Efficiently Mining Approximate Models of Associations in Evolving Databases

Conference Paper *in* Lecture Notes in Computer Science · September 2002

DOI: 10.1007/3-540-45681-3_36 · Source: DBLP

6 authors, including:

Adriano Veloso
Federal University of Minas Gerais
220 PUBLICATIONS **4,526** CITATIONS
SEE PROFILE

Wagner Meira Jr.
Federal University of Minas Gerais
617 PUBLICATIONS **12,398** CITATIONS
SEE PROFILE

Marcio Carvalho
Federal University of Minas Gerais
23 PUBLICATIONS **202** CITATIONS
SEE PROFILE

Srinivasan Parthasarathy
Structural Engineering Research Centre
215 PUBLICATIONS **9,242** CITATIONS
SEE PROFILE

Efficiently Mining Approximate Models of Associations in Evolving Databases

A. Veloso[1], B. Gusmão[1], W. Meira Jr.[1], M. Carvalho[1], S. Parthasarathy[2], and M. Zaki[3]

[1] Computer Science Department, Universidade Federal de Minas Gerais, Brazil
{adrianov,gusmao,meira,mlbc}@dcc.ufmg.br
[2] Department of Computer and Information Science, The Ohio-State University, USA
srini@cis.ohio-state.edu
[3] Computer Science Department, Rensselaer Polytechnic Institute, USA
zaki@cs.rpi.edu

Abstract Much of the existing work in machine learning and data mining has relied on devising efficient techniques to build accurate models from the data. Research on how the accuracy of a model changes as a function of dynamic updates to the databases is very limited. In this work we show that extracting this information: knowing which aspects of the model are changing; and how they are changing as a function of data updates; can be very effective for interactive data mining purposes (where response time is often more important than model quality as long as model quality is not too far off the best (exact) model.
In this paper we consider the problem of generating approximate models within the context of association mining, a key data mining task. We propose a new approach to incrementally generate approximate models of associations in evolving databases. Our approach is able to detect how patterns evolve over time (an interesting result in its own right), and uses this information in generating approximate models with high accuracy at a fraction of the cost (of generating the exact model). Extensive experimental evaluation on real databases demonstrates the effectiveness and advantages of the proposed approach.

1 Introduction

One of the main characteristics of the digital information era is the ability to store huge amounts of data. However, extracting knowledge, often referred to as data mining, from such data efficiently poses several important challenges. First, the data operated on is typically very large, so the tasks are inherently I/O intensive. Second, the computational demands are quite high. Third, many of these datasets are dynamic (E-commerce databases, Web-based applications), in the sense that they are constantly being updated (evolving datasets).

Researchers have evaluated data stratification mechanisms such as sampling to handle the first problem and memory efficient and parallel computing techniques to handle the second problem. Simply re-executing the algorithms to handle the third problem results in excessive wastage of computational resources and often does not meet the stringent interactive response times required by the data miner. In these cases, it may not be possible to mine the entire database over and over again. This has motivated

the design of incremental algorithms, i.e., algorithms that are capable of updating the frequent itemsets, and thus the associations, by taking into account just the transactions recorded since the last mining operation. In this paper we propose an approximate incremental algorithm to mine association rules that advances the state-of-the-art in this area.

Association mining is a key data mining task. It is used most often for market basket data analysis, but more recently it has also been used in such far-reaching domains as bioinformatics [8], text mining [14] and scientific computing [7]. Previous research efforts have produced many efficient sequential algorithms[6, 1, 9, 18, 19, 21], several parallel algorithms[22, 13, 20, 3], and a few incremental algorithms for determining associations [16, 15, 2, 4].

Almost all incremental algorithms by and large employ specific data structures to maintain the information previously mined so that it can be augmented by the updates. These techniques are designed to produce exact results, as would be produced by an algorithm running from scratch. However, if response time is paramount, these algorithms may still be too expensive. What is needed is a way to efficiently estimate the association parameters (support, confidence) without computing them thus saving on both computational and I/O time. Our approach relies on extracting historical trends associated with each itemset and using these trends to estimate these parameters. For instance, if an itemset support is roughly constant across time, it may not be necessary to determine its exactly frequency. An approximate value may serve. Furthermore, if an itemset shows a consistent increase or decrease trend, we may also estimate its support as a function of the number of updates after the last actual count and the slope associated with the trend.

The main contributions of this paper can be summarized as follows:

- We propose an approximate incremental algorithm, WAVE, for mining association rules, based on discovering trends in itemset frequency actual databases over time.
- We evaluate the above algorithm along the traditional axes of quality (i.e., how close to the exact model) and performance (as compared against a state-of-the-art incremental algorithm) over several real datasets.

We begin by formally presenting the problem of finding association rules in the next section. In Section 3 we present our approach for mining approximate models of associations. The effectiveness of our approach is experimentally analyzed in Section 4. Finally, in Section 5 we conclude our work and present directions for future work.

2 Problem Description and Related Work

2.1 Association Mining Problem

The association mining task can be stated as follows: Let $\mathcal{I} = \{1, 2, ..., n\}$ be a set of n distinct attributes, also called items, and let $\mathcal{D}$ be the input database. Typically $\mathcal{D}$ is arranged as a set of transactions, where each transaction T has a unique identifier TID and contains a set of items such that $T \subseteq I$. A set of items $X \subseteq \mathcal{I}$ is called an itemset. For an itemset X, we denote its corresponding *tidlist* as the set of all $TIDS$ that contain X as a subset. The support of an itemset X, denoted $\sigma(X)$, is the percentage

of transactions in $\mathcal{D}$ in which X occurs as a subset. An itemset is frequent if its support $\sigma(X) \geq minsup$, where $minsup$ is a user-specified minimum support threshold.

An association rule is an expression $A \xrightarrow{p} B$, where A and B are itemsets. The support of the rule is $\sigma(A \cup B)$ (i.e., the joint probability of a transaction containing both A and B), and the confidence $p = \sigma(A \cup B)/\sigma(A)$ (i.e., the conditional probability that a transaction contains B, given that it contains A). A rule is frequent if the itemset $A \cup B$ is frequent. A rule is confident if $p \geq minconf$, where $minconf$ is a user-specified minimum confidence threshold.

Finding frequent itemsets is computationally and I/O intensive. Let $\mid \mathcal{I} \mid = m$ be the number of items. The search space for enumeration of all frequent itemsets is 2^m, which is exponential in m. This high computational cost may be acceptable when the database is static, but not in domains with evolving data, since the itemset enumeration process will be frequently repeated. In this paper we only deal with how to efficiently mine frequent itemsets in evolving databases.

2.2 Related Work

There has been a lot of research in developing efficient algorithms for mining frequent itemsets. A general survey of these algorithms can be found in [17]. Most of these algorithms enumerate all frequent itemsets. There also exist methods which only generate frequent closed itemsets [19] and maximal frequent itemsets [6]. While these methods generate a reduced number of itemsets, they still need to mine the entire database in order to generate the set of valid associations, and therefore these methods are not efficient in mining evolving databases.

Some recent effort has been devoted to the problem of incrementally mining frequent itemsets [10, 15, 16, 2, 4, 5]. A few of these algorithms cope with the problem of determining when to update the current model, while others update the model after an arbitrary number of updates [16, 15, 2, 4]. To decide when to update, Lee and Cheung [10] propose the DELI algorithm, which uses statistical sampling methods to determine when the current model is outdated. A similar approach proposed by Ganti *et al* (DEMON [5]) monitors changes in the data stream. An efficient incremental algorithm, called ULI, was proposed by Thommas [15] *et al*. 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* [12] 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.

The proposed work, WAVE, is different from 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), WAVE requires only one scan on the incremental database and only a partial scan on the original database. Second, WAVE supports selective updates, that is, instead of determining when to update the whole set of frequent itemsets, WAVE identifies specifically which itemsets need to be updated and then updates only those itemsets. Finally, WAVE employs simple estimation procedures that gives it the ability to improve the prediction accuracy while maintaining the update costs very small. The combination of incremental techniques and on-the-fly data stream analysis makes WAVE an efficient algorithm for mining frequent itemsets and associations in evolving, and potentially streaming databases.

3 The ZIGZAG and WAVE Algorithms

In previous work [16] we presented the ZIGZAG algorithm, a method which efficiently updates the set of frequent itemsets in an evolving database paying heed to practical time and memory constraints. This accomplishment is possible by using an incremental technique based on maximal frequent itemsets, an information lossless approach. This approach results in significant I/O and computational savings, since the number of maximal itemsets is significantly smaller than the number of all frequent itemsets.

ZIGZAG keeps information about the frequent itemsets. This information is composed by the support of all maximal frequent itemsets and the tidlists of all 1-itemsets. The maximal frequent itemsets are updated by a backtracking search approach, which is guided by the results of the previous mining iteration. Further, ZIGZAG uses the updated maximal frequent itemsets[1] to incrementally construct the lattice of frequent itemsets in the database. Additional features of ZIGZAG include the ability to track stable itemsets [11].

WAVE is an extension to ZIGZAG which maintains the same data structure, but adds the ability to detect and react to changes in evolving databases. Contrasting to other incremental approaches [15, 2, 4, 5] 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 completly updated just when we cannot perform a good estimate of it anymore. Figure 1 depicts a real example that motivates our selective approach. This figure shows the correlation of two sets of popular itemsets. These popular itemsets are ranked by support (i.e., popularity ranking) and their relative positions are compared. When the set of popular itemsets is totally accurate, all the popular itemsets are in the correct position. However, when the database is evolving it is hard to continously achieve a totally accurate set of popular itemsets, since the support of those itemsets and their relative positions in the ranking are constantly changing. From Figure 1 we can see a comparison of a totally acurate set of popular itemsets and a ranked set of itemsets which is becoming outdated as the database evolves. As we can see in this figure, although there were significative changes in the support of some popular itemsets, there are also a large number of popular itemsets which remain accurate (i.e., in the correct position) and do not need to be updated, and also a large number of popular itemsets which had evolved in a systematic way, following some kind of trend. The starting point of our approach for mining popular itemsets is the set of potentially frequent itemsets according to our estimation. We distinguish three types of itemsets:

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

Predictable: It is possible to estimate the support of the itemset within a tolerance. This itemset presents some kind of trend, that is, its support increases or decreases in a systematic way over time.

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

[1] The maximal frequent itemsets solely determine all frequent itemsets

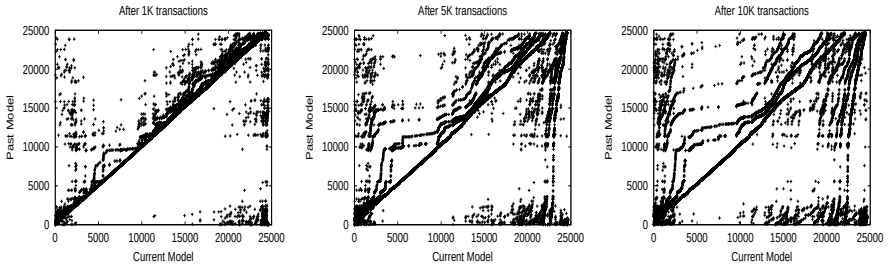

Figure 1. Evolution of Frequent Itemsets. The X-Axis represents a Totally Accurate Ranking, while the Y-Axis represents an Oudating Ranking.

In order to reduce the cost of mining popular itemsets, it is desirable to have no unpredictable itemsets, and the search for tools that better estimate the support of itemsets is probably endless, and is out of scope of this paper. Our belief is that it may not be worth the effort to employ sophisticated tools, since their cost may approach or surpass the cost of executing an incremental mining algorithm such as ZIGZAG.

We classify itemsets regarding its predictability by calculating the ratios between the number of invariant, predictable, and unpredictable itemsets. Table 1 depicts the ratios for the databases showed in Figure 2 for the sake of the evaluation of an approximate approach potential. From this table we can see that both real databases present a significant number of predictable itemsets.

Database	Invariant	Predictable	Unpredictable
WCup	7.2%	45.3%	47.5%
WPortal	9.1%	52.1%	38.8%

Table 1. Ratio between Invariant, Predictable and Unpredictable Itemsets.

Note that there exists a major difference between invariant and predictable itemsets. If there is a large number of invariant itemsets in the database, the set of popular itemsets generated will remain accurate for a long time. On the other hand, if there is a large number of predictable itemsets, the model will lose accuracy over time. However, we can generate pretty good estimations of these predictable itemsets, potentially maintaining the accuracy of the support of the popular itemsets.

WAVE is comprised of two phases. The first phase uses the tidlists associated with 1-itemsets whose union results in the itemset we want to estimate the support. The second phase analyzes the sampled results to determine whether it is necessary to count the actual support of the itemset. Each of these phases is described below.

Support Sampling – The starting point of the sampling is the tidlists associated with 1-itemsets, which are always up-to-date since they are simply augumented by the novel transactions. Formally, given two tidlists l_α and l_β associated with the itemsets α and β, we define that the exact tidlist of $\alpha \cup \beta$ is $l_{\alpha\cup\beta} = l_\alpha \cap l_\beta$. We estimate the upper bound on the merge of two tidlists as follows. We divide the tidlists into n *bins*. The upper bound of the intersection of corresponding *bins* is the lower of the two *bin* values (*bin*

value corresponding to the number of entries in the *bin*). The upper bounds associated with the *bins* are then used as input to our estimation technique, described next.

Support Estimation based on Linear Trend Detection – Trend detection is a valuable tool to predict the frequent itemsets behavior in the context of evolving databases. One of the most widespread trend detection techniques 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, and it is widely used for generating linear regression models.

To verify the goodness of the model generated by the linear regression, we must estimate the goodness-of-fit. In the absence of this estimate, we have not the slightest indication that the parameters a and b in the model have any meaning at all. The goodness-of-fit R^2 represents the proportion of variation in the dependent variable that has been 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. A R^2 value close to 1 indicates that the regression equation is very reliable. In such cases, WAVE provides an approximated technique to find the support of predictable itemsets, an approach that does not have an analog in the itemset mining research. Whenever an itemset is predictable, its support can be simply predicted using the linear regression model, rather than computed with expensive database scans. Figure 2 shows the R^2 distribution for the two databases used in the experiments. This estimate technique achieves extraordinary savings in computational and I/O requirements, as we will see in Section 4.

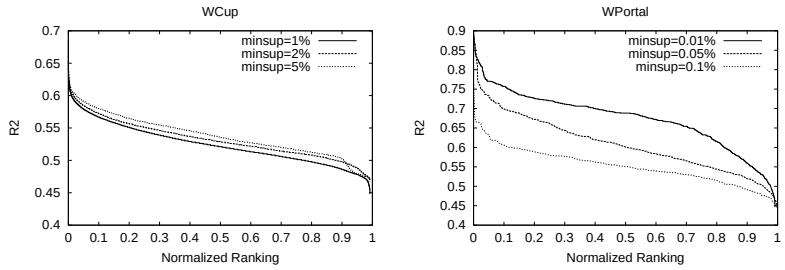

Figure 2. R^2 Distribution in WCup and WPortal Databases.

4 Experimental Evaluation

In this section we evaluate the precision, performance and scalability of WAVE and compare it to other incremental approaches. Real databases from actual applications were used as inputs in the experiments. The first database, 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 items and a standard deviation of 11.2. The second database represents the access patterns of a Web Portal. The database, WPortal, comprises 432,601 transactions over 1,182 unique items, and each transaction contains an average length of 2.9 items. Our evaluation is based on three parameters given to WAVE:

Approximation tolerance $-R^2$**:** the maximum approximation error acceptable.
Longevity: the number of transactions added to the database which triggers a complete update process.
Base length: the number of transactions effectively mined before we start the estimating process.

Thus, for each minimum support used, we performed multiple executions of the algorithm in different databases, where each execution employs a different combination of R^2, *longevity*, and *base length*. Further, we employed three metrics in our evaluation:

Precision: This metric quantifies how good the approximation 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 by WAVE when compared to ULI. We measure the elapsed time for each algorithm while mining a given database in a dedicated single-processor machine. We then calculate the work as the ratio between the elapsed time for our approach and the elapsed time for ULI.
Resource consumption: This metric quantifies the amount of memory used by each algorithm. Observing this metric is interesting for the sake of practical evaluation of the use of WAVE in large databases.

The experiments were run on an *IBM - NetFinity* 750MHz processor with 512MB main memory. The source code for ULI [15], the state-of-the-art algorithm which was used to perform our comparisons, was kindly provided to us by its authors. Timings used to calculate the work metric are based on wall clock time.

4.1 Accuracy Experiments

Here we report the accuracy results for the databases described above. Firstly, we evaluate the precision achieved by WAVE. Next, we evaluate the gains in precision provided by WAVE. We employed different databases, minimum supports, *base lengths*, *longevities*, and R^2. Figure 3(a) depicts the precision achieved by WAVE in the WCup database. From this figure we can observe that, as expected, the precision increases with the R^2 used. Surprisingly, for this database the precision decreases with the *base length* used. Further, the precision decreases with both the *longevity* and minimum support.

Slightly different results were observed for the same experiment using the WPortal database. As expected the precision decreases with the *longevity*. For *base lengths* as small as 50K transactions the lowest precision was achieved by the largest minimum support. We believe that this is because these small *base lengths* do not provide sufficient information about the database. For *base lengths* as large as 100K transactions,

the lowest precision was always achieved by the lowest minimum support. Interestingly, the highest precision was initially provided by the highest minimum support, but as we increase the R^2 value we notice a crossover point after which the second largest support value was the most precise.

We also evaluate the gains in precision achieved by WAVE. From Figure 4(a) we can observe that, using the WCup database, WAVE provides larger gains in precision for smaller values of minimum support. The opposite trend is observed when we evaluate the precision varying the *longevity*, that is, in general larger gains are achieved by larger *longevities*. It is obvious that WAVE loses precision over the time, but this result shows that WAVE can maintain a more accurate picture of the frequent itemsets for more time. Finally, the precision increases with the R^2 value, that is, increasing the precision criteria results in improved prediction precision.

The gains in precision achieved by WAVE were also evaluated using the WPortal database, and the results are depicted in Figure 4(b). In general we observe large gains for smaller values of minimum support. We can also observe that, in all cases, the higher the value of longevity, the larger is the gain in precision. One more time WAVE shows to be very robust in preserving the precision.

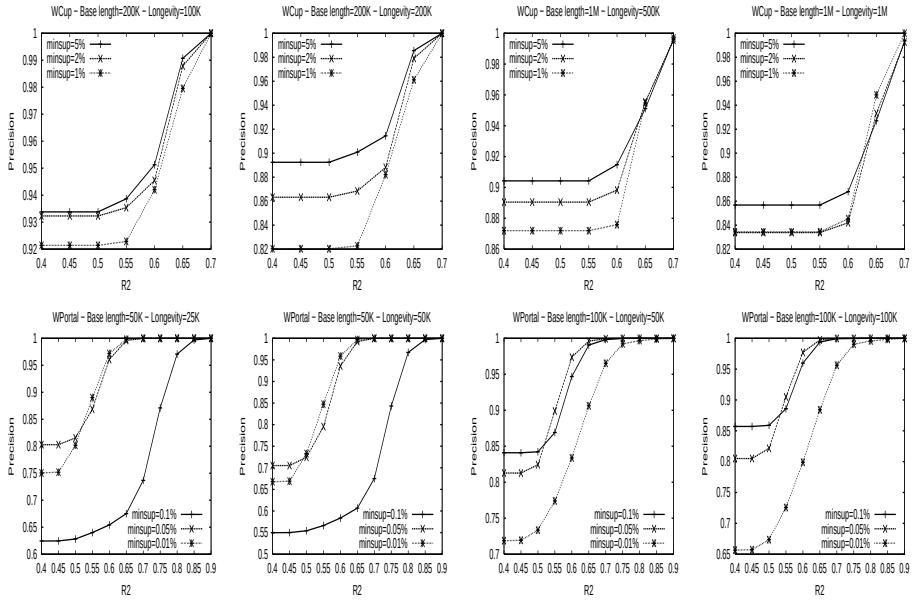

Figure 3. Precision achieved by WAVE when varying *minimum support*, R^2, *base length*, and *longevity* for a) WCup Database (top row), and b) WPortal Database (bottom row).

4.2 Performance Experiments

Now we verify the amount of work performed by WAVE in order to generate an approximate model of associations. From Figure 5(a) we can observe the results obtained

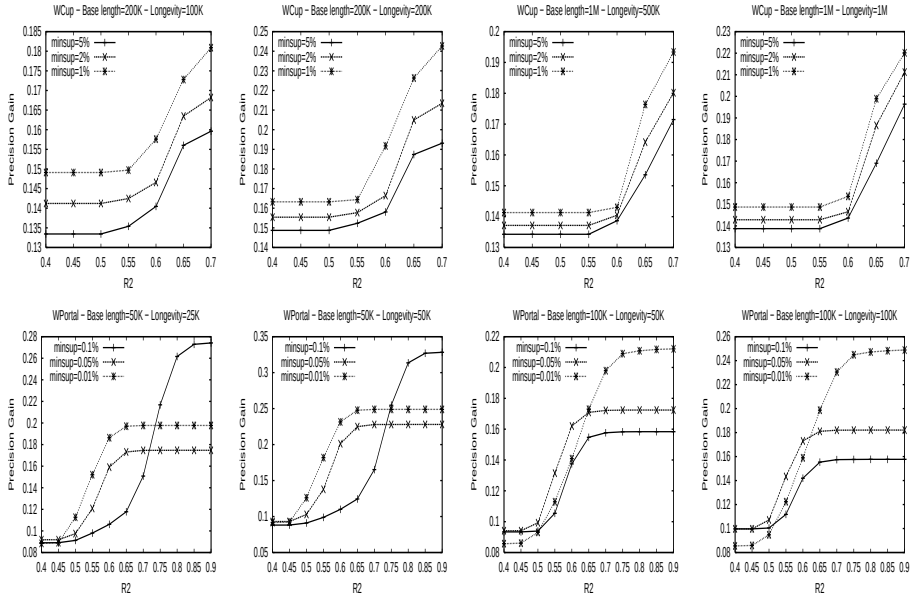

Figure 4. Precision Gains provided by WAVE when varying *minimum support*, R^2, *base length*, and *longevity* for a) WCup Database (top row), and b) WPortal Database (bottom row).

using the WCup database. WAVE performs less work for smaller values of minimum support. This is mainly because ULI spent much more time than WAVE in mining with smaller values of minimum support. We can also observe that WAVE performs the same amount of work when the R^2 threshold reaches the value 0.7, no matter how much the minimum support value is. The reason is that there are only few itemsets with an approximation as good as 0.7, and all these itemsets have a support higher than 5%, which was the highest minimum support used in this experiment.

We also verify the performance of WAVE using the WPortal database. In Figure 5(b) we can observe that in general, for this database, WAVE performs less work for smaller values of minimum support. This trend was observed when the database has a size of 50K transactions, but an interesting result arises for databases with larger sizes as 100K transactions. For smaller values of R^2, WAVE performs less work for larger values of minimum support, but when we increase the value of R^2, WAVE performs less work for smaller values of minimum support. The reason is that when the minimum support is too small, a great number of itemsets present a poor estimate. When the R^2 value is also small, even these poor estimates (not so poor as the R^2 value) are performed. However the relative number of estimates and candidates generated is higher for higher values of minimum support, and, as a consequence, more estimates were performed for higher values of minimum supports. For this database, in all cases, the larger the longevity, the smaller is the work performed by WAVE. Finally, as we can observe in this figure, WAVE performs less work for larger databases.

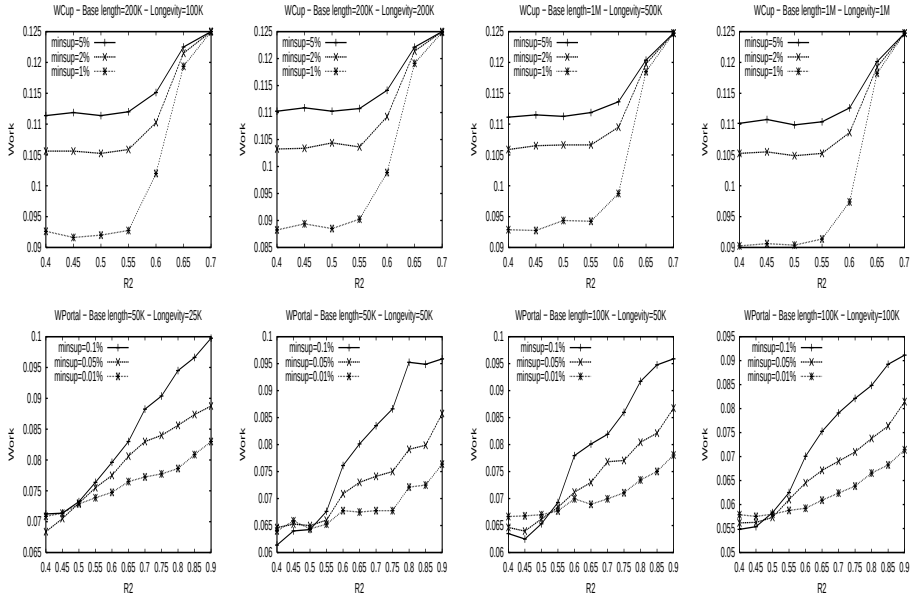

Figure 5. Work Performed by WAVE when varying *minimum support*, R^2, *base length*, and *longevity* for a) WCup Database (top row), and b) WPortal Database (bottom row).

4.3 Scalability Experiments

In this section we compare the amount of memory used by WAVE and ULI, when we employ different databases, minimum supports, *base lengths*, *longevities*, and R^2. Note that the amount of memory used by ULI does not depend on the R^2 employed. From Figure 6(a), where we plot the relative amount of memory used by WAVE and ULI to mine the WCup database, we can observe that in all cases WAVE uses less memory than ULI. The amount of memory used by WAVE exponentilly decreases with the R^2 used. This result was expected since for smaller values of R^2 a larger number of estimates are performed. When we decrease the minimum support value, the relative use of memory also decreases. This is because WAVE is more scalable than ULI, with respect to memory usage. The relative memory usage is smaller when we employ larger *longevities*. Finally, the larger the *base length* used, the less relative memory usage is observed. As can be seen in Figure 6(b), similar results were observed when we used the WPortal database.

5 Conclusions and Future Work

This paper introduced WAVE, an algorithm capable of generating high accurate approximate models of associations hidden in evolving databases. WAVE is able to efficiently maintain the model of associations up-to-date within a tolerance threshold. The resulting accuracy is similar to what would be obtained by reapplying any conventional

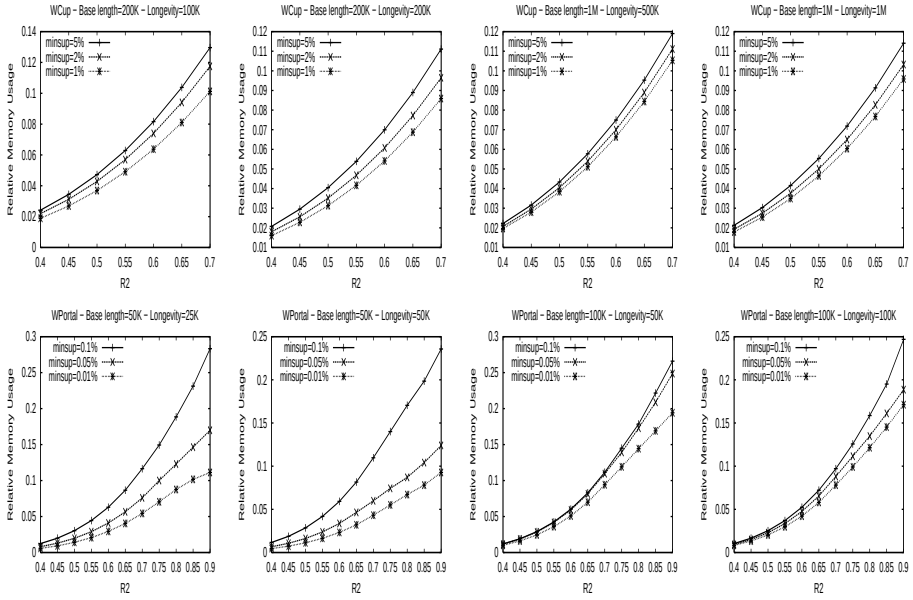

Figure 6. Relative Memory Usage when varying *minimum support*, R^2, *base length*, and *longevity* for a) WCup Database (top row), and b) WPortal Database (bottom row).

association mining algorithm to the entire database. Extensive empirical studies show that WAVE is very efficient, yielding accurate models within practical time and memory constraints. Preliminary applications of WAVE to mine real databases from actual applications show promising results.

We plan to apply WAVE to more real-world problems; its ability to do selective updates should allow it to perform very well on a broad range of tasks. Currently WAVE incrementally maintains the information about the previously frequent itemsets and discards the other ones, but in some domains these infrequent itemsets may become useful – identifying these situations based on trend detection and taking advantage of them is another area for further study.

References

1. R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proc. of the 20^{th} Int'l Conf. on Very Large Databases*, SanTiago, Chile, June 1994.
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 12^{th} Intl. Conf. on Data Engineering*, February 1996.
3. D. Cheung, K. Hu, and S. Xia. Asynchronous parallel algorithm for mining association rules on a shared-memory multipprocessors. In *ACM Symposium on Parallel Algorithms and Architectures*, pages 279–288, 1998.

4. D. Cheung, S. Lee, and B. Kao. A general incremental technique for maintaining discovered association rules. In *Proc. of the 5^{th} Intl. Conf. on Database Systems for Advanced Applications*, pages 1–4, April 1997.
5. V. Ganti, J. Gehrke, and R. Ramakrishnan. Demon: Mining and monitoring evolving data. In *Proc. of the 16^{th} Int'l Conf. on Data Engineering*, pages 439–448, San Diego, USA, 2000.
6. K. Gouda and M. Zaki. Efficiently mining maximal frequent itemsets. In *Proc. of the 1^{st} IEEE Int'l Conference on Data Mining*, San Jose, USA, November 2001.
7. V. Gupta, S. Parthasarathy, and M. Zaki. Arithmetic and logic operations with dna. In *Proc. of the 3^{rd} DIMACS Workshop on DNA Based Computers*, Philadelphia, June 1997.
8. J. Han, H. Jamil, Y. Lu, L. Chen, Y. Liao, and J. Pei. Dna-miner: A system prototype for mining dna sequences. In *Proc. of the 2001 ACM-SIGMOD Int'l. Conf. on Management of Data*, Santa Barbara, CA, May 2001.
9. J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *Proc. of the ACM SIGMOD Int'l Conf. on Management of Data*.
10. S. Lee and D. Cheung. Maintenance of discovered association rules: When to update? In *Research Issues on Data Mining and Knowledge Discovery*, 1997.
11. B. Liu, Y. Ma, and R. Lee. Analyzing the interestingness of association rules from the temporal dimension. In *Proc. of the 1^{st} IEEE Int'l Conference on Data Mining*, San Jose, USA, November 2001.
12. H. Mannila and H. Toivonen. Levelwise search and borders of theories in knowledge discovery. In *Technical Report TR C-1997-8*, U. of Helsinki, January 1997.
13. S. Parthasarathy, M. Zaki, M. Ogihara, and W. Li. Parallel data mining for association rules on shared-memory systems. In *Knowledge and Information Systems*, Santa Barbara, CA, February 2001.
14. M. Rajman and R. Besan. Text mining - knowledge extraction from unstructured textual data. In *Proc. of the 6^{th} Int'l Conf. Federation of Classication Societies*, pages 473–480, Roma, Italy, 1998.
15. S. Thomas, S. Bodagala, K. Alsabti, and S. Ranka. An efficient algorithm for the incremental updation of association rules. In *Proc. of the 3^{rd} Int'l Conf. on Knowledge Discovery and Data Mining*, August 1997.
16. 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 2^{nd} SIAM Int'l Conf. on Data Mining*, Arlington, USA, May 2002.
17. A. Veloso, B. Rocha, W. Meira Jr., and M. de Carvalho. Real world association rule mining. In *Proc. of the 19th British National Conf. on Databases (to appear)*, July 2002.
18. M. Zaki. Generating non-redundant association rules. In *Proc. of the 6^{th} ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining*, pages 34–43, New York, USA.
19. M. Zaki and C. Hsiao. Charm: An efficient algorithm for closed itemset mining. In *Proc. of the 2^{nd} SIAM Int'l Conf. on Data Mining*, Arlington, USA, May 2002.
20. M. Zaki, M. Ogihara, S. Parthasarathy, and W. Li. Parallel data mining for association rules on shared-memory multi-processors. In *Proc. of Supercomputing '96*, Pittsburg, 1996.
21. M. Zaki, S. Parthasarathy, M. Ogihara, and W. Li. New algorithms for fast discovery of association rules. In *Proc. of 3^{rd} Int'l Conf. Knowledge Discovery and Data Mining*, 1997.
22. 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.