Indonesian Journal of Electrical Engineering and Computer Science
Vol. 8, No. 2, November 2017, pp. 487 ~ 494
DOI: 10.11591/ijeecs.v8.i2.pp487-494

Towards Scalable Algorithm for Closed Itemset Mining in High-Dimensional Data

Fatimah Audah Md. Zaki*[1], Nurul Fariza Zulkurnain[2]
Department of Electrical and Computer Engineering,
Kuliyyah of Engineering, International Islamic University Malaysia
*Corresponding author, e-mail: fatimah.audah@gmail.com[1], nurulfariza@iium.edu.my[2]

Abstract

Mining frequent itemsets from large dataset has a major drawback in which the explosive number of itemsets requires additional mining process which might filter the interesting ones. Therefore, as the solution, the concept of closed frequent itemset was introduced that is lossless and condensed representation of all the frequent itemsets and their corresponding supports. Unfortunately, many algorithms are not memory-efficient since it requires the storage of closed itemsets in main memory for duplication checks. This paper presents BFF, a scalable algorithm for discovering closed frequent itemsets from high-dimensional data. Unlike many well-known algorithms, BFF traverses the search tree in breadth-first manner resulted to a minimum use of memory and less running time. The tests conducted on a number of microarray datasets show that the performance of this algorithm improved significantly as the support threshold decreases which is crucial in generating more interesting rules.

Keywords*: closed itemsets mining, association rules, high-dimensional data, scalable algorithm*

Copyright © 2017 Institute of Advanced Engineering and Science. All rights reserved.

1. Introduction

Among the numerous data mining tasks, association rules have been extensively used with the aim of describing interesting relationships between variables in a dataset. Agrawal et al. [1] introduced the Frequent Itemset Mining (FIM) problem as part of association rule discovery. An *itemset* is a collection of related items that occur together in a given dataset and is considered frequent if the number of rows which contains that itemset is not less than the minimum support, $\bar{\sigma}$. The support threshold limits the search space by pruning the itemsets that do not meet $\bar{\sigma}$. Many applications require the use of low $\bar{\sigma}$ as many interesting rules exist at the lower end of the threshold. Unfortunately, the number of extracted patterns grows exponentially as $\bar{\sigma}$ decreases. This is particularly happen with high-dimensional datasets which are usually dense; i.e. contain strongly correlated items and long frequent patterns. Therefore, mining closed frequent itemsets (CFI) was proposed in the effort to overcome the problem by removing itemsets that are not needed, while able to generate the complete set of association rules [2]. An itemset α is a CFI in a dataset D if α is frequent in D and there exists no proper super-set β such that β has the same support as α in D. Note that closed itemsets are also the maximal set of items common to a rowset.

The number of closed itemsets mined in a dense dataset is order of magnitudes smaller than the number of all frequent itemsets. More importantly, association rules extracted from closed itemsets are proved to be more meaningful for analysts as many redundancies have been discarded [3]. However, many existing closed itemset mining algorithms suffer from huge memory consumption while mining at lower support threshold. Although the number of mined closed itemsets is significantly reduced, it is still the main challenge to be addressed in any CFI algorithm. This paper introduces a new algorithm called BFF; i.e. Breadth-First Fusion, and compared it to other state-of-the-art algorithms. The algorithm exploits bottom-up row-enumeration strategy and traverses the search tree in breadth-first manner. The closure checking method and pruning strategy applied have resulted to an efficient and scalable algorithm.

Received June 29, 2017; Revised September 5, 2017; Accepted September 20, 2017

2. Frequent Itemset Mining Methodologies

The algorithms are characterized based on the ways they traverse the search space, i.e.; breadth-first or depth-first, and differ in the way internal databases are represented, i.e.; horizontal representation or vertical representation

2.1. Search Strategies

1. Breadth-first search (BFS)

Breadth-first search is used in the Apriori [1] algorithm, where it is implemented with a data structure representing a prefix tree that is built level by level. Employing breadth-first search means that, from the first level, each node will be created and scanned; starting from the leftmost node towards the right. The candidate itemsets supports are counted by scanning the database each time they were generated. Therefore, in order to identify the entire frequent itemsets, all possible candidate itemsets must be tested. However, generating and checking for all possible candidate itemsets are rather expensive, as their number can be very huge. Hence, to reduce the search space, all supersets of an infrequent itemset do not have to be considered. This is based on the *anti-monotonic* or *downward closure* property which defines that an itemset is frequent if and only if all of its subsets are frequent. Hence, infrequent itemsets are the ones that do not meet $\overline{\sigma}$. This property has become the most important pruning strategy in all frequent itemset mining algorithm.

Property 1 (*Anti-monotonic*) Given a dataset, *D*, with items *I*, let α_1 and α_2 be two itemsets such that $\alpha_1, \alpha_2 \subseteq I$, then:

$$\alpha_1 \subset \alpha_2 \Rightarrow |D_{\alpha_1}| \geq |D_{\alpha_2}| \quad (1)$$

However, the drawbacks of Apriori algorithm are the huge number of generated candidate itemsets and the need to repeatedly scanning the database to count the support of the candidates, which reduced its performance in terms of both speed and memory consumption. To overcome this, an algorithm named CLOSE [2] was developed based on Apriori framework. It generates frequent itemsets from a database in two phases, i) discovers all CFI, then ii) generates all subsets and derives their support from the CFI. Discovering CFI has reduced the number of generated candidate itemsets and scanning of database is only needed in the first phase.

2. Depth-first Search (DFS)

FP-growth [4] and ECLAT [5] employ depth-first search, which can be seen as a divide-and-conquer scheme. In FP-growth, the database is scanned to derive a list of frequent items in frequency descending order, and a frequent-pattern tree, or FP-tree, which holds the itemset association information, is built. The FP-tree pattern growth starts from frequent length-1 pattern, constructing its conditional pattern based, and then constructing its conditional FP-tree. Mining is performed recursively with the tree using depth-first search, starting at the root and following one of the branches of the tree up until the node with no children is found, and continue the search at the previous node with unexplored children. Unlike breadth-first search, it is not necessary to store all child pointers at every level, hence requires much lower memory. The method used in FP-growth has been the most approached method by researchers as it compressed large databases into a compact FP-tree which eliminates candidate generation and avoids repeated scanning of the database. Several CFI mining algorithms have been developed as extensions to the FP-growth such as CLOSET+ [6], FP-Close [7] and AFOPT [8].

2.2. Internal Representation of Dataset

1. Horizontal Data Format

Both Apriori and FP-growth methods mine frequent patterns from a set of transactions in horizontal data format. In a horizontal representation, a transaction database is stored as a list of transactions which lists the items contained in each of the transactions. This is the most obvious form of storing a transaction database.

2. Vertical Data Format

A vertical data format is an inverted version of the original dataset and is first proposed in ECLAT [5]. The database is scanned and for each item, the transaction id that contains the items is listed. The frequent itemset is generated according to the Apriori property, starting with

a single item, and employs depth-first search similar to FP-growth. The advantage of using vertical representation is that there is no need to scan the database to find the support of the (k+1)-itemset because each k-itemset has complete information to count the support. CHARM [9] is an example of algorithm that discovers CFI using vertical data format and shown to have performed better than CLOSE and CLOSET. Another example are MAFIA [10] and BitTable [11] which stores the transactional database as a series of vertical bitmaps, initially corresponds to a 1-itemset, or a single item. The bitmap structure is ideal for both candidate itemset generation and support counting. The authors of DCI_Closed [12] have also used bitmap and proposed a technique that can avoid duplicates which eliminates the need to store previously mined CFI in memory for duplication check. In addition, the concept of order preserving generators; i.e. specific generators whose closures can be computed by enumerating the items (column) according to lexicographic order were introduced.

2.3. Row-Enumeration vs Column-Enumeration

High-dimensional datasets contains huge number of columns compared to rows, making searching for CFI by column values inefficient. Therefore, row-enumeration strategy is more suitable as the search space size is smaller. CARPENTER [13] was the first method to explore the row-enumeration search space. CARPENTER does recursive generation of conditional transposed tables, performing a depth-first traversal search of the row-enumeration tree. It was shown to outperform CHARM and CLOSET; i.e. column-enumeration algorithms, in mining high-dimensional dataset. Most of well-known row-enumeration algorithms have their basis on CARPENTER ideas. FARMER [14] is an algorithm specifically developed for mining interesting rule groups from microarray datasets, i.e. a grid of DNA segment and an example of high dimensional dataset. This algorithm is enhanced with efficient search pruning strategies based on user-specified thresholds (minimum support, minimum confidence and minimum chi-square value). While TOPKRGS [15] is quite similar to FARMER in terms of row-enumeration strategy of rule groups, it differs in adopting a preference selection (top-k) to filter out significant rules, and its implementation using compact prefix-tree is more efficient. COBBLER [16] is able to switch between row and column enumeration while mining by estimating the enumeration costs for the subtrees and selecting the smallest one from both subtrees. This dynamic strategy helps COBBLER to deal with different kind of dataset including large, dense datasets that have varying characteristics on different data subsets. Based on FP-tree in FP-Growth algorithm, IsTa [17] used a prefix tree to store all closed itemsets which is updated by intersecting it with the next transactions and performs better than CARPENTER at lower support threshold.

2.4. Memory Usage

All previously mentioned algorithms require huge memory for three reasons; either to store internal representations of the dataset, or to store the currently mined CFIs for duplicates checking, or both. Several CFI mining algorithms [6– 9] have been developed as extensions to the FP-growth which uses tree-based structure to compactly represent the dataset. While others [13–17] have to recursively construct conditional database and store CFIs in a hash based table or prefix tree for duplicates checking. For MAFIA [10] and BitTable [11], the original bitmap will be pruned every time an item is found to be infrequent and the pruned bitmap is projected each time. While for ECLAT [5] and CHARM [9], currently mined CFIs need to be maintained in a hash table to avoid redundant output. Apart from all these algorithms, only DCI_Closed [12] does not need to store the CFIs in main memory and the use of bitmap to represent the dataset only once requires minimal storage. The reason for this is the efficient closure checking method proposed in DCI_Closed which eliminates the need to compare the newly discovered CFI with the already found CFIs to avoid duplicates in the output. However, DCI_Closed traverses the itemset lattice to mine the closed itemset, which is inefficient and requires longer time as the number of items in high-dimensional data is large compare to number of transactions.

3. The BFF Algorithm

Based on our studies on previous algorithms, to minimize memory usage, it is important for an algorithm to avoid the need to store all CFIs in main memory. Apart from that, to reduce database scan which adds to longer running time, an internal representation of the dataset

should be used. In order to achieve this, an efficient closure checking method and bit matrix representation of the dataset will be employed. As the aim of this work is to mine CFI in high-dimensional dataset, the search strategy is based on row-enumeration and to reduce the itemset intersections for each node while avoiding the need to store all CFIs, the search space will be traversed in breadth-first manner. The BFF algorithm is given in Algorithm 1.

Algorithm 1: BFF Algorithm

Input: Dataset T, $\bar{\sigma}$

Output: A complete set of closed itemsets

Method:

a) Creating bit matrix of the dataset T.
b) Sort transaction ids in ascending order of itemset cardinality.
c) FOR every sorted transaction id, α, and its bitset, β
Current rowset = α

Current bitset = β

Do, WHILE stack is not empty

Initialize flag=true

1. IF stacks are not empty
POP the rowset, α, and its bitset, β
Set Current rowset = α
and Current bitset = β

(Closure checking)

2. IF flag=true
FOR each α_j that comes before the first row and last row id in the rowset, intersect the bitsets, β_j
IF $\beta_j \cap \beta = \beta$, set flag=false and BREAK

3.1. The Proposed Search Strategies

The designed algorithm should be able to mine closed itemsets at lower support threshold. This is due to many interesting patterns exist at the lower end of the threshold and the number of CFI is enormous. Therefore, the proposed strategies should not involve a large storage of these CFIs in main memory, but less memory consumption with reasonable running time.

1. Bitmap Representation of the Database

The algorithm represents the dataset in the memory using a horizontal bit matrix. The element of the matrix is set to 1 if and only if item *i* appear in j^{th} transaction. Note that for each transaction, the count of 1s is exactly the cardinality of the transaction. The advantage of using bitmap representation of a dense database is lesser memory consumption, as each cell can be represented by 1 bit. To get the resulting bitset of a rowset, AND operation, or intersection of the itemsets in the rowset has to be carried out. Consider an example of a transaction dataset in Table 1, an example of intersction operation is given below;

Example 1 For rowset { 2 4 6 }, the resulting bitset is shown is 0000110, which has the support of 3; i.e. | 2 4 6 | = 3.

Table 1. Transaction Dataset.

Transaction id	Items
1	a b c e g h
2	a c d e f h
3	b e f g h
4	a b c d e f g
5	a b d f g
6	b e f h

2. Closure Checking Method

In this study, computing closure is a very important task. The algorithm adopted similar concept of closure checking method proposed by DCI_Closed [12]. By employing the right method, duplicates can be avoided and the need to compare with the ones already mined so far becomes unnecessary. With these two goals to achieve, an inclusion check for all the rows that is not included in the rowset can be performed. This inclusion check works as duplicates checking as well as closure checking as shown in the following example.

Example 2 Consider the dataset in Table 1, for rowset *X* = { 6 3 }, to compute its closure, intersections with all other rows that is not included in the rowset should be done, i.e. $f(g(X \cap 5, fgX \cap 1, fgX \cap 2, fgX \cap 4.X$ is closed iff $fgX \neq fgX \cap Xj.$

3. Pruning the Search Space

Pruning strategy is crucial to attain the best performance. This will determine the size of the search space; the smaller the better. To achieve this, the transaction ids are sorted in ascending order of their corresponding itemset's cardinality. After applying Method (b) in Algorithm 1, the dataset in Table 1 is sorted and the sorted transaction ids are 6, 3, 5, 1, 2, and 4. By sorting the transactions this way, the opportunity of pruning itemsets from the same class is maximized. In addition, most intersections involving longer patterns can be reduced since logically, shorter patterns are more likely to exist in the longer ones. In method (c) no. 2, by checking closure starting with the first sorted row id, the advantage of sorting the transactions can be further seen when the rowset can be quickly pruned if its closure contains a row id that that has been evaluated in the higher subtree. This is shown in the following example;

Example 3 Consider rowset { 3 5 2 4 }, the node is pruned as the itemset has occurred at rowset { 6 3 5 2 4 } and no further intersection is required.

4. Reusing the previous itemset intersections

Each single node of the bottom-up row-enumeration tree is a possible CFI and need to be checked for closure. Even with the proposed pruning strategy devised earlier, the number of itemset intersections for computing closure is still huge which leads to longer running time. Therefore, a method which permits reusing of the previous itemset intersections need to be designed. This can be done by storing the rowset and itemsets of each node and reuse them while generating other nodes. There are two ways to do this; i.e. depth- of breadth-first manner. In terms of space complexity, both ways will use approximately the same space. However, by traversing the tree in breadth-first manner, the number of itemset intersections is less than traversing in depth-first manner. This is because the children nodes are generated while checking the closure of their parent node and is made possible by adopting breadth-first traversal of the tree, as shown in Example 4.

Example 4 Consider row { 3 } with itemset { b e f g h }. To check its closure, its itemset must be intersected with itemsets of all other row ids. Therefore, while checking with row ids 5, 1, 2 and 4, node { 3 5 }, { 3 1 }, { 3 2 } and { 3 4 } are created, hence reducing the number of intersections to be done.

4. Performance Evaluation

This section presents the results of the performance study of BFF Algorithm and confirms that the program design has been realized. The set of experiments were performed on a PC with Intel Xeon CPU, 8 GB RAM, 64-bit operating system. The performance was evaluated by comparing it with CARPENTER and IsTa. CARPENTER [13] is the first row-enumeration algorithm. It recursively generates conditional transposed tables and performs depth-first traversal. IsTa [16] is also a depth-first row-enumeration algorithm that builds a prefix tree of all closed itemsets. It has a comparable performance with CARPENTER, but performs better at lower support threshold. The source of implementations of both algorithms is from Christian Borgelt's website [16]. The performance of these algorithms is evaluated using real high-dimensional (microarray) datasets. BFF will be compared to IsTa and CARPENTER to see the effectiveness of a breadth-first algorithm in mining CFIs. Three real datasets namely, National Cancer Institute (NCI), Leukimia and Colon were obtained from Hanchuan Peng's website, the author of the minimum-Redundancy-Maximum-Relevance feature/variable/attribute selection (mRMR) [17]. The specifications of the datasets are shown in Table 2.

Table 2. Microarray Datasets Specification.

Dataset	NCI	Leukimia	Colon
No. of Genes (Columns)	9,703	7,070	2,000
No. of Samples (Row)	60	72	62

4.1. Running Time Comparisons

Figure 3 – Figure 5 show the running time of all algorithms as a function of the support threshold, $\bar{\sigma}$. It can be seen that for all dataset, the running time of BFF is longer than CARPENTER and IsTa. However, CARPENTER and IsTa failed to run at lower support threshold; e.g. for NCI dataset, CARPENTER failed to finish the task at $\bar{\sigma} < 8$ while IsTa failed at $\bar{\sigma} < 7$. In fact, IsTa took 130s more to mine half the number of CFIs compares to BFF. For Colon dataset, though IsTa can run at $\bar{\sigma} = 7$ while CARPENTER failed, it took almost 20 times the time taken by BFF to mine the same number of CFIs. In addition, although the run time of BFF at the upper support threshold is significantly higher, the time taken is almost consistent even at the lowest $\bar{\sigma}$. Based on the results, as the threshold decreases, the run time of IsTa significantly increased. While it is unable to see the complete run time for IsTa, it can be predicted that it is going to increase more at lower threshold. This is because as the prefix tree to store the CFIs becomes bigger, bushier and longer, more time is taken to traverse the tree to check for duplicates and to merge the new CFIs into its branches.

4.2. Space Efficiency

Another issue related to the efficiency of a closed itemset algorithm is the memory usage. Figure 6 – Figure 8 plot memory requirements and the number of CFIs mined at different threshold for all algorithms.

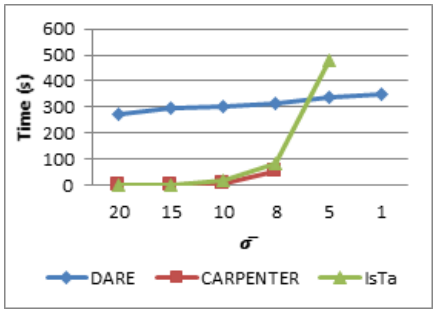

Figure 3. Run time with NCI dataset.

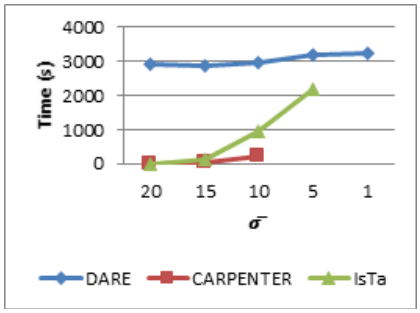

Figure 4. Run time with Leukimia dataset.

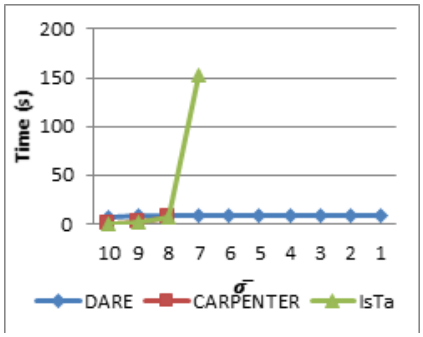

Figure 5. Run time with Colon dataset.

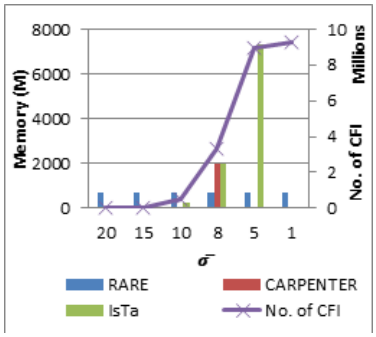

Figure 6. Memory usage with NCI dataset and the number of CFIs mined at various $\bar{\sigma}$.

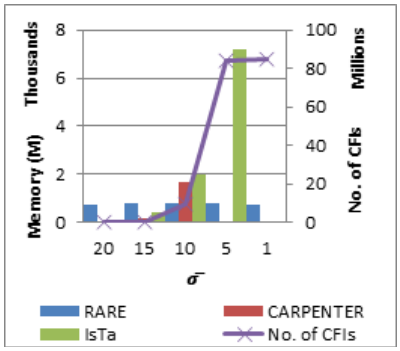

Figure 7. Memory usage with Leukimia dataset and the number of CFIs mined at various $\bar{\sigma}$.

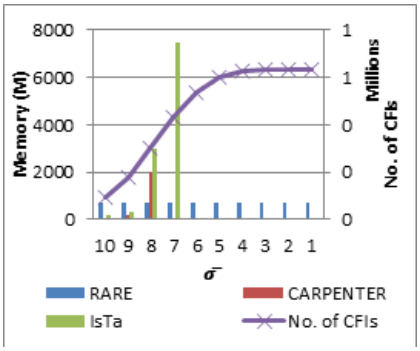

Figure 8. Memory usage with Colon dataset and the number of CFIs mined at various $\bar{\sigma}$.

For all datasets, BFF successfully mines all CFIs with adequate memory usages which are nearly constant for all thresholds as it requires minimal storage of the rowsets and their itemsets. However, note that the amount of memory used by CARPENTER and IsTa rapidly grow when the support threshold is decreased due to the huge number of CFIs generated that must be stored in the main memory. For CARPENTER, conditional transposed table is built for every node to determine the next node to traverse and to check for closure, which if found closed, will be stored in memory for duplicates checking. Therefore, the need to recursively generates the transposed table and the storage of intermediate results requires huge memory. As for IsTa, a prefix tree is built to store the items of each transaction and as the search continues to find more closed itemsets, the tree's size becomes bigger as the next rowset is merged. Therefore, for high-dimensional datasets with thousands of items, the memory requirement for the prefix tree is undoubtedly huge and will increase at lower threshold as more itemsets were added to the tree. From the point of view of space complexity, in order to check for duplicates, storing the rowsets are better than storing the itemsets as the total number of closed itemsets can be as maximum as $O(2^m)$, where m is the number of distinct items. Furthermore, in high-dimensional datasets, the number of transaction ids (row) is much smaller than the items (columns). The huge number of closed itemsets mined clearly demonstrates the limited scalability, with respect to the support threshold, of CARPENTER and IsTa, which need to store the full set of closed itemsets in the main memory.

On the other hand, the memory size required by BFF is independent of the size of the number of CFIs extracted. The amount of memory used by the BFF algorithm can be measured by taking into account all the information needed along the deepest computational path in the lattice from the root to the current node. First, the data is stored in a horizontal bit matrix with the size of $O(|m| \times |n|)$ and the stacks need $O(|n| \times |n|)$ and $O(|m| \times |n|)$ to store the rowsets and the corresponding itemsets respectively. Therefore, in addition to the bit matrix, the maximum space complexity of BFF algorithm is $O\big((2 \times |m| \times |n|) + |n^2|\big)$. Hence, from these experiments it can be concluded that BFF has the least memory requirement than CARPENTER and IsTa.

5. Conclusion

In this work we have investigated the problem in mining closed frequent itemsets from high-dimensional data. A number of algorithms have been reviewed and the limitations were acknowledged. Differing from these algorithms, the research has shown that it is possible to efficiently mine high-dimensional data which is high in density, without the need to store all sets of currently mined itemsets. By traversing the search tree in breadth-first manner, the number of itemset intersections can be reduced by checking for duplicates while generating the next nodes at the same time. In addition, reusing the itemsets which are stored in a stack further reduces the running time. In terms of storage, the designed algorithm has small memory usage, thus allowing mining large datasets. BFF has completed successfully in all experiments as reported in the previous section. Both execution times and memory requirements are almost constant with increasing size of the dataset. This means that BFF algorithm can mine any dataset as long

as its horizontal bit matrix representation fits in the main memory, disregard of the number of CFIs extracted. As the result of the efficient strategies and optimizations introduced, BFF outperforms other state-of-the-art algorithms; i.e. CARPENTER and IsTa, and requires orders of magnitude less memory while mining at low support thresholds.

References

[1] Agrawal R, Imielinski T, Swami A. *Mining association rules between sets of items in largedatabases*. In: Proceedings of the 1993 ACM-SIGMOD international conference on management DFGHJL'of data (SIGMOD'93), Washington, DC, 1993: 207–216.

[2] Pasquier N, Bastide Y, Taouil R, Lakhal L. *Discovering frequent closed itemsets for association rules*. In: Proceeding of the 7th international conference on database theory (ICDT'99), Jerusalem, Israel, 1999: 398–416.

[3] Y Bastide, N Pasquier, R Taouil, G Stumme, L Lakhal. *Mining minimal non-redundant association rules using frequent closed itemsets*. In CL '00: Proceedings of the First International Conference on Computational Logic, 2000: 972–986.

[4] Han, J, Pei, J, Yin, Y. *Mining frequent patterns without candidate generation*. Proceedings of the 2000 ACM SIGMOD international conference on Management of data. 2000: 1–12.

[5] Zaki, MJ. Scalable algorithms for association mining. *IEEE Transactions on Knowledge and Data Engineering*. 2000; 12(3): 372-390.

[6] Wang J, Han J, Pei J. *CLOSET+: searching for the best strategies for mining frequent closed itemsets*. In: Proceeding of the 2003 ACM SIGKDD international conference on knowledge discovery and data mining (KDD'03), Washington, DC, 2003: 236–245.

[7] Grahne G, Zhu J. *Efficiently using prefix-trees in mining frequent itemsets*. In: Proceeding of the ICDM'03 international workshop on frequent itemset mining implementations (FIMI'03), Melbourne, FL, 2003: 123–132.

[8] Zaki MJ, Hsiao CJ. *CHARM: an efficient algorithm for closed itemset mining*. In: Proceeding of the 2002 SIAM international conference on data mining (SDM'02). Arlington, VA. 2002: 457–473.

[9] Liu G, Lu H, Lou W, Yu JX. *On Computing, Storing and Querying Frequent Patterns*. In: Proceeding of the 2003 ACM SIGKDD international conference on knowledge discovery and data mining (KDD'03), Washington, DC, 2003: 607–612.

[10] Burdick D, Calimlim M, Gehrke J. *MAFIA: a maximal frequent itemset algorithm for transactional databases*. In: Proceeding of the 2001 international conference on data engineering (ICDE'01), Heidelberg, Germany, 2001: 443–452.

[11] András Király, Attila Gyenesei, and János Abonyi. Bit-Table Based Biclustering and Frequent Closed Itemset Mining in High-Dimensional Binary Data. *The Scientific World Journal*. 2014, Article ID 870406, 7 pages, 2014. doi:10.1155/2014/870406

[12] C Lucchese, S Orlando, R. Perego. Fast and memory efficient mining of frequent closed itemsets. in *IEEE Transactions on Knowledge and Data Engineering*, 2006; 18(1): 21-36.

[13] Pan F, Cong G, Tung AKH, Yang J, Zaki M. *CARPENTER: finding closed patterns in long biological datasets*. In: Proceeding of the 2003 ACMSIGKDD international conference on knowledge discovery and data mining (KDD'03), Washington, DC. 2003: 637–642.

[14] Cong G, Tung AK, Xu X, et al. *FARMER: finding interesting rule groups in microarray datasets*. In: Proceedings of the ACM SIGMOD International Conference on Management of Data. Paris, France: ACM Press, 2004; 143–54.

[15] Cong G, Tan K, Tung AK, et al. *Mining top-K covering rule groups for gene expression data*. In: Proceedings of the ACM SIGMOD International Conference on Management of Data. Baltimore, Maryland, USA: ACM Press, 2005; 670–81.

[16] F Pan, A Tung, G Cong, X Xu. *Cobbler: Combining column and row enumeration for closed pattern discovery*. in Scientific and Statistical Database Management, 2004. Proceedings. 16th International Conference on. 2004: 21–30.

[17] Hanchuan Peng, Fuhui Long, Chris Ding. Feature selection based on mutual information: criteria of max-dependency, max-relevance, and min-redundancy. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 2005; 27(8): 1226-1238.