International Journal of Information Technology and Knowledge Management
July-December 2009, Volume 2, No. 2, pp. 223-229

A NOVEL ALGORITHM FOR MINING FREQUENT ITEM-SETS FROM LARGE DATABASE

Akhilesh Tiwari[*], **Rajendra K. Gupta**[**] & **Dev Prakash Agrawal**[*]

Data Mining is the process of extracting interesting and previously unknown patterns and correlations from huge amounts of data. Association rule mining, a descriptive mining technique of data mining is the process of discovering items, which tend to occur together in transactions. As the data to be mined is large, the time taken for accessing data is considerable. In this paper, a new association rule mining algorithm which generates the frequent itemsets in a single pass over the database is presented. The algorithm mainly uses two approaches for association rule mining: The Partition approach, where the data is mined in partitions and merges the result, and the Apriori approach that helps to find the frequent sets within each partition. In order to evaluate the performance of new association algorithm, it is compared with the existing algorithms which require multiple passes to generate the frequent itemsets. Experiments show that time taken for the database scan is more than the time taken for candidate generation when the database size is large, which provides evidence that, focus to decrease the database access time is a viable approach to the association rule mining.

Keywords: Association Rule, Apriori, Data Mining, Frequent Itemset.

1. Introduction

Due to widespread computerization and affordable storage facilities, an enormous wealth of information is embedded in huge databases belonging to different enterprises. Such databases, whether their origin is the business enterprise or scientific experiment, have spurred a tremendous interest in the areas of Knowledge Discovery and Data Mining (Agrawal *et al.*, 1993; Bjorvand, 1998; Landau *et al.*, 1998; Hunzikar *et al.*, 1998; Aggarwal et *al*., 1999; Tiwari *et al.*, 2008). These areas have motivated allowed statisticians and data miners to develop faster analysis tools that can help sift and analyze the stockpiles of data, turning up in to valuable and often surprising information. Data mining is the act of drilling through huge volumes of data to discover relationships, or answer queries too generalized for traditional query tools (Agrawal *et al.*, 1993). Data mining is part of the process known as Knowledge Discovery in Databases (KDD), which is the automated approach of the extraction of implicit, understandable, previously unknown and potentially useful information from large databases. For extraction of such valuable information, the KDD process follows an iterative sequence of steps that include data selection and integration, data cleaning and preprocessing, data mining and algorithm selection, and, finally, post processing and knowledge presentation.

2. Association Rule Problem

The problem of mining association rules is to generate all rules that have support and confidence greater than or equal to some user specified minimum support and minimum confidence threshold respectively (Agrawal *et al*., 1994). Association rule mining involves detecting items, which tend to occur together in transactions, and the association rules that relate them. Mining frequent itemsets is a fundamental and essential operation in data mining applications including discovery of association rule, strong rules, correlations, sequential rules and episodes. Due to the huge size of data and amount of computation involved in data mining, high-performance computing is an essential component for any successful large-scale data mining applications.

Association rule mining finds the set of all subsets of items that frequently occur in many database records or transactions, and additionally extracts rule on how a subset of items influences the presence of another subset. Consider $I = \{i_1, i_2, \ldots\ldots\ldots\ i_m\}$ as a set of items. Let D, the task relevant data, is a set of database transactions where each transaction T is a set of items such that T is a subset of I. Each transaction is associated with an identifier, called TID. Let A be a set of items. A transaction T is said to contain A if and only if A is a subset of T.

An association rule is an implication of the form $A => B$, where A and B are subsets of I and $A \cap B$ is also a subset of I. The rule $A => B$ holds in the transaction set D with support S, where S is the percentage of transactions in D that contain $A \cup B$ (i.e., both A and B). This is the probability, $P\ (A \cup B)$. The rule $A => B$ has confidence C in the transaction set D if C is the percentage of transactions in D containing A that also contain B. This is taken to be the conditional probability, $P\ (B/A)$. That is,

[*] Madhav Institute of Technology & Science, Gwalior (M.P.), INDIA
[**] Union Public Service Commission, New Delhi, India,
Email: amity.tiwari@rediffmail.com

$$\text{Support } (A \cup B) = P\,(A \cup B)$$
$$\text{Confidence } (A \cup B) = P\,(B/A)$$

The definition of a frequent pattern relies on the following considerations. A set of items is referred to as an itemset (pattern). An itemset that contains K items is a K-itemset. The set $\{X,Y\}$ is a 2-itemset. The occurrence frequency of an itemset is the number of transaction that contain the itemset. This is also known as the frequency or the support count of an itemset. An itemset satisfies minimum support if the occurrence frequency of the itemset is greater than or equal to the minimal support threshold value defined by the user. The number of transaction required for the itemset to satisfy minimum support is therefore referred to as the minimum support count. If an itemset satisfies minimum support, then it is a frequent itemset.

A frequent itemset is called closed if it does not have any superset with the same support. A frequent itemset is said to be maximal if it has no supersets that are frequent. The collection of maximal frequent itemsets is a subset of the collection of closed frequent itemsets, which is a subset of the collection of all frequent itemsets. Maximal frequent itemsets are necessary for generating association rules.

The problem of mining association rules could be decomposed into two sub problems:

1. Find out all large itemsets and their support counts. A large itemset is a set of items which are contained in a sufficiently large number of transactions, with respect to a support threshold minimum support.
2. From the set of large itemsets found, find out all the association rules that have a confidence value exceeding a confidence threshold minimum confidence.

Since the solution of the second subproblem is straightforward, here we are concentrating only on the first subproblem.

3. Partition Algorithm

The partition algorithm (Han *et al*., 2000; Toivonen *et al*., 1996; Yen *et al*., 2001) is based in the observation that the frequent sets are normally very few in number compared to the set of all itemsets. As the result, if the set of transactions are partitioned in to smaller segments such that each segment can be accommodated in the main memory, then the set of frequent sets of each of these partitions can be computed. Therefore this way of finding the frequent sets by partitioning the database may improve the performance of finding large itemsets in several ways:

- By taking advantage of the large itemset property, this is that a large itemset must be large in at least one of the partitions. This idea can help to design algorithms more efficiently than those based on looking at the entire database.
- Partitioning algorithms may be able to adapt better to limited main memory. Each partition can be created such that it fits in to main memory. In addition it would be expected that the number of itemsets to be counted per partition would be smaller than those needed for the entire database.
- By using partitioning, cluster based and/or distributed algorithms can be easily created, where each partitioning could be handled by a separate machine.
- Incremental generation of association rules may be easier to perform by treating the current state of the database as one partition and treating the new entries as a second partition.

In order to achieve all the above advantages of partitioning the transaction database, the partition algorithm works as follows:

The partition algorithm uses two scans of the database to discover all frequent sets. In one scan, it generates a set of all potential frequent itemsets by scanning the database. This set is a superset of all frequent itemsets, i.e. it may contain false positives, but no false negatives are reported. During the second scan, counters for each of these itemsets are setup and their actual support is measured in one scan of the database.

The partition approach of generating frequent itemsets is given below:

```
$P$ = partition_database ($T$); $N$ = Number of partitions;
// Phase 1
for $i$ = 1 to $n$ do begin
read _in_partition ($T_i$ in $P$)
$L_i$ = generate all frequent itemsets of $Ti$ using apriori method in main memory.
End
// Merge Phase
For ($k$ = 2; $L_i^k \neq \Phi$, $i$ = 1, 2, ………, $n$; $k^{++}$) do begin
$C_k^G = Y_{i+1}^{n} L_i^k$
End
// Phase II
For $i$ = 1 to $n$ do begin
Read _ in _ partition ($T_i$ in $P$)
For all candidates $c \in C^G$ compute $s(c)T_i$
End
$L^G = \{c \in C^G / s(c)T_i \leq \sigma\}$
Answer = $L^G$
```

As given the partition algorithm above, here is the example of implementing it:

Table 1
Transaction Database

	A_1	A_2	A_3	A_4	A_5
Ψ_{t1}	1	0	0	0	1
Ψ_{t2}	0	1	0	1	0
Ψ_{t3}	0	0	0	1	1
Ψ_{t4}	0	1	1	0	0
Ψ_{t5}	0	0	0	0	1
Ψ_{t6}	0	1	1	1	0

Here is the transaction database, $A = \{A1, A2, A3, A4, A5\}$, assume $\sigma = 20$ %.

Here the database is partitioned in to 3 partitions say ξT_1, ξT_2, $\xi T_{3,}$ each containing 2 transactions. The first partition ξT_1 contains 1 to 2 transactions, ξT_2 contains 3 to 4, and ξT_3 contains 5 to 6 transactions. Here the local support is equal to the given support, which is 20%. So $\sigma = \sigma_1 = \sigma_2 = \sigma_3 = 20\%$.

The working of partition algorithm is as follows:

L_1 : = the frequent sets from the partition in $\xi T_{1,}$ which are found using the apriori algorithm on ξT_1 separately.

L_2 : = the frequent sets from the partition in $\xi T_{2,}$ which are found using the apriori algorithm on ξT_2 separately.

L_3 : = the frequent sets from the partition in $\xi T_{3,}$ which are found using the apriori algorithm on ξT_3 separately.

In phase II, the candidate set as

$$C := L_1 \cup L_2 \cup L_3$$

Later read the database once again to compute the global support of the sets in C and get the final set of frequent sets.

3.1 Problem Specification

Most of the algorithms (Mannila *et al*., 1994; Savasere *et al*., 1995; Agrawal *et al.*, 1994; Yen *et al.*, 2001; Park *et al*., 1997; Houtsma *et al*., 1995; Coenen *et al*., 2001; Tiwari *et al.,* 2008) for discovering Frequent Patterns require multiple passes over the database resulting in a large number of disk reads and placing a huge burden on the I/O subsystem. In order to reduce the burden on the I/O subsystem in the case of large databases, a new association rule mining algorithm, which uses both the partition and the apriori approach for calculating the frequent itemsets in a single pass over the database, is presented below.

4. Proposed Extensions to Partition algorithm

The following notation is used in the remainder of this approach:

Table 2
Notations used for Advanced Partitioned Approach

Notation	*Meaning*
L^i	*Local Frequent Sets:* Set of Local Frequent Itemsets of Partition *i*.
C_k^G	*Global Frequent Sets:* Set of global candidate *K*-Itemsets.
L_i^k	*Local Frequent Sets:* Set of local frequent *K*-Itemsets in partition *i*.
L^G	*Global Frequent Itemsets:* Set of global frequent Itemsets.
$S(c)T_c$	*Combined Support* Total support of candidate set *c* in all partitions.

This modified partition approach used for finding frequent itemsets in single pass over the database consists of two phases. The methodology involved in those two phases is described below:

Phase 1:

In this phase, the partition algorithm logically divides the database in to a number of non-overlapping partitions. These partitions are considered one at a time and all frequent itemsets for that partition (L^i) are generated using the apriori algorithm. In addition, when taking each partition for calculating the frequent itemsets separately the local minimum support is set to 1. Thus, if there are *n* partitions, phase I of the algorithm takes n iterations. At the end of phase I, all the local frequent itemsets of each partition are merged to generate a set of all potential frequent itemsets. In this step, the local frequent itemsets of same lengths from all *n*-partitions are combined to generate the global candidate itemsets (C_k^G), and also those global candidate itemsets has there combined support (total support of itemset if that itemset is present in more than one partition) associated with it.

Phase 2:

As the above generated global candidate itemsets have least possible frequent itemset of that partition because, during the generation of frequent itemsets using the apriori algorithm of each partition, the minimum local support was set to 1. So this phase just prune the itemsets from the global candidate itemsets list whose combined support ($S(c)_{Tc}$) (total support of an itemset in all the partitions) is less than the global minimum support. So by using the above approach the extra database pass which was needed in the phase II of the previous partition approach for calculating the support of global candidate itemsets is eliminated. So here the modified partition algorithm reads the entire database once during the phase I. And also, partition sizes are chosen such that each partition can be accommodated in the main memory.

Below is the algorithm of modified partition approach:

```
$P$ = partition_database ($T$); $N$ = Number of partitions;
// Phase 1
for $i$ = 1 to $n$ do begin
read _in_partition ($T_i$ in $P$)
$L_i$ = generate all frequent itemsets of $Ti$ using apriori
method in main memory.
End
// Merge Phase
For ($k$ = 2; $L_i^k \neq \Phi$, $i$ = 1, 2, ………, $n$; $k^{++}$) do begin
$C_k^G = Y_{i=1}{}^{n}Li^k$
End

// Phase II
$L^G = \Phi$;
for each $c \in C^G$ do begin
if $S(C)_{Tc} \geq \sigma$
$L^G = L^G \cup \{S(C)\}$
End
Answer = $L^G$
```

This above partition approach is based on the premise that the number of items in a single transaction is considerably smaller compared to the total items in the transaction database (i.e. total number of items placed in basket is less compared to total number of items available). In addition, it expects the support of frequent itemsets generated in a particular partition to be much high (more than 1). Therefore, for sufficiently large partition sizes, the number of local frequent itemsets is likely to be comparable to the number of frequent itemsets generated for the entire database. If the data characteristics are uniform across partitions, then large numbers of itemsets generated for individual partitions may be common.

5. PERFORMANCE EVALUATION OF THE PROPOSED ALGORITHM

For the purpose of implementing the above partition based association rule mining algorithm for finding the frequent itemsets, the retail market basket dataset is used. These data sets were obtained from FIMI repository for frequent itemset mining. The advanced partition approach produces the frequent itemsets in single pass over the database, where as, the previous partition approach uses two passes over the database to find the frequent itemsets. To do this, a few additional modules, including the changes in the Apriori module and partition module are created. The system is implemented as a object oriented program in the java programming language. To measure the performance of the New Partition approach with the existing Apriori and Partition algorithms, the datasets from the FIMI repository are used. The nomenclature of these datasets is of the form "*TxxDzzzK*", where "*xx*" denotes the average number of items present per transaction and "*zzzK*" denotes the total number of transaction on "*K*" (1000's). The experiments are performed on a machine running Microsoft Windows XP with 786 MB of RAM and a 1.6 GHz Pentium 4 Processor. Each experiment has been performed 4 times. The values from the first run are ignored so as to avoid the effect of the previous experiment and other database setups. The average of the next 3 runs is taken and used for analysis. This is done so as to avoid any false reporting of time due to system overload or any other factors.

5.1 Comparisons of Performances of Different Algorithms

To compare the performance of New Partition algorithm with the Apriori and Partition algorithm three different scenarios are considered:

Scenario 1: Performance of the New Partition algorithm when the average number of items per transaction increases.

Scenario 2: Performance of the New Partition algorithm when the data set size is small and the number of items per transactions are also small.

Scenario 3: Performance of the New Partition algorithm when the size of data set is large and candidate itemsets are also large.

The following subsections perform the above specified scenarios separately:

5.1.1 Scenario 1

To compare the performance of the new partition algorithm when the number of items per transaction increases, the following datasets are chosen:

Table 3
Datasets

Name	$\lvert T \rvert$	$\lvert D \rvert$	*Size in Megabytes*
*T*9*D*80*K*	9	80K	3.5
*T*13*D*80*K*	13	80K	4

Where $|T|$ is the average number of items present in the transaction and $|D|$ is the number of transactions in a dataset (in 1000's).

Figures (a) and (b) shows the execution times of Apriori and Partition and New Partition algorithms under different minimum support.

Figures (a) and (b) also shows that as the minimum support decreases, the execution times of the Apriori and Partition algorithms increase because of the increase in the total number of candidate itemsets. The New Partition algorithm execution however is independent of the increase or decrease of the minimum support (because the local support for the itemsets is treated as 1 irrespective of the global support). From figures a and b, it is evident that the New Partition approach performance is good when the average numbers of items in the transaction are less and the performance of New Partition approach is less than both the Partition and Apriori algorithm when the database size is small and even the average number of items in the transaction change.

Summary: Both the Partition and Apriori algorithms execution times beat the new partition approach almost by a magnitude of three when the database size is small and the candidate generation increases. In other words, the new partition approach performance will be good when the average number of items per transaction is less. So the next scenario sees the performance of the new partition approach when the database size changes under the different minimum support and when the average numbers of items per transaction are less.

5.1.2 Scenario 2

To analyze the performance of the New Partition algorithm when the database size increases and the minimum support decreases, the following datasets are chosen:

Table 4
Datasets

Name	$\lvert T\rvert$	$\lvert D\rvert$	*Size in Megabytes*
*T*9*D*90*K*	9	90K	4
*T*9*D*180*K*	9	180K	8
*T*9*D*360*K*	9	360K	16

Where $|T|$ are the average number of items present in the transaction and $|D|$ are the number of transaction in a datasets (in 1000's).

Figures c, d, e shows the execution times of all the three algorithms for different database sizes and having minimum percentage support of 2, 1, .50 respectively.

Figures c, d, e shows that the execution time of the New Partition algorithm increases as the size of the database increases, as do the execution times of the Apriori and the Partition algorithms. The Apriori and the Partition algorithms still have smaller execution times than the New Partition algorithm.

Summary: The performance of new partition algorithm is much less when the database sizes are small, comparatively the performance of the new partition approach increases as the minimum support decreases and the database size increases.

5.1.3 Scenario 3

To compare the performance of the New Partition algorithm when the database size is large and the minimum support decreases, the following datasets are chosen:

Table 5
Datasets

Name	$\lvert T\rvert$	$\lvert D\rvert$	*Size in Megabytes*
*T*9*D*10000*K*	9	10000K	250
*T*9*D*20000*K*	9	20000K	500
*T*9*D*40000*K*	9	40000K	1000
*T*9*D*80000*K*	9	80000K	2000
*T*9*D*160000*K*	9	160000K	4000

Where $|T|$ are the average number of items present in the transaction and $|D|$ are the number of transaction in a datasets (in 1000's).

Figures f, g, h and i shows the execution times of all the three algorithms for different database sizes and having minimum percentage support 2, 1, .50, .25 respectively. As seen in the above mentioned figures, The New Partition algorithm's execution time eventually becomes better than the execution times of the partition algorithm and almost close to the execution time of the Apriori algorithm as the database size increases and the minimum support decreases.

Summary: Scenario 3 shows that new partition algorithm will beat the performance of the other algorithms when the database size is huge and the minimum support decreases (which increase the candidate set generation). The new partition approach will beat the performance of the normal partition algorithm when the extra time needed for the candidate sets generation in new partition algorithm than the normal partition algorithm will be less than the time needed for normal partition algorithm to scan the whole database for pruning of the candidate sets.

Figure (a): Execution times of Apriori and Partition and New Partition algorithms corresponding to dataset of Table 3 having 9 items per transaction under different minimum support.

T9D80K-Figure (a) Performance T9D80K

Minimum Support (%)	*Apriori (Seconds)*	*Partition (Seconds)*	*New Partition (Seconds)*
2	76	102	449
1.5	88	116	450
1	103	141	450
.75	137	177	448
.5	154	204	451
.33	172	231	450

Figure (b): Execution times of Apriori and Partition and New Partition algorithms corresponding to dataset of Table 3 having 13 items per transaction under different minimum support.

T13D80K-Figure (b) Performance T13D80K

Minimum Support (%)	*Apriori (Seconds)*	*Partition (Seconds)*	*New Partition (Seconds)*
2	109	111	509
1.5	110	117	510
1	115	124	509
.75	118	144	508
.5	128	179	509
.33	148	288	509

Figure (c): Execution times of Apriori and Partition and New Partition algorithms corresponding to dataset of Table 4 at min_supp of 2%

Support 2% - Figure (c) Performance Support 2%

Dataset	*Apriori (Seconds)*	*Partition (Seconds)*	*New Partition (Seconds)*
T9D90K	70	115	500
T9D180K	133	230	998
T9D360K	267	458	1997

Figure (d): Execution times of Apriori and Partition and New Partition algorithms corresponding to dataset of Table 4 at min_supp of 1%

Support 1% - Figure (d) Performance Support 1%

Dataset	*Apriori (Seconds)*	*Partition (Seconds)*	*New Partition (Seconds)*
T9D90K	93	160	502
T9D180K	180	320	999
T9D360K	362	638	2000

Figure (e): Execution times of Apriori and Partition and New Partition algorithms corresponding to dataset of Table 4 at min_supp of .50%

Support .50% - Figure (e) Performance Support .50%

Dataset	*Apriori (Seconds)*	*Partition (Seconds)*	*New Partition (Seconds)*
T9D90K	140	257	500
T9D180K	280	510	1000
T9D360K	560	1000	1998

Figure (f): Execution times of Apriori and Partition and New Partition algorithms corresponding to dataset of Table 5 at min_supp of 2%

Support 2% - Figure (f) Performance Support 2%
(Large Databases)

Dataset (MB)	*Apriori (Hours)*	*Partition (Hours)*	*New Partition (Hours)*
250	0.83	1.3	5
500	1.6	2.9	10
1000	3.3	6.6	20
2000	6.6	14.9	40
4000	13.2	34	80

Figure (g): Execution times of Apriori and Partition and New Partition algorithms corresponding to datasets of Table 5 at *min_supp* of 1%

Support 1% - Figure (g) Performance Support 1%
(Large Databases)

Dataset (MB)	*Apriori (Hours)*	*Partition (Hours)*	*New Partition (Hours)*
250	1.1	1.8	5
500	2.3	4.1	10
1000	5	9.3	20
2000	10.5	21.1	40
4000	22.1	48.6	80

Figure (h): Execution times of Apriori and Partition and New Partition algorithms corresponding to datasets of Table 5 at *min_supp* of .50%

Support 1% - Figure (g) Performance Support 1%
(Large Databases)

Dataset (MB)	*Apriori (Hours)*	*Partition (Hours)*	*New Partition (Hours)*
250	1.7	2.8	5
500	3.6	6.4	10
1000	7.7	14.4	20
2000	16.2	32.2	40
4000	34	74.5	80

Figure (i): Execution times of Apriori and Partition and New Partition algorithms corresponding to datasets of Table 5 at *min_supp* of .25%

Support .25% - Figure (g) Performance Support .25%
(Large Databases)

Dataset (MB)	*Apriori (Hours)*	*Partition (Hours)*	*New Partition (Hours)*
250	3.4	3.7	5
500	7.2	8.5	10
1000	15.2	19.1	20
2000	32	43	40
4000	68	98	80

6. CONCLUSION

In this paper a new association rule mining algorithm which uses the partition approach for mining the frequent itemsets in a single pass over the database has been proposed. Experiments have been performed on real databases obtained from FIMI repository and the results have been presented. The results show that time taken for the database scan is more than the time taken for candidate generation when the database size is large, which provides evidence that, focus to decrease the database access time is a viable approach to the association rule mining.

REFERENCES

[1] R. Agrawal, T. Imielinski and A. Swami. Mining Association Rules Between Sets of Items in Large Databases. In : Proc. 1993 ACM-SIGMOD International Conference on Management of Data, Washington, D.C., (1993) 207-216.

[2] Rakesh Agrawal and R. Srikant. Fast Algorithm for Mining Association Rules in Large Databases, Proceedings of the 20th International Conference on Very Large Databases, *Santigo,* Chile, (1994), 487-499.

[3] C. C. Aggarwal and P. S. Yu. Data Mining Techniques for Associations, Clustering and Classification. Proceedings of the Third Pacific-Asia Conference, PAKDD-99, Beijing, China, (1999) 13-23.

[4] A. T. Bjorvand. Object Mining: A Practical Application of Data Mining for the Construction and Maintenance of Software Components. Proceedings of the Second European Symposium, PKDD-98, Nantes, France, (1998) 121-129.

[5] Frans Coenen, Graham Goulbourne, and Paul Leng. Computing Association Rules Using Partial Totals. 5th European Conference, PKDD 2001, *Freiburg,* Germany, Proceedings, (2001).

[6] J. Han, J. Pei, and Y. Yin., Mining Frequent Patterns Without Candidate Generation. In Proceedings of ACM SIGMOD International Conference on Management of Data, *Dallas,* TX, (2000) 1-12.

[7] M. Houtsma and A. Swami. Set Oriented Mining for Association Rules in Relational Databases. In Proceedings of 11th IEEE International Conference on Data Engineering, (1995) 25-33.

[8] Petra Hunzikar, Andreas Maier, Alex Nippe, Markus Tresch, Douglas Weers and Peter Zemp. Data Mining at a Major Bank: Lessons from a Large Marketing Application. Proceedings of the Second European Symposium, PKDD-98, Nantes, France, (1998) 345-351.

[9] D. Landau, R. Feldman, O. Zamir, Y. Aumann, M. Fresko, Y. Lindell and O. Lipshtat. Text Vis: An Integrated Visual Environment for Text Mining. Proceedings of the Second European Symposium, PKDD-98, *Nantes,* France, (1998) 56-64.

[10] H. Mannila, H. Toivonen and A. Inkeri Verkamo. Efficient Algorithms for Discovering Association Rules. AAAI Workshop on Knowledge Discovery in Databases (KDD-94), (1994) 181-192.

[11] Jong Soo. Park, Ming-Syan and Philips S. Yu. Using a Hash-Based Method with Transaction Trimming for Mining Association Rules. *IEEE Transactions on Knowledge and Data Engineering,* **9**(5) (1997).

[12] A. Savasere, E. Omieccinski and S. Navathe. An *Efficient Algorithm for Mining Association Rules in Large Databases.* Proceedings of the 21st International Conference on Very Large Databases, Zurich, Switzerland, (1995) 432-443.

[13] H. Toivonen. *Sampling Large Databases for Association Rules.* In Proceedings of International Conference on Very Large Databases, Bombay, India, (1996) 134-145.

[14] Akhilesh Tiwari, R. K. Gupta, D. P. Agrawal. *Mining Frequent Itemsets Using Prime Number Based Approach.* In Proc. 3rd International Conference on Advanced Computing and Communication Technologies (ICACCT), India, (2008) 138-141.

[15] Show-Jane Yen and Arbee L. P. Chen. A Graph–Based Approach for Discovering Various Types of Association Rules. *IEEE Transactions on Knowledge and Data Engineering,* **13**(5) (2001) 839-845.