Pruning of Apriori-Algorithm’s Pruning Steps

Prof. Dr. Ala H. Al-Hamamy   Dr. Hussein K. Al-Khafaji
Yasmeen F. Al-Wardi   Karim H. Al-Sa’di

Abstract

Association rules mining is the main task of data mining. It consists of two compulsory steps, the first step is discovery of frequent itemsets, and the second one is extracting the rules from the mined frequent itemsets. The first step is massive computational step. There are many algorithms presented to complete this step, but all of these algorithms are variations of the *Apriori* algorithm, which is the state of the art. *Apriori* requires a priori knowledge to generate the frequent itemsets and involves two time-consuming pruning steps to exclude the infrequent candidates and hold frequents. The first pruning operation is degenerating each of the (k+1)-itemsets to its k-itemset subsets. The second pruning operation is support counting for each candidate passed the first exam.

This research improves *Apriori* algorithm by pruning its pruning steps. It abridges the pruning steps to TID lists intersection and items union operations by taming the problem of frequent itemsets mining to lattice theory. It excludes the neediness to multi-scanning the database; instead, it scans the database only once.

Keywords: Data Mining, Association Rules, *Apriori* Algorithm, Lattice

1. Introduction

1.1 Association Rules

Association rules are one of the promising aspects of data mining as a knowledge discovery tool, and have been widely explored to date. They allow to capture all possible rules that explain the presence of some attributes according to the presence of other attributes. An association rule, $X \Rightarrow Y$, *is* a statement of the form "*for a specified fraction of transactions, a particular value of an attribute set X determines the value of attribute set Y as another particular value under a* *certain confidence*". Thus, association rules aim at discovering the patterns of co-occurrences of attributes in a database. For instance, in association rule in a bookshop data may be "*Whenever the customer buys Elian Rich's 'Artificial Intelligence' book, this happening for 20% of all purchases, also buys 'PROLOG manual' in 70% of the cases"*. "20%" is the support of the rule, while "70%" is the confidence. Agrawal, the founder of the problem of discovering association rules in databases defines it formally in [Rake96]

Mining of association rules was decomposed into ***two*** sub-problems [Rake96, Jiaw01, Piet98]: 1-***Discovering all frequent patterns*** (represented by large itemsets defined above), and 2- ***Generating of the association rules from those frequent itemsets***. The second sub-problem is straightforward, and can be done efficiently in a reasonable time. However, the first sub-problem is very tedious and computationally expensive for very large databases and this is the case for many real life applications. In large retailing data, the number of transactions is generally in the order of millions, and number of items (attributes) is generally in the order of thousands. *When the data contains N items, then the number of possibly large itemsets is 2^N*. However, the large itemsets existing in the database are much smaller than 2^N. Thus, brute force search techniques, which require exponential time, waste too much effort to obtain the set of large itemsets.

1.1.1 *Apriori* Algorithm

A number of algorithms are presented to mine frequent itemsets. All of these algorithms are variations of the standard algorithm, *Apriori*. Apriori requires a priori knowledge to generate the next generation of candidate itemsets, i.e., it generates the candidate (k+1)-itemsets from the frequent k-itemsets. In addition, it depends on the *Apriori fact*: "***all the subsets of a large itemset are large too***" [Rake96, Jiaw01, Piet98]. Hence, Apriori generates candidate itemsets that must pass two exams, (pruning steps), to be frequent itemsets. ***The first pruning step*** depends on the apriori fact. After generating a candidate (k+1)-itemset, it will be degenerated to its k-itemsets subsets. If any one of these subsets is not large, i.e., is not member in L_k , the candidate will be removed and regarded as small or infrequent, otherwise it must pass the second exam. The support of the candidate itemset, which delivered from the first pruning, is counted in ***the second pruning step***. If this support exceeds the minimum support, *minsup*, then the candidate will be regarded as frequent itemset. The support counting is accomplished by $|D|$ database scanning operation where |D| is the number of transactions in the database under consideration.

The second pruning step is implemented depending on a complex but novel data structure named ***hash tree*** [Rake96]. Hash tree is suggested to reduce the complexity of pruning steps. Figure (1) depicts the steps of Apriori algorithm.

The inputs to Apriori algorithm are a user-defined threshold, *minsup*, and a transaction database. Step #1 generates 1-itemsets, i.e., the frequent items. These 1-itemsets are stored in L1 list, which will be used to generate $C_{2.}$. C_2 is the list of candidate 2-itemsets. The generation operation of C_{k+1} from L_k is done by apriori_gen algorithm. Apriori_gen takes two different_last_items k-itemsets to generate candidate (k+1)-itemset by joining the similar k-1 items with the last different items. For example, let X and Y are two frequent 3-itemsets in L_3, such that X=*abc* and Y=*abd*, then the generated candidate 4-itemset is *abcd* which will be added to C_4 list. Apriori_gen performs the first complex pruning step to exclude infrequent candidates. This is done in step#21. It degenerates the generated candidate (k+1)-itemset to its k-itemset subsets. If there is any subset not member in L_k , then its candidate will be removed from C_{k+1} . Steps #8 to step#11 are responsible for the second pruning step. The core of these steps is the function *subset* that checks the subsetness of a candidate in a transaction. Subset function depends on a hash tree, which has been built gradually during the progress of mining the large itemsets in each iteration. To obtain more explanation, consider the following example.

```
Procedure **Apriori** (in D: transactional Database; minsup: minimum support)
  1   $L_1$ = {large 1-itemsets}
  2   $k = 2$
  3   **while** $L_{k-1} \neq \emptyset$ **do**
  4   **begin**
  5           $C_k = apriori\_gen(L_{k--1})$
  6         **for all** transactions $t$ in $D$ **do**
  7         **begin**
  8           $C^t = subset(C_k, t)$
  9           **for all** candidates $c \in C^t$ **do**
  10               $c.count = c.count + 1$
  11        **end**
  // 2nd pruning step
  12          $L_k = \{c \in C_k \mid c.count \geq minsup\}$
  13          $k = k + 1$
  14  **end**
```

```
 **apriori_gen** ($L_{k-1}$);
 15  $C_k = \phi$
 16  **for** all itemsets $X \in L_{k-1}$ and $Y \in L_{k-1}$ **do**
 17    **if** $X_1 = Y_1 \wedge \bullet\bullet\bullet \wedge X_{k-2} = Y_{k-2} \wedge X_{k-1} < Y_{k-1}$ **then begin**
 18      $C = X_1 X_2 \ldots X_{k-1} Y_{k-1}$
 19      add $C$ to $C_k$
 20  **end**
 // 1st pruning step
 21 delete candidate itemsets in $C_k$ whose any subset is not in $L_{k-1}$
```

Figure (1) Pseudo Code of Apriori Algorithm

Example 1 Consider the example transaction database$_{ETDB}$ in Table 2. There are six transactions in the database with Transaction IDentifiers, TIDs, 1, 2, 3, 4, 5, and 6. The set of items I = {A, B, C, D, E, F}, each item is an abbreviation of a movie or film title in CD shop sales as shown in table 1. There are totally (2^6 -1) = 63 nonempty itemsets (each non-empty subset of I is an itemset). {A} is a 1-

itemset and {AB} is a 2-itemset, and so on. Support$_{ETDB}$ (A) = 3 since three transactions include A. Let us assume that the minimum support (*minsup*) is two, (approximately taken as 33%). Then, {A, B, C, D, E, AB, AC, AE, BC, BD, BE, CD, CE, DE, ABC, ABE, ACE, BCD, BCE, BDE, CDE, ABCE, BCDE} are the set of large itemsets, since their support is greater than or equal to 2, (33% x 6), and the others are small itemsets. There are two distinct itemsets, ABCE, and BCDE, called **maximal itemsets;** all other large itemsets are subsets of one of them. Table (3) depicts all large itemsets with their supports. Let's assume that the minimum confidence (minconf) is set to 100%. Then, A $\Rightarrow$ B is an association rule with respect to the specified minsup and minconf (its support is 3, and its confidence is $\frac{SupportETDB(AD)}{SupportETDB(A)} \times$ 100 = 3/3$\times$ 100=100%). On the other hand, the rule B $\Rightarrow$ A is not a valid association rule since its confidence is 50%. This shows that the association rule is not always interchangeable sides in real world applications. If the rule sides are interchangeable, the new rule rarely has the same confidence value. This property increases the complexity of rule mining process. Table (4) depicts the association rules that can be mined from database$_{ETDB}$ according to 100% *minconf* value and 33% minsup value. It is obvious that the increment of minimum-support value reduces the number of large or frequent itemsets and vice versa. Figure (2) depicts briefly the apriori large-itemset mining process where *minsup*=3= 50%. L_k is a list of k-itemsets, while C_{k+1} is the list of candidate itemsets generated from L_k. For example, in C_3, the candidate CDE is discarded because its subset DE is not a member of L_2. BCD passes two pruning step to be validated as large itemset, all its length-2 subsets are members of L_2. In addition, its support is counted which equals 3>=*minsup*.

Item	Movie or film name

Table (1) The items abbreviations of database$_{ETDB}$

A	Titanic
B	Rain man
C	Gone With the Wind
D	Zorba
E	Speed
F	Gold Mountain

Table(2)A Transaction Database (Database$_{ETDB}$)

Transaction TID	Items-(film)
1	B,C,E
2	B,C,D,E
3	A,B,C,D,E
4	B,C,D
5	A,B,F
6	A,B,C,E

Table (3) Large itemsets with *minsup* = 33%=2

Support	Itemsets	No.
6 =100%	B	1
5 = 83%	C, BC	2
4 = 67%	E, BE, CE, BCE	4
3 = 50%	A, D, AB, BD, CD, BCD	6
2 = 33%	AC, AE, DE, ABC, ABE, ACE, BDE, CDE, ***ABCE***, ***BCDE***	10

Table (4) Association Rules with minconf=100%

$A \rightarrow B(3/3)$	$AC \rightarrow B(2/2)$	$AC \rightarrow BE(2/2)$
$C \rightarrow B(5/53)$	$AE \rightarrow B(2/23)$	$AE \rightarrow BC(2/23)$
$D \rightarrow B(3/3)$	AC → E (2/2)	DE → BC (2/2)
$E \rightarrow B(4/4)$	AE → C (2/2)	ABC → E (2/2)
$D \rightarrow C(3/3)$	DE → B (2/2)	ABE → C (2/2)
$E \rightarrow C(4/4)$	DE → C (2/2)	ACE → B (2/2)
$ABE \rightarrow C(2/2)$	ACE → B (2/2)	ABC → E (2/2)

L1	C2	L2	C3	L3	C4	L4
A	AB	AB	BCD	BCD	BCDE	{}
B	AC	BC	BCE	BCE		
C	AD	BD	BDE			
D	AE	BE	CDE			
E	BC	CD				
	BD	CE				
	BE					
	CD					
	CE					
	DE					

Figure (2) Apriori Mining Process

2. The Development of Apriori Algorithm

2.1 Set and Lattice Theory

The database under association rules mining consists of two sets, the items set I and TIDs set T. The relation that maps the first set to the second one transforms the database to *bipartite graph*. In addition, the subset binary relation is partial order on set I and set T according to the following definitions [Dave02]:

Definition 1: *Let P be a set, a partial order on P is a binary relation $\leq$, such that for all X, Y, Z $\in$P, the relation is:*

1) *Reflexive $X \leq X$.*
2) *Anti-Symmetric: $X \leq Y$ and $Y \leq X$, implies $X=Y$.*
3) *Transitive: $X \leq Y$ and $Y \leq Z$, implies $X \leq Z$.*

The set P with the relation $\leq$ is called an **ordered set** *and it is denoted as a pair $(P,\leq)$.*

Definition 2: *Let P be an ordered set, and let $S \subseteq P$. An element $X \in P$ is an* ***upper bound*** *of S if $s \leq X$ for all $s \in S$. The least upper bound, also called* ***meet***, *of S is denoted as $\vee S$.*

Definition 3: *Let P be an ordered set, and let $S \subseteq P$. An element $X \in P$ be a lower bound of S if $s \geq X$ for all $s \in S$. The greatest lower bound, also called join, of S is denoted as $\wedge S$.*

If S={x, y}, then the meet is written as $x \vee y$, and the join is written as $x \wedge y$.

Definition 4: *Let P be an ordered set. The greatest element of P, denoted as $\lceil$, is called* ***top element***, *and the least element of P, denoted as $\rfloor$, is called the* ***bottom element***.

Definition 5: *An ordered set $(L, \leq)$ is called lattice, if for any two elements x and y in L, the meet and join of x and y always exist. L is a complete lattice if $\vee S$ and $\wedge S$ exist for all $S \subseteq L$. Any finite lattice is complete. L is called meet semi-lattice if only the meet exists. L is called a join semi-lattice if only the join exists.*

To give more explanation and to tame the rule mining problem to the definitions above, let us return to example (1), where the set of frequent items is I_f= *{A, B, C, D, E}* and *T*={1,2,3,4,5,6}. Let *P* (I_f) denote the set of all subsets of I_f, (called power set of I_f) and P(*T*) is the power set of *T*. The partial orders $(P(I_f),\subseteq)$ the set of all possible itemsets and $(P(T),\subseteq)$ the set of all possible tidsets is both complete lattices where the join is given by set intersection and meet is given by set union. The ordered set $(P(I_f),\subseteq)$ *and* $(P(T),\subseteq)$ is represented in figure (3). Suppose a set $S\subseteq P(I_f)$ *={ABC, ABE, ACE, BCD}*. Then the itemsets ***abcde*** is the upper bound of S and the empty set {} is the lower bound and the join of S. The meet of S is ***abcde***. The top element of (P(I_f),$\subseteq$) is *abcde* and the bottom element is {}. The set of all frequent itemsets is a join semi-lattice, consider figure (3), which depicts the join of all frequent itemsets extracted in example (1) of CD-shop sales database. The join of any two frequent itemsets is frequent, but their meet may or may not frequent. This phenomenon is consistent with the *apriori-fact* that "*all the subsets of a frequent itemset are frequent too*". In figure (3), the join of the itemsets AC and CD is C which is frequent, $AC\wedge CD= AC\cap CD=C$, but the meet $AC \vee CD= AC \cup CD =ACD$ is not frequent. On the other hand, the meet $AC \vee AE= AC \cup AE =ACE$ is frequent.

As mentioned previously, there are two maximal itemsets in example (1), ***ABCE*** and ***BCDE*** that are shown in grayed circle; therefore, each of them is a top element of the semi graph. To distinct the subsets of *ABCE* and *BCDE* in figure (3), we draw the arcs that connect the subsets of *ABCE* as dashed lines and the arcs that connect the subsets of *BCDE* as bold lines. Double lines, dashed and bold lines, connect the itemsets, which are subsets of *ABCE* and *BCDE*.

According to this taming of association rules mining to the graph theory, any (k+1)-itemset can be obtained from the union of any of two its k-itemsets subsets, also its tidlist can be obtained by intersection the tidlists of these two k-itemsets.

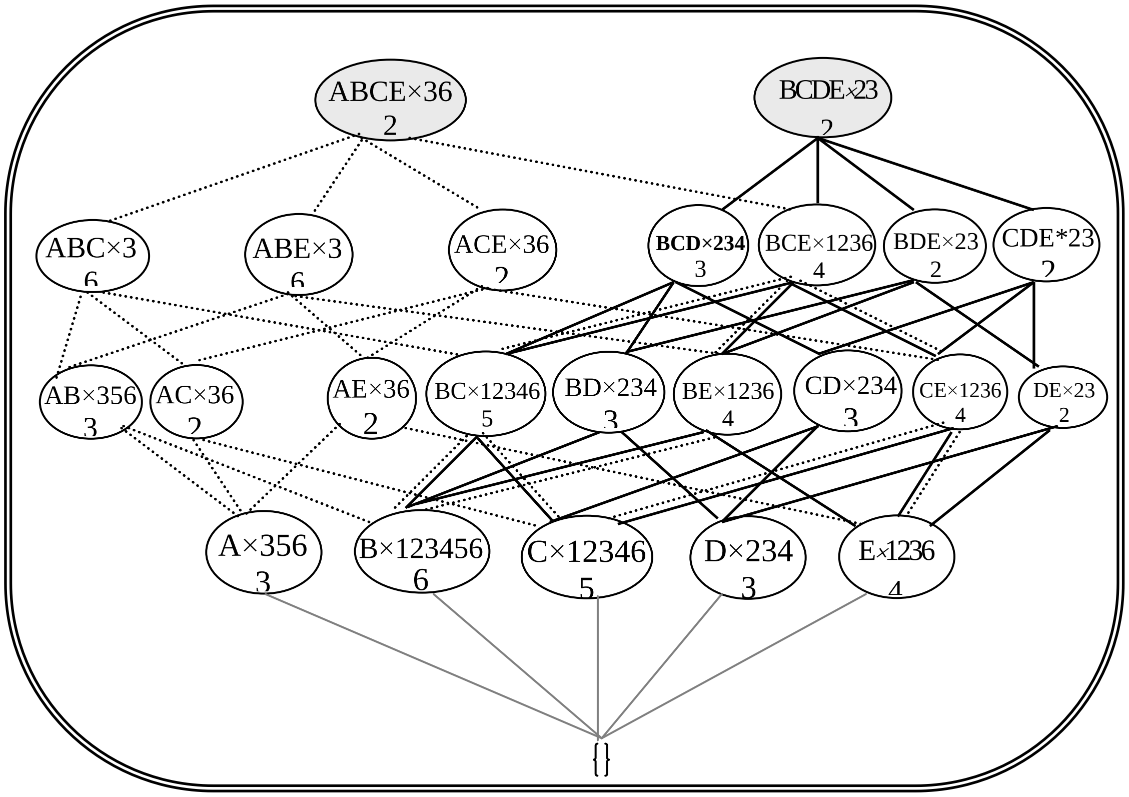

Figure (3) join semi-lattice of frequent itemsets

2. 2 The Developed Algorithm

The developed algorithm excludes the apriori’s pruning steps that previously mentioned. It does not generate candidate itemsets. It generates an itemset with its tidlist by union and intersection operation respectively. The generated itemset will excluded or regarded as large itemset directly and simply by counting the length of its tidlist. Figure (4) depicts the developed apriori algorithm.

The developed algorithm scans the database only once. It loads the frequent items, i.e., 1-itemsets, and there tidlists and then generates all frequent itemsets from these 1-itemsets without re-scanning the database. It utilizes the concept of vertical database to represent the database.

```
Procedure Developed_Apriori ()
 1  $L_1$ = {large 1-itemsets}
 2  $k = 2$
 3  while $L_{k-1} \neq \varnothing$ do
 4  begin
 5      $L_k$ = developed_apriori_gen ($L_{k-1}$)
 6      $k = k + 1$
 7   end %while
 8  end % apriori

 Developed_apriori_gen ($L_{k-1}$);
 9  $C_k = \phi$
10  for all itemsets $X \in L_{k-1}$ and $Y \in L_{k-1}$ do
11    if $X_1 = Y_1 \wedge \bullet\bullet\bullet \wedge X_{k-2} = Y_{k-2} \wedge X_{k-1} < Y_{k-1}$ then begin
12      $C$ = union(X,Y);
13      $C_{TID}$ =intersect ($X_{TID}$, $Y_{TID}$);
14      If $|C_{TID}| >= minsup$ then add C to $L_k$;
15     Else ignore C;
16   end;
17 end; %apriori_gen
```

Figure (4) Pseudo code of developed algorithm

It is easy to see that the main portion of apriori algorithm, step # 3 to step # 4 of figure (1), is converted to simple iteration to call *developed_apriori_gen*, as shown in step #9 to step #14 of figure (4).

Step # 11 and step # 12 make a union operation of two large (k-1)-itemsets to generate a candidate k-itemset, i.e., finding the meet of the (k-1) th level. Step #13 intersects the tidlists of itemset X and Y to find the tidlist the candidate C, i.e., it finds the join of the tidlists. The number of TIDs in a tidlist of C is the support of C, which is computed and checked according to the *minsup* threshold in steps # 13 and 14. To give full explanation, recall example (1). Let us trace the behavior of the developed apriori depending example (1) with minsup=3. Firstly, the algorithm selects the 1-itemsets, frequent items. Then the algorithm joins the frequent items and intersects their tidlists to find the candidate 2-itemsets; C2. The infrequent candidates are excluded easily by checking the cardinality of their tidlists with the *minsup*. In figure (5), the bold cells in C's columns are the candidates with supports greater than or equal to the minsup=3; therefore they are regarded as frequent itemsets, hence they appear in the next level L_n. In turn, these frequent itemsets are used to generate the next level of candidates. The shaded cells in figure (5) of C's columns are excluded, i.e., they are regarded as infrequent itemsets.

2.2.1 Support Counting

As mentioned previously, the developed apriori algorithm regards the support of a mined frequent k-itemset as the cardinality of its TID-list which obtained by intersection of its frequent (k-1)-items subsets. To prove the validity of this support counting, which excludes the frequent database scanning, consider the following definitions and theorems [Dave02]:

Definition (6) *A lattice is said to be distributive if for all X, Y, Z* $\in$ *L,* $X \wedge (Y \vee Z) = (X \wedge Y) \vee (X \wedge Z)$.

Definition (7) *Let L be a lattice with bottom element* $\rfloor$. *Then* $X \in L$ *is called an atom if X covers* $\rfloor$. *The set of atoms of L is denoted by A (L).*

Definition (8) *A lattice L is called a Boolean lattice if*

1) It is distributive.

2) It has $\lceil$ *and* $\rfloor$ *elements.*

3) Each member X of the lattice has a complement.

Theorem (1) *For a finite Boolean lattice L, with* $X \in L$, $X = \vee \{Y \in A(L) \mid Y \leq X\}$.

This means that every element of a Boolean lattice is given as a join of the set of atoms. Since the powerset lattice *P(I)* is a Boolean lattice with the join operation corresponding to set union, we get the following lemma:

Lemma (1) *For any* $X \in P(I)$, *let* $J=\{Y \in A(P(I)) \mid Y \leq X\}$. *Then* $X = \cup_{Y \in J}{}^{Y}$, $\Sigma(X) = |\cap_{Y \in J}{}^{\lambda(X)}|$.

Lemma 1 stipulates that any obtained itemset is a join of some atoms of a specified graph, and the support of the itemset can be obtained by intersecting the tidset of the atoms. This lemma can be generalized to a set of itemsets:

Lemma (2) *For any* $X \in P(I)$, *let* $X = \cup_{Y \in J}{}^{Y}$, *then* $\Sigma(X) = |\cap_{Y \in J}{}^{\lambda(X)}|$.

Lemma 2 states that if an itemset is obtained as the union of a set of itemset in J, then its support is counted by intersecting the tidsets of elements in J. Depending on Lemma 1 and Lemma 2, we can compute the support of an itemset of length k, i.e., k-itemset, by intersecting the tidsets of any two (k-1)-itemsets of its subsets. Therefore, the frequentness of the generated itemset is determined by simple checking of its cardinality. The developed version of apriori scans the database only once by selection of frequent items, 1-k-itemsets, and keeps the tidlist of each one. To generate the next generation of candidate itemsets, the only required is union of two last-item-differ itemset, to intersect their tidlists, and counting the cardinality of result tidlist.

3. Experimental Results

Experiments were performed on 1.2 GHz Tualatin PC with 320 MB of memory, running windows^{xp} .Apriori and developed-apriori was implemented by using PL/SQL of Oracle DBMS. These implementations are executed by using five real and one artificial database, which recently made publicly available on (www.ecn.purdue.edu/kddcup). The characteristics of these databases are shown in table (5).

L1	C2	L2	C3	L3	C4	L4
A 3,5,6	A∪B=AB $\lvert 3,5,6\rvert \cap \lvert 1,2,3,4,5,6\rvert = \lvert 3,5,6\rvert \geq 3$	AB 3,5,6	BC∪BD=BCD $\lvert 1,2,3,4,6\rvert \cap \lvert 2,3,4\rvert = \lvert 2,3,4\rvert \geq 3$	BCD 2,3,4	BCD∪BCE=BCDE $\lvert 2,3,4\rvert \cap \lvert 1,2,3,6\rvert = \lvert 2,3\rvert \leq 3$	{}
B 1,2,3,4,5,6	A∪C=AC $\lvert 3,5,6\rvert \cap \lvert 1,2,3,4,6\rvert = \lvert 3,6\rvert \leq 3$	BC 1,2,3,4,6	BC∪BE=BCE $\lvert 1,2,3,4,6\rvert \cap \lvert 1,2,3,6\rvert = \lvert 1,2,3,6\rvert \geq 3$	BCE 1,2,3,6		
C 1,2,3,4,6	A∪D=AD $\lvert 3,5,6\rvert \cap \lvert 2,3,4\rvert = \lvert 3\rvert \leq 3$	BD 2,3,4	BD∪BE=BDE $\lvert 2,3,4\rvert \cap \lvert 1,2,3,6\rvert = \lvert 2,3\rvert \leq 3$			
D 2,3,4	A∪E=AE $\lvert 3,5,6\rvert \cap \lvert 1,2,3,6\rvert = \lvert 3,6\rvert \leq 3$	BE 1,2,3,6	CD∪CE=CDE $\lvert 2,3,4\rvert \cap \lvert 1,2,3,6\rvert = \lvert 2,3\rvert \leq 3$			
E 1,2,3,6	B∪C=BC $\lvert 1,2,3,4,5,6\rvert \cap \lvert 1,2,3,4,6\rvert = \lvert 1,2,3,4,6\rvert \geq 3$	CD 2,3,4				
	B∪D=BD $\lvert 1,2,3,4,5,6\rvert \cap \lvert 2,3,4\rvert = \lvert 2,3,4\rvert \geq 3$	CE 1,2,3,6				
	B∪E=BE $\lvert 1,2,3,4,5,6\rvert \cap \lvert 1,2,3,6\rvert = \lvert 1,2,3,6\rvert \geq 3$					
	C∪D=CD $\lvert 1,2,3,4,6\rvert \cap \lvert 2,3,4\rvert = \lvert 2,3,4\rvert \geq 3$					
	C∪E=CE $\lvert 1,2,3,4,6\rvert \cap \lvert 1,2,3,6\rvert = \lvert 1,2,3,6\rvert \geq 3$					
	D∪E=DE $\lvert 2,3,4\rvert \cap \lvert 1,2,3,6\rvert = \lvert 2,3\rvert \leq 3$					

Figure (5) Developed_Apriori Mining Process

Table (5) Database Characteristics

Database Name	Number of Items	Number of Record	Database Type
Chess	76	3196	Real/dense
Connect	130	67557	Real/dense
Mushroom	120	8124	Real/dense
Pumsb*	7117	49046	Real/very dense
Pumsb	7117	49046	Real/dense
T40I10D100K	1000	100000	Artificial/sparse

Figure (6) depicts a comparison of developed version of apriori versus apriori algorithm. Each database is carried out on different *minsup* values, x-axis. Y-axis represents the execution time in second. It is easy to see that the developed apriori outperforms the original apriori in all the experiments.

Pumsb* database is very dense database. Apriori is not efficient with dense databases [Jiaw01], therefore, the execution curve of apriori of Pumsb* database parallels the curve of the developed apriori in far region or grids of x-axis. This curve reflects the efficiency of the development to deals with dense databases with small minsup values.

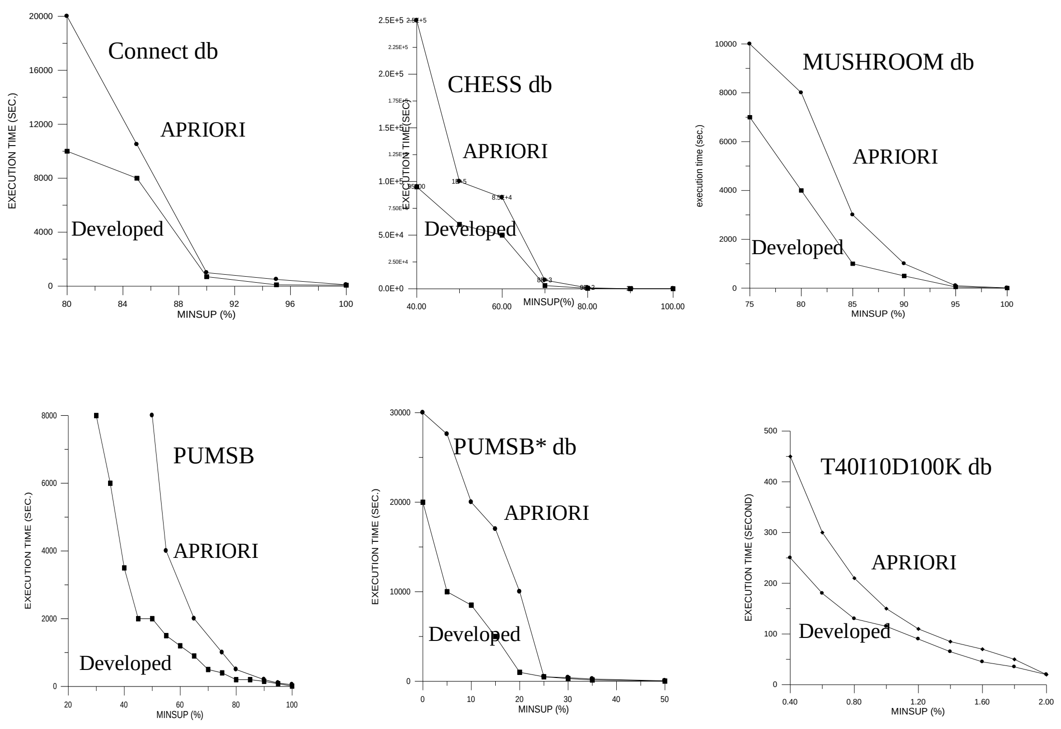

Figure (6) Developed Apriori versus Apriori Algorithm

4. Discussion and Conclusion

We presented and evaluated an efficient developed version of apriori algorithm. This algorithm scans the database only once by exploring the itemsets and tidlists simultaneously depending on vertical layout instead of horizontal one. It excludes the needing to construct and explore the complex data structure, hash tree, of apriori algorithm, and in addition excludes the time-consuming pruning steps. This development transfers the complexity of apriori from exponential $O(r.\ n.\ 2^{l})$ to linear complexity $O(C_n)$, where C_n is the number of candidate itemsets to be frequent, $|T| = n$ is the number of transactions in a database, l is the length of the longest frequent itemset, and r is the number of maximal frequent itemsets. An extensive set of experiments confirm that the developed apriori outperforms apriori algorithm. However, the developed apriori appears high appealing to RAM to maintain itemset × tidset pairs of vertical database layout. After the development, apriori becomes very suitable to design DBMS-embedded rules miners and can be implemented with smaller amount of programming code.

References

[Dave02] B. A. Davey, H.A. Priestly, "**Introduction to Lattices and order**", 2nd ed., Cambridge Uni. Press, 2002.

[Jiaw01] Jiawei Han, Micheline Kamber, "**Data Mining Concepts and Techniques**", MK, 2001.

[Rake96] Rakesh Agrawal, Heikki Mannila, Ramakrishnan Srikant, "**Fast discovery of association rules.** In Usama Fayyad, Gregory Piatetsky-Shapiro, Padhraic Smyth, and Ramaswamy Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining,* pages 307-328. AAAI/MIT Press, 1996.

[Piet98] Pieter Adriaans, Dolf Zantinge, "**Data Mining**", Addison Wesley, 1998.

تـشــــذيب خطوات الــتــشـــذيب في خوارزمية الأبريوري

ا.د. علاء الحمامي   ا.م. د. حسين الخفاجي   ياسمين فرحان   كريم هاشم

الخلاصة

استكشاف قواعد الأرتباط هو واحد من اهم اغراض تعدين البيانات. لأستكشاف قواعد الأرتباط في قواعد البيانات، هنالك اجرائين لازمين هما: (١) استكشاف المجموعات الكبيرة للعناصر ، (٢) واستخراج القواعد من هذه المجاميع المستكشفة. الأجراء الأول اعقد بكثير من الثاني والذي يحتاج الى اجراءات حسابية كبيرة. هنالك العديد من الخوارزميات المقترحة لانجاز الأجراء الأول، كلها تعتبر تنوعا من خوارزمية قياسية تدعى ابريوري. خوارزمية ابريوري تحتاج الى معرفة مسبقة لتوليد مستوى جديد من مجموعات العناصر الكبيرة. انها تولد مجموعات عناصر مرشحة لتكون كبيرة. لأستبعاد المرشحات التي لاتتواقر فيها الشروط،، الخوارزمية تستخدم عمليتي تشذيب معقدة؛ الأولى هي تهديم مجموعة العناصر

المتولدة المرشحة بطول (ن) الى مجموعاتها الجزئية بطول (ن-١) والتي يجب ان تكون كلها كبيرة وخلافا لذلك فان المرشحة سوف تستبعد. والغير مستبعدة تجتاز عملية تشذيب اخرى، حيث تحسب تكراراتها في القاعدة التي يجب تتجاوز خط عتبة قيمة الدعم الأصغر. لذلك فان الخوارزمية تحتاج الى جولات متككرة من البحث في قاعدة البيانات والتي تستهلك زمنا كبيرا.

البحث يقدم خوارزمية جديدة او تطويرا لخوارزمية الأبريوري، تهدف لأستبعاد عمليات التشذيب واختزال عمليات التوليد والتشذيب الى عملية تقاطع بين رموز الصفقات لمجموعتي عناصر كبيرة بطول (ن) لتوليد المجاميع ذات طول (ن+١). هذا التطوير اعتمد اخضاع مشكلة استكشاف قواعد الأرتباط الى نظريات المخططات الشبكية. قواعد بيانات عديدة استخدمت لأثبات جدوى هذه الخوارزمية المطورة وتفوقها على سابقتها من ناحية زمن التنفيذ والتعقيد البرمجي الذي ينعكس على زمن الأنجاز.