which is based on agglomerative clustering of items for finding approximative frequent item sets is known as AFISA [5].

Our paper is organized as follows: Section 2 presents definitions and notations on frequent item sets and association rules. Section 3 shows some benefits of vertical data set format in binary. Special purpose binary compression techniques are presented in Section 4. Following, we present mining frequent item set process with compressed bit vectors in Section 5. Experimental test data sets and experimental results can be found in Section 6. Finally, we conclude with Section 7.

2. Frequent Item Set Detection and Association Rules

Let $I = \{I_1, I_2, \ldots, I_m\}$ be a set of items in data set D. A transactional data set consists of transactions $T = \{T_1, T2, \ldots, T_n\}$, where each T is a set of items such that $T \subseteq I$. Let A be a set of items which satisfies the condition $A \subseteq I$. For a transaction $T_i \in T$, if the condition $A \subseteq T_i$ is satisfied, then it is said that T_i contains A. Number of transactions that contains A is called $support\ count$ of A and is denoted with $support\_count(A)$. Frequency is determined by a user specified threshold called $minimum\ support\ count$, min_sup. A is said to be frequent k-item set, if and only if $|A| = k$, and $support\_count(A) \geq min\_sup$ hold.

As mentioned earlier, finding frequent item sets enables inferring relationships between set of items. Association rule analysis can be performed after detecting frequent item sets, and it is widely used in market basket data sets. Given two frequent item sets A and B, an association rule of the form $A \Rightarrow B$ indicates that customers who bought the item set A are likely to by the item set B. Of course, association rules are not limited to market basket data sets, wireless telecommunication companies can use association rules in their operational data sets for increasing quality of service

as well. The rule $A \Rightarrow B$ has the following confidence

$$confidence(A \Rightarrow B) = \frac{support\_count(A \cup B)}{support\_count(A)} \quad (1)$$

3. Mining Frequent Item Sets on Vertical Data Set Format

Most of the frequent item set detection algorithms operate on horizontal data sets as shown in Table 1, where each row represents a transaction and each transaction contains a set of items. It is possible to represent this data set in binary format as shown in Table 2. In binary representation 1 indicates the existence of the corresponding item in the corresponding transaction.

Table 1: A transactional Data Set in Horizontal Format

Transaction ID	Items
T_1	I_1,I_2,I_3
T_2	I_2, I_3
T_3	I_1, I_2, I_4
T_4	I_1, I_4
T_5	I_3, I_4, I_5
T_6	I_1, I_2, I_3, I_4

Table 2: Binary Representation of a Horizontal Transactional Data Set

Transaction ID	Items				
	I_1	I_2	I_3	I_4	I_5
T_1	1	1	1	0	0
T_2	0	1	1	0	0
T_3	1	1	0	1	0
T_4	1	0	0	1	0
T_5	0	0	1	1	1
T_6	1	1	1	1	0

Table 3: A vertical data set

Items	Transaction ID
I_1	T_1, T_3, T_4, T_6
I_2	T_1, T_2, T_3, T_6
I_3	T_1, T_2, T_5, T_6
I_4	I_3, I_4, T_5, T_6
I_5	T_5

Vertical data set format is introduced in many resources, such as [4]. Table 1 can be transformed into vertical format as shown in Table 3. In vertical data set format it is easier to obtain the occurrence information of an item, and perform set operations such as AND, OR, XOR. A vertical data set can be represented in binary as shown in Table 4. In this representation each item constitutes a bit vector, and bitwise set operations such as AND, OR, XOR can be performed. It should be noted that bitwise set operations are extremely fast, on CPUs with 64 bit architecture 64 bits are ANDed, ORed, or XORed at once. Most of the existing algorithms can be implemented on binary representation of vertical data set format to utilize the advantages of bitwise set operations.

Table 4: A vertical data set with binary representation

Transaction ID	Items				
	I_1	I_2	I_3	I_4	I_5
T_1	1	1	1	0	0
T_2	0	1	1	0	0
T_3	1	1	0	1	0
T_4	1	0	0	1	0
T_5	0	0	1	1	1
T_6	1	1	1	1	0

Unfortunately, sparse data sets waste a lot of space when represented by bit vectors. But, very good compression ratios can be obtained on sparse data sets since they mostly consist 0s. In order to save space and speed up operations we use compression techniques on bit vectors. Constantly compressing items for saving storage space and decompressing them for performing any operation is very costly. Therefore, we investigate and use compression techniques that allow bitwise set operations on the compressed bit vectors which avoids redundant and costly compression and decompression operations.

4. Compressing Binary Data Sets

For very large data sets bit vector representation of the data can be troublesome to fit into the main memory. Although bit vectors can be compressed, using general purpose compression schemes are not preferred because of costly compression and decompression overheads at each operation.

Special purpose compression techniques have been studied in data mining, although not very widespread. MAFIA [6] uses a technique that compresses the data adaptively according to its context. Compression is done on the fly with respect to a cost estimation method. VIPER [7] utilizes a compression technique called *skinning* which is based on Golomb encoding scheme and a novel candidate generation method.

Some other compression techniques are specifically designed to compress bit vectors while being able to apply bitwise set operations directly on the compressed forms-without the overhead of decompression. Even though such compression techniques may not compress the item bit vectors as good as the general purpose compression methods, their overall performance is higher by eliminating considerable overheads.

Byte-aligned Bitmap Coding (BBC) [8] is a Run Length Encoding (RLE) method that compresses the bit vectors in bytes. BBC considers bytes instead of bits since CPUs work more effectively on bytes than arbitrary number of bits. Word Aligned Hybrid (WAH) [9] compression algorithm is another RLE based technique that uses word-length atomic units for compression and utilizes modern CPU architectures. Enhanced Word Aligned Hybrid (EWAH) [10] tries to improve the compression ratio of WAH by sorting these

words. Position List Word Aligned Hybrid (PLWAH) [11] decreases the storage space requirement and improves the performance of WAH.

The compression technique to be utilized has a major effect on speed. Even though, in theory, any lossless compression technique can be used for compression, choices are limited when aim is the speed improvement. Since uncompressed bit map operations are very fast, it is possible that the operational advantages of compressed bit maps are overshadowed by the compression/decompression cost. In general, soft computing compression techniques are not deterministic in terms of time and efficiency, therefore they are not eligible for the tests. Aligned Bit Map Compression techniques offers a good balance between compression ratio and compression/decompression time.

4.1 Aligned Bit Map Compression

BBC, WAH and EWAH implement a special version of RLE which encodes an input sequence by representing each element with its occurrence. RLE scheme works best on sequences having lots of repetitions. For a sequence $S = 1,1,1,1,1,3,3,4,4,4,2,2$, basic RLE compression produces a compressed sequence $S_C = 5,1,2,3,3,4,2,2$. In S_C elements with odd indices represent the number of occurrences of their consecutive element. For sequence S, compression ratio of basic RLE is $\frac{|S_C|}{|S|} = \frac{8}{12}$.

Modern CPUs process the data in byte or word sized units. Thus, in most situations processing the whole word can be faster than processing just one bit. Byte aligned and word aligned compression methods exploit this behavior and instead of working directly on bits they group the bit sequences into bytes or word sized chunks. If two or more consecutive chunks are same, they will be replaced with a *fill byte/word* which consists of number of repetition and the repeated chunk. If a chunk is unique compared with its left and right chunks then it is added to the compressed sequence unmodified as a *literal byte/word*. More details on bit vector compression algorithms can be found in [12].

5. Mining Frequent Item Sets by Using Compressed Item Bit Vectors

We represent data sets in binary format where each item is a bit vector. Furthermore, we compress each item and operate directly on the compressed bit vectors. Using compressed form of the data set has the advantages of improved processing speed and reduced storage space. Bitwise operations on WAH compressed bit vectors are much faster than operations on uncompressed bit vectors if the compression level reaches to a certain level [9]. Similar results are reported on EWAH as well. We choose to use EWAH for compression since it is superior to WAH and its implementation is publicly available by its author.

As explained in detail in Section 4, RLE compression techniques looks for the repetitious patterns in the data and express them as a repetition number and the repeated string. Even though the repetitions in a data set, thus compression ratio, can not be known exactly in advance, a correlation exists between the density of the data set and the compression ratio.

If the data set is sparse, or dense, there is a high probability for existence of long sequences of 0's, or 1's. Long sequences are effectively compressed by RLE compression techniques, therefore, ratio of compression is much better on the dense or sparse data sets.

6. Experimental Results

We conducted experiments for mining all the frequent item sets by representing binary formatted vertical data sets with compressed bit vectors. Experiments are performed on a computer having Quadcore Intel Xeon CPU, 32GB main memory which runs on Linux operating system. Our choice of implementation language is Java, which provides built-in support for bit vectors and bitwise operations. For compressing item bit vectors, we used the Java implementation of EWAH which is obtained from http://code.google.com/p/javaewah.

Properties of test data sets are presented in Table 5. Compressed and original size of bit vectors can be found in Table 6, where compression is calculated as the ratio of output size to the input size as described in [13]-smaller values indicate better compression. Table 6 clearly shows that compression dramatically reduces the space requirement on all the data sets.

Our experimental evaluations show that by using compression, execution time of frequent item set detection process improves considerably on some data sets. Run time results are shown in Figures 1, 2, 3, and 4. Data sets are organized according to their number of transactions and items. On test data sets having compression ratio of 6.5×10^{-3} mining process works 4 to 6 times faster with compressed bit vectors, which is remarkable. On the data sets having compression ratio of 32×10^{-3} results are comparable. And, on the data sets having compression ratio of 65×10^{-3} compression looses its speed advantage. We noticed that although compression is very useful for saving space on all test data sets, it reduces run time considerably for data sets with compression ratio of 32×10^{-3} or better.

Mining frequent item sets by using compression techniques is very useful on data sets having good compression ratios, which can be obtained with dense or sparse data sets. These kind of data sets are common in telecommunication, retail, banking, and finance industries as well as on some data streams.

Table 5: Properties of Test Data Sets

Data Set Name	Transactions	Items	Transactions/Item (average)
100MT-100I-0.01	100M	100	10099
100MT-100I-0.05	100M	100	50486
100MT-100I-0.1	100M	100	100949
100MT-1KI-0.01	100M	1000	10099
100MT-1KI-0.05	100M	1000	50487
100MT-1KI-0.1	100M	1000	100948
10MT-100I-0.01	10M	100	1009
10MT-100I-0.05	10M	100	5048
10MT-100I-0.1	10M	100	10095
10MT-1KI-0.01	10M	1000	1009
10MT-1KI-0.05	10M	1000	5048
10MT-1KI-0.1	10M	1000	10094

Table 6: Compression Ratios of Test Data Sets

Data Set Name	Original size in bytes	Compressed size in bytes	Compression ratio
100MT-100I-0.01	12500000	80788	6.5×10^{-3}
100MT-100I-0.05	12500000	403884	32×10^{-3}
100MT-100I-0.1	12500000	807588	65×10^{-3}
100MT-1KI-0.01	12500000	80788	6.5×10^{-3}
100MT-1KI-0.05	12500000	403892	32×10^{-3}
100MT-1KI-0.1	12500000	807580	65×10^{-3}
10MT-100I-0.01	1250000	8068	6.5×10^{-3}
10MT-100I-0.05	1250000	40380	32×10^{-3}
10MT-100I-0.1	1250000	80756	65×10^{-3}
10MT-1KI-0.01	1250000	8068	6.5×10^{-3}
10MT-1KI-0.05	1250000	40380	32×10^{-3}
10MT-1KI-0.1	1250000	80748	65×10^{-3}

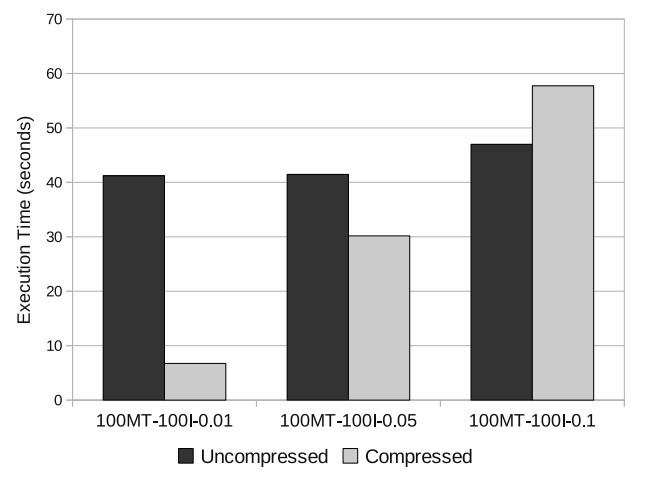

Fig. 1: Runtime results for the 100M-100I-0.01, 100M-100I-0.05, 100M-100I-0.1 data sets

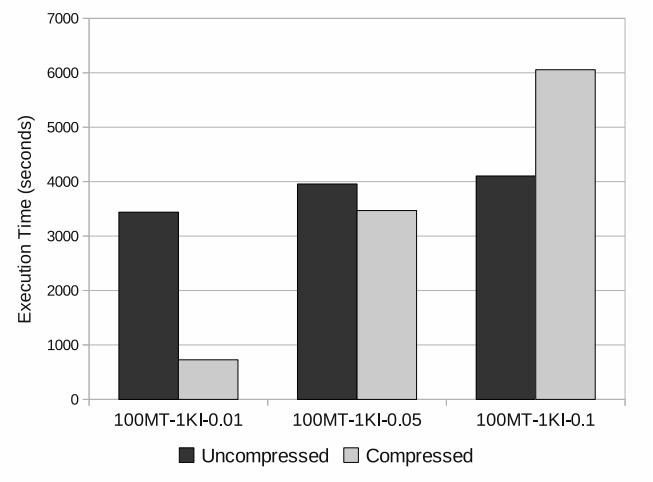

Fig. 2: Runtime results for the 100M-1KI-0.01, 100M-1KI-0.05, 100M-1KI-0.1 data sets

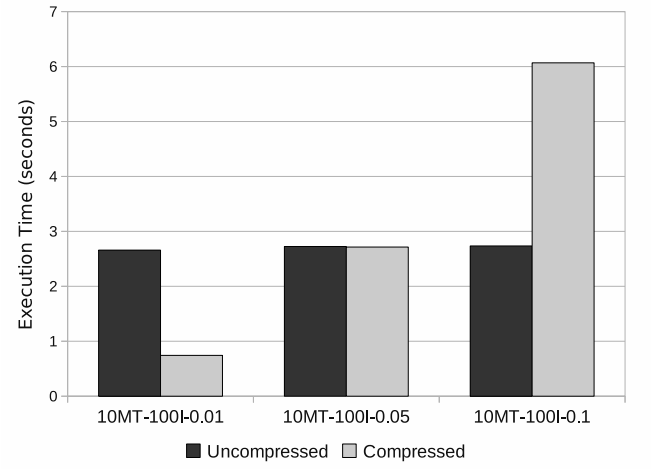

Fig. 3: Runtime results for the 10M-100I-0.01, 10M-100I-0.05, 10M-100I-0.1 data sets

7. Conclusion

We studied the benefits of using special compression techniques on binary data sets for mining frequent item sets which allow applying bitwise set operations directly on the compressed bit vectors. It is fair to say that compression is very useful for saving space. And, compression is very useful for obtaining better run times on data sets with good compression ratios. Our work can be widely applied in practice with very good results, since most of the data sets in telecommunication, retail, banking, and finance industries are very sparse with very good compression ratios.

References

[1] J. Han and M. Kamber, *Data mining: concepts and techniques*. Morgan Kaufmann, 2006.

[2] R. Agrawal and R. Srikant, "Mining sequential patterns," in *Data Engineering, 1995. Proceedings of the Eleventh International Conference on*. IEEE, 2002, pp. 3–14.

[3] J. Han, J. Pei, Y. Yin, and R. Mao, "Mining frequent patterns without candidate generation: A frequent-pattern tree approach," *Data mining and knowledge discovery*, vol. 8, no. 1, pp. 53–87, 2004.

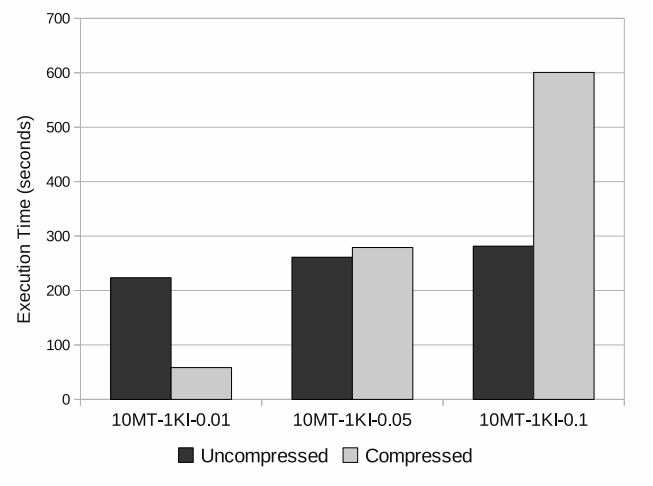

Fig. 4: Runtime results for the 10M-1KI-0.01, 10M-1KI-0.05, 10M-1KI-0.1 data sets

[4] M. Zaki, "Scalable algorithms for association mining," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 12, no. 3, pp. 372–390, 2002.

[5] S. Mimaroglu and D. Simovici, "Clustering and Approximate Identification of Frequent Item Sets," in *Proceedings of the Twentieth International Florida Artificial Intelligence Research Society Conference*, 2007, pp. 502–506.

[6] D. Burdick, M. Calimlim, J. Flannick, J. Gehrke, and T. Yiu, "MAFIA: a maximal frequent itemset algorithm," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 17, no. 11, pp. 1490–1504, 2005.

[7] P. Shenoy, J. R. Haritsa, S. Sudarshan, G. Bhalotia, M. Bawa, and D. Shah, "Turbo-charging vertical mining of large databases," *ACM SIGMOD Record*, p. 22–33, 2000, ACM ID: 335376.

[8] G. Antoshenkov, "United states patent: 5363098 - byte aligned data compression," Nov. 1994. [Online]. Available: http://patft.uspto.gov/netacgi/nph-Parser?Sect1=PTO1&Sect2=HITOFF&d=PALL&p=1&u=%2Fnetahtml%2FPTO%2Fsrchnum.htm&r=1&f=G&l=50&s1=5363098.PN.&OS=PN/5363098&RS=PN/5363098

[9] K. Wu, E. J. Otoo, and A. Shoshani, "Optimizing bitmap indices with efficient compression," *ACM Transactions on Database Systems (TODS)*, vol. 31, p. 1–38, Mar. 2006, ACM ID: 1132864.

[10] O. Kaser, D. Lemire, and K. Aouiche, "Histogram-aware sorting for enhanced word-aligned compression in bitmap indexes," *Proceeding of the ACM 11th international workshop on Data warehousing and OLAP*, p. 1–8, 2008, ACM ID: 1458434.

[11] F. Deliège and T. B. Pedersen, "Position list word aligned hybrid: optimizing space and performance for compressed bitmaps," *Proceedings of the 13th International Conference on Extending Database Technology*, p. 228–239, 2010, ACM ID: 1739071.

[12] K. Wu, E. Otoo, A. Shoshani, and H. Nordberg, "Notes on design and implementation of compressed bit vectors," Lawrence Berkeley National Laboratory, Technical Report LBNL/PUB-3161, Sept. 2001. [Online]. Available: http://crd.lbl.gov/~kewu/fastbit/publications.html

[13] K. Sayood, *Introduction to data compression*. Morgan Kaufmann, 2000.