The Item-Set Tree: A Data Structure for Data Mining*

Alaaeldin Hafez[1], Jitender Deogun[2], and Vijay V. Raghavan[1]

Abstract. Enhancements in data capturing technology have lead to exponential growth in amounts of data being stored in information systems. This growth in turn has motivated researchers to seek new techniques for extraction of knowledge implicit or hidden in the data. In this paper, we motivate the need for an incremental data mining approach based on data structure called the item-set tree. The motivated approach is shown to be effective for solving problems related to efficiency of handling data updates, accuracy of data mining results, processing input transactions, and answering user queries. We present efficient algorithms to insert transactions into the item-set tree and to count frequencies of itemsets for queries about strength of association among items. We prove that the expected complexity of inserting a transaction is ≈ O(1), and that of frequency counting is O(n), where n is the cardinality of the domain of items.

1 Introduction

Association mining that discovers dependencies among values of an attribute was introduced by Agrawal et al.[1] and has emerged as a prominent research area. The association mining problem also referred to as the *market basket* problem can be formally defined as follows. Let $I = \{i_1, i_2, \ldots, i_n\}$ be a set of items as $S = \{s_1, s_2, \ldots, s_m\}$ be a set of transactions, where each transaction $s_i \in S$ is a set of items that is $s_i \subseteq I$. An *association rule* denoted by $X \Rightarrow Y$, where $X, Y \subset I$ and $X \cap Y = \Phi$, describes the existence of a relationship between the two itemsets X and Y.

Several measures have been introduced to define the *strength* of the relationship between itemsets X and Y such as support, confidence, and interest. The definitions of these measures, from a probabilistic model are given below.

- **I.** *Support* $(X \Rightarrow Y) = P(X, Y)$, or the percentage of transactions in the database that contain both X and Y.
- **II.** *Confidence* $(X \Rightarrow Y) = P(X, Y) / P(X)$, or the percentage of transactions containing Y in transactions those contain X.
- **III.** *Interest(* $X \Rightarrow Y) = P(X, Y) / P(X)P(Y)$ represents a test of statistical independence.

* This research was supported in part by the U.S. Department of Energy, Grant No. DE-FG02-97ER1220, and by the Army Research Office, Grant No. DAAH04-96-1-0325, under DEPSCoR program of Advanced Research Projects Agency, Department of Defense.

[1] ahafez(raghavan)@cacs.usl.edu, The Center for Advanced Computer Studies, University of SW Louisiana, Lafayette, LA 70504, USA.

[2] Deogun@cse.unl.edu, The Department of Computer Science, University of Nebraska, Lincoln, NE 68588, USA.

Many algorithms [1,2,3,4,5,6,7,8], have been proposed to generate association rules that satisfy certain measures. A close examination of those algorithms reveals that the spectrum of techniques that generate association rules, has two extremes:

- A transaction data file is repeatedly scanned to generate large itemsets. The scanning process stops when there are no more itemsets to be generated.
- A transaction data file is scanned only once to build a complete transaction lattice. Each node on the lattice represents a possible large itemset. A count is attached to each node to reflect the frequency of itemsets represented by nodes.

In the first case, since the transaction data file is traversed many times, the cost of generating large itemsets is high. In the later case, while the transaction data file is traversed only once, the maximum number of nodes in the transaction lattice is 2^n , n is the cardinality of *I*, the set of items. Maintaining such a structure is expensive.

Many knowledge discovery applications, such as on-line services and world wide web, require accurate mining information from data that changes on a regular basis. In world wide web, every day hundreds of remote sites are created and removed. In such an environment, frequent or occasional updates may change the status of some rules discovered earlier. Also, many data mining applications deal with itemsets that may not satisfy data mining rules. Users could be interested in finding correlation between itemsets, not necessarily satisfying the measures of the data mining rules.

Discovering knowledge is an expensive operation. It requires extensive access of secondary storage that can become a bottleneck for efficient processing. Running data mining algorithms from scratch, each time there is a change in data, is obviously not an efficient strategy. Building a structure to maintain knowledge discovered could solve many problems, that have faced data mining techniques for years, that is *database updates*, *accuracy of data mining results*, *performance*, and *ad-hoc queries*.

In this paper, we propose a new approach, that represents a compromise between the two extremes of the association mining spectrum. In the context of the proposed approach two algorithms are introduced. The first algorithm builds an item-set tree by traversing the data file once, that is used to produce mining rules. While the second algorithm allows users to apply on-line ad hoc queries on the item-set tree.

The item-set tree approach is introduced in section 2. In section 3, counting frequencies of itemsets is given. The item-set tree approach is evaluated and the paper is concluded in section 4.

2 The Item-Set Tree

The item-set tree *T* is a graphical representation of the transaction data file *F*. Each node $s \in T$ represents a transaction group *s*. All transactions that are having the same itemset, belong to the same transaction group. Let $I=\{i_1, i_2,...,i_n\}$ be an ordered set of items. For two transactions $s_i=\{a_1, a_2,...,a_l\}$ and $s_j=\{b_1, b_2,...,b_k\}$, let $s_i \leq s_j$ iff $a_p \leq b_p$ for all $1 \leq p \leq min(l,k)$. We call *l* and *k*, the lengths of s_i and s_j, respectively.

Each node in tree *T* represents either an encountered transaction, i.e., a transaction in the transaction file, or a subset of an encountered transaction. Node s_i is ancestor node of node s_j, if $s_i \subset^e s_j$ that is $s_i=\{a_1, a_2,...,a_l\}$ and $s_j=\{a_1, a_2,...,a_k\}$, for some $l<k$. Moreover a node s_i direct ancestor of node s_j if s_i is an ancestor of s_j and

there is no other node s_k such that $s_i \subset^e s_k \subset^e s_j$. Frequency of a node s is denoted by *f(s)* representing the count of transactions that have the same transaction group *s*. The item-set tree is constructed by transactions inserting process: *The root node r represents the null itemset {}. A transaction s is inserted by examining (in order) the children of the root node r. Each time a node is inserted , f(r) is incremented by 1*. The insertion process successfully ends with one of the following cases.

Case 1: All nodes s_j (children of r) are such that these do share no leading elements in *s*. When a leaf node *s* is inserted as a son of *r*, *f(s)* is initiated to 1.
Case 2: $s=s_j$, the node already exists. $f(s_j)$ is incremented by 1.
Case 3: $s \subset^e s_j$, *s* is an ordered subset of node s_j. A node *s*, representing *s*, is inserted as a child of *r* and as a parent of s_j. $f(s) = f(s_j) + 1$.
Case 4: $s_j \subset^e s$, node s_j is an ordered subset of *s*. The subtree, that has s_j as a root, is examined and the procedure starts over again
Case 5: $s \cap^e s_j \neq \phi$,there exists an ordered intersection between *s* and s_j. Two nodes are inserted. A node s_i, $s_i = s \cap^e s_j$, is inserted between *r* and s_j, and a node *s* is inserted as a child of s_i. $f(s_i) = f(s_j)+1$, and *f(s)* is initiated to 1.

Algorithm Construct (s,T)
s is an input itemset
T is the itemset tree
begin
r=*root*(T)
increase *f(r)*
if s = *items (r)* **then exit**
choose T_s=*subtree*(r) such that s and *items*(*root*(T_s)) are comparable
if T_s does not exist **then**
create a new son *x* for *r*, *items(x)* =s and *f(x)* =1
else if *root*(T_s) $\subset^e$ s **then** call **Construct (s, T_s)**
else if s $\subset^e$ *root*(T_s) **then**
create a new node *x,* as a son of *r* and a father of *root*(T_s),
items(x) =s and *f(x)* = *f*(*root*(T_s))+1
else create two nodes *x* and *y*, *x* as the father of *root*(T_s),s.t.*items(x)* = s $\cap^e$ *root*(T_s), *f(x)* = *f*(*root*(T_s))+1, and *y* as a son of *x*, s.t., *items(y)* = s , *f(y)* = 1
end

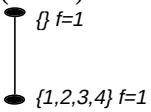
Figure 1: Algorithm Construct

Example 1: Let *I={1,2,3,4}* and *F={{1,2,3,4}, {1,2}, {1,3}, {2,3}}* be a transaction file that has 4 transactions. In this example, we assume that all transaction in the transaction file *F* have occurred only once. The item-set tree T is fully constructed in 4 steps (for the 4 transactions). Various steps of the solution are shown in Figures 2.

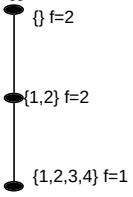
Inserting all transaction of the transaction data file F, using algorithm Construct(s,T), requires scanning file F only once. An important characteristic of the Construct(s,T) algorithm, is that, no matter what the sequence of the inserted transactions is, the item-set tree T is always the same.

In sections 4.1 and 4.2, we study the performance of algorithm Construct(S,T).

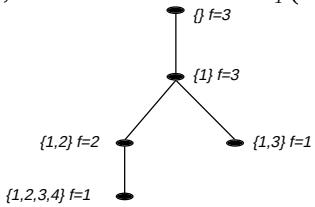
Step1: *s={1,2,3,4}*, *s* is added as a child of *{}* (case 1).



Step2: *s={1,2}*, *s* is added as a child of *{}* and as a father of *{1,2,3,4}* (case 3).



Step3: *s={1,3}*, s_1*={1}* (s_1*={1,2}*$\cap^e$*{1,3})* is added as a child of {} and as a father of *{1,2}*, *s* is added as a child of s_1 (case 5).



Step4: *s={2,3}*, *s* is added as a child of *{}* (case 1).

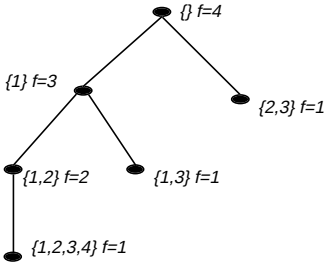


Figure 2: Steps 1 and 4 of example 1.

3 Frequency Counting

In order to answer ad hoc queries, we introduce algorithm Count. Algorithm Count calculates the frequency of an itemset *s* by adding up frequencies of those encountered itemsets, that contain *s*. In the example 2, we demonstrate how to count frequencies of itemsets. Algorithm Count is given in Figure 3.

```
Algorithm Count(s,T)
input:    An item set  s, and an item-set tree T.
Output:Frequency f of item set s.
begin
        r=root(T)
        if s⊆r then f(s)=f(s)+f(r) ; end
        while r<s and last-item(r)<last-item(s) do
                traverse subtrees; T' ,of r
                call Count(s,T')
        enddo
end
```

Figure 3: Algorithm Count

Example 2: Let *T* be the item-set tree constructed in example 1, and *s={2,3}* be the itemset to be counted. To count the frequency of itemset *s*, the item-set tree *T* is traversed in order as shown in the following steps,

- Start from the smallest subtree with root node {1}. In this case, s>{1}& s⊄{1}.

- The subtree of {1} is orderly traversed; starting with node {1,2}. s>{1,2}&s $\not\subset$ {1,2}
- The subtree of {1,2} is orderly traversed; starting with node {1,2,3,4}. s $\subset${1,2,3,4}, f=1.
- Go back to next-subtree of {1}, node {1,3}. s $\not\subset$ {1,3}, and the last element in {1,2,4} equals the last element in s. No further traversing through this subtree.
- Go back to next-subtree of {}, node {2,3}. s equals {2,3}. f=1+1, and no further traversing through this subtree. The procedure ends with f({2,3})= 2.

4 Performance Results

In this section, we study the performance of algorithms Construct and Count. We assume that, items are uniformly distributed over all transactions. In section 4.1, we give the expected number of nodes in the item-set tree T after inserting N transactions. In sections 4.2 and 4.3, the expected number of iterations to insert a transaction, and the expected number of iterations to count the frequency of an itemset, respectively, are given. In section 4.4, we discuss the results of our analytical study.

4.1 Number of Nodes in The Item-set tree

Lemma 1. *Given an ordered set $I=\{i_1,i_2, \ldots, i_n\}$, of n items, and a set of transaction nodes $V_k \in T$, $1\le k \le K$, $1\le K \le 2^n-1$, $V_k=\{a_1,a_2, \ldots, a_l\}$, $a_1<a_2< \ldots < a_l$, and items $a_i \in I$, $1\le i \le l$, $1\le l \le n$ which are uniformly distributed over itemset domain I, and an itemset $s_j=\{b_1,b_2, \ldots, b_r\}$ with items $b_1<b_2< \ldots < b_r$, $b_i \in I$, $1\le i \le r$, $1\le r \le n$ which are uniformly distributed over itemset domain I Algorithm* **Construct**, *produces an item-set tree T, with expected number of nodes K such that*

$$K \le N(1+\tfrac{1}{48}) - \tfrac{N}{16}((n-1)(\tfrac{1}{2})^n + \tfrac{1}{3}(\tfrac{1}{2})^{2n-2})$$

where N is number of inserted transaction.

Proof. Before proving lemma 1, we first state and prove the following lemma. The following lemma makes the proof easier to describe.

Lemma 2. *Given an ordered set $I=\{i_1,i_2, \ldots, i_n\}$, of n items, and a set of transaction nodes $V_k \in T$, $1\le k \le K$, $1\le K \le 2^n-1$, $V_k=\{a_1,a_2, \ldots, a_l\}$, $a_1<a_2< \ldots < a_l$, and items $a_i \in I$, $1\le i \le l$, $1\le l \le n$ are uniformly distributed over itemset domain I. Let $s_j=\{b_1,b_2, \ldots, b_r\}$ be an itemset with items $b_1<b_2< \ldots < b_r$, $b_i \in I$, $1\le i \le r$, $1\le r \le n$ which are uniformly distributed over items domain I. Given that s_j is not an empty itemset, the probability that there exist a node $V_k \in T$ such that the order intersection of s_j and V_k equals an item set Z, where $Z\neq\Phi$, $Z\neq s_j$, and $Z\neq V_k$, is*

$$P(S_j \cap^e V_k = Z,\ \ Z \neq V_k, Z \notin T) = \left(\frac{\frac{1}{3}-(n-1)(\frac{1}{2})^n-\frac{1}{3}(\frac{1}{2})^{2n-2}}{((1-(\frac{1}{2})^n)^2}\right) * \left(1-\frac{K}{2^n-1}\right) * \left(\frac{K}{2^n-1}\right)$$

Proof. First we state the assumptions:

- A transaction group (node) V_k is in T with probability $P(V_k \in T) = \frac{K}{2^n-1}$, where K is the number of nodes in T.

- A transaction s_j and a transaction group V_k are each represented as a set of 1's and 0's, where 0 in position i means item $a_i \in I$ does not exist, and 1 in position i means item $a_i \in I$ does exist.
- Both V_k and s_j are not empty itemsets, i.e., the probability is conditioned, the probability of both V_k and s_j are not empty itemsets, is

$$P(V_k \neq \Phi \quad and \quad S_j \neq \Phi) = P(V_k \neq \Phi) * P(S_j \neq \Phi) = (1-(\tfrac{1}{2})^n)^2$$

- The item-set tree T has already K nodes, and each node either represents a transactions group or an ordered intersection of two transactions groups.
- All K nodes in T are distinct, i.e., $V_k \neq V_l$ for all nodes k,l *in T*.
- both V_k and s_j are not empty itemsets,

We use the following table to demonstrate all the requirements needed ,

	Shared items	X		OR		
s_j	At least 1	1	0's or 1's		0	At least 1
V_k	At least 1	0	At least 1		1	0's or 1's

The following formula gives the required probability,

$$\sum_{x=2}^{x=n-1} ((\tfrac{1}{2}*\tfrac{1}{2}+\tfrac{1}{2}*\tfrac{1}{2})^{(x-1)} - (\tfrac{1}{2}*\tfrac{1}{2})^{(x-1)}) * (\tfrac{1}{2}*\tfrac{1}{2}+\tfrac{1}{2}*\tfrac{1}{2}) * (1-(\tfrac{1}{2})^{(n-x)})$$

which could be written as

$$\tfrac{1}{3}-(n-1)*(\tfrac{1}{2})^n - \tfrac{1}{3}(\tfrac{1}{2})^{2n-2}$$

Since we assume that both V_k and s_j are not empty itemsets, the above formula should be divided by the probability of both V_k and s_j are not empty itemsets. Also, it should multiplied by $P(V_k \in T) = \frac{K}{2^n-1}$ & $P(Z \notin T) = 1 - \frac{K}{2^n-1}$. Now, the complete formula could be written as

$$(\frac{\frac{1}{3}-(n-1)(\frac{1}{2})^n - \frac{1}{3}(\frac{1}{2})^{2n-2}}{((1-(\frac{1}{2})^n)^2}) * (1-\frac{K}{2^n-1}) * (\frac{K}{2^n-1})$$

Proof of Lemma 1. We use the same assumptions given in the proof of Lemma 2. For each new encountered transaction group, algorithm **Construct** inserts either 1 node or 2 nodes. So, the cost function should equal to $\sum_{s_j=s_1}^{s_j=s_N}[1+P(s_j \quad inserts \ 2 \quad nodes)]$.

To insert two nodes in T, the following conditions must be satisfied; $\exists$ node $V_k \in T$ such that, $S_j \cap^e V_k = Z$, $Z \neq \Phi$, $Z \neq V_k$, $Z \neq S_j$, $V_k \in T$, and $Z \notin T$.

By using Lemma 2,

$$P(s_j \quad inserts \ 2 \quad nodes) = P(S_j \cap^e V_k = Z \quad and \quad Z \neq \Phi, Z \neq S_j, Z \neq V_k, Z \notin T)$$

Or

$$Expected \quad number \quad of \quad nodes = \sum_{K=1}^{k=N}[1+(\frac{\frac{1}{3}-(n-1)(\frac{1}{2})^n - \frac{1}{3}(\frac{1}{2})^{2n-2}}{((1-(\frac{1}{2})^n)^2}) * (1-\frac{k}{2^n-1}) * (\frac{k}{2^n-1})]$$

In the above formula, the following inequality, is always true for $n \geq 1$,

$$(1 - (\tfrac{1}{2})^n)^2 \geq \tfrac{1}{4}$$

Also, since values of k could have any number between 1 and 2^n-1, which means the following inequality always hold

$$(1 - \tfrac{k}{2^n - 1}) * (\tfrac{k}{2^n - 1}) \leq \tfrac{1}{4}$$

Using the above two inequalities, the upper bound of the expected number of nodes K in an item-set tree with N transactions is

$$K \leq N(1 + \tfrac{1}{48}) - \tfrac{N}{16}((n-1)(\tfrac{1}{2})^n + \tfrac{1}{3}(\tfrac{1}{2})^{2n-2})$$

4.2 Number of Iterations to Insert a Transaction

Lemma 3: *Given an ordered set $I=\{i_1,i_2, \ldots, i_n\}$, of n items, and a set of transaction nodes $V_k \in T$, $1 \leq k \leq K$, $1 \leq K \leq 2^n$-1, $V_k = \{a_1,a_2, \ldots, a_l\}$, $a_1 < a_2 < \ldots < a_l$, and items $a_i \in I$, $1 \leq i \leq l$, $1 \leq l \leq n$ are uniformly distributed over itemset domain I. Let $s_j=\{b_1,b_2, \ldots, b_r\}$ be an itemset with items $b_1 < b_2 < \ldots < b_r$, $b_i \in I$, $1 \leq i \leq r$, $1 \leq r \leq n$ which are uniformly distributed over items domain I. Given that all $V_k \in T$ and s_j are not empty itemsets, the expected number of iterations algorithm* Construct *takes to enter a transaction into the item-set tree T is less than*

$$1 + n((n-2) * 2^{n-1} + 1) * \frac{K}{(2^n - 1)^3}$$

where K is the number of nodes in T.

Proof. In order to insert a transaction s_j, with length l, in exactly one iteration, i.e., first level in the item-set tree T, there are two cases. First case, there exits a node $V_k \in T$ in first level of T, such that $V_k = S_j$, while the second case, where neither s_j nor all ordered subset nodes of s_j, are in T. In other words, $V_k \notin T$ for all $V_k \subseteq^e S_j$. Let $P_\Phi = P(V_k \notin T, V_k \subseteq^e S_j)$, and $P_e = P(V_k \in T, S_j = V_k)$. The cost of inserting such transaction is less than

$$1 * (P_e + P_\Phi)$$

Now to insert a transaction s_j, with length l, in exactly two iterations, i.e., second level in the item-set tree T, there are two cases. First case, exactly one order subset of s_j, does exist in T, and there exits a node $V_k \in T$ in second level of T, such that $V_k = S_j$, while the second case, there exists exactly one order subset of s_j, and neither s_j nor all other ordered subset nodes of s_j, are in T. Let $P_s = P(V_k \in T, V_k \subset^e S_j)$. The cost of inserting such transaction is less than

$$2 * (C_1^{l-1}(P_s P_e + P_s P_\Phi))$$

Since the maximum number of iterations is l, the expected cost of inserting transaction s_j is $(P_e + P_\Phi)\sum_{i=1}^{l} i * (C_{i-1}^{l-1} P_s^{i-1})$ **(1)**

By following the same assumptions given in the proof of lemma 2, the expected value of P_s is $\left(\frac{(n-2)(\frac{1}{2})^{n+1}+(\frac{1}{2})^{2n}}{((1-(\frac{1}{2})^n)^2}\right) * \left(\frac{K}{2^n-1}\right)$

and, the expected value of $(P_\Phi + P_e)$ *is* $1-P_s$. Formula (1) could be written as

$$(1-P_s)*(P_s*l*(1+P_s)^{l-1}+(1+P_s)^l) \qquad \textbf{(2)}$$

Since $(1+P_s)^l = 1+lP_s+l(l-1)P_s^2+....,$

Formula (2) could be written as

$$(1-P_s)*(lP_s+l(l-1)P_s^2+l(l-1)(l-2)P_s^3+...+1+lP_s+l(l-1)P_s^2+l(l-1)(l-2)P_s^3+...$$

By ignoring higher terms, the above formula could be

$$1 + (2l - 1)P_s$$

Since *l* could go take any value between *1* and *n*, then the expected number of iterations is $\frac{1}{n}\sum_{l=1}^{l=n}(1+(2l-1)*(\frac{(n-2)(\frac{1}{2})^{n+1}+(\frac{1}{2})^{2n}}{((1-(\frac{1}{2})^n)^2})*(\frac{K}{2^n-1}))$

Which could be written as $1 + n((n-2)*2^{n-1}+1)*\frac{K}{(2^n-1)^3}$

4.3 Number of iterations to Count The Frequency of an Itemset

Lemma 4. *Given an ordered set I={i_1,i_2, . . . , i_n}, of n items, and a set of transaction nodes $V_k \in T$, $1 \le k \le K$, $1 \le K \le 2^n$-1, V_k ={a_1,a_2, . . . , a_l}, $a_1 < a_2 < \ldots < a_l$, and items $a_i \in I$, $1 \le i \le l$, $1 \le l \le n$ are uniformly distributed over itemset domain I. Let s_j={b_1,b_2, . . . , b_r} be an itemset with items $b_1 < b_2 < \ldots < b_r$, $b_i \in I$, $1 \le i \le r$, $1 \le r \le n$ which are uniformly distributed over items domain I. Given that all $V_k \in T$ and s_j are not empty itemsets, the expected number of iterations algorithm* Count *takes to count an itemset frequency in the item-set tree T, with K nodes is*

$$\frac{n-1}{2} + \left(\frac{1}{n}\right) * \left(\frac{\frac{4}{3}-(n+\frac{7}{3})(\frac{1}{2})^{n+1}-(\frac{1}{2})^{2n-2}-\frac{n(n-1)}{2}(\frac{1}{2})^{2n-1}}{(1-(\frac{1}{2})^n)^2}\right) * \left(\frac{K}{2^n-1}\right)$$

Proof. In order to count the frequency of an itemset s_j, with length *l*, where O_1 and O_l are orders of first element and last elements in s_j, i_1, $i_l \in s_j$, respectively, all itemsets S_j^p with first element has order O_1 , and last element has order O_k , which could have s_j as part of them should be checked. The number of such checks (or iterations) is $2^{O_l - O_1}$. The count stops when we reach the full set of s_j , we will call it S_j^f .

So, to count the frequency of itemset s_j in exactly one iteration, there should be a node $V_k \in T$ such that $S_j^f \subseteq^e V_k$, or, with unsuccessful count, when the first visited node $V_k \in T$ such that $S_j^f \not\subset^e V_k$ and $V_k \not\subset^e S_j^p$. Let $\tilde{P}_e = P(V_k \in T, V_k \subseteq^e S_j^f)$ and $\tilde{P}_\Phi = P(V_k \notin T, V_k \subset^e S_j^p)$. The cost of counting such transaction is $1 * (\tilde{P}_e + \tilde{P}_\Phi)$

Generally speaking, to count the frequency of an itemset s_j with length l, where O_1 and O_l are orders of first element and last elements in s_j, i_1, $i_l \in s_j$, respectively in exactly i iteration, The cost of counting is

$$i*(C_{i-1}^{2^{O_l-O_1}}\tilde{P}_e\tilde{P}_s^{\,i-1}+C_{i-1}^{2^{O_l-O_1}}\tilde{P}_s^{\,i-1}\tilde{P}_\Phi)$$

Since the maximum number of iterations is $2^{O_l-O_1}$, the expected cost of counting frequency of itemset s_j is

$$(\tilde{P}_e+\tilde{P}_\Phi)\sum_{i=1}^{2^{O_l-O_1}} i*(C_{i-1}^{2^{O_l-O_1}}\tilde{P}_s^{\,i-1}) \qquad \textbf{(1)}$$

The expected value of $\tilde{P}_s$ is

$$\left(\frac{(n-2)(\frac{1}{2})^{n+1}+(\frac{1}{2})^{2n}}{((1-(\frac{1}{2})^n)^2}\right)*\left(\frac{K}{2^n-1}\right)$$

since, $\tilde{P}_\Phi+\tilde{P}_e=1-\tilde{P}_s$, formula (1) could be written as follows

$$(1-\tilde{P}_s)\sum_{i=1}^{2^{O_l-O_1}} i*(C_{i-1}^{2^{O_l-O_1}}\tilde{P}_s^{\,i-1})$$

which equals

$$(1-\tilde{P}_s)*(\tilde{P}_s(2^{O_l-O_1})(1+\tilde{P}_s)^{2^{O_l-O_1}-1}+(1+\tilde{P}_s)^{2^{O_l-O_1}}) \qquad \textbf{(2)}$$

Since, $(1+P)^x=1+xP+x(x-1)P^2+.....$

$\tilde{P}_s(2^{O_l-O_1})(1+\tilde{P}_s)^{2^{O_l-O_1}-1}+(1+\tilde{P}_s)^{2^{O_l-O_1}}$ could be written as

$$1+\left(\frac{(\frac{1}{2})^{2n-O_l+O_1-1}}{(1-(\frac{1}{2})^n)^2}\right)*\left(\frac{K}{2^n-1}\right)+\left(\frac{(\frac{1}{2})^{4n-2O_l+2O_1-1}}{(1-(\frac{1}{2})^n)^2}\right)*\left(\frac{K}{2^n-1}\right)^2-\left(\frac{(\frac{1}{2})^{4n-O_l+O_1-1}}{(1-(\frac{1}{2})^n)^2}\right)*\left(\frac{K}{2^n-1}\right)^2+....$$

Algorithm Count, applies the search for all other itemsets, start with lower order items, i.e., items with order less than O_1 , one at a time. Number of such itemsets, including s_j, is O_1 .By neglecting higher terms, and sum over all possible itemsets, our formula could be written as

$$O_1+O_1\left(\frac{(\frac{1}{2})^{2n-O_l+O_1-1}}{(1-(\frac{1}{2})^n)^2}\right)*\left(\frac{K}{2^n-1}\right)$$

Taking an average over O_l, which ranges from O_1 to n, the above formula is converted to

$$O_1+\left(\frac{O_1}{n-O_1+1}\right)*\left(\frac{(\frac{1}{2})^{n+O_1-2}-(\frac{1}{2})^{2n-1}}{(1-(\frac{1}{2})^n)^2}\right)*\left(\frac{K}{2^n-1}\right)$$

For simplification reason, since the minimum number of O_1 is 1, we will divide the second term by 1. Average value over O_1 , which ranges from *1* to n, the above formula , will be

$$\frac{n-1}{2}+\left(\frac{1}{n}\right)*\left(\frac{\frac{4}{3}-(n+\frac{7}{3})(\frac{1}{2})^{n+1}-(\frac{1}{2})^{2n-2}-\frac{n(n-1)}{2}(\frac{1}{2})^{2n-1}}{(1-(\frac{1}{2})^n)^2}\right)*\left(\frac{K}{2^n-1}\right)$$

4.4 Conclusions and Discussion

In this paper, we have introduced a new approach for association mining, called the item-set tree approach. The new approach solves some of the problems inherent in traditional data mining techniques, such as, data updates, accuracy of data mining results, performance, and user queries. The spectrum of techniques that generate association rules, has been studied, and two extreme cases have been analyzed. The main assumption in our study is that all items are equally likely to appear in an itemset. Although this assumption does not reflect the real life, but it gives a good indication about the performance of the item-set tree approach.

We have discussed the item-set tree approach in details. In our approach, the transaction file is read only once. The item-set tree approach maintains a structure to handle frequency counting of transaction data, that allows future updates. Two algorithms; first, to insert transactions into the item-set tree, and second, to count frequencies of itemsets are investigated. Our investigations of the two algorithms show that the costs of insertion and counting do not depend on the number of transactions. The expected cost of inserting a transaction is $\approx O(1)$, and the expected cost of counting the frequency of an itemset is $O(n)$, where n is the cardinality of the domain of items. We conclude that those items that are queried most by users should have low order values, while those items which rarely queried by users should have high order values. This can be accomplished by using prior knowledge of the pattern of user queries.

References

[1] R. Agrawal, T. Imilienski, and A. Swami, "Mining Association Rules between Sets of Items in Large Databases," Proc. of the ACM SIGMOD Int'l Conf. On Management of data, May 1993.

[2] R. Agrawal, and R. Srikant, "Fast Algorithms for Mining Association Rules," Proc. Of the 20th VLDB Conference, Santiago, Chile, 1994.

[3] R. Agrawal, J. Shafer, "Parallel Mining of Association Rules," IEEE Transactions on Knowledge and Data Engineering, Vol. 8, No. 6, Dec. 1996.

[4] C. Agrawal, and P. Yu, "Mining Large Itemsets for Association Rules," Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, 1997.

[5] S. Brin, R. Motwani, J. Ullman, and S. Tsur, "Dynamic Itemset Counting and Implication Rules for Market Basket Data," SIGMOD Record (SCM Special Interset Group on Management of Data), 26,2, 1997.

[6] S. Chaudhuri, "Data Mining and Database Systems: Where is the Intersection," Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, 1997.

[7] H. Mannila, H. Toivonen, and A. Verkamo, "Efficient Algorithms for Discovering Association Rules," AAAI Workshop on Knowledge Discovery in databases (KDD-94) , July 1994.

[8] M. Zaki, S. Parthasarathy, M. Ogihara, and W. Li, " New Algorithms for Fast Discovery of Association Rules," Proc. Of the 3rd Int'l Conf. On Knowledge Discovery and data Mining (KDD-97), AAAI Press, 1997.