Int. J. of Computers, Communications & Control, ISSN 1841-9836, E-ISSN 1841-9844
Vol. V (2010), No. 4, pp. 567-577

# A New Rymon Tree Based Procedure for Mining Statistically Significant Frequent Itemsets

P. Stanišić, S. Tomović

**Predrag Stanisic, Savo Tomovic**
University of Montenegro
Department of Mathematics and Computer Science
Dzordza Vasingtona bb, Podgorica, Montenegro
E-mail: pedjas@ac.me, savica@t-com.me

**Abstract:** In this paper we suggest a new method for frequent itemsets mining, which is more efficient than well known Apriori algorithm. The method is based on special structure called Rymon tree. For its implementation, we suggest modified sort-merge-join algorithm. Finally, we explain how support measure, which is used in Apriori algorithm, gives statistically significant frequent itemsets.
**Keywords:** frequent itemset mining, association analysis, Apriori algorithm, Rymon tree

## 1 Introduction

Finding frequent itemsets in databases is fundamental operation behind association rule mining. The problem of mining association rules over transactional databases was introduced in [1]. An example of such rule might be that "85% of customers who bought milk also bought bread". Discovering all such rules is important for planning marketing campaigns, designing catalogues, managing prices and stocks, customer relationships management etc.

The supermarket is interested in identifying associations between item sets; for example, it may be interested to know how many of customers who bought milk also bought bread. This knowledge is important because if it turns out that many of the customers who bought milk also bought bread, the supermarket will place bread physically close to milk in order to stimulate the sales of bread. Of course, such a piece of knowledge is especially interesting when there is a substantial number of customers who buy two items together and when large fraction of those individuals who buy milk also buy bread.

For example, the association rule $milk \Rightarrow bread$ [support=20%, confidence=85%] represents facts:

- 20% of all transactions under analysis contain milk and bread;
- 85% of the customers who purchased milk also purchased bread.

The result of association analysis is strong association rules, which are rules satisfying a minimal support and minimal confidence threshold. The minimal support and the minimal confidence are input parameters for association analysis.

The problem of association rules mining can be decomposed into two sub-problems [1]:

- Discovering frequent itemsets. Frequent itemsets have support higher than minimal support;
- Generating rules. The aim of this step is to derive rules with high confidence (strong rules) from frequent itemsets. For each frequent itemset $l$ all nonempty subsets of $l$ are found; for each $a \subset l \wedge a \neq \varnothing$ the rule $a \Rightarrow l - a$ is generated, if $\frac{support(l)}{support(a)} > minimal\ confidence$.

Overall performances of mining association rules are determined by the first step; we do not cosider the second step in this paper. Efficient algorithms for solving the second sub-problem are presented in [12].

The paper is organized as follows. Section 2 provides formalization of frequent itemsets mining

Copyright © 2006-2010 by CCC Publications

problem. Section 3 describes Apriori multiple_num algorithm which is a modification of well known Apriori algorithm [1]. Section 4 presents a new candidate generation procedure which is part of Apriori multiple_num. In section 5 we use hypothesis testing to validate generated frequent itemsets.

## 2 Preliminaries

Suppose that $I$ is a finite set; we refer to the elements of $I$ as items. We primarily use notions from [10].

**Definition 1.** A transaction dataset on I is a function T: {1, ...,n} $\rightarrow$ P(I), where P(I) is set of all subsets of I. The set T(k) is the $k^{th}$ transaction of T. The numbers 1,...,n are the transaction identifiers (TIDs). [10]

Given a transaction data set T on the set I, we would like to determine those subsets of I that occur often enough as values of T. [10]

**Definition 2.** Let T: 1, ..., n$\rightarrow$P(I) be a transaction data set on set of items I, where P(I) is set of all subsets of I. The support count of subset K of set of items I in T is the number $suppcount_T$(K) given by:

$$suppcount_T(K) = |\{k|1 \leq k \leq n \wedge K \subseteq T(k)\}|. \tag{1}$$

The support of an item set K (in the following text instead of "item set K" we will use "itemset K") is the number:

$$support_T(K) = suppcount_T(K)/n. \tag{2}$$

[10]

The following rather straightforward statement is fundamental for the study of frequent itemsets. It is known as Apriori principle [1]. Proof is presented in order to introduce anti-monotone property.

**Theorem 3.** *Let T: 1, ..., n$\rightarrow$ P(I) be a transaction data set on a set of items I, where P(I) is set of all subsets of I. If K and K' are two itemsets, then $K' \subseteq K$ implies $support_T(K') \geq support_T(K)$. [10]*

**Proof:** The previous theorem states that $support_T$ for an itemset has the anti-monotone property. It means that support for an itemset never exceeds the support for its subsets. For proof, it is sufficient to note that every transaction that contains K also contains K'. The statement from the theorem follows immediately. □

**Definition 4.** An itemset K is $\mu$-frequent relative to the transaction data set T if $support_T(K) \geq \mu$. We denote by $F_T^\mu$ the collection of all $\mu$-frequent itemsets relative to the transaction data set T and by $F_{T,r}^\mu$ the collection of $\mu$-frequent itemsets that contain r items for $r \geq 1$ (in the following text we will use r-itemset to denote itemset that contains r items). [10]

Note that $F_T^\mu = \bigcup_{r\geq 1} F_{T,r}^\mu$. If it is clear what $\mu$ and $T$ are, we can omit them.

In this paper we will propose new algorithm for frequent itemsets mining which is based on special structure: Rymon tree. The Rymon tree was introduced in [8] in order to provide a unified search-based framework for several problems in artificial intelligence; the Rymon tree is also useful for data mining algorithms. In Definition 5 and 6 we define necessary concepts and in Definition 7 we define the Rymon tree.

**Definition 5.** Let $S$ be a set and let $d : S \rightarrow N$ be an injective function. The number $d(x)$ is the index of $x \in S$. If $P \subseteq S$, *view of P* is subset $view(d,P) = \{s \in S|d(s) > max_{p\in P}d(p)\}$.

**Definition 6.** A collection of sets $C$ is *hereditary* if $U \in C$ and $W \subseteq U$ implies $W \in C$.

**Definition 7.** Let $C$ be a hereditary collection of subsets of a set $S$. The graph $G = (C, E)$ is a Rymon tree for $C$ and the indexing function $d$ if:

- the root of the $G$ is $\emptyset$
- the children of a node $P$ are the sets of the form $P \cup \{s\}$, where $s \in view(d, P)$

If $S = \{s_1, ..., s_n\}$ and $d(s_i) = i$ for $1 \leq i \leq n$, we will omit the indexing function from the definition of the Rymon tree for $P(S)$.

Let $S = \{i_1, i_2, i_3, i_4\}$ and let $C$ be $P(S)$, which is clearly a hereditary collection of sets. Finally, let $d$ be injective mapping: $d(i_k) = k$ for $1 \leq k \leq 4$. The Rymon tree for $C$ and $d$ is shown in Fig. 1.

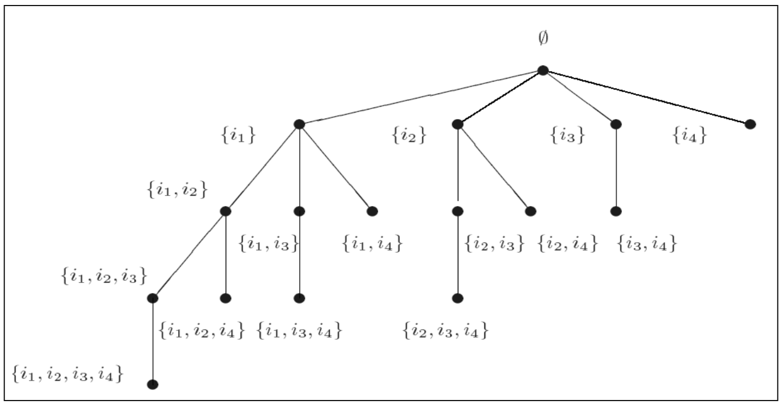

Figure 1: Example of Rymon tree

A key property of a Rymon tree is stated next.

**Theorem 8.** *Let* G *be a Rymon tree for a hereditary collection* C *of subsets of a set* S *and an indexing function* d*. Every set* P *of* C *occurs exactly once in the tree.*

Note that in the Rymon tree of a collection $P(S)$, the collection $S_r$, that consists of sets located at distance $r$ from the root, denotes all subsets of the size $r$ of $S$.

## 3 Apriori multiple_num Algorithm

Apriori multiple_num algorithm generates frequent itemsets starting with frequent 1-itemsets (itemsets consisted of just one item). Next, the algorithm iteratively generates frequent itemsets to the maximal length of frequent itemset. Each iteration of the algorithm consists of two phases: *candidate generation* and *support counting*.

In candidate generation phase potentially frequent itemsets or candidate itemsets are generated. The Apriori principle [1] is used in this phase. It is based on anti-monotone property of the itemset support (see Theorem 3) and provides elimination or pruning of some candidate itemsets without calculating its support. According to the Apriori principle, if $X$ is frequent itemset, then all its subsets are also frequent. This fact is used in candidate generation phase in a way that the candidate containing at least one not frequent subset is being pruned immediately (before support counting phase).

Support counting phase consists of calculating support for all previously generated candidates (which are not pruned according to the Apriori principle in the candidate generation phase). Calculating candidate support requires one database scan and efficient determination if the candidates are contained in particular transaction $t \in T$. For candidates contained in $t \in T$, it's support will be incremented. On account of that, the candidates are organized in hash tree. The candidates which have enough support are termed as frequent itemsets.

The main difference between iterations in original Apriori algorithm [1] and Apriori multiple_num algorithm is that iterations in later one are "longer", which is determined by *multiple_num* parameter. Actually, in original Apriori algorithm in the iteration $k$ set $F_k$(containing all frequent itemsets with $k$ items) is generated, while Apriori multiple_num algorithm in the iteration $k$ generates sets $F_{k+i}, 0 \leq i \leq multiple\_num$. If $k_{Max} < multiple\_num$ is true, where $k_{Max}$ is the maximal length of frequent itemset, Apriori multiple_num algorithm terminates in just two iterations, or just two database scans.

In addition, all candidate $k$-itemsets (itemsets containing $k$ items) will be signed as $C_k$, and all frequent $k$-itemsets as $F_k$. Pseudocode for Apriori multiple_num algorithm is given bellow.

```
Apriori multiple_num Algorithm
Input: T-transactional database; $\mu$ -minimal support;
Output: F-frequent itemsets in T
Method:
   1. $F_1 = all\_large\_1itemsets(T, \mu)$
   2. $multiple\_num = maximal\_length\_of\_transactions$
   3. $C_2 = apriori\_gen(F_1, F_1)$
   4. FOR i=3 TO multiple_num
          $C_i = apriori\_gen(C_{i-1}, C_{i-2})$
      END FOR
   5. FOR i=2 TO multiple_num
          $createCandidateHashtree(C_i)$
      END FOR
   6. FOR EACH $t \in T$ DO
          FOR i=2 TO multiple_num
              $traverseHashtree(C_i, t)$
          END FOR
      END FOR
   7. FOR i=2 TO multiple_num
          $F_i = \{c \in C_i | support(c) \geq \mu\}$
      END FOR
   8. $F = \bigcup_k F_k$
```

Let us explain the most important steps briefly. Generating frequent 1-itemsets is done in the same way as in original Apriori algorithm [1]. This step requires one database scan. Then, parameter *multiple_num* is set to the length of the longest transaction from the database $T$, which ensures that the algorithm will need just one more database scan. Steps 3 and 4 are concerned with candidate generation: in step 3 set $C_2$ is generated by calling apriori_gen function, then loop in step 4 generates all other candidates $C_i, 3 \leq i \leq multiple\_num$, by calling apriori_gen function, but with the following difference. According to original Apriori algorithm [1] candidate itemsets $C_{k+1}$ (candidate itemsets containing $k$+1 items) is formed from the set $F_k$(frequent itemsets containing $k$ items) in iteration $k$+1. However, we want to generate all itemsets in just one loop in order to reduce number of iterations (database scans) to two, but we do not have the necessary frequent sets. As the solution, arguments are candidate sets $C_{k-1}$ and $C_{k-2}$, which is known at this moment. The next section describes modification of apriori_gen function and its fast implementation.

The support counting phase comes next. All candidate itemsets $C_i, 2 \leq i \leq multiple\_num$ are organized in separate hash trees in order to make support counting process efficient. Then, we scan database and calculate support for candidates by traversing corresponding hash trees. At the end of support counting phase frequent itemsets $F_i, 1 \leq i \leq multiple\_num$ are generated. As we stated earlier, we will not further consider support counting phase.

Apriori algorithm [1] performs $k_{max}+1$ iterations, where $k_{max}$ is the maximal length of frequent itemsets, and in each iteration it scans whole database. Apriori multiple_num algorithm finishes after 2 iterations and performs 2 database scans.

## 4 New Procedure for Candidate Generation

We assume that any itemset $K$ is kept sorted according to some relation $<$, where for all $x, y \in K$, $x < y$ means that object $x$ is in front of object $y$. Also, we assume that all transactions in database $T$ and all subsets of $K$ are kept sorted in lexicographic order according to relation $<$.

For candidate generation we suggest an original method by which the set $C_{T,k}$ is calculated by joining $C^{\mu}_{T,k-1}$ with $C^{\mu}_{T,k-2}$, for $k \geq 3$. Candidate $k$-itemset is created from one candidate ($k$-1)-itemset and one candidate ($k$-2)-itemset in the following way. Let $X = \{x_1, ..., x_{k-1}\} \in C^{\mu}_{T,k-1}$ and $Y = \{y_1, ..., y_{k-2}\} \in C^{\mu}_{T,k-2}$. Itemsets $X$ and $Y$ are joined if and only if the following condition is satisfied:

$$x_i = y_i, (1 \leq i \leq k-3) \bigwedge x_{k-1} < y_{k-2} \tag{3}$$

producing the candidate $k$-itemset $\{x_1, ..., x_{k-2}, x_{k-1}, y_{k-2}\}$.

We will prove the correctness of the suggested method. In the following text we will denote this method by $C_{T,k} = C^{\mu}_{T,k-1} \times C^{\mu}_{T,k-2}$. Let $I = i_1, ..., i_n$ be a set of items that contains n elements. Denote by $G_I = (P(I), E)$ the Rymon tree of $P(I)$. The root of the tree is $\emptyset$. A vertex $K = \{i_{p_1}, ..., i_{p_k}\}$ with $i_{p_1} < i_{p_2} < ... < i_{p_k}$ has $n - i_{p_k}$ children $K \bigcup j$, where $i_{p_k} < j \leq n$. Let $S_r$ be the collection of itemsets that have $r$ elements. The next theorem suggest a technique for generating $S_r$ starting from $S_{r-1}$ and $S_{r-2}$. It is a modification of Theorem 7.8. from [10].

**Theorem 9.** *Let $G$ be the Ryman tree of $P(I)$, where $I = i_1, ..., i_n$. If $W \in S_r$, where $r \geq 3$, then there exists a unique pair of distinct sets $U \in S_{r-1}$ and $V \in S_{r-2}$ that has a common immediate ancestor $T \in S_{r-3}$ in $G$ such that $U \bigcap V \in S_{r-3}$ and $W = U \bigcup V$.*

**Proof:** Let $u$ and $v$ and $p$ be the three elements of $W$ that have the largest, the second-largest and the third-largest subscripts, respectively. Consider the sets $U = W - \{u\}$ and $V = W - \{v, p\}$. Note that $U \in S_{r-1}$ and $V \in S_{r-2}$. Moreover, $Z = U \bigcup V$ belongs to $S_{r-3}$ because it consists of the first $r$-3 elements of $W$. Note that both $U$ and $V$ are descendants of $Z$ and that $U \bigcup V = W$(for $r$=3 we have $Z = \emptyset$).

The pair $(U, V)$ is unique. Indeed, suppose that $W$ can be obtained in the same manner from another pair of distinct sets $U_1 \in S_{r-1}$ and $V_1 \in S_{r-2}$ such that $U_1$ and $V_1$ are immediate descendants of a set $Z_1 \in S_{r-3}$. The definition of the Rymon tree $G_I$ implies that $U_1 = Z_1 \bigcup \{i_m, i_q\}$ and $V_1 = Z_1 \bigcup \{i_y\}$, where the letters in $Z_1$ are indexed by a number smaller than $min\{m, q, y\}$. Then, $Z_1$ consists of the first $r$-3 symbols of $W$, so $Z_1 = Z$. If $m < q < y$, then $m$ is the third-highest index of a symbol in $W$, $q$ is the second-highest index of a symbol in $W$ and $y$ is the highest index of a symbol in $W$, so $U_1 = U$ and $V_1 = V$. □

The following theorem, together with the obvious fact $C^{\mu}_{T,k} \subset F^{\mu}_{T,k}$ for all $k$, directly proves correctness of our method $C_{T,k} = C^{\mu}_{T,k-1} \times C^{\mu}_{T,k-2}$. It is modification of Theorem 7.10. from [10].

**Theorem 10.** *Let $T$ be a transaction data set on a set of items $I$ and let $k \in N$ such that $k > 2$. If $W$ is a $\mu$-frequent itemset and $|W| = k$, then there exists a $\mu$-frequent itemset $Z$ and two itemsets $\{i_m, i_q\}$ and $\{i_y\}$ such that $|Z| = k-3$, $Z \subseteq W$, $W = Z \bigcup \{i_m, i_q, i_y\}$ and both $Z \bigcup \{i_m, i_q\}$ and $Z \bigcup \{i_y\}$ are $\mu$-frequent itemsets.*

**Proof:** If $W$ is an itemset such that $|W| = k$, than we already know that $W$ is the union of two subsets $U$ and $V$ of $I$ such that $|U| = k-1$, $|V| = k-2$ and that $Z = U \bigcap V$ has $k$-3 elements (it follows from Theorem 2). Since $W$ is a $\mu$-frequent itemset and $Z$, $U$, $V$ are subsets of $W$, it follows that each of these sets is also a $\mu$-frequent itemset (it follows from Theorem 1). □

Apriori algorithm [2] generates candidate $k$-itemset by joining two large ($k$-1)-itemsets, if and only if they have first ($k$-2) items in common. Because of that, each join operation requires ($k$-2) equality comparisons. If a candidate $k$-itemset is generated by the method $C_{T,k} = C^{\mu}_{T,k-1} \times C^{\mu}_{T,k-2}$ for $k \geq 3$, it is enough to process ($k$-3) equality comparisons.

The method $C_{T,k} = C^{\mu}_{T,k-1} \times C^{\mu}_{T,k-2}$ can be represented by the following SQL query:

```
INSERT INTO $C_{T,k}$
SELECT $R_1.item_1,...,R_1.item_{k-1},R_2.item_{k-2}$
FROM $C^{\mu}_{T,k-1}$ AS $R_1$, $C^{\mu}_{T,k-2}$ AS $R_2$
WHERE $R_1.item_1 = R_2.item_1 \wedge ... \wedge R_1.item_{k-3} = R_2.item_{k-3} \wedge R_1.item_{k-1} < R_2.item_{k-2}$
```

For the implementation of the join $C_{T,k} = C^{\mu}_{T,k-1} \times C^{\mu}_{T,k-2}$ we suggest a modification of sort-merge-join algorithm (note that $C^{\mu}_{T,k-1}$ and $C^{\mu}_{T,k-2}$ are sorted because of the way they are constructed and lexicographic order of itemsets).

By the original sort-merge-join algorithm [9], it is possible to compute natural joins and equi-joins. Let $r(R)$ and $s(S)$ be the relations and $R \bigcap S$ denote their common attributes. The algorithm keeps one pointer on the current position in relation $r(R)$ and another one pointer on the current position in relation $s(S)$. As the algorithm proceeds, the pointers move through the relations. It is supposed that the relations are sorted according to joining attributes, so tuples with the same values on the joining attributes are in consecutive order. Thereby, each tuple needs to be read only once, and, as a result, each relation is also read only once.

The number of blocks transfers is equal to the sum of the number of blocks in both sets $C^{\mu}_{T,k-1}$ and $C^{\mu}_{T,k-2}$, $n_{b1} + n_{b2}$.

The modification of sort-merge-join algorithm we suggest refers to the elimination of restrictions that join must be natural or equi-join. First, we separate the condition (3):

$$x_i = y_i, 1 \leq i \leq k-3 \tag{4}$$

$$x_{k-1} < y_{k-2}. \tag{5}$$

Joining $C_{T,k} = C^{\mu}_{T,k-1} \times C^{\mu}_{T,k-2}$ is calculated according to the condition (4), in other words we compute natural join. For this, the described sort-merge-join algorithm is used, and our modification is: before $X = \{x_1,...,x_{k-1}\}$ and $Y = \{y_1,...,y_{k-2}\}$, for which $X \in C^{\mu}_{T,k-1}$ and $Y \in C^{\mu}_{T,k-2}$ and $x_i = y_i, 1 \leq i \leq k-3$ is true, are joined, we check if condition (5) is satisfied, and after that we generate candidate $k$-itemset $\{x_1,...,x_{k-2},x_{k-1},y_{k-2}\}$.

The pseudocode of **apriori_gen** function comes next.

```
FUNCTION apriori_gen($C^{\mu}_{T,k-1}$,$C^{\mu}_{T,k-2}$)
    1.  i = 0
    2.  j = 0
    3.  while $i \leq |C^{\mu}_{T,k-1}| \wedge j \leq |C^{\mu}_{T,k-2}|$
          $iset_1 = C^{\mu}_{T,k-1}[i++]$
          $S = \{iset_1\}$
          done = false
          while $done = false \wedge i \leq C^{\mu}_{T,k-1}$
```

```
            $iset_{1a} = C^{\mu}_{T,k-1}[i++]$
            if $iset_{1a}[w] = iset_1[w], 1 \leq w \leq k-2$ then
               $S = S \bigcup \{iset_{1a}\}$
               $i++$
            else
               done = true
            end if
         end while
         $iset_2 = C^{\mu}_{T,k-2}[j]$
         while $j \leq |C^{\mu}_{T,k-2}| \wedge iset_2[1,...,k-2] \prec iset_1[1,...,k-2]$
            $iset_2 = C^{\mu}_{T,k-2}[j++]$
         end while
         while $j \leq |C^{\mu}_{T,k-2}| \wedge iset_1[w] = iset_2[w], 1 \leq w \leq k-2$
            for each $s \in S$
               if $iset_1[k-1] \prec iset_2[k-2]$ then
                  $c = \{iset_1[1],...,iset_1[k-1], iset_2[k-2]\}$
                  if c contains-not-frequent-subset then
                     DELETE c
                  else
                     $C_{T,k} = C_{T,k} \bigcup \{c\}$
                  end if
            end for
            j++
            $iset_2 = C^{\mu}_{T,k-2}[j]$
         end while
      end while
```

## 5 Statistical Test for Validating Frequent Itemset

Frequent itemset mining algorithms have the potential to generate a large number of patterns. For example, even if we assume that no customer has more than five items in his shopping cart and that there are 10000 items, there are $\sum_{i=1}^{5} \binom{10000}{5}$ possible contents of this cart, which corresponds to the subsets having no more than five items of a set that has 10,000 items, and this is indeed a large number. As the size and dimensionality of real commercial databases can be very large, we could easily end up with thousands or even millions of patterns, many of which might not be interesting. It is therefore important to establish a set of well-accepted criteria for evaluating the quality of patterns.

In Apriori multiple_num algorithm support measure is used to determine whether an itemset is frequent: an itemset $X$ is considered frequent in the data set $T$, if $supp_T(X) > minsup$, where *minsup* is a user-specified threshold. Support measure is kind of *objective interestingness measure*, which is data-driven and domain-independent approach that uses statistics derived from data for evaluating the quality of association patterns [12].

Now, we will explain how statistical hypothesis testing can be applied to validate frequent itemsets generated with support measure.

Hypothesis testing is a statistical inference procedure to determine whether a hypothesis should be accepted or rejected based on the evidence gathered from data. Examples of hypothesis tests include verifying the quality of patterns extracted by many data mining algorithms and validating the significance of the performance difference between two classification models.

In hypothesis testing, we are usually presented with two opposite hypothesis, which are known, respectively, as the null hypothesis and the alternative hypothesis. The general procedure for hypothesis testing consists of the following four steps [12]:

- Formulate the null and alternative hypotheses to be tested.
- Define a test statistic $\theta$ that determines whether the null hypothesis should be accepted or rejected. The probability distribution associated with the test statistic should be known.
- Compute the value of $\theta$ from the observed data. Use the knowledge of the probability distribution to determine a quantity known as $p$-value.
- Define a significance level a which controls the range of $\theta$ values in which the null hypothesis should be rejected. The range of values for $\theta$ is known as the rejection region.

Frequent itemsets mining problem can be formulated into the hypothesis testing framework in the following way. To validate if the itemset $X$ is frequent in the data set $T$, we need to decide whether to accept the null hypothesis, $H_0 : supp_T(X) = minsup$, or the alternative hypothesis $H_1 : supp_T(X) > minsup$. If the null hypothesis is rejected, then $X$ is considered as frequent itemset. To perform the test, the probability distribution for $supp_T(X)$ must also be known.

**Theorem 11.** *The measure $supp_T(X)$ for the itemset X in transaction data set T has the binomial distribution with mean $supp_T(X)$ and variance $\frac{supp_T(X)*(1-supp_T(X))}{n}$, where n is the number of transactions in T.*

**Proof:** We will use measure $suppcount_T(X)$ and calculate mean and variance for it and later derive mean and variance for the measure $supp_T(X)$. The measure $suppcount_T(X) = X_n$ presents the number of transactions in $T$ that contain itemset $X$, and $supp_T(X) = suppcount_T(X)/n$ (Definition 2).

The measure $X_n$ is analogous to determining the number of heads that shows up when tossing $n$ coins. Let us calculate $E(X_n)$ and $D(X_n)$.

Mean is $E(X_n) = n * p$, where $p$ is the probability of success, which means (in our case) the itemset $X$ appears in one transaction. According to Bernoulli low, the following holds:

$$\forall \varepsilon > 0, lim_{N\to\infty} P\{|\frac{X_n}{n} - p| \leq \varepsilon\} = 1. \tag{6}$$

Freely speaking, for large $n$ (we work with large databases so $n$ can be considered large), we can use relative frequency instead of probability. So, we now have:

$$E(X_n) = np \approx n\frac{X_n}{n} = X_n \tag{7}$$

For variance we compute:

$$D(X_n) = np(1-p) \approx n\frac{X_n}{n}(1-\frac{X_n}{n}) = \frac{X_n(n-X_n)}{n} \tag{8}$$

Now we will compute $E(supp_T(X))$ and $D(supp_T(X))$. Recall that $supp_T(X) = \frac{X_n}{n}$. We have:

$$E(supp_T(X)) = E(\frac{X_n}{n}) = \frac{1}{n}E(X_n) = \frac{X_n}{n} = supp_T(X). \tag{9}$$

$$D(supp_T(X)) = D(\frac{X_n}{n}) = \frac{1}{n^2}D(X_n) = \frac{1}{n^2}\frac{X_n(n-X_n)}{n} = \frac{1}{n}supp_T(X)(1-supp_T(X)) \tag{10}$$

□

The binomial distribution can be further approximated using normal distribution if $n$ is sufficiently large, which is typically the case in association analysis.

Regarding previous paragraph and Theorem 4, under the null hypothesis $supp_T(X)$ is assumed to be normally distributed with mean *minsup* and variance $\frac{minsup(1-minsup)}{n}$. To test whether the null hypothesis should be accepted or rejected, the following statistic can be used:

$$W_N = \frac{supp_T(X) - minsup}{\sqrt{\frac{minsup(1-minsup)}{n}}}. \tag{11}$$

The previous statistic, according to the Central Limit Theorem, has the distribution $N(0,1)$. The statistic essentially measures the difference between the observed support $supp_T(X)$ and the *minsup* threshold in units of standard deviation.

Let N=10000, $supp_T(X) = 0.11$, *minsup*=0.1 and $\alpha = 0.001$. The last parameter is the desired significance level. It controls Type 1 error which is rejecting the null hypothesis even though the hypothesis is true.

In the Apriori algorithm we compare $supp_T(X) = 0.11 > 0.1 = minsup$ and we declare $X$ as frequent itemset. Is this validation procedure statistically correct?

Under the hypothesis $H_1$ statistics $W_{10000}$ is positive and for rejection region we choose
$R = \{(x_1, ..., x_{10000}) | w_{10000} > k\}, k > 0$.

Let us find $k$.

$0.001 = P_{H_0}\{W_{10000} > k\}$
$P_{H_0}\{W_{10000} > k\} = 0.499$
$k = 3.09$

Now we compute $w_{10000} = \frac{0.11-0.1}{\sqrt{\frac{0.1*(1-0.1)}{10000}}} = 3.33\cdots$

We can see that $w_{10000} > k$, so we are in rejection region and $H_1$ is accepted, which means the itemset $X$ is considered statistically significant.

## 6 Conclusion

In this section we compare the proposed method with original Apriori [1] and with Apriori Multiple which we introduced in [11].

Sections 3 and 4 of the paper contain comparison with the original Apriori algorithm [1]. The main advantages of the new algorithm are: it finishes in just two database scans and it uses more efficient candidate generation procedure.

The algorithm from [11] also finishes in two database scans and it uses similar procedure for candidate generation as the Apriori multiple_num algorithm proposed here. But, the Apriori multiple_num algorithm is more efficient. The main advantage of the Apriori multiple_num algorithm in comparison with algorithm from [11] is in the following. The Apriori multiple_num uses Rymon tree structure for definition of candidate join procedure as it is explained in Section 4. Because of that, candidate sets are stored in Rymon tree structure before joining instead of storing candidates in array as it is done in [11]. The following experiment confirms that Rymon tree based implementation is more efficient.

We implemented the Apriori multiple_num, the original Apriori [1] and the Apriori Multiple [11] algorithms in C in order to evaluate its performances. Experiments are performed on PC with a CPU Intel(R) Core(TM)2 clock rate of 2.66GHz and with 2GB of RAM. Also, run time used here means the total execution time, i.e., the period between input and output instead of CPU time measured in the experiments in some literature. In experiments dataset which can be found on www.cs.uregina.ca is used. It contains 10000 binary transactions. The average length of transactions is 8.

We did not compare number of I/O operations because the algorithm proposed here finishes in just two database scans, while the original Apriori requires at least $k_{max}+1$ scans, where $k_{max}$ is the length of the longest frequent itemset (as explained in Section 3).

Figure 1 shows that the original Apriori algorithm from [1] is outperformed by both the Apriori Multiple [11] and the Apriori multiple_num presented here. Also, it can be seen that Apriori multiple_num with Rymon tree based implementation is significantly better than the algorithm from [11].

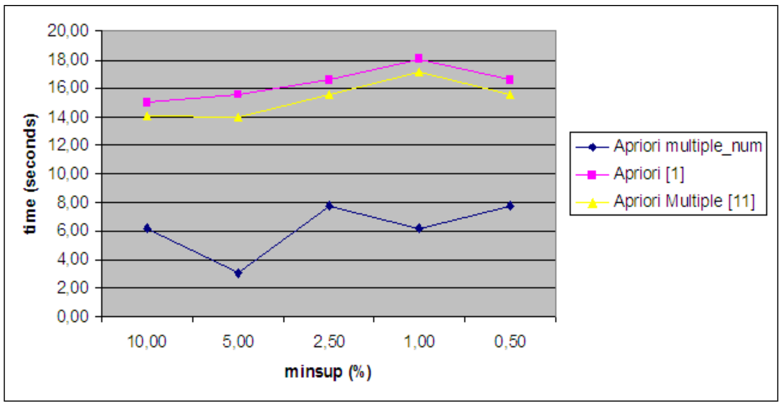

Figure 2: Execution times for different algorithms

## Bibliography

[1] Agrawal, R., Srikant, R., Fast Algorithms for Mining Association Rules, Proceedings of VLDB-94, 487-499, Santiago, Chile (1994)

[2] Coenen, F.P., Leng, P., Ahmed, S., T-Trees, Vertical Partitioning and Distributed Association Rule Mining, Proceedings ICDM-2003, 513-516 (2003)

[3] Coenen, F.P., Leng, P., Ahmed, S., Data Structures for Association Rule Mining: T-trees and P-trees, IEEE Transactions on Data and Knowledge Engineering, Vol. 16, No 6, 774-778 (2004)

[4] Coenen, F.P., Leng, P., Goulbourne, G., Tree Structures for Mining Association Rules, Journal of Data Mining and Knowledge Discovery Vol. 8, No. 1, 25-51 (2004)

[5] Goulbourne, G., Coenen, F., Leng, P., Algorithms for Computing Association Rules Using a Partial-Support Tree, Journal of Knowledge-Based Systems Vol. 13, 141-149 (1999)

[6] Grahne, G., Zhu, J., Efficiently Using Prefix-trees in Mining Frequent Itemsets, Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations (2003)

[7] Han, J., Pei, J., Yu, P.S., Mining Frequent Patterns without Candidate Generation, Proceedings of the ACM SIGMOD Conference on Management of Data, 1-12 (2000)

[8] Rymon, R., Search Through Systematic Set Enumeration, Proceedings of 3rd International Conference on Principles of Knowledge Representation and Reasoning, 539-550 (1992)

[9] Silberschatz, A., Korth, H. F., Sudarshan, S., Database System Concepts, Mc Graw Hill, New York (2006)

[10] Simovici, A. D., Djeraba, C., Mathematical Tools for Data Mining (Set Theory, Partial Orders, Combinatorics), Springer-Verlag London Limited (2008)

[11] Stanisic, P., Tomovic, S., Apriori Multiple Algorithm for Mining Association Rules, Information Technology and Control Vol. 37, No. 4, 311-320 (2008)

[12] Tan., P.N., Steinbach, M., Kumar, V., Introduction to Data Mining, Addicon Wesley (2006).

**Dr. Predrag Stanisic** is a professor in the Faculty of Science - Department of Mathematics and Computer Science at University of Montenegro. He received his B.Sc. degree in Mathematics and Computer Science from University of Montenegro in 1996, his M.Sc degree in Computer Science from University of Belgrade Serbia in 1998 and his Ph.D. degree in Computer Science from Moscow State University M.V. Lomonosov in 1999. He is currently the dean of Faculty of Science at University of Montenegro and he teaches a wide variety of undergraduate and graduate courses in several computer science disciplines, especially database systems, operating systems and programming.

**M.Sc Savo Tomovic** is a teaching assistant in the Faculty of Science - Department of Mathematics and Computer Science at University of Montenegro. He received his B.Sc. degree in Mathematics and Computer Science from University of Montenegro in 2006, his M.Sc degree in Computer Science from University of Montenegro in 2007. He is currently a Ph.D. student in Computer Science at University of Montenegro.