A Comparative Study of Association Rules Algorithms on Large Databases

Ahmed Alhamzi, **Mona Nasr, Shaimaa Salama**
Faculty of Computers and Information, Helwan University, Cairo, Egypt
ahmed_alhamzi@yahoo.com, m.nasr@helwan.edu.eg, chaimaa_salama@yahoo.com

Abstract

The task of mining association rules consists of two main steps. The first involves finding the set of all frequent itemsets. The second step involves testing and generating all high confidence rules among itemsets. This paper presents a comparative study of association rules algorithms on large databases. Five algorithms have been chosen for this comparative study. The Apriori, Close, FP-growth, Top-k rules, and TNR algorithms have been chosen because these are the most commonly used in the literature. Moreover, these algorithms differ in the number of dataset scanning which affects the performance. In addition, some of these algorithms generate redundant association rules while others don't. All these algorithms are implemented and compared on different datasets. Experimental results show that the FP-Growth algorithm has the best performance, while the TNR algorithm has the best generated non-redundant association rules, and the Top-k rules algorithm has the best performance when the minimum confidence is high.

Keywords: *Data mining; Association rules; Apriori algorithm; FP-Growth algorithm; Close algorithm; Top-k rules algorithm; TNR algorithms.*

1. Introduction

Since its introduction, association rule mining [1], has become one of the core data mining tasks and has attracted tremendous interest among data mining researchers and practitioners. It has an elegantly simple problem statement; that is, to find the set of all subsets of items (called itemsets) that frequently occur in many database records or transactions, and to extract the rules telling us how a subset of items influences the presence of another subset. There are many algorithms to generate association rules such as AIS [1], Apriori [2], Close [3], FP-Growth [4], Top-k rules [5], TNR [6], etc. The first algorithm for mining association rules was the AIS algorithm. Shortly after that, the algorithm was improved and renamed Apriori. The Apriori algorithm is the most classical and important algorithm for mining, but it requires multiple passes over the database. The Apriori algorithm was improved for reducing passes, shrinking candidate items, and facilitating support counting of candidates. Improving the performances of Apriori algorithm was made by the introduction of a novel, compact data structure, referred to as frequent pattern tree, or FP-tree, and the associated mining algorithm, FP-growth. Another two algorithms for the efficient generating of large frequent candidate sets are Matrix and BCAR algorithms. The Matrix algorithm was proposed by Yubo and Tingzhu [7]. The Matrix algorithm generates a matrix with entries 1 or 0 by passing over the database only once, and then the frequent candidate sets are obtained from the resulting matrix. Finally, association rules are mined from the frequent candidate sets. The Boolean algebra and compression technique for association rules (BCAR) [8] algorithm was proposed by S. Anekritmongkol et al. which it compress database and reduce the data of each candidate.

A serious problem in association rule discovery is that the set of association rules can grow to be unwieldy as the number of transactions increases, especially if the support and confidence thresholds are small. As the number of frequent itemsets increases, the number of rules presented to the user typically increases proportionately. Many of these rules may be redundant. To overcome the "too many frequent itemsets" disadvantage, the closed itemset concept was proposed [9]. The set of closed itemsets is said to be closed if it has no superset with the same frequency (support). Several algorithms have been proposed in the literature, including A-Close, FP-Close [10], B-Miner & C-Miner [11], etc.

Ashrafi et al. [12] presented several methods to eliminate redundant rules and to produce a small number of rules from any given frequent or frequent closed itemsets generated. Ashrafi et al. [13] present additional redundant rule elimination methods that first identify the rules that have similar meaning and then eliminate those rules. Furthermore, their methods eliminate redundant rules in such a way that they never drop any higher confidence or interesting rules from the resultant rule set.

The rest of the paper is organized as follows: Section 2 describes the association mining task and presents the main aspects of Apriori, FP-growth, Close, Top-k rules and TNR algorithms; Section 3 shows measuring efficiency of the algorithms; and the paper is concluded in Section 4.

2. Association rules

Association rule mining is to find association rules that satisfy the predefined minimum support and confidence from a given database. The problem is usually decomposed into two sub problems. One is to find those itemsets whose occurrences exceed a predefined threshold in the database called support; those itemsets are called frequent or large itemsets. The second problem is to generate association rules from those large itemsets with the constraints of minimal confidence. Support and confidence are important measures for association rules. Association rule mining is one of the most important technique of the data mining. Its aim is to extract interesting correlations, frequent patterns and association among set of items in the transaction database. [14]

The problem of discovering association rules was first introduced, and then an algorithm called AIS (Artificial Immune System) was proposed for mining association rules. For the last few years, many algorithms for rule mining have been proposed. Most of them follow the representative approach of the Apriori algorithm. Various researches have been done to improve the performance and scalability of Apriori.

2.1 Apriori Algorithm

In the Apriori algorithm, the items are sorted in lexicographic order. Frequent itemsets are computed iteratively, in ascending order of their size. The process takes k iterations, where k is the size of the largest frequent itemsets. For each iteration $i \leq$ k, the database is scanned once and all frequent itemsets of size i are computed. The first iteration computes the set L_1 of frequent 1-itemsets. A subsequent iteration i consists of two phases. First, a set C_i of candidate i-itemsets is created by joining the frequent $(i-1)$-itemsets in L_{i-1} found in the previous iteration. This phase is realized by the Apriori-gen function. Then the database is

scanned for determining the support of candidates in C_i and frequent i-itemsets are extracted from the candidates. This process is repeated until no more candidates can be generated. [2]

Definition 1: (Association rule on frequent itemsets)

Rule $X \Rightarrow Y$ is an association rule on frequent itemsets if both X and XUY are frequent itemsets. The complete set of association rules on frequent itemsets can be generated by an adapted version of Apriori rule generation algorithm [3].

2.2 FP-Growth Algorithm [15]

The FP-Growth algorithm, proposed by Jiawei Han, et al., is an efficient and scalable method for mining the complete set of frequent patterns by pattern fragment growth, using an extended prefix-tree structure for storing compressed and crucial information about frequent patterns named frequent-pattern tree (FP-tree). The FP-Growth is an alternative way to find frequent itemsets without using candidate generation, thus improving performance. In their study, they proved that their method outperforms other popular methods for mining frequent patterns; e.g. the Apriori, Tree Projection [16], PRICES [17] algorithms. In another paper [15], some optimizations are proposed to speed up FP-growth so that a single path FP-tree has been further developed for performance improvements.

In the following we present basic steps for FP-Growth algorithm:

- The algorithm avoids repeated database scans so it can scan the database in only two scans.
- Scan the database to find all frequent itemsets and order them in descending frequent order.
- Scan the database again to construct FP-tree according to FP-Tree construction. The FP-Tree is adequate for mining frequent patterns and can replace the database.
- FP-Growth starts scanning the tree to compute the support of k-itemset and to examine items under form the conditional pattern base.
- Generate association rules by adapted version of Apriori rule generation algorithm.

2.3 Close Algorithm [3]

In 1993, Agrawal et al. developed a method for mining traditional association rules (TAR) [2]. Following that, the Apriori algorithm was proposed. Because TAR contains a number of redundancies, Nicolas Pasquier, et al. proposed the Close algorithm to mine frequent closed itemsets and added an adapted version of Apriori rules algorithm to generate association rules from frequent closed itemsets. Additionally, the number of frequent closed itemsets is often much smaller than the number of frequent itemsets, so the time for generating rules from frequent closed itemsets is reduced significantly.

Nicolas Pasquier, et al. described the concept of frequent closed itemsets and added this concept to the Apriori algorithm in order to find frequent closed itemsets. They also generated valid association rules by an adapted version of Apriori rule generation algorithm.

Definition 2: An itemset *X* is closed in data set *S* if there exists no proper super-itemset *Y* such that *Y* has the same support count as *X* in *S*.

Definition 3: A closed itemset *X* is a frequent closed itemset if and only if its support is no less than the given minimum support threshold.

The advantages of using closed frequent itemset mining are (a) the number of frequent closed itemsets is less when compared to number of frequent itemsets; and (b) all information of frequent itemsets can be derived from closed frequent itemsets. The advantage of Close algorithm reduces frequent itemsets and association rules so that it can find frequent closed itemsets and generate association rules from frequent closed itemsets.

2.4 Top-k rules algorithm [5]

Top-k Rules algorithm was proposed by Philippe Fournier-Viger, et al. It is used to mine the Top-k association rules, where k is the number of association rules to be found and is set by the user. Mining the Top-k association rules has two challenges. First, Top-k rules cannot rely on minimum support to prune the search space, but they could be modified to mine frequent itemsets with minimum support = 1 to ensure that all top-k rules can be generated in the generating rules step. Second, top-k association rules mining algorithm cannot use the two steps process to mine association rules [1], but they would have to be modified to be able to find the top-k rules by generating rules and keeping top-k rules. They defined the efficient approach for generating association rules as "rules expansions" that does not rely on the two steps process to mine association rules [1]. The Top-k rules algorithm is an advantageous alternative to classical association rule mining algorithms for users who want to control the number of association rules generated. Generally, the Top-k rules algorithm contains the following steps:

- Scan database once to calculate tids({c}) for each single item c in the database.
- The algorithm generates all rules by joining pair of items i, j, where i and j each have at least minimum support. The support of itemset (i, j) are calculated by intersect tids(i) and tids(j). If support of pairs items is not satisfy minimum support, the rule cannot be created with i, j. The rule is valid if the rule satisfies the minimum support and minimum confidence. The support of rules $\{i\} \rightarrow \{j\}$ and $\{j\} \rightarrow \{i\}$ are calculate by dividing the support of itemset (i,j) by number of transactions. The confidence of $\{i\} \rightarrow \{j\}$ are calculated by dividing support of itemset (i,j) by support of itemset(i).
- All valid rules are added in the list L. Also all the rules are added to the set R where each support of rules have at least minimum support.
- When a new valid rule is found, the rule adds to L. Then, if the list L contains more than k rules and support of rule is greater than minimum support, the rules that are equal minimum support can be removed until only k rules are kept. Finally, the minimum support is changed to the lowest of support the rule in L.
- The set R start loop to expand all rules by selected rule with the highest support until there is no more rules in R. For each rule, flag expandLR indicates if the rule should be left and right expanded or just left expand.

2.5 TNR Algorithm [6]

Top-k non-redundant association rules (TNR) algorithm is combined the idea of mining a set of non-redundant rules with the idea of Top-k association rules. The TNR algorithm is based on "rule expansion" and two strategies to avoid generating redundant rules. It is similar to the Top-k rules algorithm, but it added two strategies when save rules on L. After the search procedure is modified, the every generated rule r_a is added to L only if sup $(r_a) \geq$

minsupp and r_a is not redundant with respect to another rule. There are two strategies to determine r_a redundant as follows:

Strategy 1:

For each rule r_a that generated such that sup (r_a) ≥ minsupp, if $\exists r_b \in L|$ sup(r_b)=sup(r_a) ∧conf(r_b)=conf(r_a) ∧conf(r_b) ∧left(r_b) ⊆ left(r_a) ∧right(r_a)⊆ right(r_b) and r_a is redundant with respect r_b, then $r_a$is not added to L. r_a otherwise, is added to L.

Strategy 2:

For each rule r_b that generated such that sup (r_b) ≥ minsupp, if $\exists$ $r_a \in L|$ sup(r_b)=sup(r_a) ∧conf(r_b)=conf(r_a) ∧conf(r_b) ∧left(r_a) ⊆ left(r_b) ∧right(r_b)⊆ right(r_a) and r_a is redundant with respect r_b, then $r_a$is removed from L. if remove the rule r_a , it may have forced variable. If that happened, then the algorithm may have missed some rules that have a support lower than r_a but are non-redundant. To solve this problem they added a parameter that they named Δ that increases by the number of rules k necessary to raise the internal minimum support variable.

3. Measuring Efficiency of the Algorithms

In this section, we compare previously described algorithms in the previous section and discus observed differences in performance behavior.

3.1 Experiment Results

The Apriori, FP-Growth, Close, Top-k rules and TNR algorithms were implemented in Java and tested on several datasets [18]. Table (1) summarizes the datasets characteristics. The platform's specifications used for this test were: Intel core i3 (3*2.40 GHz), 4GBs RAM memory, Windows 8. In order to obtain more realistic results, a Microsoft SQL 2008 Server was used. To study the performance, four datasets has been used to support factors between 40% and 90%, and also confidence factor between 40% and 90%. Taking into account these considerations, the datasets depend on the number of items in a transaction, number of items in a frequent itemset, etc.

Table (1) Datasets' Characteristics

Datasets	Number of Transactions	Number of distinct Items
Mushroom	8124	119
Pumsb	49046	2097
T10I4D100K	100000	870
Retail	88162	16470

Some of the results of the comparison between the Apriori, FP-growth, Close, Top-k rules, and TNR algorithms for minimum support and minimum confidence factors between 40% and 90% and for different datasets are presented in Tables 2-13.

Table (2) Number Rules of Five Algorithms on Mushroom

MinSup &MinConf	Frequent itemsets by Apriori and FP-Growth algorithms	Frequent closed itemsets by Close algorithm	Number of Rules by Close algorithm	Number of Rules by Apriori, FP-Growth and Top-k rules algorithms	Number of Rules by TNR algorithm
40%	565	140	2654	7020	6546
50%	153	45	516	1148	1113
60%	51	19	136	266	253
70%	31	12	90	180	170
80%	23	10	54	88	82
90%	9	5	12	14	14

Table (3) Execute Time of Five Algorithms on Mushroom.

MinSup &MinConf	Execute time of Apriori (ms)	Execute time of FP-Growth (ms)	Execute time of Close (ms)	Execute time of Top-k rules (ms)	Execute time of TNR (ms)
40%	1442189	194537	1437958	1876629	14322037
50%	1315282	193373	1058754	477997	2332565
60%	1265774	192328	1054246	224818	676054
70%	1096240	192646	1055215	194669	575561
80%	863433	191910	867130	146355	352596
90%	755820	192119	677265	104866	127540

Table (4) Maximum Memory of Five Algorithms on Mushroom

MinSup &MinConf	Maximum memory usage of Apriori (mb)	Maximum memory usage of FP-Growth (mb)	Maximum memory usage of Close (mb)	Maximum memory usage of Top-k rules (mb)	Maximum memory usage of TNR (mb)
40%	17.08866	14.21042	17	84.74023	440.4
50%	18.6	15.24411	16.88	27.28	56.2315
60%	13.2	11.61	16.2	13.8	18.8
70%	16.4	18.0332	13.6	12.644	16.9
80%	10.6	12.601	10.8	11.15	13.4
90%	16.5	16.9	16.6	9.75	9.6

Table (5) Number Rules of the Five Algorithms on Retail

MinSup &MinConf	Frequent itemsets by Apriori and FP-Growth algorithms	Frequent closed itemsets by Close algorithm	Number of Rules by Close algorithm	Number of Rules by Apriori, FP-Growth and Top-k rules algorithms	Number of Rules by TNR algorithm
40%	95	16	316	1024	1003
50%	71	13	210	640	601
60%	35	7	70	190	185
70%	31	6	64	180	174
80%	11	3	14	22	22
90%	7	2	8	12	12

Table (6) Execute Time of Five Algorithms on Retail.

MinSup &MinConf	Execute time of Apriori (ms)	Execute time of FP-Growth (ms)	Execute time of Close (ms)	Execute time of Top-k rules (ms)	Execute time of TNR (ms)
40%	93140175	14312046	93087818	64094737	75028720
50%	93125614	14301127	93003989	64011955	75023807
60%	78959334	14253814	78899075	46929071	57717263
70%	78896862	14224032	78808594	46499056	57609598
80%	50149892	14219850	50140615	38517208	49518278
90%	50145794	14208197	50140068	33519667	43529381

Table (7) Maximum Memory of Five Algorithms on Retail.

MinSup &MinConf	Maximum memory usage of Apriori (mb)	Maximum memory usage of FP-Growth (mb)	Maximum memory usage of Close (mb)	Maximum memory usage of Top-k rules (mb)	Maximum memory usage of TNR (mb)
40%	14.16	16.66	13.37	303.9	414.23
50%	13.95	16.95	13.21	238.7108	381.514
60%	15.11	16.83	14.83	213.8	257.5365
70%	13.11	15.41	12.02	212.2	216.9
80%	17.7	16.7	16.7	188.32	191.6
90%	16.6	14.3	12.1	177.302	179.803

Table (8) Number Rules of Five Algorithms on T10I4D100K

MInSup &MinConf	Frequent itemsets by Apriori and FP-Growth algorithms	Frequent closed itemsets by Close algorithm	Number of Rules by Close algorithm	Number of Rules by Apriori, FP-Growth and Top-k rules algorithms	Number of Rules by TNR algorithm
40%	127	49	784	1932	1902
50%	127	49	784	1932	1902
60%	63	28	318	602	585
70%	63	28	318	602	602
80%	31	16	130	180	180
90%	9	5	8	8	8

Table (9) Execute Time of Five Algorithms on T10I4D100K.

MinSup &MinConf	Execute time of Apriori (ms)	Execute time of FP-Growth (ms)	Execute time of Close (ms)	Execute time of Top-k rules (ms)	Execute time of TNR (ms)
40%	80724201	10682009	80704601	40795514	45985017
50%	80743111	10681000	80713111	40779798	45899703
60%	69845174	10718988	69841174	32299910	39512793
70%	69944142	10712970	69743172	32374701	39499887
80%	59136272	10645311	59126373	26875612	30987690
90%	26749101	10642572	26658092	9423634	13569837

Table (10) Maximum Memory of Five Algorithms on T10I4D100K.

MinSup &MinConf	Maximum memory usage of Apriori (mb)	Maximum memory usage of FP-Growth (mb)	Maximum memory usage of Close (mb)	Maximum memory usage of Top-k rules (mb)	Maximum memory usage of TNR (mb)
40%	15.75	16.95	15.86	402.5	489.8
50%	16.2	17.1	16.1	402.7	489.9
60%	14.03312	15.83312	14.211	143.8301	147.9
70%	14.92	15.92	14.73	144.534	156.35
80%	11.813	13.0111	12.1	112.92	115.92
90%	16.451	13.6233	15.501	47.835	50.01

Table (11) Number Rules of Five Algorithms on Pumsb.

MinSup &MinConf	Frequent itemsets by Apriori and FP-Growth algorithms	Frequent closed itemsets by Close algorithm	Number of Rules by Close algorithm	Number of Rules by Apriori, FP-Growth and Top-k rules algorithms	Number of Rules by TNR algorithm
70%	356	79	1222	4546	4430
90%	7	3	4	4	4

Table (12) Execute Time of Five Algorithms on Pumsb.

MinSup &MinConf	Execute time of Apriori (ms)	Execute time of FP-Growth (ms)	Execute time of Close (ms)	Execute time of Top-k rules (ms)	Execute time of TNR (ms)
70%	70156171	9474078	68967495	91332058	112697223
90%	47195453	9184603	46987351	6092669	9467830

Table (13) Maximum Memory of Five Algorithms on Pumsb.

MinSup &MinConf	Maximum memory usage of Apriori (mb)	Maximum memory usage of FP-Growth (mb)	Maximum memory usage of Close (mb)	Maximum memory usage of Top-k rules (mb)	Maximum memory usage of TNR (mb)
70%	14.027	18.205	14.05	903.84	989.87
90%	9.627	16.94	14	20.03223	189.1

3.2 Discussion

Tables 2, 5, 8 and 11 show the number of rules generated by algorithms, while tables (3, 6, 9 and 12) and Figure (1, 3, 5 and 7) show the execution times for the five algorithms. The best performance of algorithms is FP-growth algorithm when the datasets contain small or large frequent itemsets. Also Top-k rules and TNR algorithms are better than Apriori and Close algorithms when the datasets have small frequent itemsets as in Retail and T10I4D100K. For high minimum support, minimum confidence of 90%, and datasets of Mushroom and Pumsb, the Top-k rules algorithm has the best performance.

Tables (4, 7, 10 and 13) and Figures (2, 4, 6 and 8) show the maximum memory usage of five algorithms on four datasets. The Top-k rules and TNR algorithms have the highest memory usage, but Apriroi and Close algorithms have the lowest memory usage.

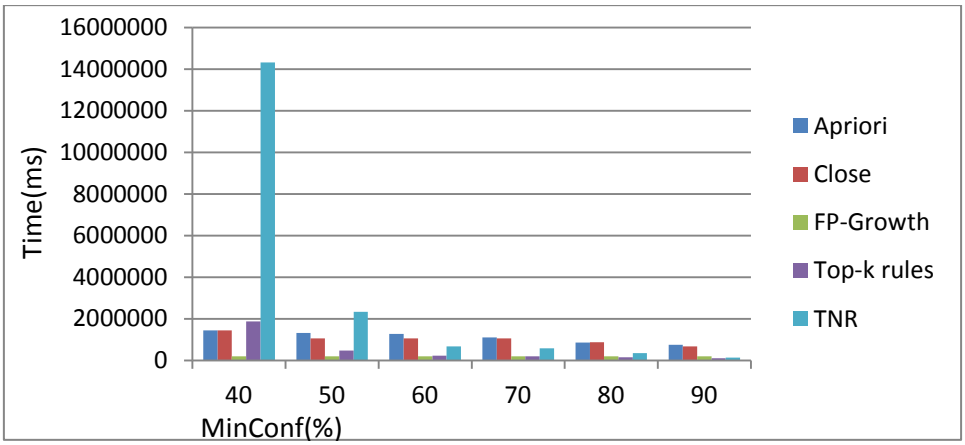

Figure (1) Execution Time of Five Algorithms on Mushroom

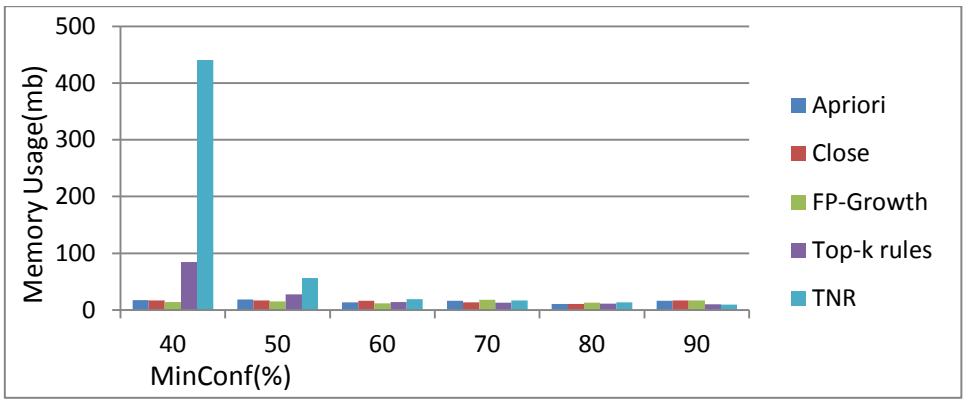

Figure (2) Maximum Memory Usage (MB) of Five Algorithms on Mushroom

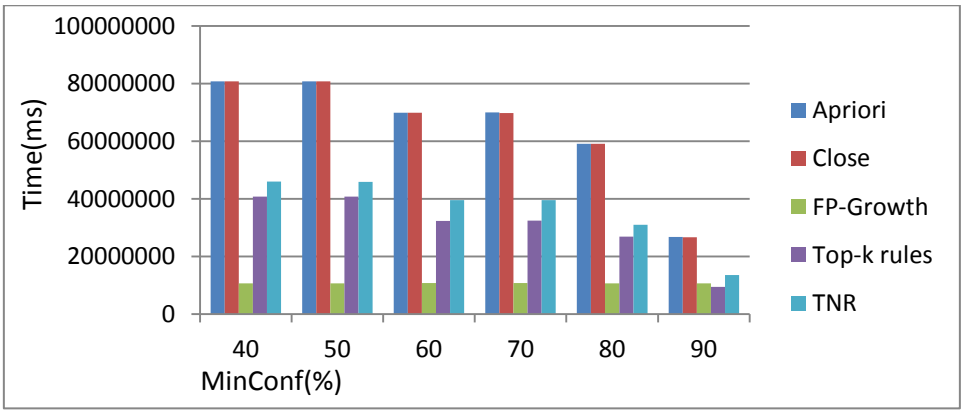

Figure (3) Execution Time of Five Algorithms on T10I4100K

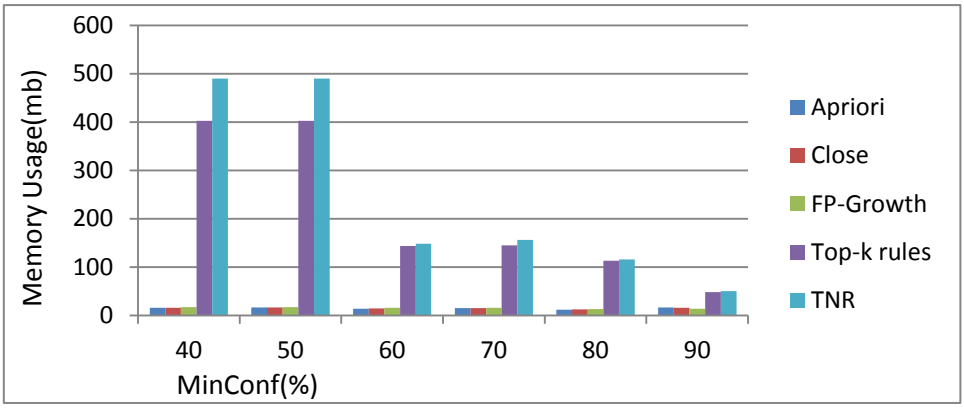

Figure (4) Maximum Memory Usage (MB) of Five Algorithms on T10I4100K

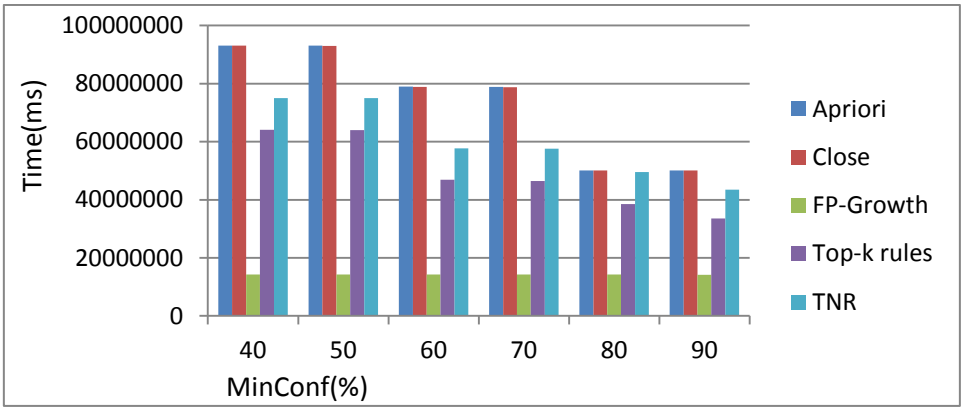

Figure (5) Execution Time of Five Algorithms on Retail.

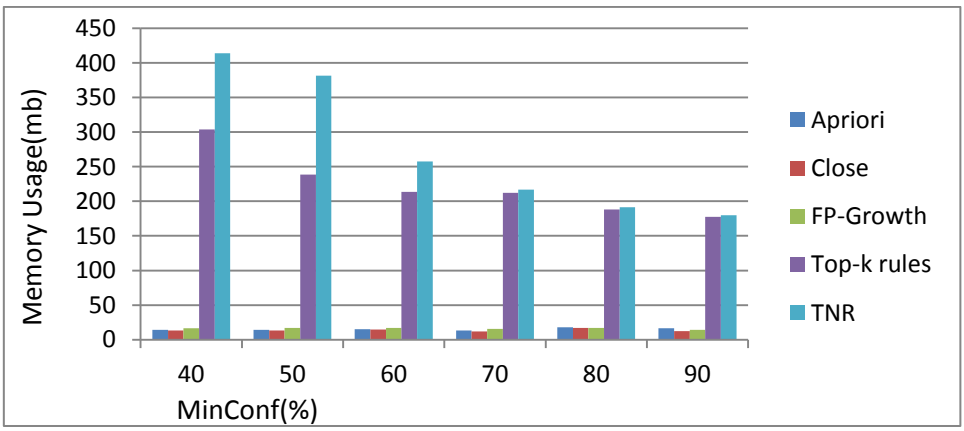

Figure (6) Maximum Memory Usage (MB) of Five Algorithms on Retail

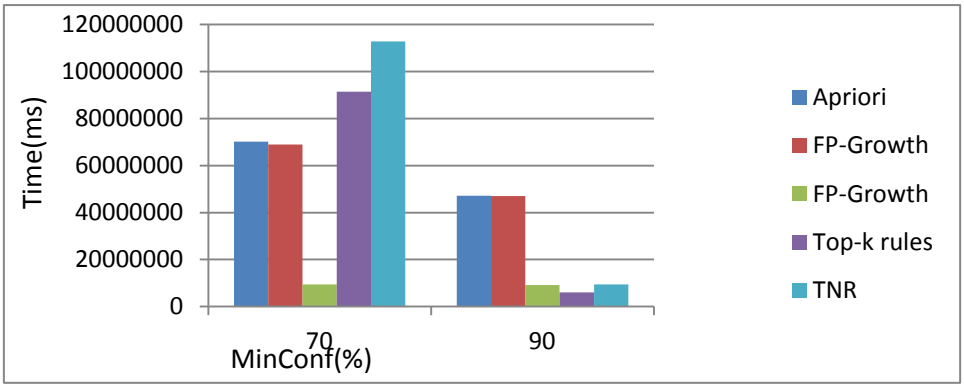

Figure (7) Execution Time of Five Algorithms on Pumsb.

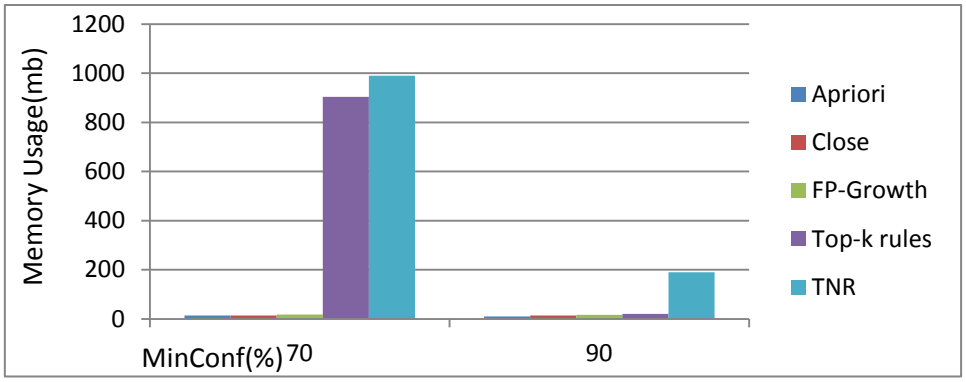

Figure (8) Maximum Memory Usage (MB) of Five Algorithms on Pumsb

4. Conclusion and Future Work

Association rules can be used to find the link between different products (items) in the transaction database, through which the buying behaviour patterns of customers can be discovered. This paper measured the efficiency of association rules algorithms on large datasets because the size of frequent itemsets differ depending on the datasets. The best performance is the FP-growth algorithm on different datasets, while the TNR algorithm has the best generated non-redundant association rules. The Top-k rules algorithm has the best performance on different datasets when the minimum support and minimum confidence are high.

In future, we will develop efficient algorithms for the fast mining of non-redundant association rules to be generated by the combination of FP-growth and TNR algorithms. Subsequently, efficient algorithms for mining frequent closed itemsets from large databases will be discussed.

References

[1] Rakesh Agrawal, Tomasz Imielinski and Arun Swami, "Mining Association Rules between Sets of Items in Large Databases", Proceedings of the 1993 ACM SIGMOD Conference Washington DC, USA, May 1993.

[2] Rakesh Agrawal and Ramakrishnan Srikant, "Fast Algorithms for Mining Association Rules", Proc. of the 20th VLDB conference Santiago, Chile, 1994.

[3] Nicolas Pasquier,Yves Bastide, Rafik Taouil and Lotfi Lakhal, "Efficient mining of association rules using closed itemset lattice", Information Systems, Vol. 24, No.1, Pages 25-46,1999.

[4] Jiawei Han, Jian Pei and Yiwen Yin, "Mining frequent patterns without candidate generation", In Proc. 2000 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'00), Dallas, TX, Pages 1–12, 2000.

[5] Philippe Fournier-Viger, Cheng-Wei Wu and Vincent S. Tseng, "Mining top-k association rules", In Proceedings of the 25th Canadian conference on Advances in Artificial Intelligence, Pages 61-73, 2012.

[6] Philippe Fournier-Viger and Vincent S. Tseng, "Mining Top-K Non-Redundant Association Rules", In Proceedings of the 20th international conference on Foundations of Intelligent Systems, Pages 31-40,2012.

[7] Yubo Yuan and Tingzhu Huang, "A Matrix Algorithm for Mining Association Rules", Lecture Notes in Computer Science, vol.3644, Pages 370 – 379, Sep 2005.

[8] Somboon Anekritmongkol and M.L.Kulthon Kasemsan, "Effective Candidates Generate Algorithm for Association Rules", International Conference on Future Information Technology (IPCSIT), vol.13, 2011.

[9] Nicolas Pasquier,Yves Bastide, Rafik Taouil and Lotfi Lakhal, "Discovering Frequent Closed Itemsets for Association Rules", Proceeding 7th International Conference Database Theory, Pages 398-416, 1999.

[10] G. Grahne and J. Zhu, "Fast algorithms for frequent itemset mining using FP-trees", Knowledge and Data Engineering, IEEE Transactions on , vol.17, No 10, Pages 1347 - 1362, Oct 2005.

[11] Liping Ji, Kian-Lee Tan, K H. Tung, "Compressed Hierachical Mining of frequent Closed Patterns from Dense Data Sets", IEEE Transaction on Knowledege and Engineering, vol.19, NO.9, Sep 2007.

[12] Mafruz Zaman Ashrafi, David Taniar, and Kate Smith, "A New Approach of Eliminating Redundant Association Rules", Lecture Notes in Computer Science, vol.3180, Pages 465 – 474, 2004.

[13] Mafruz Zaman Ashrafi, David Taniar and Kate Smith, "Redundant Association Rules Reduction Techniques", Lecture Notes in Computer Science, vol.3809, Pages 254 – 263, 2005.

[14] Jiawei Han and Micheline Kamber, "Data Mining Concepts and Techniques", Morgan Kaufmann, 2006.

[15] Jiawei Han, Jian Pei, Yiwen Yin and Runying Mao, "Mining Frequent Patterns without Candidate Generation: A Frequent-Pattern Tree Approach", Data Mining and Knowledge Discovery, vol.8, Pages 53–87, 2004.

[16] Ramesh C. Agarwal, Charu C. Aggarwal, V.V.V. Prasad, " A tree projection algorithm for generation of frequent itemsets", In J. Parallel and Distributed Computing, 2000.

[17] Chuan Wang, Christos Tjortjis, "PRICES: An Efficient Algorithm for Mining Association Rules", Lecture Notes in Computer Science, vol.3177, Pages 352 – 358, Jan 2004.

[18] http://fimi.ua.ac.be/data/, 2014.