Parallel, Incremental and Interactive Mining for Frequent Itemsets in Evolving Databases

Adriano Veloso
Wagner Meira Jr.
Márcio Bunte de Carvalho
Computer Science Department
Universidade Federal de Minas Gerais
Belo Horizonte – MG – Brazil
{adrianov,meira,mlbc}@dcc.ufmg.br

Srinivasan Parthasarathy
Computer and Information Science Department
The Ohio-State University
Columbus – OH – USA
srini@cis.ohio-state.edu

Mohammed Zaki
Computer Science Department
Rensselaer Polytechnic Institute
Troy – NY – USA
zaki@cs.rpi.edu

Abstract

This paper deals with new approaches to maintaining frequent itemsets in evolving databases. Our new approaches make use of incremental techniques to provide significant I/O reduction, and parallel techniques to provide computational savings. At the same time, our approaches are able to effectively handle online data updates (deletions/insertions) and interactive response times (approximate/partial results). Some additional highlights of the proposed approaches include extending the validity of the itemsets (generating approximate models of itemsets), and performing selective updates (tracking stable and predictable itemsets). These features allow our approaches to mine evolving data stored in warehouses as well as (potentially) streaming data. Extensive experimental benchmarking on evolving data demonstrates the potential advantages of the proposed approaches. We believe that this work can have high impact in application areas such as electronic commerce, web mining, and network intrusion detection.

1 Introduction

Stimulated by progress in computer technology and electronic data acquisition, recent years have seen the growth of huge databases, in fields ranging from supermarket sales and banking, through astronomy, chemistry, medicine and biology. These databases are viewed as critical resources. There is much valuable information hidden in them. The need of the hour is to extract it efficiently.

Data mining and knowledge discovery (KDD) lies at the interface of statistics, database technology, machine learning, high-performance computing and other areas. It is concerned with the computational and data intensive process of deriving interesting and useful information or patterns from massive databases. An important data mining task is frequent itemset mining. *Frequent* itemsets are used most often to generate correlations and association rules [1], but more recently they have also been used in such far-reaching domains as bio-informatics [7, 12] and information retrieval [8].

Several problems arise in the itemset discovery task, mostly as a consequence of the large size of the databases involved in this process. Moreover, many organizations today have more than large databases; they have databases that change and grow continuously. For example, retail chains record millions of transactions, telecommunications companies connect thousands of calls, and popular web sites log millions of hits.

In all these applications the evolving database is updated with a new block of data at regular time intervals. For large time intervals, we have the common scenario in many data warehouses. For small intervals, we have streaming data. Issues like interactivity and quick response times, are paramount. Providing interactivity and quick response times when the database is evolving is a challenging task, since changes in the data can invalidate the model of frequent itemsets and make data understanding and knowledge discovery difficult. Simply using traditional approaches to update the model can result in an explosion in the computational and I/O resources required.

Recognizing the dynamic nature of most operational databases, much effort has been devoted to the problem of mining frequent itemsets in evolving databases and several researchers [10, 6, 9, 2, 3, 11] have proposed interesting solutions and efficient algorithms. Generally these algorithms are incremental in the sense that they re-use previously mined information and try to combine it with the fresh data to reduce I/O requirements. Also there are efficient parallel approaches [15] designed to reduce computational requirements.

In this paper we present a new approach for mining frequent itemsets in evolving databases. Our approach differs from existing approaches in several ways. To the best of our knowledge, it is the first approach that combines both parallel and incremental techniques to provide significant computational and I/O savings. These features are very effective to facilitate data understanding and knowledge discovery in evolving databases. Our approaches can be used to mine evolving data arriving in large blocks of transactions, or even streaming data. We present extensive experimental results, highlighting the efficiency of our approaches.

Related Work

There has been a lot of research in developing efficient algorithms for mining frequent itemsets. Most of them enumerate all frequent itemsets. There also exist methods which only generate frequent closed itemsets [13] and maximal frequent itemsets [5]. While these methods generate a reduced number of itemsets, they still need to mine the entire database in order to generate the model of frequent itemsets, and therefore these methods are not efficient in mining evolving databases.

Much effort has been devoted to the problem of incrementally mining frequent itemsets [6, 9, 10, 2, 3, 4]. Some of these algorithms cope with the problem of determining when to update, while the others simply treat arbitrary insertions and deletions of transactions. Lee [6] proposed the DELI algorithm, which uses statistical sampling methods to determine when to apply the updating process. Another interesting approach is DEMON [4], which helps adapt incremental algorithms to work effectively with evolving data by monitoring changes in the stream. Our approaches are different from all the above approaches in several ways. First, while these approaches need to perform $O(n)$ database scans (n is the size of the largest frequent itemset), our approaches require only one scan on the incremental database and only a partial scan on the original database. Second, we support selective updates, that is, instead of determining when to update the entire set of frequent itemsets, we determine which particularly itemsets need to be updated. But the main difference stands from the fact that our approaches are also parallel. The combination of incremental techniques, on-the-fly data stream analysis, and parallel techniques make our approach unique.

2 Problem Statement

The problem of mining frequent itemsets can be formally stated as: Let $\mathcal{I} = \{i_1, i_2, ..., i_m\}$ be a set of m distinct attributes, also called items. Each transaction $\mathcal{T}$ in the database $\mathcal{D}$, has a unique identifier (*tid*), and contains a set of items (itemset), such that $\mathcal{T} \subseteq \mathcal{I}$. An itemset with exactly k items is called a *k-itemset*. The set of *tids* which

contains a given itemset, say X, is called the *tidlist* of X, and is denoted as $\mathcal{L}_{\mathcal{D}}(X)$. The vertical projection of $\mathcal{D}$ is the set of the tidlists of all items in $\mathcal{D}$. An itemset is said to have a *support s* if *s*% of the transactions in $\mathcal{D}$ contain this itemset. An itemset is frequent if its support is no less than a user-specified frequency, called *minimum support*. 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 or *positive border*. The *negative border*, on the other hand, refers to the minimal infrequent itemsets. Mining frequent itemsets is a very hard problem. Given m items, there are potentially 2^m frequent itemsets. Some modest databases have few thousands of items and many hundreds of thousands of transactions. So, discovering the lattice of frequent itemsets requires a lot of computation power, memory and disk I/O.

This high computational and space cost may be acceptable when $\mathcal{D}$ is static since the discovery is done offline, and many mechanisms such as sampling, and memory and parallel computing techniques have been presented in the literature.

Incremental Model Maintenance

A model has a concrete definition in the context of data mining. It is a tool which summarizes the data and can be used to predict unknown future values of attributes of interest, based on known values of some attributes in the database. When the data is changing, the model loses accuracy and becomes obsolete. Model maintenance aims to maintain an accurate data mining model undergoing insertion and deletion of blocks of data.

The central idea of incremental model maintenance relies on the re-use of previously mined information to enhance the performance of future interactions by reducing the number of database scans. Let $s_{\mathcal{D}}$ be the minimum support threshold used when mining $\mathcal{D}$, and $L_{\mathcal{D}}$ be the set of frequent itemsets obtained. Let $\mathcal{P}$ be the information kept from the current mining operation that will be used to improve the next mining operation. Using as a starting point the original database $\mathcal{D}$, 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., $\Delta = (\mathcal{D} \cup d^+) - d^-$. An incremental model maintenance algorithm must find the set L_Δ, the frequent itemsets in Δ, with respect to a minimum support s_Δ and, more importantly, using $\mathcal{P}$ and minimizing access to $\mathcal{D}$ (the original database) to enhance the algorithm performance and scalability.

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

3 The ZIGZAG and WAVE Algorithms

In previous work [10] we presented the ZIGZAG algorithm, which is able to find L_Δ within practical time and memory constraints. This accomplishment is possible by using an incremental technique based on the MFI, an information lossless approach. This approach results in significant I/O savings, since the number of maximal frequent itemsets can be significantly smaller than the negative border.

3.1 Determining Frequent Itemsets

To find L_Δ, ZIGZAG first employs a backtracking search for the MFI in Δ. 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, P_j. Initially I is empty; it is extended one item at a time, as the lattice is traversed. The length of I is the same as the depth of the corresponding node in the search tree. Given a candidate itemset of length l, $I_l = \{i_0, i_1, ..., i_{l-1}\}$, the possible values for the next item i_l comes from a subset $C_l \subseteq P_l$ called the *combine set*. If $y \in P_l - C_l$, then nodes in the search tree with root node $I_l = \{i_0, i_1, ..., i_l - 1, y\}$ will not be considered by the backtracking algorithm. Each iteration of 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 extract the new possible set of extensions, P_{l+1}, which consists only of items in C_l that follow x. The new combine set, C_{l+1}, consists of those items in the possible set that produce a frequent itemset when used to extend I_{l+1}.

To efficiently perform this backtracking search, the algorithm must satisfy two main properties, as follows. First, the ability to perform fast support computation. This property is associated with the cost of processing a candidate. 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, faster the search would be. These two properties determine the amount of work done in the search. ZIGZAG utilizes a set of techniques to enhance support computation and pruning effectiveness.

Fast Support Computation – While searching for the MFI, ZIGZAG continuously generates candidates. For each

candidate generated a new combine set must be computed to make possible to process extensions of it. In order to generate new combine sets, some support computations must be applied. To perform support computation, ZIGZAG is based on the associativity of itemsets, which is defined as follows. Let X be a k-itemset of items $X_1 \ldots X_k$, where $X_i \in I$, and $\delta(X)$ the support of X. According to [14], 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 the vertical projections of d^+, d^-, and Δ were already constructed in the ZAG phase, ZIGZAG is able to compute the support of any itemset in d^+, d^- and Δ. The desire of course, is to maximize the number of retained itemsets. These itemsets have their support counts in $\mathcal{D}$ already stored in $\mathcal{P}$. To perform fast support computation we first verify if the support in $\mathcal{D}$ of the itemset is already stored in $\mathcal{P}$. If so, it is a retained itemset and its support can be computed just over d^+, d^-, and using $\mathcal{P}$.

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 explore 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.

Reordering the elements in the combine set to achieve these two goals is a very effective mean of cutting down the search space. The basic heuristic is to sort the combine set in increasing order of support; it is likely to produce small combine sets in the next level. However, traditional algorithms can apply just a static ordering once at the first level, since the supports of longer itemsets are not determined yet. As the dependencies between the partial solutions and the combine sets usually change after each iteration of the algorithm, the efficiency of this heuristic can goes down as the search evolves. Guided by $\mathcal{P}$, ZIGZAG can continuously order the elements in the combine sets generated in subsequent levels of the search space, since it has free access to estimates of the support of itemsets that can be generated in the search, potentially capturing as early as possible some changes on the dependencies between the partial solution and its respective combine set. This allows ZIGZAG to get a better ordering of elements to produce smaller branches.

3.2 Updating the Support of Frequent Subsets

To avoid scanning the entire database for support 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 an emerged itemset, its support has to be computed over Δ. Otherwise, if the subset generated is a retained itemset, its support is computed just over d^+ and d^-, by summing its already known support count in $\mathcal{D}$ to its support count in d^+, and subtracting its support in d^-. The incremental approach makes this top-down enumeration very efficient since we have the support counts in $\mathcal{D}$ of a possibly large number of subsets generated (i.e., all the retained itemsets), avoiding performing expensive operations over Δ to determine them. This process iterates generating smaller subsets until there are no more subsets to be checked.

In contrast to other incremental approaches [9, 2, 3, 4] which generally monitor changes in the database to detect the best moment to update the entire set of itemsets, we choose instead to perform selective updates, that is, the support of every single itemset is completely updated just when we cannot perform a good estimate of it anymore. We distinguish three types of itemsets, regarding its actual support:

Invariant: The support of the itemset does not change significantly as we add new transactions (i.e., it varies within a predefined threshold). This itemset indicates a stable pattern and its support does not need to be updated.

Predictable: It is possible to estimate the support of the itemset within a tolerance. Several approximation tools may be employed, from a traditional linear estimation to sophisticated time-series analysis algorithms.

Unpredictable: It is not possible, given a set of approximation tools, to obtain a good estimate of the support of the itemset.

The algorithm works in two phases. The first phase samples the tidlists associated with 1-itemsets whose union results in the itemset we want to estimate the support. The second phase analyzes the sampling in order to determine whether it is necessary to count the actual support of the itemset. Each of these phases is described in the following.

Support Sampling – The starting point of the support sampling are the tidlists associated with 1-itemsets, which are always up-to-date. Given two tidlists l_α and l_β associated with the itemsets α and β, we define that $\delta = \alpha \cup \beta$ and $l_\delta = l_\alpha \cap l_\beta$. Formally, our sampling is based on estimating the upper bound on the merge of two tidlists.

Support Estimation based on Linear Trend Detection – One of the most widespread trend detection technique is linear regression, which finds which itemsets follow a linear trend, i.e., which of them fit reasonably a function to a straight line. The model used by the linear regression is expressed as the function $y = a + bx$, where a is the y-intercept and b is the slope of the line that represents the linear relationship between x and y. In our scenario the x variable represents the number of transactions while the y variable represents the estimated support. The *method of least squares* determines the values of a and b that minimize the sum of the squares of the errors. To verify how good is the model generated by the linear regression, we must estimate the goodness-of-fit. The goodness-of-fit R^2 represents the proportion of variation in the dependent variable that has been explained or accounted for by the regression line. This R^2 indicator ranges in value from 0 to 1 and reveals how closely the estimated y-values correlate to its actual y-values. When the R^2 value is above a given threshold, the itemset is classified as predictable, and its support can be simply predicted using the linear regression model, rather than computed with expensive database scans. We refer to this as the WAVE algorithm [11].

4 Parallel Technique for Incremental and Interactive Mining

Incrementally mining frequent itemsets can still be a very costly computation process. Hence, there is a practical need to develop parallel incremental algorithms for this task. In this section we describe a parallel technique for incremental and interactive mining of frequent itemsets. Our parallel technique was designed for *shared-memory multi-processors*, that is, processors communicate through shared variables.

The technique is based on *adaptive tidlist interval distribution*. In this approach, each candidate has its tidlist divided into n partitions, where n is the number of processors. Therefore, each processor counts the support of the candidate against its own tidlist interval. Results from each processor are joined to obtain the resultant tidlist, which is stored in a shared structure in memory. This process is adaptive since the tidlist interval may vary from candidate to candidate. Figure 1 depicts the process of support counting for a candidate itemset. The tidlist A is partitioned among four processors $P1$, $P2$, $P3$ and $P4$. Each processor performs the intersection of its partition with the tidlist B. The results $R1$, $R2$, $R3$ and $R4$ are joined to form the resultant tidlist C.

This approach does not reduce pruning effectiveness neither estimation quality, since the reordering of the combine set is not affected and the *tids* of the resultant tidlist will still be chronologically ordered.

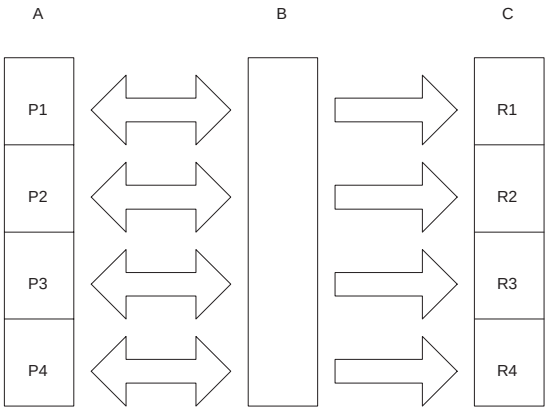


Figure 1: Parallel Support Counting. The tidlist intersection is made in parallel and the result of each processor is joined to obtain the resultant tidlist.

5 Evaluation

In this section we evaluate our algorithm and compare it to other approaches. A real database from an actual application were used as input in the experiments. WCup comes from click stream data from the official site of the 1998 World Soccer Cup. WCup was extracted from a 62-day log, comprising 2,128,932 transactions over 4,768 unique items with an average transaction length of 8.5.

5.1 Incremental Mining

The basic parameter of our evaluation is the block size, that is, the number of transactions added to the database which triggers a complete update. Thus, for each minimum support employed, we performed multiple executions of the algorithm, where each execution employs a different block size. Further, we employed two metrics in our evaluation:

Candidates Processed and Retained Itemsets: The total number of candidates processed and the number of retained itemsets in a given mining operation.

Execution Time: The elapsed time for mining the frequent itemsets in Δ. We compare the result with a non-incremental algorithm, ECLAT, to demonstrate the advantages of our algorithms for mining evolving databases.

The experiments were run on an *IBM - NetFinity* 750MHz processor with 512MB main memory.

From Figure 2 we can observe that the number of retained itemsets is always very close to the number of frequent itemsets. It means that a large number of operations were made just over d^+, reducing I/O requirements and improving the performance. The percentage of retained itemsets is larger for smaller block sizes, as expected. Analyzing Figure 3 we observe the advantages of incremental mining. When the evolving database reaches a certain size, ZIGZAG outperforms ECLAT by more than an order of magnitude. Most importantly, ZIGZAG can mine the database in a nearly constant time. The gains in performance are larger for smaller minimum supports and smaller block sizes.

5.2 Interactive Mining

Our evaluation is based on two parameters:

Approximation Tolerance $-R^2$**:** the maximum approximation error acceptable.

Block Size: the number of new transactions added to the database which triggers a complete update.

Thus, for each minimum support employed, we performed multiple executions of the algorithm, where each execution employs a different combination of R^2 and block size. Further, we employed two metrics in our evaluation:

Accuracy: This metric quantifies how good the approximated model is. It is the linear correlation of two ordered sets of itemsets. The ranking criteria is the support, that is, two ordered sets are totally correlated if they are of the same length, and the same itemset appears in corresponding positions in both sets.

Work: This metric quantifies the amount of work performed. We measure work as the elapsed time for mining the database. The work is the ratio between the time spent by WAVE and ZIGZAG algorithms.

We evaluate the accuracy and also the gains in accuracy provided by WAVE. We employed different minimum supports, block sizes, and R^2. Figure 4 depicts the accuracy achieved by WAVE in the WCup database. From this figure we can observe that, as expected, the accuracy increases with the R^2 used, and the accuracy increase with the block size used. Further, the accuracy decreases with the minimum support. We also verified how WAVE treats the trade-off between accuracy and work. From Figure 5 we can observe that WAVE performs less work for smaller values of minimum support. From Figure 6 we can observe that WAVE provides larger gains in accuracy for smaller values of minimum support. The opposite trend

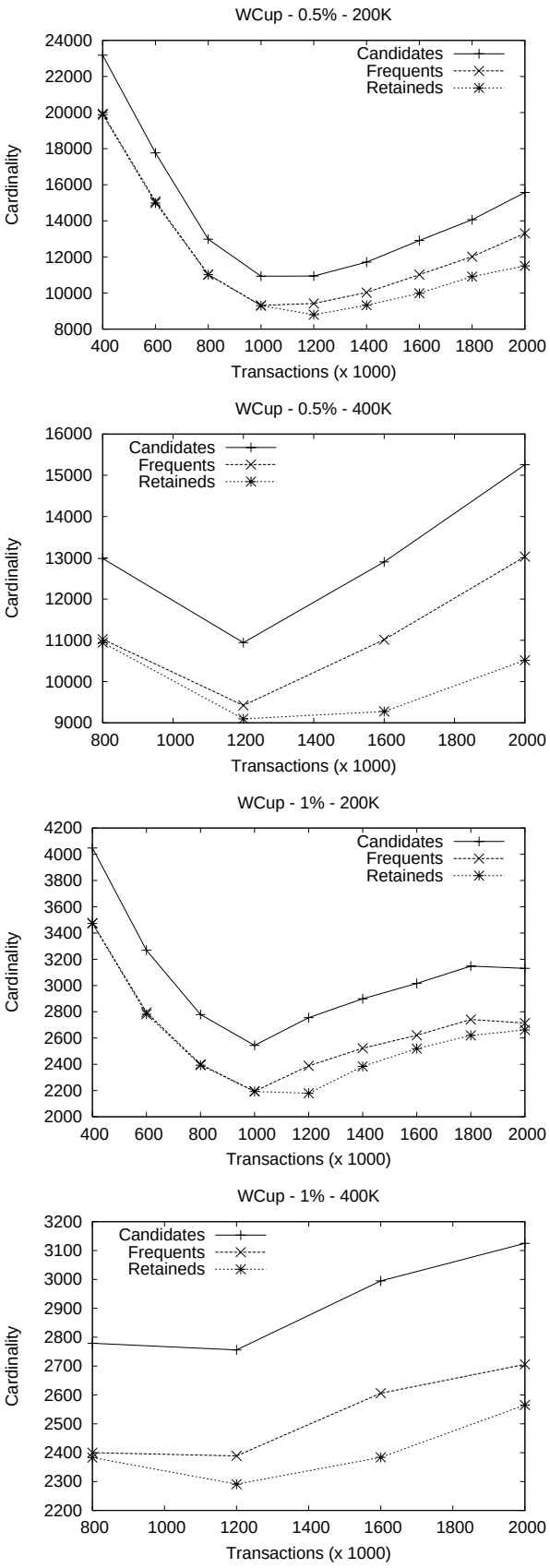


Figure 2: Total Number of Candidate, Frequent and Retained Itemsets.

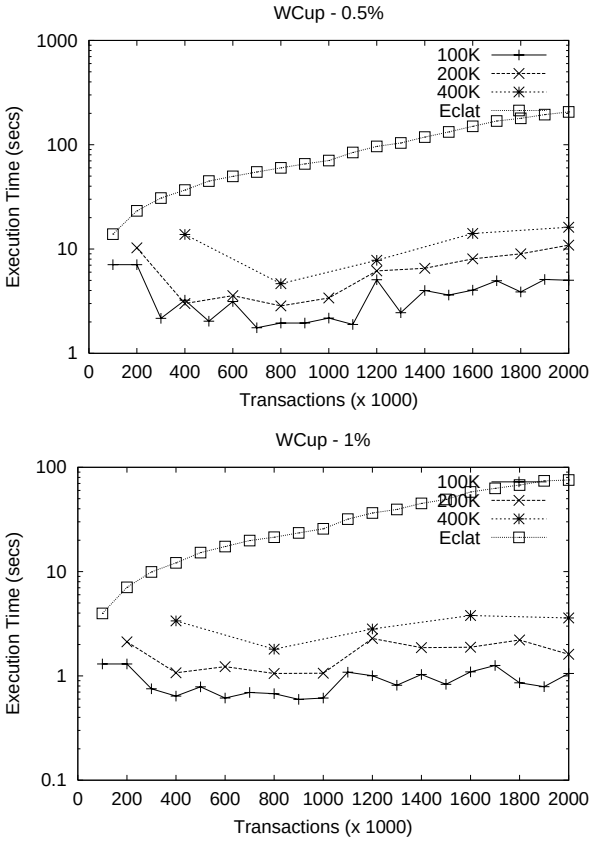


Figure 3: Total Elapsed Time for Mining Frequent Itemsets.

is observed when we evaluate the accuracy by varying the block size, that is, larger gains are achieved by larger blocks. Finally, the gain increases with the R^2 value.

5.3 Parallel and Incremental Mining

Now the aspect we want to study is the merit of parallelism in incremental mining. Our evaluation is based on the number of processors used in the mining operation. Thus, for each minimum support employed, we performed executions of the algorithm, where each execution employs a different number of processors. Further, the basic metric employed in our evaluation is the *speedup*, that is, the relative performance of our parallel algorithm and ZIGZAG, its serial version. The experiments were run on a 4-processor *IBM*. Each processor run 200MHz and has 256 MB main memory.

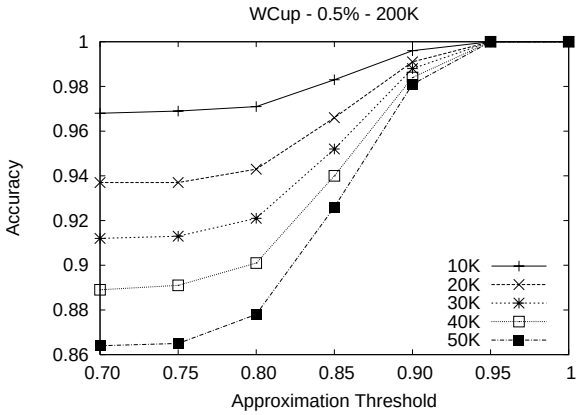
Figure 7 shows the percentage of the total execution time that is spent on the support counting task. As we can see, it is clearly the dominant task, spending always more than 90% of the total execution time. Figure 8 shows the relative response times. It can be seen that the speedup of our parallel algorithm is closer to the ideal speedup when we reduce the minimum support threshold. This is mainly because for smaller supports the tidlists are larger, and the parallel task becomes more significant.

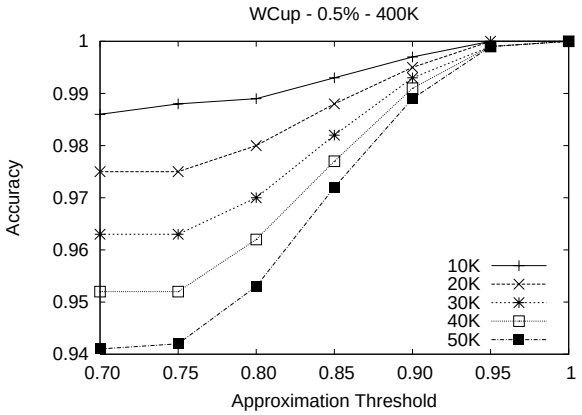
6 Conclusions and Future Work

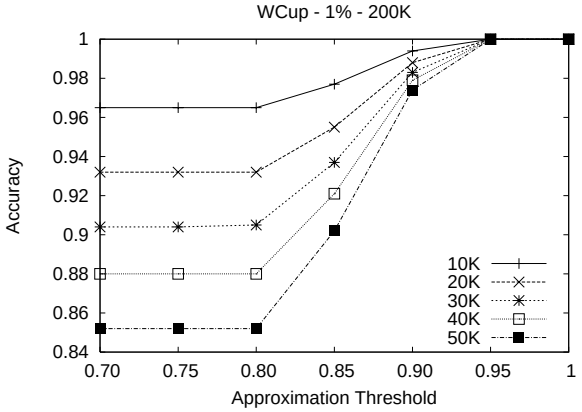
In this paper we presented techniques for efficiently mining frequent itemsets in evolving databases. Our techniques are incremental, interactive and parallel. The incremental approach makes use of the MFI to perform the incremental mining operations. The interactive approach makes use of *selective updates* to avoid updating the entire model of frequent itemsets. The parallel approach is based on the *adaptive tidlist interval distribution* technique, which continuously assigns partitions of the tidlist among the different processors. These techniques combined are able to mine from evolving data stored in warehouses to streaming data. We intend to continue this work evaluating the parallel algorithm in a distributed environment.

References

[1] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In *Proc. of the ACM SIGMOD Int'l Conf. on Management of Data*, Washington, USA, May 1993.







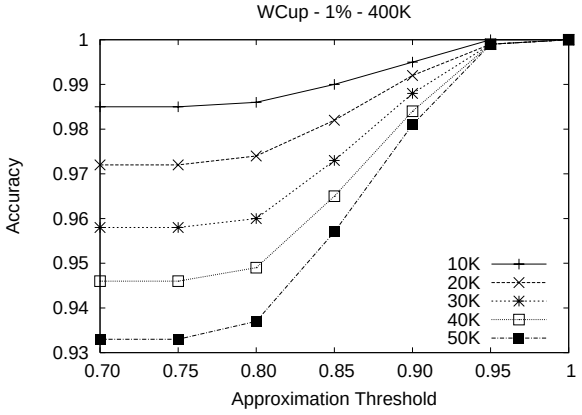
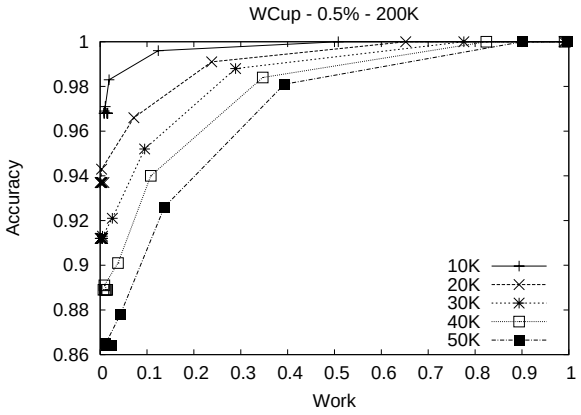
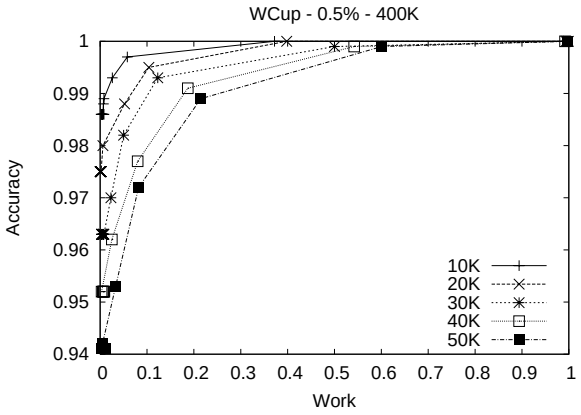
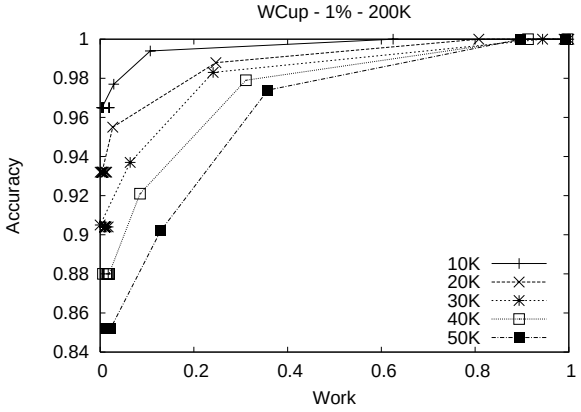


Figure 4: Accuracy of the Model generated by WAVE.







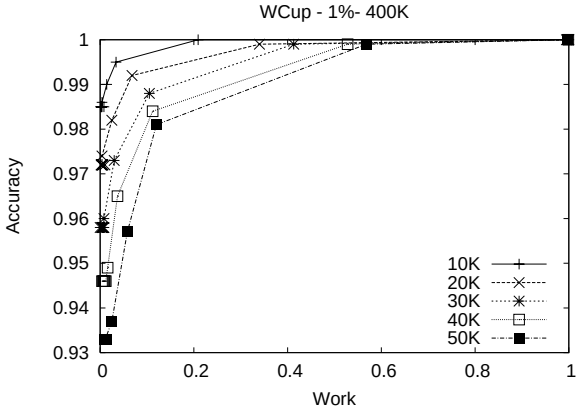


Figure 5: Trade-off between Accuracy and Work.

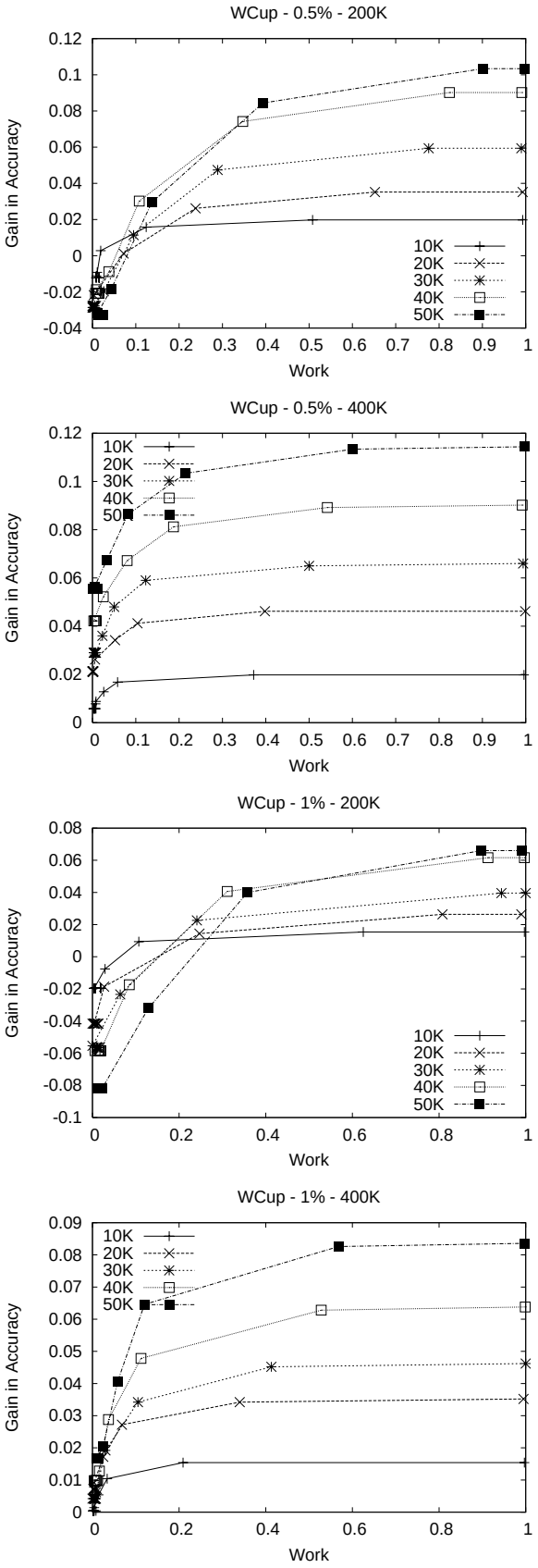


Figure 6: Accuracy Gains provided by WAVE.

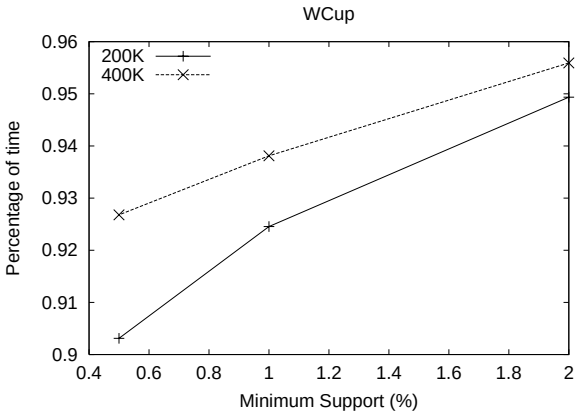


Figure 7: Time spent in Support Counting

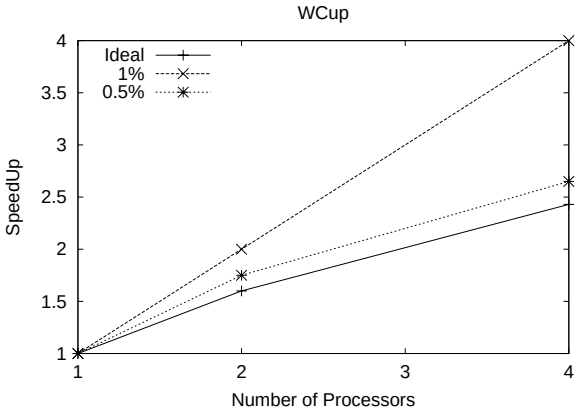


Figure 8: Speedup

[2] 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.

[3] D. Cheung, S. Lee, and B. Kao. A general incremental technique for maintaining discovered association rules. In *Proc. of the 5th Intl. Conf. on Database Systems for Advanced Applications*, pages 1–4, April 1997.

[4] V. Ganti, J. Gehrke, and R. Ramakrishnan. Demon: Mining and monitoring evolving data. In *Proc. of the 16th Int'l Conf. on Data Engineering*, pages 439–448, San Diego, USA, 2000.

[5] K. Gouda and M. Zaki. Efficiently mining maximal frequent itemsets. In *Proc. of the 1st IEEE Int'l Conference on Data Mining*, San Jose, USA, November 2001.

[6] S. Lee and D. Cheung. Maintenance of discovered association rules: When to update? In *Research Issues on Data Mining and Knowledge Discovery*, 1997.

[7] S. Parthasarathy and M. Coatney. Efficient discovery of common substructures in macromolecules. In *Proc. of the 3rd IEEE International Conference on Data Mining*, Maebashi City, Japan, December 2002.

[8] B. Possas, N. Ziviani, W. Meira Jr., and B. Ribeiro-Neto. Set-based model: A new approach for information retrieval. In *Proc. of the 25th International ACM SIGIR Conference on Research and Development in Information Retrieval*, Tempere, Finland, August 2002.

[9] S. Thomas, S. Bodagala, K. Alsabti, and S. Ranka. An efficient algorithm for the incremental updation of association rules. In *Proc. of the 3rd ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining*, August 1997.

[10] A. Veloso, W. Meira Jr., M. B. de Carvalho, B. Pôssas, S. Parthasarathy, and M. Zaki. Mining frequent itemsets in evolving databases. In *Proc. of the 2nd SIAM Int'l Conf. on Data Mining*, Arlington, USA, May 2002.

[11] A. Veloso, W. Meira Jr., M. B. de Carvalho, B. Rocha, S. Parthasarathy, and M. Zaki. Efficiently mining approximate models of associations in evolving databases. In *Proc. of the 6th Int'l Conf. on Principles and Practices of Data Mining and Knowledge Discovery in Databases*, Helsinki, Finland, August 2002.

[12] M. Zaki. Directions in protein contact map mining. In *NSF Workshop on Next Generation Data Mining*, Baltimore, USA, November 2002.

[13] M. Zaki and C. Hsiao. Charm: An efficient algorithm for closed itemset mining. In *Proc. of the 2nd SIAM Int'l Conf. on Data Mining*, Arlington, USA, May 2002.

[14] M. Zaki, S. Parthasarathy, M. Ogihara, and W. Li. New algorithms for fast discovery of association rules. In *Proc. of the 3rd ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining*, August 1997.

[15] M. Zaki, S. Parthasarathy, M. Ogihara, and W. Li. New parallel algorithms for fast discovery of association rules. *Data Mining and Knowledge Discovery: An International Journal*, 4(1):343–373, December 1997.