Mining Frequent Itemsets in Evolving Databases

*A.A. Veloso**, *W. Meira Jr.**, *M.B. de Carvalho**, *B. Pôssas**, *S. Parthasarathy*†, *M. Javeed Zaki*‡

1 Introduction

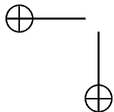
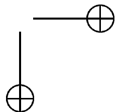
The field of knowledge discovery and data mining (KDD), spurred by advances in data collection technology, is concerned with the process of deriving interesting and useful patterns from large datasets. The KDD process is computational and data-intensive and is inherently interactive and iterative in nature. In fact, interactivity is often the key to facilitating effective data understanding and knowledge discovery. In such an environment, response time is crucial because lengthy time delay between responses of consecutive user requests can disturb the flow of human perception and formation of insight. The task of guaranteeing quick response times is more complicated in dynamic datasets, where there is a constant influx of data. Changes to the data can invalidate existing patterns or introduce new. Simply re-executing algorithms from scratch when a database is updated can result in an explosion in the computational and I/O resources required. What is needed is a way to process the data incrementally and update the information that is gleaned while being cognizant of the interactive requirements of the process. In this paper we present such an approach for a key data mining task: association rule mining.

Discovery of association rules is an important problem in data mining. The prototypical application is the analysis of sales or *basket* data [1, 2], but besides the retail sales example, association rules have been shown to be useful in domains such as decision support, telecommunications alarm diagnosis and prediction, university enrollments, etc. The problem of mining association rules can be decomposed into two subproblems. Conceptually, the goal of the first subproblem is to find all the co-

*Department of Computer Science, Universidade Federal de Minas Gerais, Brazil adrianov@dcc.ufmg.br meira@dcc.ufmg.br mlbc@dcc.ufmg.br bavep@dcc.ufmg.br

†Department of Computer and Information Science, The Ohio State University, USA srini@cis.ohio-state.edu

‡Computer Science Department, Rensselaer Polytechnic Institute, USA zaki@cs.rpi.edu

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited

occurring items (henceforth referred as itemsets) which are present in a significant number of transactions; a number greater than a user-specified threshold value called minimum support. Many efficient algorithms have been proposed to solve this first subproblem[2, 5, 7] for static datasets. The goal of the second subproblem is to generate implication (probabilistically determined) rules, based on the frequent itemsets, with respect to another user-specified threshold; the minimum confidence.

Now consider the problem of computing association rules on an evolving dataset, like those found in e-commerce or web-based domains. The datasets in such domains tend to be very dynamic, i.e., constantly updated with fresh data. Let us assume that at some point in time we have computed all the relevant association rules for such a database. Later, if the database is updated or if the user decides to change the parameters of the rule generating program, the set of associations previously computed is no longer valid (some of the rules may still hold, but not all of them). A naive approach to compute the new set of association rules would be to re-execute a traditional algorithm on the updated database or with the new parameters. However, this process is not efficient since it is memoryless (i.e., it ignores the already discovered knowledge), essentially duplicating part of the work that have already been done.

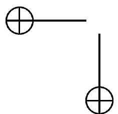
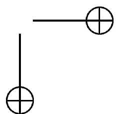
To address this problem, several researchers [3, 4, 6, 9, 10, 16, 17] have proposed incremental association mining algorithms. These algorithms re-use the previously mined information and try to combine this information with the fresh data to efficiently re-compute the new set of association rules. In this paper we present a novel technique that advances the state-of-the-art in incremental association rule mining. Our algorithm, called ZIGZAG[1], like other approaches, uses previously discovered knowledge to reduce the cost of updating the frequent itemsets. However, it introduces some significant improvements over previous incremental mining algorithms [3, 4, 6]. We highlight these improvements next.

Our approach maintains only the maximal frequent itemsets to incrementally construct the lattice of frequent itemsets. The maximal frequent itemsets are updated by a novel backtracking search, which is guided by the results of the previous mining iteration, yielding a very efficient way for determining frequent itemsets. The approach is also very efficient in supporting drill-down/drill-up interactions based on modifying user parameters such as minimum support. Due to the exploratory nature of data mining this support for interactivity is crucial. Further, our algorithm is extermely efficient regarding I/O.

The algorithm can also support windowed mining operations or short term mining, wherein old transactions are discarded from the database and new ones added, keeping the set of association rules coherent with respect to the most recent data. Short-term mining is desirable for applications such as WEB mining, where the user behavior may vary significantly across time and old access patterns may be no longer relevant. This feature enables a high degree of interactivity, encouraging a flexible exploration of the database through the use of a fixed-length time windows.

Another important feature of our approach is its ability to track and use stable association patterns[12]. This feature is useful in two ways. First, users may be interested in identifying such associations because of the stability property[12], which

[1] The name zigzag was inspired on the behavior of the algorithm, which searches the solution space for maximal frequent itemsets on an upward fashion and then enumerates their subsets on a downward fashion.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited

is regarded to the discovery of reliable association rules. Second, these associations can be used to speed up the incremental mining process at a small cost to accuracy. Variations of the popularity of the frequent itemsets are considered in order to determine which subsets are to be updated as a result of data modifications. For instance, the algorithm will update only those frequent itemsets whose variation in popularity is above a pre-defined relaxation threshold. Note that if the popularity of an itemset does not change much over time (a stable association), the rules from its subsets will likely remain a fairly good approximation of the actual rules, even in the presence of new data updates. This enables the algorithm to postpone a full update operation for some maximal frequent itemsets, which increases the algorithm efficiency significantly.

Extensive experiments are reported in this work comparing the performance of ZIGZAG against ULI [16] running on real (from the E-Commerce domain) and synthetic databases under different characteristics. We also evaluate the use of ZIGZAG for short-term mining, and the impacts of relaxation.

2 Preliminaries and Related Work

Association Rules Let $I = \{I_1, I_2, ..., I_m\}$ be a set of m items and an itemset X is a non-empty ordered set of items and, in particular, if it contains k items, it will be denoted by k-itemset. A transaction T is a set of items, $T \subseteq I$ uniquely identified by *tid* and D is a database of transactions. The set of *tids* which contain a given itemset, X, is called the *tidlist* of X and is denoted as $\mathcal{L}_D(X)$.

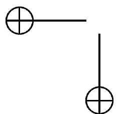
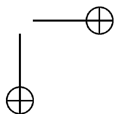
The *support* of X, denoted $\sigma_D(X)$, is the percentage of transactions in D that contains X as subset, i.e., $\sigma_D(x) = \|\mathcal{L}_D(X)\|/\|D\|$. An itemset X is *frequent* if its support is no less than a user-specified minimum support threshold (*minsup*).

A frequent itemset is *maximal* if it is not a subset of any other frequent itemset. The set of all maximal frequent itemsets is denoted as MFI. MFI is also referred to as the *positive border*. The *negative* border, on the other hand, refers to the minimal not frequent itemsets. The knowledge of MFI is sufficient to determine all sets that are frequent, but not their exact supports. The exact frequency can, however, be obtained by counting all the distinct subsets of maximal sets against the database.

An association rule is an implication of the form $X \Rightarrow Y$, where $X \subseteq I$, $Y \subseteq I$ and $X \bigcap Y = \emptyset$. The support of the rule is given as $\sigma_D(X \cup Y)$, i.e., the joint probability of X and Y, while its *confidence* is given as $\sigma_D(X \cup Y)/\sigma_D(X)$, i.e., the conditional probability of Y given X. A rule is *confident* if its confidence is above a *minconf* threshold. Given the support of frequent sets it is straight-forward to generate and test the rules that are confident. Thus, from now on, we will concentrate on mining frequent itemsets in evolving databases.

Updating Frequent Itemsets Incremental mining of association rules was first introduced in [4]. The central theme of incremental mining relies on the re-use of previously mined computations to enhance the performance of future interactions.

Let s_D be the support threshold used when mining D, and L_D be the corresponding set of frequent itemsets. Let P be the information kept from the current mining that will be used in the next updated operation. In our case, P consists of all the single items (along with their tidlists) and all the frequent itemsets (along with their support counts) in the original database D. Using as a starting point D, a set of new transactions d^+ is added and a set of old transactions d^- is removed, forming the new set of transactions Δ, i.e., $\Delta = (D \cup d^+) - d^-$. Since modification

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited

of an existing transaction may be handled as a deletion followed by a insertion we will assume, without loss of generality, that there are no transaction modifications.

The problem of *updating frequent itemsets* is to find the set L_Δ, the frequent itemsets in Δ, with respect to a minimum support s_Δ and, more important, using P and minimizing access to D, to enhance the algorithm performance.

An itemset X is frequent in Δ if its support is no less than s_Δ. Notice that a frequent itemset in D may not be frequent in Δ (defined as *a declined* itemset), on the other hand, an itemset X not frequent in D, may become a frequent itemset in Δ (defined as *emerged* itemset). If a frequent itemset in D remains frequent in Δ it is called a *retained* itemset.

Backtracking Search Our incremental ZIGZAG algorithm is based on some ideas from GENMAX [13], which is an algorithm that uses a backtrack search to enumerate all the maximal frequent itemsets in a given database. We briefly review the backtracking paradigm in the context of enumerating MFI in the original database D. We will subsequently modify this procedure for incremental mining.

Backtracking algorithms are useful for many combinatorial problems where the solution can be represented as a set $I = \{i_0, i_1, ...\}$, where each i_j is chosen from a finite *possible set*, S_j. Initially I is empty; it is extended one item at a time, as the search space is traversed. The length of I is the same as the depth of the corresponding node in the search tree.

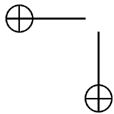
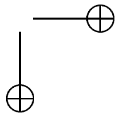
Given a partial solution of length l, $I_l = \{i_0, i_1, ..., i_{l-1}\}$, the admissable values for the next item i_l comes from a subset $C_l \subseteq S_l$ called the *combine set*. If $y \in S_l - C_l$, then the nodes $I_l = \{i_0, i_1, ..., i_l - 1, y\}$ will not be considered by the backtracking algorithm. Since such subtrees have been pruned away from the original search space, the determination of C_l is also called *pruning*.

At each iteration, the algorithm tries extending I_l with every item x in the combine set C_l. An extension is valid if the resulting itemset I_{l+1} is frequent and is not a subset of any already known maximal frequent itemset. The next step is to compute the new possible set of extensions, S_{l+1}, which consists only of items in C_l that follows x. The new combine set, C_{l+1}, consists of those items in S_{l+1} that produce a frequent itemset when used to extend I_{l+1}. Any item not in the combine set refers to a pruned subtree. As presented, the backtrack search performs a depth-first traversal of the search space. GENMAX also uses additional optimizations to speed up the maximality checking and for fast frequency computations (see [13] for additional details).

2.1 Previous Work

Recognizing the dynamic nature of databases, much effort has been devoted to the problem of incrementally mining the frequent sets. Here we give an overview of the methods proposed for solving this problem, and compare them to ZIGZAG.

The first incremental association mining algorithm, FUP, was presented by Cheung *et al* [4], as an algorithm for maintaining association rules. During each iteration FUP starts by mining d^+ and uses the results for guiding the update. In subsequent work, FUP2 [9], the authors extended the above idea to handle both deletions and updates. However, a major limitation of these algorithms is that they may require $O(k)$ database scans on Δ, where k is the size of the largest frequent itemset. To decide whether an update is necessary and to minimize update costs Lee and Cheung[10] proposed an extension to the above, called DELI, which uses

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited

statistical sampling methods to determine when to apply the updation process. Our approach is different from the above in terms of the number of database scans required (only one on the incremental database and only a partial scan on the original database), in terms of support for selective updates (on only some of the maximal frequent itemsets), and in terms of allowing users to modify the parameters (like minimum support) while computing the update.

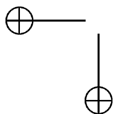
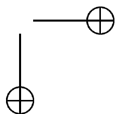
Ayan *et al*[3] present an algorithm called UWEP (Update With Early Pruning) that employs a dynamic look-ahead strategy in updating the frequent itemsets in D by detecting and removing those itemsets that will no longer remain frequent in Δ. UWEP scans D at most once and Δ exactly once. In [17], two incremental algorithms were presented – the *Pairs* approach stores the set of frequent 2-sequences, while the *Borders* algorithm keeps track of the frequent set and the negative border. An approach very similar to the Borders algorithm was also proposed in [6]. These approaches may require only a single scan of the original data, but they still make $O(k)$ scans of the increment database. In contrast to the above, our algorithm makes only one scan of the incremental database, and usually requires only a partial scan of the original database (only those tid-lists that are needed). Like in FUP, these approaches do not allow changes in the user parameters.

Although these incremental algorithms are better than re-executing a traditional algorithm (e.g., Apriori) from scratch on the updated database they all have limitations when compared with ZIGZAG. The amount of information maintained in the case of ZIGZAG is much smaller (since we incrementally maintain only maximal itemsets). Many of the above approaches require several scans over the incremental and original databases. Many of them do not support interactive operations (such as modifying the minimum support etc.) as part of the process. Further, windowed transactional operations (or short term mining operations), useful for domains like WEB mining, are not supported by any of the incremental methods described above.

3 The ZIGZAG Algorithm

In this section we describe our algorithm, ZIGZAG, which connects new techniques of incremental mining, short-term mining and interactive mining for association rules.

ZIGZAG is inspired by the GENMAX algorithm. The main idea is to incrementally compute the maximal frequent itemsets in Δ using previous knowledge P. This avoids the generation and testing of many unnecessary candidates that are usually examined in other approaches, i.e., the candidates that are counted and subsequently turn out to be infrequent (the negative border, to be exact). Having the new MFI is sufficient to know which sets are frequent; their exact support can be obtained by examining d^+, d^- and using previous knowledge P, or, where this is not possible, by performing tidlist intersections over Δ. ZIGZAG has three main steps: (1) recording novel transactions and discarding obsolete ones, (2) updating the maximal frequent itemsets in the updated database, (3) updating the support of the other frequent itemsets in the updated database. Determining association rules, as mentioned, is relatively simple if we have the frequent itemsets along with their supports. The first task is how to record the information associated with the novel transactions, and how to "forget" the information associated with the obsolete ones. Based on this information, we perform the second and third steps, that are finding all frequent itemsets and updating their supports in the updated database.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited

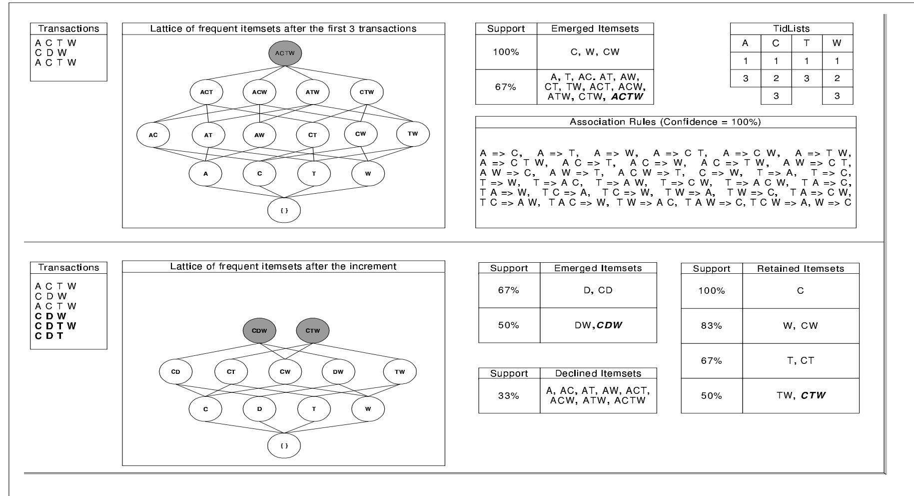


Figure 1. *Updating Process*

We illustrate the main operations of the algorithm execution in Figure 1. For this example, the value of minimum support is 50%. The shaded itemsets are the maximal frequent itemsets in each lattice. The top lattice is obtained by mining the original database D with three transactions. Here $ACTW$ is the sole maximal frequent itemset. Rules with 100% confidence are shown on the right.

The bottom lattice shows what happens when an increment of three new transactions is added to D. Here there is no decrement d^-, so that $\Delta = D \cup d^+$. With the same *minsup*, The first lattice shows **ACTW** as the updated lattice has **CTW** and **CDW** as the two new maximal frequent itemsets. Notice that even in this simple example, the update affected significantly the results, allowing us to grasp important characteristics of the updating process. First, it is possible that the set of maximal frequent itemsets changes completely as a consequence of the update, with corresponding changes in the frequent itemsets. Second, there may be completely novel items and itemsets that become frequent in the updated database.

The main goal of our algorithm is to reduce the number of candidate itemsets that are examined on the updated database Δ to decide whether they are frequent or not. As we will see later, by utilizing the stored information P, some itemsets can be examined just over d^+ and d^- in order to discover their support, potentially reducing the frequency computation costs. Further, as a very important characteristic of the maximal-based search employed by our algorithm, it is not necessary to explicitly examine every single frequent itemset, potentially reducing the number of candidates examined for finding all frequent itemsets.

3.1 Recording and Discarding Transactions

The novel transactions are recorded by creating, for each item in Δ, its tidlist in d^+, so that at the end of the process we have the vertical increment database projection. Similarly, we discard the obsolete transactions, by examining each item in Δ and creating its tidlist in d^-, forming the vertical decrement database projection.

The vertical projection of the updated database Δ also has to be updated with the novel transactions. So for each item in Δ we augment its tidlist with the transactions in d^+ and remove the tids in d^-, that contain this item, i.e., for an

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited

item X, its updated tidlist is given as $\mathcal{L}_\Delta(X) = (\mathcal{L}_D(X) \cup \mathcal{L}_{d^+}(X)) - \mathcal{L}_{d^-}(X)$.

This process may be performed on the fly, that is, whenever a transaction is added to or removed from the database, it is recorded in or discarded from the projections. Or, on the other hand one can perform block updates over a given interval (i.e., a set of transaction additions and deletions).

This strategy has several advantages for sake of incremental and short-term mining of association rules. First, by augmenting the tidlists, it is easy to determine which information is not the current update operation, since the tidlists will be always chronologically ordered. For the same reason it is also easy to determine and discard the obsolete information, performing short-term mining. Second, it is easy to identify emerging patterns and contrasts between the itemsets over mining operations. Since d^+ reflects the most recent events, we can also verify the impact of decisions like personalizations, in WEB mining applications, for instance.

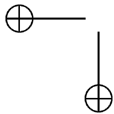
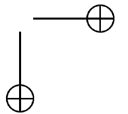
3.2 Determining the Frequent Itemsets

After recording the novel transactions and discarding the obsolete ones, we have to determine the frequent itemsets in the updated database Δ. The set of frequent itemsets in Δ is composed by the union of the set of retained itemsets and the set of emerged itemsets, and it can be solely determined by the maximal frequent itemsets in Δ [13].

In this step, ZIGZAG employs a search for the maximal frequent itemsets in Δ. An efficient maximal-based search must satisfy two main properties, as follows. First, the ability to perform fast frequency computation. This property is associated with the cost of processing a candidate itemset. To satisfy this property extant approaches generally utilize compressed structures for representation of the tidlist, like diffsets [13] or compressed bit vectors. Second, the ability to quickly remove large branches of the search space from consideration. This property is associated with the number of candidates generated in the search. The smaller the number of candidates generated, the faster the search would be. These two abilities determine the amount of work done in the search. As mentioned earlier, ZIGZAG employs a backtracking search inspired by GENMAX, which utilizes a set of techniques to enhance the frequency computation and the pruning effectiveness. However, better results can be achieved by making use of the incremental approach, and we will present how ZIGZAG achieves these properties in the next two sections.

Fast Frequency Computation Techniques While searching for the maximal frequent itemsets, ZIGZAG continuously generates partial solutions, and as a new partial solution arises, a new combine set must be computed to make possible to process extensions of this partial solution. In order to generate new combine sets, some frequency computations must be applied. To perform the frequency computation, ZIGZAG is based on the associativity of itemsets, which is defined as follows.

Let X be a k-itemset of items $X_1 \dots X_k$, where $X_i \in I$, $\mathcal{L}(X)$ be its tidlist and $\delta(X) = \mid \mathcal{L}(X) \mid$ is the length of $\mathcal{L}(X)$ and thus the support count of X in a given database. According to [7], any itemset can be obtained by joining its atoms (individual items) and the support can be obtained by intersecting the tidlist of each atom. Since we constructed the vertical projections of d^+ and d^-, and updated the vertical projection of Δ, we have free access to the tidlists of each atom in d^+, d^- and Δ, being able to compute the support of any itemset in d^+, d^- and Δ.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited

The main goal of this step is to maximize the number of itemsets that have their frequency computed based just on d^+ and d^-. These itemsets are called retained itemsets, and their support counts in D are already stored in P. In order to quickly discover if a given candidate is a retained itemset, we store the support counts of the retained itemsets using a hash table.

Combine_Set(I_{l+1}, S_{l+1})

$C = \emptyset$
For each $y \in S_{l+1}$ *do*
  $y' = y$
  If $I_{l+1} \cup \{y\}$ *is a retained itemset*
    then $\sigma(y') = \sigma_D(I_{l+1} \cup \{y\}) + \sigma_{d^+}(I_{l+1} \cup \{y\}) - \sigma_{d^-}(I_{l+1} \cup \{y\})$
  Else, $\sigma(y') = \sigma_\Delta(I_{l+1} \cup \{y\})$
  If $\sigma(y') \geq$ *minsup*
    then $C = C \cup \{y'\}$
return C

Algorithm 3.1: **Optimized Generation of the Combine Set**

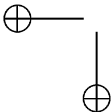
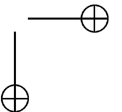
We describe the routine where the frequency of an extension is tested in Algorithm 3.1. The partial solution I_{l+1} is combined with each item y in the possible set S_{l+1}, generating an extension. If this extension is frequent then y' is valid and must be added to the new combine set.

In order to decide if a given extension is valid, we must check its support. To perform a fast frequency computation, we first verify if the extension $I_{l+1} \cup \{y\}$ is a retained itemset. If so, as mentioned, its frequency can be computed just over d^+, d^-, and using P, thereby enhancing the frequency computation process.

Pruning Techniques Two general principles for efficient search using backtracking are that: (1) It is more efficient to make the next choice of a branch to be the one whose combine set has the fewest items. This usually minimizes the number of candidates generated. (2) If we are able to remove a node as early as possible from the backtracking search tree, we effectively prune many branches from consideration.

Reordering the elements in the current combine set to achieve these two goals is a very effective mean of cutting down the search space. The basic heuristic employed by GenMax (and also in MaxMiner [14]) is to sort the combine set in increasing order of support; it is likely to produce small combine sets in the next level, since the items with lower support are less likely to produce frequent itemsets.

Guided by the previous mining results, ZIGZAG can continuously order the elements in the combine sets generated in the subsequent levels of the search space, since it has free access to estimations of the itemset supports that can be generated in the search, potentially capturing as early as possible some changes on the dependencies between the partial solution and its combine set. This allows ZIGZAG to get a better ordering of elements to produce smaller branches. In fact, ZIGZAG can use previous information to compute *correlation measures*, between a partial solution and items in the combine set, instead of support. Correlation can be used to generate statistical dependences for both the presence and absence of items (i.e., items of the combine set) in an itemset (i.e., a partial solution), and its value can be computed by: $\frac{\sigma_D(\alpha \cup x)}{\sigma_D(\alpha) * \sigma_D(x)}$, where α is a partial solution and x is an item in the combine set. If one sorts the combine set in increasing order of correlation, one

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited

produces small combine sets at the next level, leading to a large degree of pruning.

The use of both optimizations does not affect the correctness of the algorithm, since the reordering of the combine set just changes the order in which the maximal frequent itemsets are generated. However, these optimizations drastically improve the efficiency of the search, reducing the number of candidates generated, since the next choice of a branch to explore in the backtracking search is likely to be good approximation of the best choice while mining Δ. In summary, the main idea behind the efficiency of the search employed by ZIGZAG stems from the fact that it eliminates branches that are subsumed by an already mined maximal frequent itemset, and it is very likely that the maximal frequent itemsets that are generated earlier subsume a large number of candidate itemsets that would be generated if the order in which these maximal patterns were generated was different.

All the discussed steps are now described in Algorithm 3.2.

Opt_Gen_Max(I_l, C_l, l)

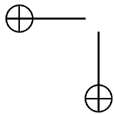
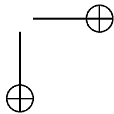
For each $x \in C_l$ *do*
  $I_{l+1} = I \cup \{x\}$
  $S_{l+1} = \{y : y \in C_l \text{ and } y > x\}$
  If $I_{l+1} \cup S_{l+1}$ *has a superset in MFI then return*
  C_{l+1} = *Combine_Set*(I_{l+1}, S_{l+1})
  If C_{l+1} *is empty*
    then If I_{l+1} *has no superset in MFI then MFI* = *MFI* $\cup I_{l+1}$
  Else
    Sort C_{l+1} *based on the correlation between* I_{l+1} *and each item of* C_{l+1}
    Opt_Gen_Max(I_{l+1}, C_{l+1}, $l+1$)

Algorithm 3.2: **Optimized Search for Maximal Frequent Itemsets**

3.3 Updating the Support of the Frequent Itemsets

Just determining the maximal frequent itemsets is not sufficient to generate all association rules, since we do not have the support associated with each subset. This is the main negative aspect of all the extant maximal-based approaches, since a complete scan over the entire database is required in order to compute the frequency of each subset. Considering that in a dynamic environment the entire database generally is enormous, this step can be time consuming.

To avoid scanning the entire database for frequency computation of all subsets, again ZIGZAG makes use of both maximal and incremental approaches, traversing the frequent itemset lattice in a top-down fashion as follows. It breaks each maximal frequent k-itemset into k subsets of size $(k-1)$. If the frequent subset generated is a a retained itemset, its frequency in Δ is computed just over d^+ and d^-, by summing its already known support count in D to its support count in d^+, and subtracting its support in d^-. Otherwise, if the subset generated is an emerged itemset, its frequency has to be computed over the entire updated database. The incremental approach makes this top-down enumeration very efficient since we have the support counts in D of a possibly large number of subsets generated (i.e., all the retained itemsets), avoiding performing intersections over the entire database to determine them. We just need to perform intersections over d^- and d^+. This process iterates generating smaller subsets and computing their frequencies until there are no more subsets to be checked. We illustrate the entire process in Algorithm 3.3.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited

Update_Subsets_Support(θ)

For each subset $\beta \subseteq \theta$ *do*
If $\sigma(\beta)$ *was not updated yet then*
If β *is a retained itemset then* $\sigma(\beta) = \sigma_D(\beta) + \sigma_{d^+}(\beta) - \sigma_{d^-}(\beta)$
Else, $\sigma(\beta) = \sigma_\Delta(\beta)$
Update_Subsets_Support(β*)*

Algorithm 3.3: **Support Update Process**

Nevertheless, for some really dense databases, even this incremental updating process becomes unfeasible, since the number of subsets can grow too large. In these scenarios, one possible strategy is to *relax* the updating process, as follows: If the popularity variation (i.e., support) of a given itemset is above a *relaxation threshold*, then it is necessary to update its subsets. On the other hand, if the popularity of the itemset did not change much, we only adjust its popularity according to the same relative variation observed in its superset, making its support a good approximation of its real support in Δ. Later on, when sufficient change happens (i.e., exceeds the relaxation threshold), the set, along with its subsets, will be updated with their exact frequency.

All steps of ZIGZAG are presented in Algorithm 3.4.

ZigZag(S, D, d^+, d^-, *minsup*, *minconf***)**

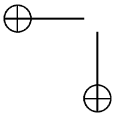
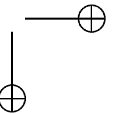
Scan P *and recover the information collected in the previous mining.*
Scan d^+ *and* d^-*. Create the incremental and decremental vertical projection.*
Update the vertical projection of Δ*.*
F_1 = Frequent items in Δ.
Sort F_1 *in increasing order of support.*
MFI $= \emptyset$
Opt_Gen_Max($\emptyset$*,* F_1*, 0)*
For each $\theta \in$ *MFI do*
Update_Subsets_Support(θ*)*
Generate the association rules with confidence $\geq$ *minconf*

Algorithm 3.4: **General Description of** ZIGZAG

3.4 Discussion

The gains provided by ZIGZAG vary according to the differences between D and Δ, which determine the number of retained, declined and emerged itemsets. Since the retained itemsets enhances both the frequency computation and the pruning effectiveness, the more the number of retained itemsets is, the better the efficiency would be, and vice versa. At first, the number of declined itemsets does not affect significantly the performance of the algorithm, since these itemsets are not processed during the search, but they correspond to an unnecessary use of storage resources.

For instance, if the differences between D and Δ are not significant, it is also very likely that the continuous ordering of the combine sets makes the search much more efficient, since it may correct some "wrong" paths visited while mining the original database D. On the other hand, if we observe a significant variation between D and Δ, the performance of ZIGZAG is clearly bounded by the performance of GENMAX. First because, like GENMAX, ZIGZAG will follow the same combine set

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited

ordering. Second because, like GenMax, all the candidates generated by ZIGZAG in order to find the maximal frequent itemsets will be checked over the entire database.

Finally, typical algorithms for updating the frequent itemsets in dynamic databases make use of positive or negative borders [11] to perform the update while reducing the number of candidates generated. However, these algorithms must update the entire border across mining operations in order to give the correct results, and depending on the size of the border employed and on the variations between D and Δ, this update is likely to be even more costly than our incremental search for maximal frequent itemsets. First because more candidates will be generated and more itemsets will be checked over the entire database for being frequent. Second because the declined itemsets must be also computed in order to decide if it is really not frequent anymore, corresponding to unnecessary computation, since they do not contribute to the set of frequent itemsets in the updated database.

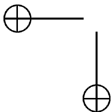
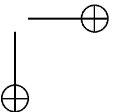
4 Experimental Results

In this section we present the results of the performance evaluation of ZIGZAG using several performance metrics and compare its performance against a state-of-the-art incremental algorithm [16]. Both synthetic and actual datasets were used as inputs in the experiments. The synthetic datasets were generated using the procedure decribed in [2]. The synthetic dataset, called T25.I10.D100K, comprises 100,000 transactions over 1,000 unique items, each transaction has 25 items on average, and the size of the longest maximal frequent itemset is 12. We also evaluated the performance of our approach using real data from actual applications. The real dataset, VBook, was extracted from a 45-day log from an electronic bookstore and comprises approximatelly 100,000 customer transactions. A transaction contains the books examined, added to the shopping cart and/or bought by a customer during a single visit to the bookstore. The second actual dataset is called Gazelle and comes from click-stream data from a small dot-com company called Gazelle.com, a legware and legcare retailer, which no longer exists. A portion of this dataset was used in the KDD-Cup 2000 competition. It has 59,601 transactions over 498 items with an average transaction length of 2.5 and a standard deviation of 4.9. All the datasets were divided into equal-sized non-overlapping partitions, which size is equal to the smallest interval employed in the experiments (1,000 transactions).

We varied two parameters in our experiments: minimum support and update interval. Thus, for each minimum support employed we performed multiple executions of the algorithm, where each execution employs a different update interval. The experiments were run on a PC 750MHz processor with 1GB main memory. The source code for ULI used in the experiments was kindly provided to us by its authors. Timings in the figures are based on total wall clock time.

Incremental Mining Figure 2(a) shows the performance of ZIGZAG on the real dataset Gazelle, for two different minimum supports and different update intervals. Note that the vast majority of maximal frequent itemsets in this dataset are of length 2, and they represent approximatelly 10% of all frequent itemsets.

We can observe that, in general, the shorter the update interval is, the smaller the update execution time. Nevertheless, its higher update frequency may result in higher overall costs. In fact, the best update interval also depends on the minimum support employed and will vary as the database grows.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited

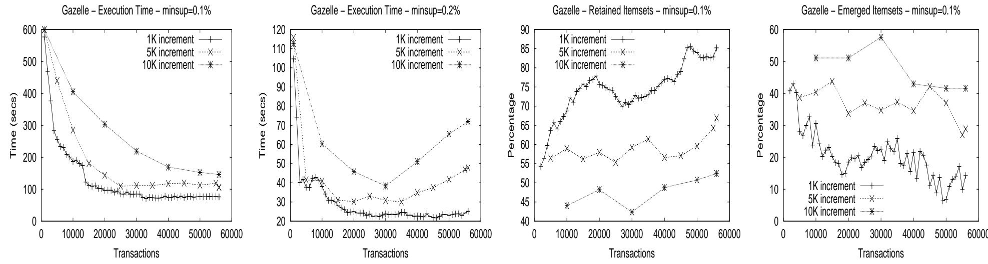


Figure 2. *a) Total Execution Time (left graphs), b) Retained and Emerged Itemsets (right graphs)*

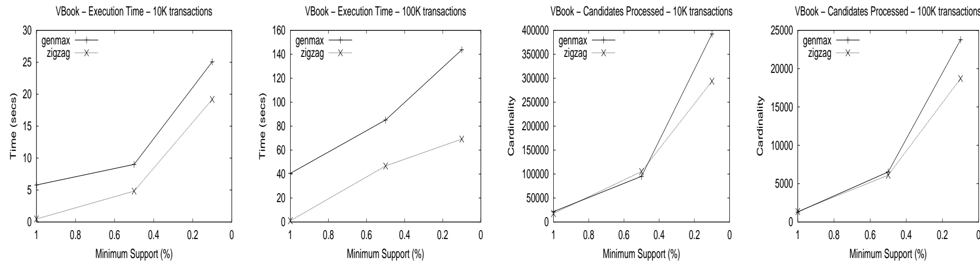


Figure 3. *a) Execution Time (left graphs), b) Itemsets Processed (right graphs)*

We can also observe that, after a given number of transactions, the execution time of processing increments tends to stabilize. This phenomenon is due to the large number of retained itemsets and the small number of emerged ones, as can be seen in Figure 2(b). This result means that a large number of operations have been performed just over d^+, explaining the constant execution time observed, since the increment has a fixed size.

Supporting Interactive and Incremental Processing As discussed earlier, ZIGZAG supports incremental processing hand-in-hand with interactive operations. We evaluate the performance of this here. In these experiments we use the real dataset, VBook, and measure the time spent to find the maximal frequent itemsets. We compare our algorithm against the base-line non-incremental version, GENMAX. Note that, at any given iteration (after the first one) ZIGZAG has information about the data, and on a data update, if a large number of the frequent itemsets are retained, then it can re-use this information effectively.

Figure 3(a) shows the time spent by each algorithm to discover the maximal frequent itemsets. In the first iteration almost all candidates generated by ZIGZAG are retained itemsets. Since there is no increment, the support of a retained itemset is not modified, and ZIGZAG almost instantaneously re-discovers the updated solution. In the following iterations, as the minimum support decreases, there may be some emerged itemsets, and ZIGZAG will have to compute them over the entire dataset. Even so, as we can see in the figure, there is a great improvement provided by the retained itemsets, since we do not have to compute their frequency.

The improvement provided by ZIGZAG does not come just from the optimized frequency computation. The dynamically adapting pruning strategies are also quite effective. From Figure 3(b), we can see that almost always ZIGZAG processes a smaller number of candidates (anywhere from 10-20% less) than GENMAX.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited

We next consider the impact of data updates on the performance of these two approaches. Figure 4(a) depicts the speed up achieved by by ZIGZAG over GENMAX at different minimum support values for 1K increments. The speed up is higher for smaller support values. The gains provided by ZIGZAG comes also from the pruning technique utilized while searching for the maximal frequent itemsets. As seen before, ZIGZAG evaluates far fewer candidates when compared to GENMAX, as shown in Figure 4(b), for different update intervals. This result once again highlights the importance of the pruning strategies in ZIGZAG.

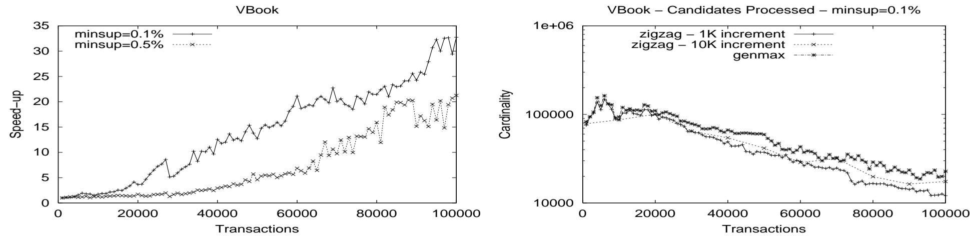


Figure 4. *a)* ZIGZAG *speed up relatively to* GENMAX, *and b) Itemsets Processed*

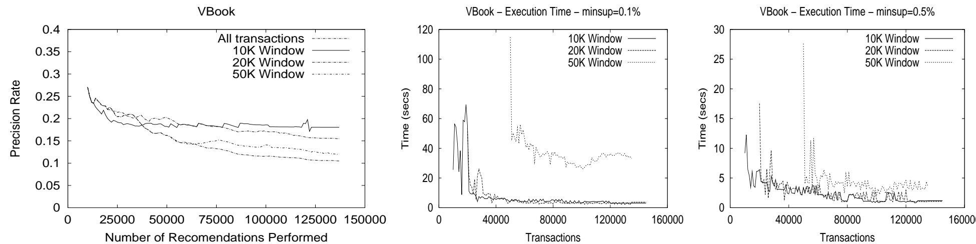


Figure 5. *a) Coherence with Recent Data, and b) Variyng Window Size*

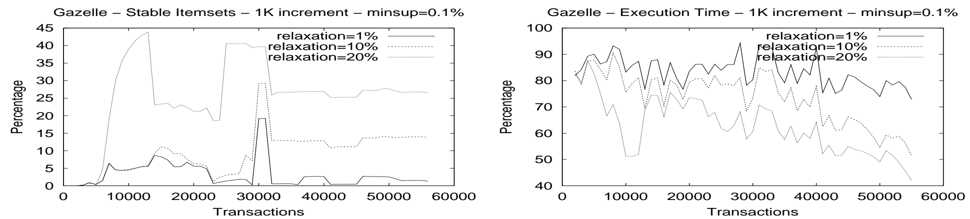


Figure 6. *a) Stable Itemsets (left graph) and b) Execution Time (right graph)*

Effectiveness of Windowed Operations on Real Data To demonstrate the effectiveness of short-term mining operations, we use a real example where the transactions vary significant over the time. Our goal is to recommend books to customers based on the VBook dataset. Here, the items selected by the customer, while performing a transaction, form an itemset. For each possible recommendation (ordered by support) we check its correlation with the appraised itemset, and recommend the first item that exceeds a correlation threshold. For each recommendation performed, we verify whether the customer really selects the book recommended. We used three different window sizes for this experiment (10K,20K,50K). We compared the recommendation precision using each size against the base case (no windowing in effect). The precision is given by the rate of correct recommendations performed. As we can see in Figure 5(a), the best window size varied as the dataset evolves. However, after a certain number of transactions processed and recommendations

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited

performed, we observed that the best window size is the one with the smallest fixed size employed. Furthermore, the precision reached when all transactions are relevant is going down, while the precision obtained by the short-term mining operation becomes constant.

The execution time also depends on the window size employed. As shown in Figure 5(b), the smaller the window size the faster the execution time, because for smaller windows there are less relevant transactions to be processed. Also the execution time becomes constant as the dataset evolves.

Impact of Tracking Stable Associations In this experiment we evaluate the ability of our algorithm to track stable associations and its use in approximately updating the maximal frequent itemsets as a function of data updates. Figure 6 documents these results for the Gazelle dataset. The Y axis of Figure 6(a) represents the percentage of frequent itemsets that remains stable as a result of this optimization for various relaxation thresholds. Interestingly, for this dataset this percentage stabilizes after a certain number of transactions (35,000 in this case) for all the relaxation thresholds. The number of missing associations for the smallest threshold is under 3% for a relaxation threshold of 1%. The corresponding execution times under these relaxation thresholds, compared to the execution times obtained whithout the relaxation, is also shown in Figure 6(b). Clearly, the higher the threshold value, the lower the execution time. For the lowest threshold value (1%) the average improvement in execution time is around 20%.

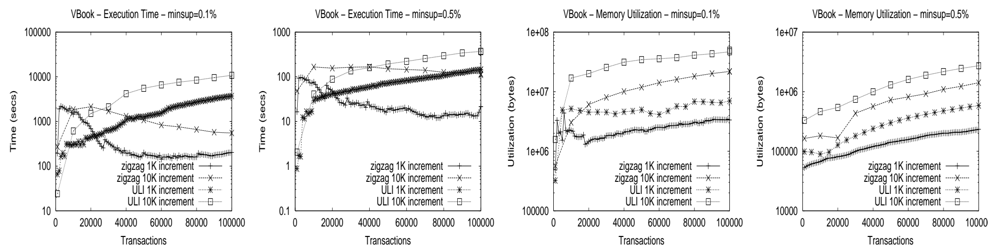


Figure 7. *a) Execution Time (left graphs) and b) Memory Usage (right graphs)*

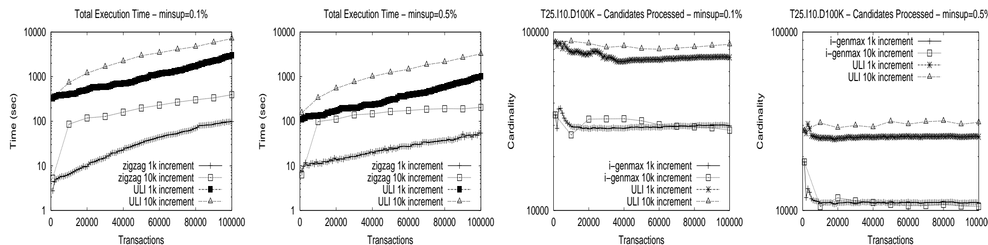


Figure 8. *a) Execution Time (left graphs), b) Itemsets Processed (right graphs)*

Comparative Results In this section we present the comparison of ZIGZAG against ULI using different values of minimum support and update interval. We employed two different datasets, VBook and T25.I10.100K. ULI strives to reduce the I/O requirements for updating the set of frequent itemsets by maintaining the previous frequent itemsets and the negative border along with their support counts. The whole database is scanned just once, but the incremental database must be scanned as many times as the size of the longest frequent itemset.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited

Figure 7(a) shows the comparative performance of ZIGZAG and ULI on the dataset VBook. Note that the Y-axis represents a log scale of execution time to process each fixed sized increment (10K,1K). During the first 10,000 transactions this dataset generates a large number of maximal frequent itemsets, and ZIGZAG has to perform a lot of operations to find them. As the dataset evolves, the number of changes to the set of maximal frequent itemsets decreases, and ZIGZAG has significantly better performance than ULI(an order of magnitude), even for larger update intervals. Further, we can observe that ZIGZAG provides higher gains for smaller support values, since it is less costly to run ULI for larger support values.

Using a pure vertical database format to compute the frequent 2-itemsets, ZIGZAG will perform $O(n^2)$ operations, in the worst case, where n is the number of frequent items. On the other hand, the horizontal approach performs only $O(n)$ operations. The pure vertical approach is clearly not efficient, since the intermediate results of these operations will not fit in memory for large databases. To overcome this problem, ZIGZAG computes the set of frequent items and the 2-itemsets, using a vertical-to-horizontal recovery method [13]. Figure 7(b) shows the peak of memory usage of the two algorithms per update operation. As we can see, ZIGZAG uses less memory than ULI for the same support threshold and update interval. As expected, the smaller the support value employed, the more memory will be used to compute the intermediate results. The updated interval also affects the memory utilization, since the incremental projections will be larger. Finally, the size of the dataset is also important, since the size of the tidlists used as intermediate results to compute the emerged itemsets increases linearly with the dataset size.

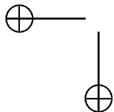
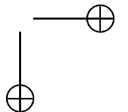
We next evaluated both approaches on a synthetic dataset T25.I10.D100K. This dataset T25.I0.D100K has a huge negative border for smaller support values. Figure 8(a) shows the execution times of ZIGZAG and ULI. ZIGZAG performs significantly better than ULI for this type of database. As we can see, it clearly out performs ULI by more than an order of magnitude. Further, ZIGZAG is able to handle lower support values in dense datasets.

As an important characteristic of a maximal-based search, the number of candidates processed is greatly reduced. Figure 8(b) shows the number of candidates processed by each algorithm. This result validates the gains provided by the search employed by ZIGZAG.

5 Summary

In this paper we presented ZIGZAG, a new algorithm for mining frequent itemsets in evolving databases. Our approach maintains only the maximal itemsets to incrementally construct the lattice of frequent itemsets. The maximal frequent itemsets are updated by a backtracking search, which is guided by the results of the previous mining iteration, yielding a very efficient way for determining frequent itemsets, when compared with current alternatives. The approach is also very efficient in supporting drill-down/drill-up interactions based on modifying user parameters.

ZIGZAG can support short-term mining, wherein old transactions may be discarded from the database and new ones added, keeping the set of association rules coherent with respect to the most recent data. Short-term mining is desirable for applications such as WEB mining, where the user behavior may vary significantly across time and old access patterns may be no longer relevant. Another important feature of our approach is its ability to track and use stable association patterns.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited

Bibliography

[1] R. Agrawal, T. Imielinski, and A. Swami. Mining Association Rules between Sets of Items in Large Databases. In *Proc. of the ACM-SIGMOD Intl. Conf. on Management of Data*, May 1993.

[2] R. Agrawal and A. Swami. Fast Algorithms for Mining Association Rules. In *Proc. of the 20th Intl. Conf. on Very Large Data Bases*, September 1994.

[3] N. Ayan, A. Tansel, and E. Arkun. An Efficient Algorithm to Update Large Itemsets with Early Pruning. In *Proc. of the 5th ACM-SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining*, August 1999.

[4] D. Cheung, J. Han, V. Ng, and C. Y. Wong. Maintenance of Discovered Association Rules in Large Databases: An Incremental Updating Technique. In *Proc. of the 12th Intl. Conf. on Data Engineering*, February 1996.

[5] S. Brin, R. Motwani, J. Ullman, and S. Tsur. Dynamic Itemset Counting and Implication Rules for Market Basket Data . In *Proc. of the ACM-SIGMOD Intl. Conf. on Management of Data*, 1997.

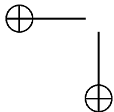
[6] S. Thomas and S. Chakravarthy. Incremental Mining of Constrained Associations. In *Proc. of the 7th Intl. Conf. of High Performance Computing (HiPC), p. 547-558*, December 2000.

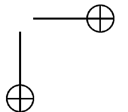
[7] M. J. Zaki, S. Parthasarathy, M. Ogihara, and W. Li. New Algorithms for Fast Discovery of Association Rules. In *Proc. of the 3rd Intl. Conf. on Knowledge Discovery and Data Mining*, 1997.

[8] S. Parthasarathy, M. Zaki, M. Ogihara, S. Dwarkadas. Incremental and Interactive Sequence Mining, In *8th International Conference on Information and Knowledge Management*, November 1999.

[9] D. Cheung, S. D. Lee, and B Kao. A General Incremental Technique for Maitaining Discovered Association Rules. In *Proc. of the 5th Intl. Conf. on Databases Systems for Advanced Applications*, April 1997.

[10] S. D. Lee, and D. Cheung. Maintenance of Association Rules: When to Update? In *Proc. of SIGMOD WorkShop on Research Issues in Data Mining and Knowlwdge Discovery* , May 1997.





Copyright © by SIAM. Unauthorized reproduction of this article is prohibited

[11] H. Mannila and H. Toivonen. Levelwise search and borders of theories in knowledge discovery. *Technical Report TR C-1997-8, Dept. of Computer Science*, U. of Helsinki, Jan. 1997.

[12] B. Liu and Y. Ma Analyzing the Interestingness of Association Rules from the Temporal Dimension. *In 1st IEEE International Conference on Data Mining*, November 2001.

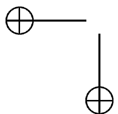
[13] K. Gouda and M. J. Zaki Efficiently Mining Maximal Frequent Itemsets. *In 1st IEEE International Conference on Data Mining*, November 2001.

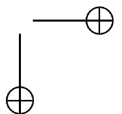
[14] R. J. Bayardo. Efficiently mining long patterns from databases. In *Proc. of ACM-SIGMOD Intl. Conf. on Management of Data*, June 1998.

[15] M. J. Zaki and C. J. Hsiao. CHARM: An Efficient Algorithm for Closed association Rule Mining. *Technical Report 99-10, Computer Science Dept., Rensselaer Polytechnic Institute*, October 2000.

[16] S. Thomas, S. Bodagala, K. Alsabti and S. Ranka. An Efficient Algorithm for the Incremental Updation of Association Rules in Large Databases. *In Proc. of the 3rd International conference on Knowledge Discovery and Data Mining (KDD 97)*, August 1997.

[17] R. Feldman, Y. Aumann, A. Amir and H. Mannila. Efficient Algorithms for Discovering Frequent Sets in Incremental Databases. In *Proc. of SIGMOD WorkShop on Research Issues in Data Mining and Knowlwdge Discovery* , May 1997.





Copyright © by SIAM. Unauthorized reproduction of this article is prohibited