Knowledge Management in Association Rule Mining

Adriano Veloso Bruno Pôssas Wagner Meira Jr. Márcio B. de Carvalho
Computer Science Department
Federal University of Minas Gerais
Belo Horizonte, Brazil
{adrianov, bavep, meira, mlbc}@dcc.ufmg.br

Abstract

Most current work on discovery of *association rules* assumes that the database from which the rules are determined is static. The mining operation is performed just once and therefore there is no need of knowledge management integration techniques. However, there are several domains where the database is updated on a regular basis. In these dynamic databases, it is hard to maintain the discovered rules since the updates may not only invalidate some existing rules but also make other rules relevant. We present an efficient algorithm, PELICAN, for incrementally updating the association rules when new *transactions* are added to or old transactions are removed from a dynamic database. Its efficiency comes from addressing several knowledge management issues in its implementation. The *incremental mining* operation is enabled by the maintenance and utilization of the knowledge discovered in the previous mining operations, which provides more interactivity, performance and scalability. Furthermore, PELICAN also allows obsolete transactions to be discarded, so that it can be used for *short-term* mining, and support counting to be relaxed, either for handling noises in the transaction stream or keeping the computational costs low. Extensive experimental data is presented that demonstrates the advantages of PELICAN.

1 Introduction

Data Mining is a new and interdisciplinary field merging ideas from statistics, artificial inteligence, and databases. Its main motivation is the phenomenal growth of data in all domains of human knowledge, and the need to extract valid and useful information from the collected data.

Discovering interesting information in large databases may be CPU and I/O intensive. The high computational cost associated with these procedures may be acceptable when the database is static since the discovery is performed off-line, and several approaches to this problem have been presented in the literature [1, 2, 5, 7]. However, there are several domains where the databases are updated continuously. In these dynamic databases, the knowledge previously acquired must be preserved to facilitate the incorporation of new useful information, otherwise an organization will have to rediscover this knowledge. In order to make data mining algorithms cost effective in such environments, there are some knowledge management issues that must be addressed:

1. **Reuse previous knowledge:** The already discovered knowledge must be reused to prime, guide and improve the knowledge discovery process. However the current techniques for data mining present problems regarding flexibility and scalability while handling these dynamic databases. Furthermore, the high computational cost of data mining discourages interative exploratory mining operations, preventing the user to exploit full potential of data mining.

2. **Update the knowledge base:** Incorporating novel and useful information into previously acquired knowledge is not just a matter of juxtaposing these two knowledge bases. The novel knowledge must be compatible with the existing knowledge base.

3. **Tolerate noisy input:** Noisy data may affect significantly the mining results, either by taking a turn for the worse the data mining performance or by providing non-useful information. A data mining algorithm must be able to handle transient effects associated with the input data.

4. **Manage resource usage:** As mentioned, determining association rules is a computationally intensive task, and one strategy for managing the usage of resources is to relax the task constraints, reducing the amount of work to be performed. Although an essential feature for on-line applications such as eletronic commerce, web mining, and fault monitoring, relaxation of the generation of association rules is not trivial.

5. **Discard obsolete knowledge:** Knowledge becomes obsolete in several application domains, such as news and consumer habits. In this case, it is necessary to discard the obsolete knowledge, which is a task that depends on how the previous acquired knowledge is stored.

In this paper we address these challenges, proposing an approach that exploits sinergies between Knowledge Management and Association Rule Mining, a key Data Mining task which has an elegantly simple problem statement, that is, to find the sets of all subsets of items that frequently occur as database records or transactions, and to infer rules showing us how a subset of items is a function of the presence of another subset. The problem of discovering association rules was first explored in [1] on supermarket basket data, and today it is the most widely studied of all Data Mining functions.

The association rule mining process is essentially an exploratory activity wherein each new mining is guided by the results of the previous mining operation. Therefore we must design novel techniques that take into account the possibility of using the knowledge discovered on the previous mining operations to minimize the amount of work done during each new mining operation. Further, due to the mining exploratory nature, these new techniques must also provide a more flexible mechanism for exploring the databases. Such techniques should be specially useful in dynamic systems, which are constantly updated with novel data, and the mined patterns are rapidly outdated. Current approaches used in these systems generally apply a traditional algorithm on the updated database. However, this process is not efficient since it repeats the work performed previously, ignoring the discovered knowledge.

In this paper we present an efficient algorithm called PELICAN [1] that uses previously discovered knowledge to incrementally update the association rules, while being more flexible and able to discard obsolete knowledge, to speedup the mining operation at a small cost to accuracy, and to handle noisy input data stream.

[1] The name Pelican was inspired on the fishing strategy adopted by this bird, which dives for grabbing its fishes.

Extensive experiments are reported in this paper and demonstrate the performance of PELICAN, when comparing it against ECLAT [7] running on the updated database to find the new frequent association rules. PELICAN is found to be up to 16 times faster than re-running ECLAT on the entire database. More importantly, the number of candidate sets processed is found to be 1-2% of that in ECLAT. We also evaluated the use of PELICAN for short-term mining, and the impact of the relaxed update.

The remaining of the paper is organized as follows. The problem description is given in the next section. PELICAN is described in Section 3 and its performance is discussed in Section 4. Related work is discussed in Section 5, and Section 6 concludes our paper.

2 Formal Problem Description

2.1 Association Rule Mining

To state the problem of mining association rules, as it was proposed in [1], we need some definitions. Let $I = \{I_1, I_2, ..., I_m\}$ be a set of m items and an itemset X is a non-empty lexicographically 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, identified by *tid* and D is a set of transactions.

The support of an itemset X is the number of transactions in D that contains X as subset. An itemset X is frequent if its support is not smaller than the minimum support threshold. Two k-itemsets belong to the same equivalence class F_k, if they share a common k - 1 length prefix. The motivation of this definition is that equivalence classes may be processed independently, as will be discussed later.

An association rule is an implication of the form $X \Rightarrow Y$, where $X \subseteq I$ and $Y \subseteq I$. The association rule $X \Rightarrow Y$ holds in D with *confidence c* if c% of the transactions in D that contains X also contains Y. The association rule $X \Rightarrow Y$ has *support s* in D if s% of the transactions in D contain $X \bigcup Y$. Given a minimum confidence threshold and a minimum support threshold, the problem of mining association rules is to find all the *strong* association rules, that is, the ones whose confidence and support values are greater than or equal the thresholds. A *weak* association rule is a rule that does not meet the thresholds.

The problem of mining association rules can be decomposed into two subproblems. The first problem is to find all frequent itemsets. Many efficient algorithms have been proposed for finding all the frequent itemsets in a database [2, 5, 7]. The second problem is to generate all strong association rules from the frequent itemsets found. Since it is relatively easy to generate association rules if the frequent itemsets are available, major efforts in the research community have been focused on finding efficient algorithms to compute the frequent itemsets, as in the work presented in this paper.

The task of finding all frequent itemsets in dynamic databases is quite challenging mainly because the database can grows to an enormous size. This indicates that the promise of successful data mining lies in the integration of the mining operation and the long-term knowledge management, as the ability to efficiently handle and maintain a large amount of knowledge over a significant long period of time.

2.2 The ECLAT algorithm

ECLAT [7] is an efficient algorithm for determining frequent itemsets that is based on two key features: vertical database format and lattice decomposition. In the vertical format, we mantain for each itemset

X an ordered list of transaction identifiers, which is the $tidlist$ of X. The support of X is given by the length of the corresponding tidlist. Note that if we take the *union* of two k-itemsets from the same equivalence class, we obtain a k+*1*-itemset whose tidlist is the *intersection* of the tidlists of these two k-itemsets. Lattice decomposition divides the search space into sub-lattices that can be divided in smaller sub-lattices, where each one is processed independently in memory, minimizing the secondary memory operations. Each sub-lattice corresponds to an equivalence class, where all itemsets share the same prefix and may be kept in memory for more efficiency. ECLAT finds the itemsets within each class employing a bottom-up search, which starts from 1-itemsets and keeps merging tidlists of candidates.

2.3 Incremental Association Rule Mining

The central idea of the Incremental Association Rule Mining refers to optimizations that can be done across mining computations over updated dataset based on previously stored knowledge. In order to better understand the complexity of incrementally updating association rules, we analyze it formally in the remaining of this section.

Let s_D be the *minsup* (support threshold) used when mining D, and L_D be the set of frequent itemsets obtained. Let P be the information kept from the current mining that will be used on the next incremental mining operation. In our case, P is formed by all the items (along with their tidlists) and all the frequent itemsets (along with their support counts) in D. Using as a starting point the original database 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. The problem of *updating association rules* is to find the set L_Δ, the frequent itemsets in Δ, with respect to a minimum support s_Δ and, most importantly, using P 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.

3 The PELICAN Algorithm

In this section we describe our algorithm, PELICAN, which is based on the ECLAT algorithm, briefly discussed in Section 2.2. Our purpose is to minimize the re-computation and re-scanning of Δ, while handling incremental, short-term and relaxed mining of association rules. The incremental mining is divided into three tasks: (1) recording and discarding novel transactions, (2) determining retained itemsets and (3) determining emerged itemsets. Finding association rules, as mentioned, is relatively simple if we have the frequent itemsets, so we focus on determing the frequent itemsets. The first task is how to record the information associated with the novel transactions so that we can update the frequent itemsets. Based on this information, we perform the second and third tasks, that are, discarding all itemsets that used to be frequent and finding the new frequent itemsets for Δ.

We illustrate the main operations of the algorithm execution in Figure 1. For this example, the value of $minsup$ is 50%, the lattice induced by the original database D is on the left and the lattice induced by the updated database Δ is on the right. The shaded itemsets compose the positive border [10] in each lattice, while the line crossing the lattice divides it into frequent and not frequent itemsets. The left lattice shows $\{1, 2, 3\}$ as the unique itemset composing the positive border of the database D. The right lattice shows $\{2, 3\}$ and $\{3, 4\}$ as the new positive border.

Notice that even in this simple example, the update affected significantly the results, allowing us to grasp important characteristics of the update process. First, it is possible that the positive border changes completely as a consequence of the update, and thus the frequent itemsets. Second, there may be completely novel itemsets that become frequent in the updated database.

The main goal of PELICAN is to reduce the number of candidates examined over Δ to decide whether they are frequent. As mentioned, by utilizing the knowledge discovered on previous mining operations, some candidates are examined just over d^+ and d^-, reduncing computation and I/O costs.

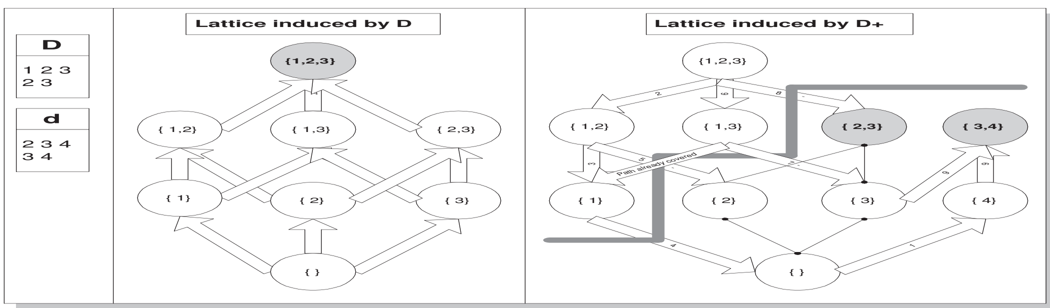

Figure 1: Updating the lattice

3.1 Recording and Discarding Novel Transactions

The novel transactions are recorded by simply augmenting the proper tidlists associated with each item. The recording may be performed on the fly that is, whenever a transaction is performed, it is recorded, or from an interval of time, at the beginning of the process. This strategy has several advantages for sake of incremental mining, since by augmenting the tidlists, it is easy to determine which information are from the last interval.

3.2 Determining Retained Itemsets

After recording the novel transactions, we have to determine which itemsets remain frequent in Δ, called retained itemsets. The starting point is the positive border associated with D, from which it is possible to determine all frequent itemsets of D [7]. Although the positive border may have changed from D to Δ, many of the frequent itemsets in D remain frequent in Δ, and we determine them by checking the itemset lattice in a top-down fashion as follows. For each itemset present in the positive border, we check its support, and if it is still frequent, it is included in the frequent sets of Δ, which we call F_Δ. On the other hand, if it is declined, we break the k-itemset into k $k-1$-itemsets. Each of these $k-1$-itemsets are checked for being frequent, and the process iterates until there are no more itemsets to be checked, when F_Δ contains the retained itemsets, that is, the frequent itemsets in D that remain frequent in Δ. This search is very efficient since we have all the support counts in D of the subsets generated, avoiding performing intersections over the entire database to determine it. We just need to perform intersections over d^- and d^+. So for each frequent itemset in D we sum its support count in D to its support count in d^+, and the support count obtained is subtracted from the support count for this itemset in d^-, determining its support in Δ. At the end of the process the retained itemsets will have the new support count, and the declined itemsets will be discarded.

3.3 Determining Emerged Itemsets

Once we have determined which itemsets remain frequent in Δ, we have to determine the new positive border. Let the I and I^{Δ} be the set of frequent items in D and Δ, respectively. If $I^{\Delta} \subseteq I$ then F_{Δ} may be used as starting point of our algorithm. Otherwise, we also consider the novel frequent items ($I^{\Delta} - I$) for determining the frequent itemsets from Δ. In both cases, our algorithm 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 $X_i \in I$, $\mathcal{L}(X)$ be its tidlist and $\sigma(X) = \mid \mathcal{L}(X) \mid$ is the length of $\mathcal{L}(X)$ and thus the support of X. 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.

Our algorithm considers the retained itemsets (F_{Δ}) and the set of items I_n that became frequent in Δ, i.e., $I_n = I^{\Delta} - I$, where $I^{\Delta} = \{X_i \mid X_i \in \Sigma(F_{\Delta})\}$ and $I = \{X_i \mid X_i \in \Sigma(F_D)\}$, where $\Sigma(F)$ is the set of items that compose the itemsets in F. If I_n is empty, then we build equivalence classes from F_{Δ}. Otherwise, we build the 2-itemsets that include at least one item from I_n, and proceed generating candidate itemsets and testing for frequency until all frequent itemsets are determined. The frequency computation for the emerged itemsets must be performed over the entire database. Although the strategy presented is not necessarily optimal when the frequent itemsets are determined from scratch, the gains may be significant when part of the itemset lattice is already covered, as discussed in Section 4.

3.4 Addressing Knowledge Management Issues with PELICAN

In this section we discuss how each of the knowledge management issues is addressed by PELICAN. The first two issues are addressed by the incremental mining algorithm described. The third and fourth issues (tolerate noisy input and manage resource usage) are addressed by a version of PELICAN that relaxes the association rules constraints. The last issue (discard obsolete knowledge) is addressed by the short-term mining extension.

Other open issues in the integration of Association Rule Mining and Knowledge Management, are the ability to incorporate new knowledge, to handle noisy data, and to manage computational resources. Relaxing the minimum support requirement to determine the frequent itemsets is a very effective stragtegy for coping with these issues. This relaxed update operation handles transient variations in the database, potentially reducing costs since the number of itemsets to be enumerated over the database may be smaller. In this case, if the support variation of a given itemset is within a pre-defined tolerance, the state (frequent or not) of the itemset does not change, and the popularity of this itemset in Δ is simply adjusted to remains the same as it was in D. Thus, small variations on the popularity of an itemset are not considered instantaneously for sake of updating its subsets, since the rules containing its subsets are still a good approximation, not being worth the cost of performing a full update, which is postponed.

This ability is implemented in the top-down search employed by PELICAN. For each itemset present in the positive border, we verify its support in Δ and compares it with the support in D. As explained, if the popularity variation of this itemset is above the tolerance threshold, the support of all its subsets are not computed over Δ.

We define short-term mining as the ability to discard old transactions from the database, considering just a recent portion of the transactions for determining the frequent itemsets, and keeping the set of association rules coherent with recent data. This feature, as discussed in the examples, is desirable for applications such as WEB mining, where the user behavior may vary significantly across time and obsolete knowledge may not be relevant anymore. This feature also represents more interactivity, encouraging fair and flexible exploration of the database through windows of transactions, and its implementation is straightforward in PELICAN, since in the vertical database format the tidlists are always temporally ordered, making easier to remove old transactions, performing short-term mining.

In the next section we illustrate all these features and the cost-benefit trade-offs they imply.

4 Experimental Results

In this section we present the performance evaluation of PELICAN and compare it to ECLAT. We employ two basic metrics for comparing the algorithms: execution time and itemsets processed. The execution time is the elapsed time for generating the association rules. The number of itemsets processed is the sum of itemsets declined and emerged. This metric may be considered an approximation of the elapsed time and helps us to understand the execution time differences observed. We also evaluate the extensions proposed to the algorithm, that are, varying window size and relaxed support counting in terms of computational costs and precision.

We employ three datasets in our experimental evaluation. The first dataset is synthetic and was generated using the procedure described in [2]. This dataset mimics the transactions in a retailing environment, where people tend to buy sets of items. The synthetic dataset, called T25.I20.D100K, comprises 100,000 transactions comprising 1,000 unique items, each transaction has 25 items on average. The other datasets are extracted from an actual 45-day log from a virtual bookstore and comprises 45,178 and 133,937 transactions respectively. In these cases, a transaction contains the books verified, added to cart or bought by a customer during a single visit to the bookstore.

We divide the datasets into equal-sized non-overlapping partitions, whose size is equal to 1,000 transactions. We varied two parameters in our experiments: support and update interval. Thus, for each support we performed multiple executions of the algorithm over the whole dataset. Each execution employs a different update interval, and is compared to the re-execution of ECLAT for all transactions so far. All experiments were performed in a 500MHz AMD-Atlhon with 768MB main memory.

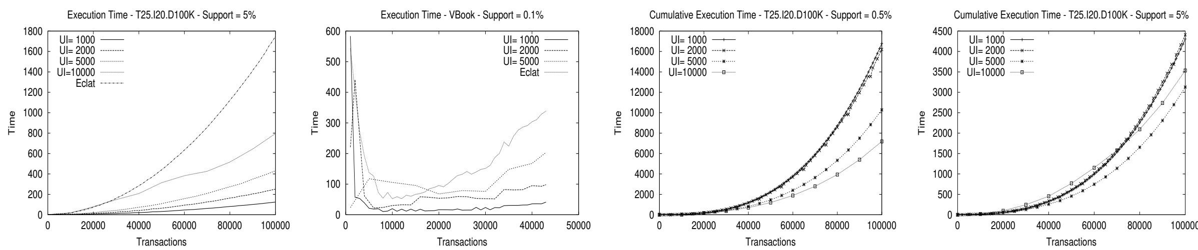

Figure 2: a) Total Execution Time (Left), b) Cumulative Execution Time (Right)

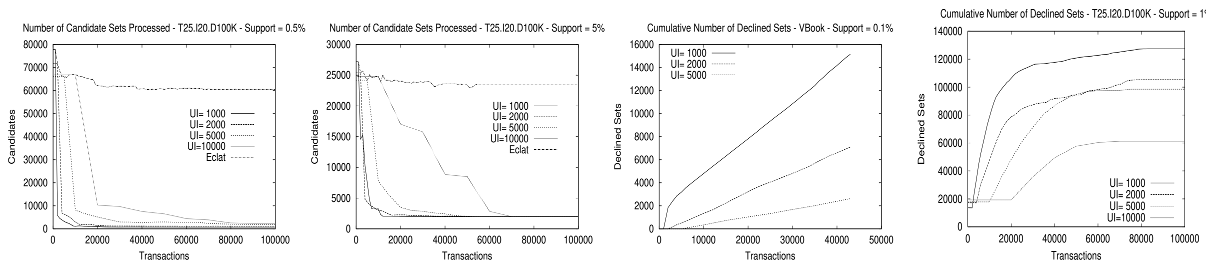

Figure 3: a) Number of Candidates (Left), b) Cumulative Declined Itemsets (Right)

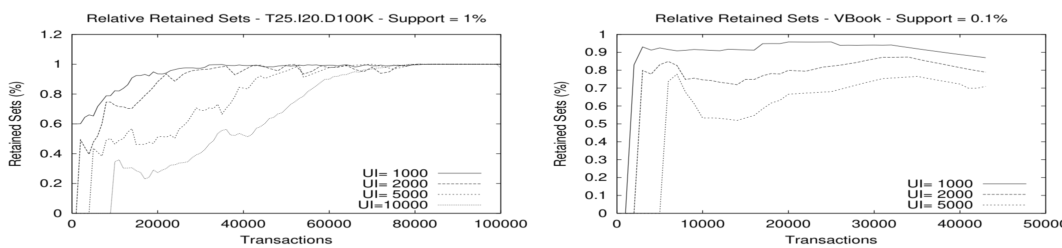

Figure 4: Retained Itemsets

4.1 Incremental Mining

In this section we discuss the improvements, both in execution time as in the number of candidates, provided by PELICAN when compared to ECLAT. Figure 2(a) compares the execution time of PELICAN and ECLAT for varying minimum supports and update intervals. PELICAN outperformed ECLAT, mainly when the number of transactions processed is larger, reaching a speedup factor at most 16 in our experiments. We also observe that, the larger the interval is, the higher the execution time. Other experiments show that PELICAN provides higher gains for smaller supports, since it is less costly to rerun ECLAT for larger support values. Although smaller update intervals present a small cost per update, its higher frequency may result in a higher overall cost, as can be seen in Figure 2(b), where the two last graphs in the Figure present, for two support values, the cumulative execution time of PELICAN. It is remarkable that the best update interval is a function not only of the database, but also the support value employed. Further, the best interval may change as the number of transactions processed increases.

We have seen that PELICAN outperforms ECLAT in terms of execution time for updating association rules. As discussed, the gains of employing PELICAN come from the small number of itemsets checked over the entire database for being frequent. On the other hand, it is important to verify the number of declined itemsets, which we consider non-useful computation, since they do not contribute for the update. In this section we determine how each of these factors contribute for the observed gains.

Figure 3(a) shows the number of candidates in each execution, that is, the number of declined sets plus the number of emerged sets. As expected, PELICAN reduces the number of candidates as the number of transactions increases. Further, the smaller the update interval is, the faster is the reduction in the number of candidates that must be processed, and the graphs also indicate that PELICAN effectiveness is inversely proportional to the support value employed. We can also observe that the number of candidates tends to be constant, as a consequence of the small variance of the synthetic dataset. This observation is confirmed by Figure 3(b), where we plot the cumulative number of declined sets for the synthetic and real logs. Notice that the number of declined sets (and the same was observed for emerged sets) increases continuously for the virtual bookstore, while it stabilizes in the synthetic dataset.

We then verified how the number of retained itemsets varied as a function of minimum support and update interval. As mentioned, our goal is to maximize the number of retained itemsets, since the frequency of these itemsets can be computed solely over d^+ and d^-. Further, notice that the retained itemsets cardinality is usually small, making the gains quite significative, since these itemsets are associated with the largest tidlists, and thus having higher intersection costs.

Figure 4 shows the relative number of retained itemsets for the various configurations evaluated. We can observe that the smaller the update interval is, the larger is the percentage of retained itemsets. Again, we can see that, after a potentially low number of transactions of the synthetic dataset, almost all itemsets are retained, justifying the increasing gains of PELICAN. The higher variance of the bookstore log can be observed again by percentages of retained sets as low as 70%, for more than 40,000 transactions. Other experiments also showed that the number of retained itemsets approaches the number of frequent itemsets for smaller support values.

We then evaluated whether the magnitude of the support variations on the positive border are a good estimator for the amount of work performed during an update. Thus, for each itemset in the positive border we determine its support variation between consecutive updates and plot its average against the number of candidates processed during the update. Figure 5 shows these plots for both declined and emerged sets for an execution employing an update interval of 1,000. In this experiment we employed the real dataset and we can see clearly that there is a strong correlation, which may be exploited for sake of restricting the amount of processing performed during each update, as we discuss in Section 4.3.

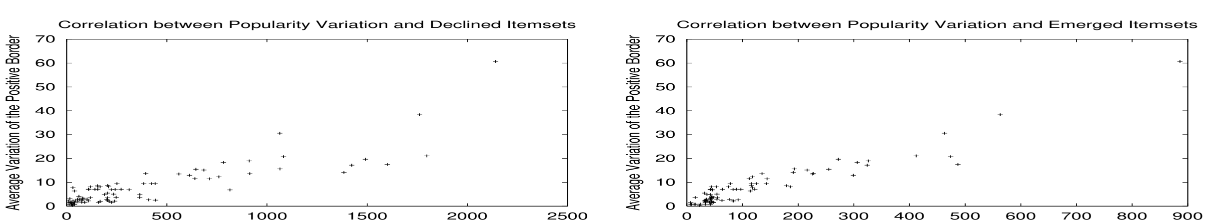

Figure 5: Correlation between the Variation on the Border and Number of Itemsets Processed

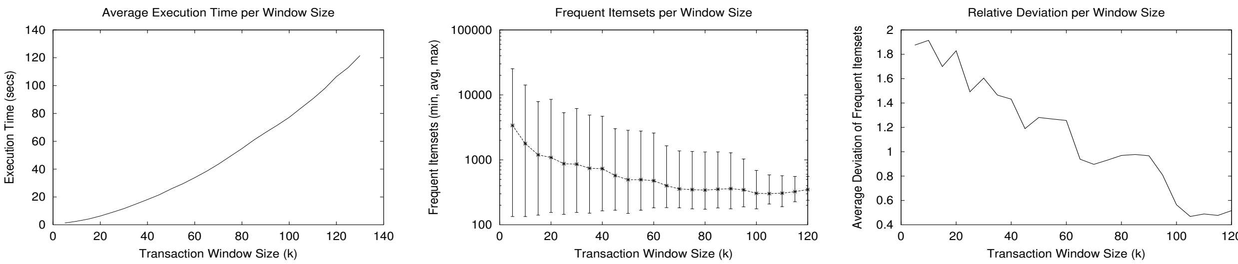

Figure 6: a) Average Cost, b) Frequent Itemsets, and c) Relative Deviation per Window Size

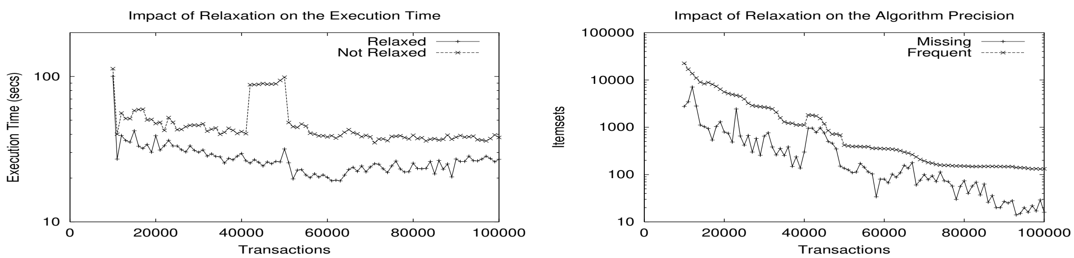

Figure 7: Evaluating the Impact of Relaxation

4.2 Varying the Window Size

As mentioned in Section 3, PELICAN supports transparently short-memory data mining, which is performed by discarding transactions that are not within the last w transactions, where w is the size of the window. The appropriate window size depends on the purpose of the mining task and on the characteristics of the dataset. We evaluated the impact of varying the size of the window both in terms of the number of itemsets processed and the update cost. For both analysis we employed the virtual bookstore database, which illustrates an actual use of our algorithm. Figure 6(a) plots the average update cost as a function of the window size. As expected, the cost increases almost linearly with the window size, as has been observed to ECLAT, which is the basis of PELICAN. Figure 6(b) shows how the number of frequent itemsets processed per update varies as a function of the window size. The graph shows one errobar for each window size tested, and we can see that both the average number of itemsets processed (the line) and the variance on the number of itemsets for a given window size decrease as the window size increases. This behavior is expected as a consequence of the increase of the population of potential itemsets, which become less popular. Figure 6(c) shows how the relative standard deviation of the number of itemsets processed changes with the size of window. Again, confirming the tendency observed in the left graph, the deviation reduces as the window size increases, but presenting an interesting step-based pattern, which can be used as a basis for choosing the window size to be employed as a function of the amount of work that has to be performed on average.

4.3 Relaxing the Update

Another feature of PELICAN is the possibility of relaxing the determination of retained itemsets, as described in Section 3. Relaxing implies a compromise between computational cost and the precision of the itemsets determined. We estimated the precision of an approximate solution by the number of itemsets that differ between sets of frequent itemsets generated by different executions using the same stream of transactions. Again we use the virtual bookstore dataset, aiming to verify the performance and implications of relaxing in an actual use. Figure 7 compares a non-relaxed execution to a relaxed execution where the popularity variations of up to 0.3% are not considered, further, the update interval employed was 1,000 transactions and the window size was 10,000 transactions.

The left graph shows the execution time for each update interval for both configurations. As expected, the relaxed configuration reduced the amount of work performed in all updates, although the gains varied across executions. However, it is remarkable the spike around the transaction 50,000 and we investigated it further. After analyzing the logs, we found out that there is a sudden and intensive robot access to the site at that point. Robots are well known for disrupting the reference locality in the WEB and we can see clearly in the graph the effects of robot accesses, which cause an abrupt increase in the number of itemsets to be evaluated and thus on the corresponding execution time. Notice that the spike in terms of execution time lasts about 10,000 transactions, which is exactly the size of the window employed. On the other hand, our relaxed configuration handled the noise introduced by the robots much better, since they were not significantly affected.

These observations can be confirmed by observing the right graph, where we plot the number of frequent itemsets for each update and we can again see the spike around transaction 50,000. In this graph however, we can see a similar, although smaller, spike in the number of itemsets for the relaxed version as well, which was efficiently handled by the relaxed configuration, as observed. These experiments showed not only that relaxation may be a powerful resource for coupling with noise, but also provides a easy strategy for bounding computational costs in actual applications such as WEB mining.

5 Related Work

In this section we present an overview of the algorithms for mining of association rules, which employ some knowledge management techniques, and compare them to PELICAN. The FUP [4] algorithm is the first approach to incremental mining. It is iterative, and each iteration performs a scan of Δ. Each scan starts by mining d^+ and using the results for guiding the update of D. FUP requires k passes over Δ, where k is the cardinality of the largest frequent itemset, while PELICAN scans the vertical database only when necessary. In [8] the same authors of FUP proposed FUP2, a more general technique for maintaining the association rules while inserting and deleting transactions. When there are just insertions, FUP2 is similar to FUP and when there are just deletions, FUP2 is complementary to FUP. Like FUP, FUP2 is not capable to determine in advance whether the update must be performed or not. [3] presents an algorithm called UWEP 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 [9], the authors proposed DELI, which uses samples of the database to estimate differences between the rules in consecutive updates to decide when perform the mining. We should note that the approaches mentioned usually address the first two issues, while PELICAN handles all five knowledge management described.

6 Summary

We presented an efficient incremental algorithm, PELICAN, which addresses several knowledge management issues while generating association rules. The major advantages of PELICAN over previously proposed algorithms are that it minimizes response time, is able to handle noises in the transaction stream, and it allows changes in the transaction database, either when transactions are added or when they are discarded.

We illustrated the usefulness of PELICAN through extensive experiments using both synthetic and real data. PELICAN outperforms ECLAT in terms of execution time in all cases, by a factor of up to 16 in our experiments, showing the usefulness of integrating the Data Mining and Knowledge Management. The gains are higher for smaller values of support and databases with smaller variance, such as the synthetic datasets used. We also showed how PELICAN may be used for performing short-term mining and handling noises in the database while keeping the computational costs low. We intend to continue our work by developing other incremental approaches that enumerate all the frequent itemsets using a $maximal$ itemset based search method, which we believe can be much more efficient than updating the positive border, since the number of maximal itemsets is significantly smaller than the positive border, and most importantly, we do not have to explicitly compute the declined itemsets. Finally, we will also continue investigating the correlation between support value and update interval, and the usefulness of short-term mining in actual applications such as WEB personalization.

References

[1] R. Agrawal, T. Imielinski, and A. Swami. Mining Association Rules between Sets of Items in Large Databases. In *Proc. of the ACM 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 Intl. Conf. on Knowledge Discovery and Data Mining*, 1999.

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

[5] S. Brin, R. Motwani and J. Ullman. Dynamic Itemset Counting and Implication Rules for Market Basket Data . In *Proc. of the ACM 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 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] D. Cheung and S. Lee. A General Incremental Technique for Maitaining Discovered Association Rules. In *Proc. of the 5th Intl. Conf. on Databases Systems for Advanced Applications*, 1997.

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

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