Incremental, Online, and Merge Mining of Partial Periodic Patterns in Time-Series Databases

Walid G. Aref, *Member*, *IEEE*, Mohamed G. Elfeky, and Ahmed K. Elmagarmid, *Senior Member*, *IEEE*

Abstract—Mining of periodic patterns in time-series databases is an interesting data mining problem. It can be envisioned as a tool for forecasting and prediction of the future behavior of time-series data. Incremental mining refers to the issue of maintaining the discovered patterns over time in the presence of more items being added into the database. Because of the mostly append only nature of updating time-series data, incremental mining would be very effective and efficient. Several algorithms for incremental mining of partial periodic patterns in time-series databases are proposed and are analyzed empirically. The new algorithms allow for online adaptation of the thresholds in order to produce interactive mining of partial periodic patterns. The storage overhead of the incremental online mining algorithms is analyzed. Results show that the storage overhead for storing the intermediate data structures pays off as the incremental online mining of partial periodic patterns proves to be significantly more efficient than the nonincremental nononline versions. Moreover, a new problem, termed merge mining, is introduced as a generalization of incremental mining. Merge mining can be defined as merging the discovered patterns of two or more databases that are mined independently of each other. An algorithm for merge mining of partial periodic patterns in time-series databases is proposed and analyzed.

Index Terms—Data mining, time-series databases, incremental mining, online mining.

1 Introduction

Data mining is defined as the application of data analysis and discovery algorithms to large databases with the goal of discovering (predicting) patterns [12]. A time-series database is a database that contains data over time, e.g., weather data that contains several measures (e.g., the temperature) at different times per day. Other examples of time-series databases are the stock prices and the power consumption.

Early work in time-series data mining addresses the similarity matching problem [1], [11]. Agrawal et al. [2] develop a model of similarity of time sequences that can be used for mining periodic patterns. Recent studies toward similarity matching of time sequences include [8], [20], [22]. Other studies in time-series data mining concentrate on discovering special kinds of patterns. Agrawal et al. [4] define a shape definition language for retrieving user-specified shapes contained in histories (time-series data). Agrawal and Srikant [6], [23] develop an apriori-like [5] technique for mining sequential patterns, which is extended by Garofalakis et al. in [15]. In [7], Bettini et al. develop effective algorithms for discovering temporal patterns. Recently, Han et al. [17], [18] define the notion of partial periodic patterns and present two algorithms for mining this kind of patterns in time-series databases.

• *The authors are with the Department of Computer Science, Purdue University, 250 N. University St., West Lafayette, Indiana 47907-2066. E-mail: {aref, mgelfeky, ake}@cs.purdue.edu.*

Manuscript received 20 Aug. 2001; revised 31 May 2002; accepted 19 Nov. 2002.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number 114815.

Partial periodic patterns, which are the patterns of interest in this paper, specify the behavior of the time series at some, but not all the points in time [17]. For example, a pattern disclosing that the prices of a specific stock are high every Friday and low every Tuesday is a partial periodic pattern. It is partial since it does not describe any regularity for the other week days. As another example, consider a temperature time series of a specific town, a partial periodic pattern may discover that the average temperature is very high in August and very low in December, but not regular in the other months.

One of the important research issues in data mining is *incremental mining*, which is defined as how to maintain the discovered patterns over time as data is continuously being added to the database. The term is originally proposed by Agrawal and Psaila in [3]. In [9], [13], [24], incremental techniques are proposed for the maintenance of frequent sets that are discovered in transaction databases. Ester et al. [10] develop a scalable incremental clustering algorithm. Utgoff [25] develops ID5, an incremental version of the decision tree classifier ID3 [21], yet it is not scalable. In [16], Gehrke et al. develop a scalable incremental algorithm for maintaining decision tree classifiers. In [26], Wang and Tan present an incremental mining algorithm for finding sequential patterns. Ganti et al. [14] describe algorithms for incremental mining of frequent sets and clusters with respect to a new dimension called the *data span dimension* that allows user-defined selections of a temporal subset of the database. To the best of our knowledge, the problem of incremental mining of periodic patterns in time-series databases is not studied before.

In practice, expert users need to provide appropriate thresholds to obtain useful data mining results. *Online*

1041-4347/04/$17.00 © 2004 IEEE Published by the IEEE Computer Society

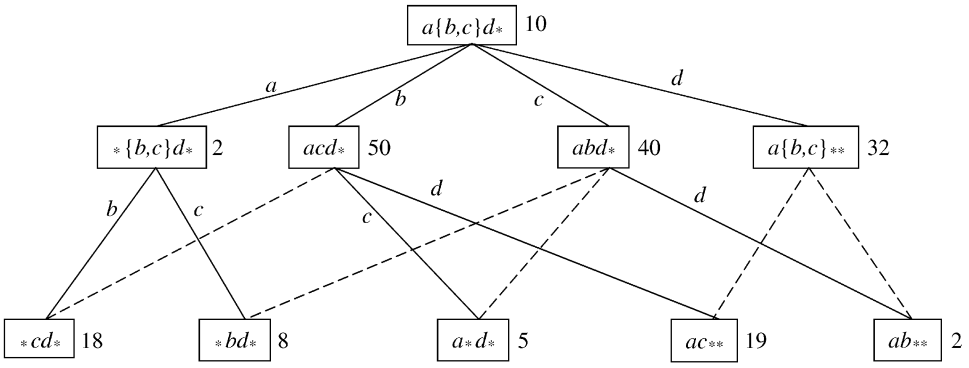

Fig. 1. An example of a max-subpattern tree.

mining considers providing the user with the ability to change the thresholds while the mining process is in progress. Generally, an online mining algorithm outputs continuous results as long as the user keeps changing the thresholds. In the context of association rules [5], an algorithm for online mining is proposed by Hidber [19]. To the best of our knowledge, the issue of online mining of periodic patterns in time-series databases is not addressed before.

In this paper, we present new algorithms for incremental and online mining of partial periodic patterns in time-series databases. Our results show that our algorithms perform an order of magnitude better than the nonincremental nononline ones both for real and synthetic data sets. We define a new operation, termed *merge mining*, that is a technique for merging the mining results of a collection of databases that are each mined separately. We present an algorithm for performing the merge mining operation as an adaptation of our incremental online mining algorithm.

The rest of this paper is organized as follows: In Section 2, the mining problem of partial periodic patterns is introduced along with the notation that is used throughout the paper. Section 3 presents the new algorithms for incremental mining of partial periodic patterns. Section 4 shows how these algorithms can be adapted to address the online mining problem. The merge mining operation is defined in Section 5 along with an efficient algorithm for performing it. In Section 6, a comparison of the performance of the algorithms is reported. Finally, we conclude our study in Section 7.

2 Mining Partial Periodic Patterns

2.1 Notation

Assume that a sequence of n time-stamped features have been collected in a time-series database. For each time instant i, let D_i be the feature collected and L be the set of all features. Thus, the time series of features is represented as, $S = D_1, D_2, \ldots, D_n$. For example, in a time-series database for power consumption, the features collected may be the power consumption rates of a certain customer per hour, or in a time-series database for stock prices, the features collected may be the stock prices of a specific company. Hence, if we quantize the time-series into levels and denote each level (e.g., high, medium, etc.) by a letter, then the set of features $L = \{a, b, c, \ldots\}$, and S is a string of length n over L.

A **pattern** is a sequence $s = s_1 \ldots s_p$, such that p is the length of the pattern and $\forall i = 1 \ldots p$, $s_i \subseteq L$. The L-length of a pattern s is defined as $\sum |s_i|$. A pattern with L-length j is also called a j-pattern. A pattern $s' = s'_1 \ldots s'_p$ is called a **subpattern** of another pattern s if, for each position i, $s'_i \subseteq s_i$. For example, the pattern $a\{b,c\} * \{d,e\}f$, which is of length 5, is a 6-pattern; both of the two patterns $ac * * f$ and $* * * df$ are subpatterns of $a\{b,c\} * \{d,e\}f$, and none of the two patterns $abc * f$ and $ac * f *$ are subpatterns of $a\{b,c\} * \{d,e\}\ f$. Note that the symbol $*$ is used instead of ϕ, and that we omit the brackets when any s_i is a singleton (contains only one element). Clearly, the L-length of any pattern is greater than or equal to the L-length of any of its subpatterns.

The sequence S can be divided into disjoint patterns of equal length p, i.e., $S = S_1, S_2, \ldots, S_i, \ldots$, where $S_i = D_{ip+1}, \ldots, D_{ip+p}$ for $i = 0 \ldots \lfloor n/p \rfloor - 1$, and p is the **period** of that sequence. Each pattern S_i is called a **period segment**. For example, for the mentioned power-consumption time-series database, a typical value for the period is 24, which divides the sequence into patterns each of which represents a day.

A period segment S_i **matches** a pattern s if s is a subpattern of S_i. The **frequency count** of a pattern in a time series is the number of period segments of this time series that matches that pattern. The **confidence** of a pattern is defined as the division of its frequency count by the number of period segments in the time series ($\lfloor n/p \rfloor$). A pattern is called **frequent** if its confidence is greater than or equal to a minimum threshold. For example, in the series $abbaebdced$, if the period is 3, then there are three period segments and the pattern $\{a,d\} * b$, has a frequency count of 2, and a confidence of $2/3$.

2.2 The Max-Subpattern Hit Set Algorithm

In this section, we overview the algorithm given in [17], termed the max-subpattern hit set algorithm, that mines for partial periodic patterns in a time-series database. The algorithm builds a tree, called the max-subpattern tree (refer to Fig. 1 for illustration), whose nodes represent candidate frequent patterns for the time-series. A node is a parent to another node if the following two conditions are satisfied: 1) the pattern represented by the parent node has an L-length that is larger than the L-length of the pattern

represented by the child node by exactly 1, and 2) the pattern represented by the child node is a subpattern of the pattern represented by the parent node. These two conditions mean that the child node pattern is similar to the parent node pattern after removing one letter. The link between a parent and a child node is labeled by that letter. Each node has a count value that reflects the number of occurrences of the pattern represented by this node in the entire time-series. Hence, when a pattern from the time-series is encountered, the corresponding node count is incremented by 1. Yet, this is not enough since all the counts of the nodes that correspond to its subpatterns should also be incremented. For example, when the pattern $acd*$ is encountered, it also means that the pattern $*cd*$ is encountered, and so on, for all its subpatterns recursively. This is a costly operation. Instead, the node that corresponds to the encountered pattern is the only one whose count will be incremented and, then, the exact frequency count of any pattern will be the summation of the count of its corresponding node and all the counts of its parent nodes along the path of the tree to reach this node.

The resulting data structure will be a graph rather than a tree since one pattern can be a subpattern of two or more patterns. For example, the pattern $*cd*$ is a subpattern of the pattern $acd*$ as well as the pattern $*\{b,c\}d*$. In order to preserve the data structure as a tree and decrease the complexity of the insertion and search operations, not all the parent-child links will be kept. This will imply an additional step to determine all the candidate parents of a given child in order to calculate its exact frequency count. Fig. 1 gives an example of the max-subpattern tree. Notice that the dotted lines represent those parent-child links that are not kept.

Clearly, the root node of the tree will represent the candidate frequent pattern that all the other candidate frequent patterns are subpatterns of. Therefore, this pattern will be having the maximum L-length among all the candidate frequent patterns and so it is called the **max-pattern** $C_{\max}$. To determine this pattern, all the 1-patterns are extracted from the time-series and are kept in a list called L_1 along with their respective counts. Then, the set of frequent 1-patterns, termed F_1, is generated. $C_{\max}$ is defined to be the union of all the frequent 1-patterns such that the union operation between two patterns s and t is defined as $(s \cup t)_i = s_i \cup t_i$. For example, the union of the two patterns $a*cd*$ and $b*e*f$ is $\{a,b\}*\{c,e\}df$. For example, if $F_1 = \{a**, *b**, *c**, **d*\}$, then $C_{\max} = a\{b,c\}d*$.

The max-subpattern tree is built incrementally as follows: The sequence is divided into period segments. For each period segment, its candidate frequent pattern is determined and a search operation in the tree for the node of this pattern is performed. If that node exists, then its count is incremented. Otherwise, a new node is created for this pattern and its count is set to 1. The candidate frequent pattern in a period segment is called a **hit** and is defined to be the intersection between the period segment and the max-pattern $C_{\max}$, such that the intersection operation between two patterns s and t is defined as $(s \cap t)_i = s_i \cap t_i$. For example, if $C_{\max} = a\{b,c\}d*$ and $S_i = abed$, then its hit is $ab**$.

Therefore, the algorithm for mining the partial periodic patterns in a time-series database can be summarized in the following steps:

1. The time-series is divided into a number of period segments according to the value of the period.
2. All the 1-patterns are inserted into the list L_1 along with their respective counts, and then the patterns that happen to be frequent according to the specified threshold are inserted into the set F_1.
3. The patterns in F_1 are unioned to form the max-pattern $C_{\max}$.
4. Each period segment is intersected with $C_{\max}$, and the resulting pattern is either inserted in the tree if the pattern is a new one, or its corresponding node count is incremented.
5. Finally, the exact frequency count of each pattern is calculated by adding the count of its node to the counts of all the nodes of the patterns that this pattern is a subpattern of.

As an example for the last step, in Fig. 1, the frequency count of $*cd*$ is 80 (18 for itself, 2 for $*\{b,c\}d*$, 50 for $acd*$, and 10 for $a\{b,c\}d*$). Notice that the frequent 1-patterns are not inserted into the tree since they are already stored in F_1.

3 New Algorithms for Incremental Mining of Partial Periodic Patterns

In this section, we define the incremental mining of partial periodic patterns problem and present new algorithms for it.

3.1 Problem Definition

Generally, the problem of incremental mining takes as input an augmented database S' that is composed of a database S and new data items that are added to S, i.e., if the time-series $S = D_1, D_2, \ldots D_n$ is incremented by a data block of m features, then $S' = D_1, D_2, \ldots D_n, D_{n+1}, D_{n+2}, \ldots, D_{n+m}$. The incremental mining problem assumes that the database S is previously mined. Hence, sufficient information about the database S is collected during the mining process and is currently available. The incremental mining problem intends to discover the frequent patterns in S' making use of the available information. According to the algorithm in Section 2.2, this available information includes the max-subpattern tree T and the 1-patterns list L_1.

First, the unit by which the data is incremented should be defined. Clearly, the database will be considered incremented if at least one period segment is added. Note that one period segment contains a number of features (letters) that is equal to the length of the period. This means that the database will not be considered incremented if the number of the features added is less than the length of the period, i.e., when $m < p$.

Each added period segment contains a number of 1-patterns that is equal to the value of the period. These 1-patterns are inserted into the 1-patterns list L_1. If the inserted 1-pattern is not already there, it is added; otherwise, its count is incremented. Assume that there are k period segments added to the data (i.e., $m = kp$). Hence, any 1-pattern is incremented by a value that is at least 0 and

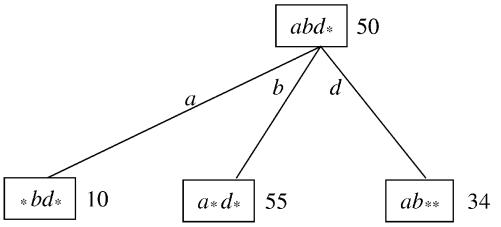

Fig. 2. The max-subpattern tree of Fig. 1 after deleting the letter c.

is at most k. Since the number of period segments in the database is also incremented by k, the set F_1 may be affected in the following way. Let c be the total number of period segments in the database S, t be the confidence threshold, x and x' be the frequency counts of any given 1-pattern in S and S', respectively. For each 1-pattern in one of the added period segments, there are four cases:

1. Case 1: The 1-pattern is infrequent in both S and S' (i.e., $x/c < t$ and $x'/(c+k) < t$).
2. Case 2: The 1-pattern is infrequent in S and becomes frequent in S' (i.e., $x/c < t$ and $x'/(c+k) \geq t$) and, hence, is added to the set F_1.
3. Case 3: The 1-pattern is frequent in S and it remains frequent in S' (i.e., $x/c \geq t$ and $x'/(c+k) \geq t$).
4. Case 4: The 1-pattern is frequent in S and it becomes infrequent in S' (i.e., $x/c \geq t$ and $x'/(c+k) < t$) and, hence, is removed from the set F_1.

Cases 2 and 4 result in updating the set F_1. Recall that the max-pattern $C_{\max}$, which is the root node of the max-subpattern tree T, is calculated by unioning all the patterns of the set F_1. Hence, if F_1 is updated, $C_{\max}$ is also updated, and the max-subpattern tree T should be updated accordingly.

3.2 Entire-Segments (ES) Incremental Algorithm

Let $C'_{\max}$ denote the max-pattern $C_{\max}$ after update. Let c_j be the component at position j of $C_{\max}$, and c'_j be the component at the same position of $C'_{\max}$. If $c'_j \neq c_j$, then updating c_j to c'_j implies *deletion* and/or *insertion* of one or more letters. For example, if $C_{\max} = a\{b,c\}d*$ and F_1 is updated as follows: The 1-pattern $*c**$ is removed and the two 1-patterns $*e**$ and $***f$ are added, then $C'_{\max} = a\{b,e\}df$, i.e., the letter c at position 2 is deleted and then the letters e at position 2 and f at position 4 are inserted.

Hence, in order to reflect the update of $C_{\max}$, the proposed incremental algorithm contains two main steps to update the max-subpattern tree T by handling the deletion and insertion events. The first step updates the max-subpattern tree T such that the resulting tree T^t contains no pattern that contains one of the deleted letters. The second step updates the max-subpattern tree T^t such that the resulting tree T' contains all the patterns that contain at least one of the inserted letters.

Let $C^t_{\max}$ be the pattern resulting from removing the deleted letters from $C_{\max}$, e.g., in the previous example, $C^t_{\max} = abd*$. $C^t_{\max}$ can be calculated easily by intersecting $C_{\max}$ and $C'_{\max}$. Clearly, if $C^t_{\max}$ equals $C_{\max}$, then there are no deleted letters; otherwise, $C^t_{\max}$ is a subpattern of $C_{\max}$. Hence, if there is a node in T that represents $C^t_{\max}$, then that node will become the new root of T^t. Otherwise, a new node is created. Consider the max-subpattern tree given in Fig. 1. Furthermore, assume that $C'_{\max} = a\{b,e\}df$. Hence, $C^t_{\max} = abd*$, and the tree T^t initially contains only two nodes: one for the pattern $abd*$ with a count of 40 and one for the pattern $ab**$ with a count of 2. There are two aspects concerning the resulting tree T^t; the counts must be fixed, and the nonlinked children from T should be added. To consider both of these two aspects concurrently, the max-subpattern tree T is scanned and, for each node, the intersection of its pattern with $C^t_{\max}$ is inserted into T^t with the same count. We call this operation *tree update* operation.

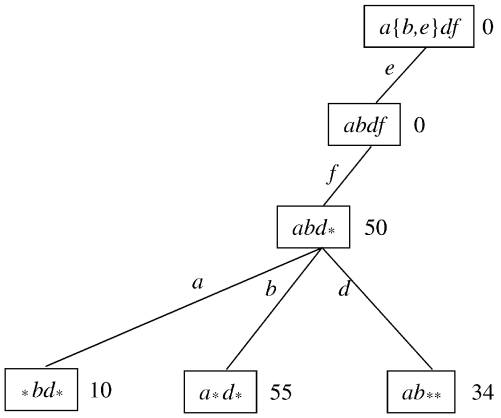

Fig. 3. The max-subpattern tree of Fig. 2 after inserting the two letters e and f.

For example, in Fig. 1, $C_{\max} = a\{b,c\}d*$. Assume that $C'_{\max} = a\{b,e\}f$, so $C^t_{\max} = abd*$. The initial tree T^t has only two nodes that contain the patterns $abd*$ with a count of 40 and the pattern $ab**$ with a count of 2. Performing the *tree update* operation results in inserting the following patterns in T^t ($abd*10$, $*bd*2$, $a*d*50$, $abd*40$, $ab**32$, $*bd*8$, $a*d*5$, $ab**2$). Fig. 2 gives the resulting max-subpattern tree T^t. Note that the counts of the nodes in the resulting tree T^t must be reset to zero before the *tree update* operation starts.

Recall that $C'_{\max}$ is the updated max-pattern due to updating F_1. Therefore, $C'_{\max}$ should be the pattern of the root node of the max-subpattern tree T' that must result from the second step. The input max-subpattern tree for this step is T^t that results from the previous step. Since $C^t_{\max}$, the root node pattern of T^t, is calculated by intersecting $C_{\max}$ and $C'_{\max}$, then $C^t_{\max}$ is a subpattern of $C'_{\max}$. Hence, as an initial action, if $C^t_{\max} \neq C'_{\max}$, a new root node for T' is created that contains the pattern $C'_{\max}$, and the root node of T^t becomes a child for that new root node. The problem now is that all the information we have (the max-subpattern tree T and the list L_1) is not enough to determine the counts of the patterns that contain at least one of the inserted letters since these patterns were not originally included in T. For example, consider Fig. 3 that gives the result of applying the initial action on the max-subpattern tree given in Fig. 2. Both the patterns $a\{b,e\}df$ and $abdf$ have appeared as nodes in the new tree, but their counts cannot be determined. Also, there might be other frequent patterns that should appear in T', e.g., $ab*f$. Hence, the time-series S should be scanned again in the same way as the scanning step in the max-subpattern hit set mining algorithm to determine

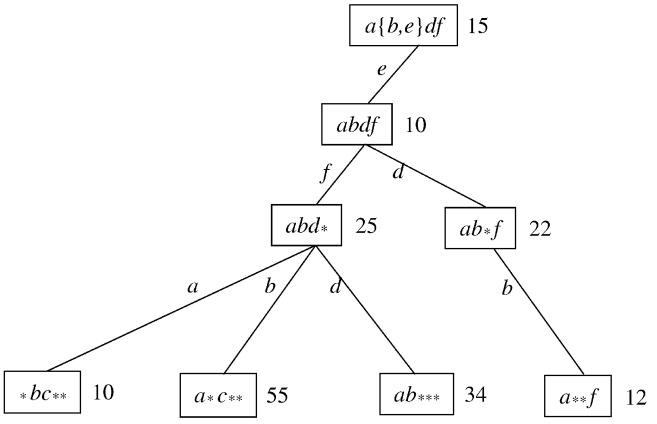

Fig. 4. The max-subpattern tree of Fig. 3 after scanning the time-series.

the frequent patterns that should appear in T' as well as their exact counts. Fig. 4 gives the tree T' after this scan.

In the tree given in Fig. 2, the pattern $abd*$ results from different data segments. Two of them are $abde$ and $abdf$. Rescanning the time-series S will reconsider both those segments. Reconsidering $abde$ is easy since its hit (its intersection with $C'_{\max}$) is still $abd*$ which is already a node in the tree. Yet, reconsidering the segment $abdf$ is a bit nontrivial since its hit now is $abdf$ and, hence, the count of its node should be incremented, while the count of the node $abd*$ should be decremented in order to correctly maintain the counts of the nodes. Therefore, we can notice that the only segments that need to be considered in the scanning step are those segments that contain at least one of the inserted letters. If we maintain an *inverted list* associated with each 1-pattern in L_1 that contains the period segments in which this 1-pattern appears, then we can avoid the scan over the time series S and consider only the segments in the lists of the 1-patterns that are added to F_1. This approach will avoid the additional scan over the time series in our proposed algorithm.

Since the max-subpattern tree is updated to reflect the change of the max-pattern $C_{\max}$, the added period segments are scanned and their hits are inserted in the tree. Therefore, the max-subpattern tree T' now reflects the new time-series database S'.

The ES algorithm for incremental mining of partial periodic patterns is outlined in the following steps:

1. The list L_1 is updated to include the 1-patterns of the added period segments and then the set F_1 is updated consequently (only if needed).
2. The patterns in F_1 are unioned to form the new max-pattern $C''_{\max}$.
3. The tree is updated appropriately according to the previous discussion to reflect the change of the max-pattern (the pattern of the root node).
4. Each one of the added period segments is intersected with $C'_{\max}$, and the resulting pattern is either inserted in the tree if it is new, or its corresponding node count is incremented.
5. Finally, the exact frequency count of each pattern is calculated by adding the count of its node to the counts of all the nodes of the patterns that this pattern is a subpattern of.

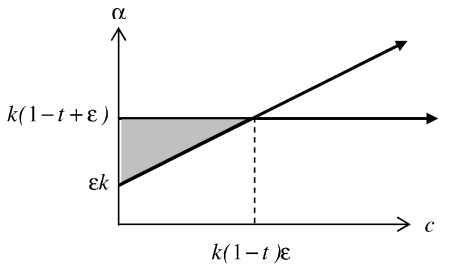

Fig. 5. A plot of the inequalities relations between α and c for one 1-pattern.

3.3 Analysis

A quick analysis of the ES algorithm shows that there is at most one scan over the database if there are inserted letters. This scan is an expensive operation. Although we can avoid scanning the database by maintaining inverted lists, we prove below that the probability of performing this scan is very low if the data size is very large, which is the case in most practical applications.

Assume that the added period segments will be uniformly distributed over the 1-patterns they have. Hence, for the 1-pattern s_i whose frequency count is x_i, the new count of this pattern will be $x'_i = x_i + k(x_i/c)$. If this pattern was not already in F_1 since $x_i/c < t$, where t is the confidence threshold, then $x'_i/(c+k) = \frac{x_i+k(x_i/c)}{c+k} = x_i/c < t$ and, hence, it will not be added to F_1, i.e., the scan is not required. If this uniform distribution is missed by a value of α_i, i.e., $x'_i = x_i + k(x_i/c) + \alpha_i$, then, in order to add this pattern to F_1, $x_i/c + \alpha_i/(c+k)$ should be larger than t. Assuming that $x_i/c = t - \varepsilon_i$, then it should be that $\alpha_i > \varepsilon_i(c+k)$. But, $k(x_i/c) + \alpha_i < k$, then $\alpha_i < k(1-t+\varepsilon_i)$. Fig. 5 gives a plot of these two inequalities for a 1-pattern segment s_i. The solution of these inequalities lies in the shaded area of the plot. From the figure, we deduce that, for large values of c, i.e., $c > \frac{1-t}{\varepsilon_i} k \; \forall i$, no solution is found for the inequalities. Therefore, the probability that new patterns are added to F_1 is very low and, hence, the probability of performing the additional scan over the database is very low as well. In other words, when the database size is large, the cost of the scan operation will be high. However, according to our discussion here, the scan operation will be unlikely to happen in this case. Fig. 5 also shows that increasing the value of t will decrease the shaded area and, hence, decrease the probability of performing the additional scan.

The inverted lists approach will avoid the additional scan over the database in our proposed algorithm, but it requires additional space. This additional space is given by the sum of the counts of all 1-patterns in L_1, which equals to the space occupied by the original time-series times the value of the period. Yet, any information that can be obtained from the original time-series can also be obtained now from L_1 with the inverted lists. Therefore, this representation is an alternative for the original representation of the data and, hence, we no longer require the original time-series. It is worth mentioning that this inverted lists approach was proposed in a similar way in [14] for incremental mining of frequent itemsets.

3.4 Block (BL) Incremental Mining

The *Entire-Segments (ES) Incremental Mining* algorithm discussed in the previous sections assumes that all the k new segments will be dealt with simultaneously. Another algorithm based on the same approach is to divide the k new period segments into blocks of equal size, say b, and to consider the same updating technique several times over each block independently. This algorithm is called *Block (BL) Incremental Mining*. The special case when $b = 1$ is called *Single-Segment (SS) Incremental Mining*. Section 6 gives a comparison among these algorithms based on the performance analysis.

3.5 Merge (ME) Incremental Mining

Another approach for incremental mining of partial periodic patterns is to consider the newly added period segments as a stand-alone time-series database and apply the original mining algorithm over this time-series separately from the original time series database. The result will be new instances of the structures (the list L_1 and the max-subpattern tree T). Then, the algorithm merges these structures with the corresponding ones of the original time-series database. We term this algorithm the *Merge Incremental Mining Algorithm*. Let T and T' denote the max-subpattern trees of the original time-series database and the added time-series, respectively. Then, the max-subpattern tree of the overall time-series is $T'' = T \cup T'$. This union operation is described below. Similarly, Let L_1 and L'_1 denote the 1-patterns lists of the original time-series and the added time-series, respectively. Then, the 1-patterns list of the overall time-series is $L''_1 = L_1 \cup L'_1$. The latter union operation is simple to perform. For each pattern $s \in L_1$ with count x and $s \notin L'_1$, add s to L''_1 with the same count x. For each pattern $s \in L'_1$ with count x and $s \notin L_1$, add s to L''_1 with the same count x. Finally, for each pattern $s \in L_1$ with count x and $s \in L'_1$ with count x', add s to L''_1 with count $x + x'$.

To merge the two max-subpattern trees ($T \cup T'$) into one tree (T''), first we determine the root node pattern (max-pattern) of T''. Following the same notation, let $C_{\max}$ and $C'_{\max}$ be the max-patterns of T and T', respectively. Recall that those max-patterns capture the frequent 1-patterns of their corresponding time-series. Clearly, it is not possible that a pattern would be frequent in the overall time-series without being frequent in at least one of the two sub time-series (the original and the added). Therefore, the max-pattern of the overall time-series can be determined by unioning $C_{\max}$ and $C'_{\max}$, i.e., $C''_{\max} = C_{\max} \cup C'_{\max}$. Then, each tree is updated such that its root node pattern contains the calculated max-pattern $C''_{\max}$ using the same technique of the previously proposed algorithm. Recall that the update operation may trigger a rescan of the corresponding database and this scan can be avoided using the inverted lists approach. Now, the two trees T and T' have the same root node pattern. Without loss of generality, let T' be the tree with smaller number of nodes. Initially, T'' is set equal to T. Then, T' is scanned and, for each node, its pattern is inserted into T''. Notice that, if a subpattern does not belong to the tree T or the tree T', then it cannot belong to T''.

3.6 Data Span Dimension

In [14], Ganti et al. introduce a new dimension to the problem of incremental mining called the *data span dimension* that offers two options. The *unrestricted window* option is similar to the regular incremental mining problem where all the data is considered for mining. In the *most recent window* option, a specified number w of the most recently segments is the only data to be considered for mining. The proposed model in [14] (GEMM) to allow for this most recent window option can be generalized for any incremental model maintenance algorithm. Yet, the full generality of GEMM comes to the fore for classes of models that cannot be maintained under deletions of tuples [14].

Our proposed incremental model can be maintained easily under deletion of segments. Therefore, if the user specifies a window w of segments, then the segments that lie before this window should be removed from the maintained model. Recall that the maintained model includes the 1-patterns list L_1, the max-subpattern tree, and the inverted lists (if used). The deletion operation of a segment from the maintained model is analogous to the insertion operation discussed so far. Deleting a segment from L_1 is performed by decrementing the count of all the 1-patterns in L_1 that are contained in this segment, and also deleting this segment from the inverted list associated with the 1-patterns. Deleting a segment from the max-subpattern tree is performed by decrementing the count of the node that represents the hit pattern of this segment.

4 Online Mining

Online mining can be defined as maintaining the discovered patterns over the whole range of thresholds. Practically, under online mining, the user should be able to change the thresholds during the mining process in order to refine the mining results. Online mining should not restart the mining process each time the user changes the thresholds; otherwise, it would be so inefficient and will not be considered online.

4.1 Online Mining of Partial Periodic Patterns

Recall that the mining algorithm of partial periodic patterns proposed in [17] and discussed in Section 2.2 uses the confidence threshold in extracting the frequent 1-patterns set F_1 from the 1-patterns list L_1 and in extracting the frequent patterns from the max-subpattern tree T. Hence, changing the confidence threshold during building the 1-patterns list L_1 has no effect. Yet, changing the confidence threshold during building the tree is the main issue since it may result in updating the set F_1. As discussed in Section 3.1, updating F_1 will result in updating the root node, $C_{\max}$, of T. In other words, the max-subpattern tree T should be rebuilt.

Let l be the number of features scanned so far from the time-series $S = D_1, D_2, \ldots D_{l-1}, D_l, D_{l+1}, \ldots, D_n$. Clearly, since the time series is first divided into period segments of length p, $l = kp$ for a certain integer k. It can be shown now that the problem is transferred to an incremental problem in which $n - l$ features are added to the time-series database of which l features were mined before.

In other words, if the user changes the confidence threshold value, F_1 is updated accordingly and so will the

max-pattern $C_{\max}$. Let $C'_{\max}$ denote the new max-pattern after the update. The process of building the max-subpattern tree is stopped (l features are scanned so far). The tree is updated to have a new root node that contains the pattern $C'_{\max}$ in a similar way to what was discussed in Section 3.2. Then, the process resumes considering the remaining $n - l$ features of the time-series.

This online algorithm has at most one rescan over the l features per every change of the confidence threshold. Although this algorithm is theoretically costly, the following analysis shows that it is practically not.

Let t and t' be the confidence thresholds before and after the change, respectively. If $t' < t$, then more patterns become frequent and new 1-patterns will be added to F_1. Accordingly, letters will be inserted into $C'_{\max}$ and a rescan of the database will be needed. If $t' > t$, some patterns will be removed from F_1. Accordingly, there will be deleted letters from $C'_{\max}$ and no inserted letters. Hence, no rescan is needed. In practice, it is more appropriate that the user starts the mining process with lower threshold values, gets many frequent patterns, and then tries to increase those values in order to obtain less frequent patterns. Hence, the proposed online algorithm is practically efficient. Note that using the inverted lists approach discussed before will avoid a rescan under all conditions.

5 Merge Mining

Merge mining can be defined as merging the discovered patterns of two or more databases that are mined independently of each other. This operation can be viewed as a generalization of incremental mining. The input is two or more previously mined databases and it is required to discover the patterns from the combined database without applying the mining algorithm again. This problem arises in practice, e.g., in a multilocation company that has a database for each one of its branches. Merging the databases and running the mining algorithm again over the entire database may not be appropriate and is time consuming. Instead, merge mining is how to merge the mining results of the databases to discover the patterns of the combined database. Another application of merge mining is in parallel data mining where the database is composed into a number of smaller databases; each is mined separately and the results are combined using merge mining. This procedure can be recursively performed when mining each of the small databases.

The input to the merge mining algorithm is a number of max-subpattern trees and the same number of 1-pattern lists, each belongs to one of the databases to be merged. The proposed algorithm is the same as the algorithm described in Section 3.5 for merge incremental mining. We obtain the max-pattern $C_{\max}$ of the combined time-series by the union operation $C_{\max} = \bigcup_{i=1}^{m} C^i_{\max}$ such that m denotes the number of time-series databases to be combined and $C^i_{\max}$ is the max-pattern of the time-series number i. Then, each tree is updated to have a root node that contains the combined max-pattern. Now that all the trees have the same root node pattern, the one that has more nodes is selected and all the other trees are inserted into that one, which becomes the combined tree that contains the aggregate mining results of the combined time-series databases.

TABLE 1
Sample Results Using Power Consumption Database

Confidence Threshold *(%)*	Non-Incremental Mining Time *(seconds)*	Time Overhead *(milliseconds)*	ES Incremental Mining Time *(milliseconds)*
10	28.53	39.02	30.20
20	28.57	33.32	30.38
30	28.56	33.36	29.86
40	28.64	33.18	29.52
50	28.54	33.34	29.32
60	27.76	33.20	23.08
70	27.76	33.18	23.12
80	27.29	33.20	21.98

6 Performance Study

In our experiments, we use two real databases. The first one is a relatively small database that contains the daily power consumption rates of some customers over a period of one year. It is made available through the CIMEG[1] project. The database size is approximately 5 Megabytes. The appropriate period for this data is 7 that corresponds to weekly power consumption in units of days. The second database contains sanitized data of timed sales transactions for some Wal-Mart stores over a period of 15 months. The timed sales transactions data has a size of 130 Megabytes. An appropriate period for this data is 24 that corresponds to daily transactions in units of the number of transactions per hour. In the experiments using the power consumption database, the time-series data is incremented by 100 segments of period 7, i.e., 700 features, while in the experiments using the timed sales transactions database, the data is incremented by 30 segments of period 24, i.e., one month of data. In both databases, the numeric data values are quantized into five levels: *very low*, *low*, *medium*, *high*, and *very high*. For the power consumption data, quantization is done based on discussions with domain experts (*very low* corresponds to less than 6,000 Watts/Day, and each level has a 2,000 Watts range). For the timed sales transactions data, quantization is based on manual inspection of the values (*very low* corresponds to transactions per hour, *low* corresponds to less than 200 transactions per hour, and each level has a 200 transactions range).

We study the performance of algorithms ES, BL, SS, and ME, described in Sections 3.2, 3.4, and 3.5. Table 1 gives the results for different values of confidence threshold using the power consumption database. The results show that all the incremental algorithms, proposed in the paper, perform better than the nonincremental mining algorithm.

While the nonincremental mining algorithm takes approximately 30 seconds for mining the entire database, for different values of confidence threshold, the proposed incremental algorithm ES takes around 30 milliseconds. There is an additional overhead time that is paid once by

1. CIMEG: Consortium for the Intelligence Management of the Electric Power Grid (http://helios.ecn.purdue.edu/~cimeg).

TABLE 2
Sample Results Using Timed Sales Transactions Database

Confidence Threshold (%)	Non-Incremental Mining Time (seconds)	ES Incremental Mining Time (seconds)	ME Incremental Mining Time (seconds)
10	160.23	5.54	6.51
20	165.54	5.93	6.98
30	167.03	6.06	6.94
40	170.45	6.09	7.12
50	166.89	6.11	7.03
60	121.64	1.57	1.81
70	116.85	0.63	0.68
80	109.15	0.18	0.21

the incremental algorithms that involve storing the max-subpattern tree and the 1-patterns list (column 3 of Table 1) (30-40 milliseconds). Also, there is some additional storage needed to store these data structures, but is very minor compared to the size of the database (less than 0.1 percent of the size of the database). Note that the inverted lists approach is not used in this experiment.

Using the timed sales transactions database for the same experiment gives similar results (Table 2). While nonincremental mining takes more than 100 seconds, the two proposed incremental algorithms, ES and ME, take time that ranges from about six seconds for small confidence threshold values to less than one second for large confidence threshold values.

Table 3 gives a comparison of the running time of the proposed incremental mining algorithms while varying the period size (for a constant confidence threshold of 10 percent) using the power consumption database. The execution times of the nonincremental mining algorithm are given in the second column of this table. The table shows that the single-segment incremental mining algorithm (SS) is worse than the other algorithms, especially with large period sizes. This is clear since the SS algorithm considers the added segments one-by-one and may require more than one scan over the database. These results show that the entire-segments incremental algorithm (ES) is better than the merge incremental algorithm (ME) for large period values.

Fig. 6 gives a comparison of the incremental mining algorithms while varying the confidence threshold for a constant period size of 7 for the power consumption database. The figure shows that the single segment incremental mining algorithm (SS) is worse than the other two incremental algorithms. Note that the high drop in execution time for SS, which happens at confidence threshold 55 percent, occurs because of the drop in the max-subpattern tree size. In this case, the number of frequent

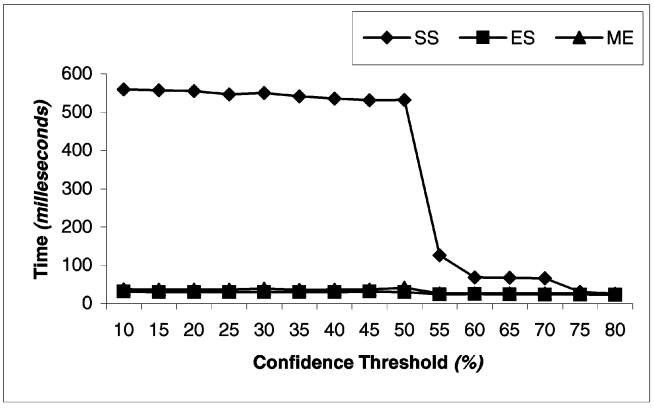

Fig. 6. Time comparison with respect to the confidence threshold (power consumption database).

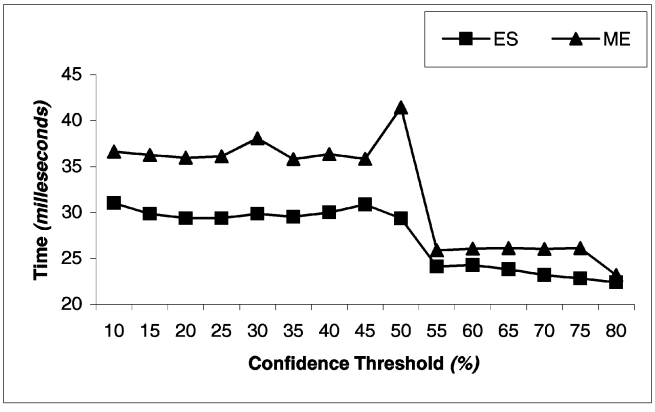

Fig. 7. Excluding single-segment incremental mining algorithm from Fig. 6.

1-patterns decreases and so is the L-length of the max-pattern $C_{\max}$ (the root node).

The scale of Fig. 6 does not show a clear comparison between algorithms ES and ME. In Fig. 7, we exclude algorithm SS from the comparison. Fig. 8 gives the results of the same study using the timed sales transactions database. The results show that algorithm ES performs better than ME. Note that, in Fig. 7, a high value is encountered at confidence threshold 50 percent for algorithm ME. The reason is that at the 50 percent confidence threshold, the number of frequent patterns in the original database happens to be less than the number of frequent patterns in the added period segments and, hence, the max-subpattern tree size of the original database is less than the max-subpattern tree size of the added segments.

There are other factors that may affect the performance of the ES and ME algorithms. Namely, we study the following

TABLE 3
Results with Respect to the Period Size Using Power Consumption Database

Period	Time (seconds)			
	Max-Subpattern Hit Set Mining	Single-Segment (SS) Incremental Mining	Entire-Segments (ES) Incremental Mining	Merge (ME) Incremental Mining
5	29.49	0.30	0.02	0.02
10	50.02	15.97	0.27	0.36
15	82.43	301.27	6.41	7.87
20	99.72	117.34	3.78	8.07

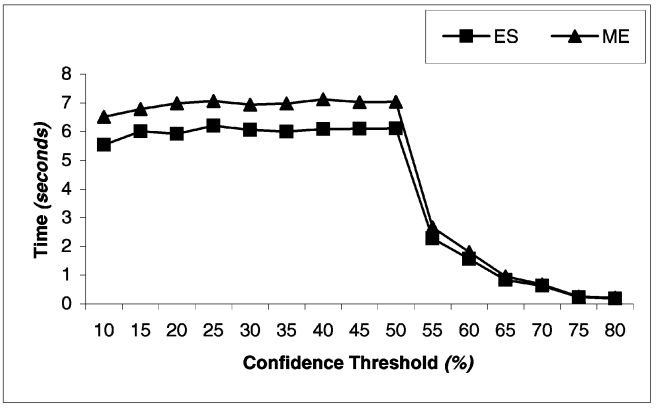

Fig. 8. Time comparison with respect to the confidence threshold (timed sales transactions database).

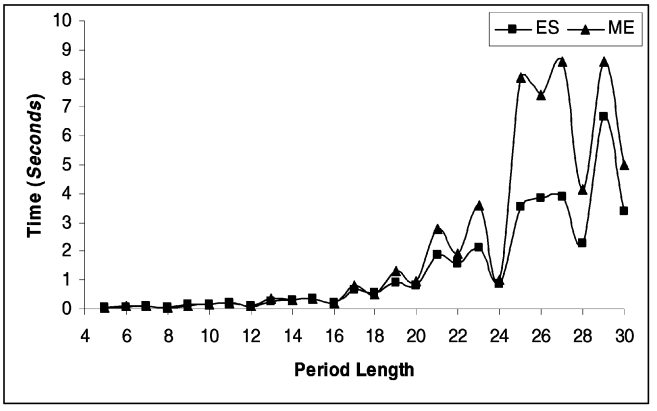

Fig. 10. Time comparison with respect to the period length at 50 percent confidence threshold.

factors: the number of levels into which the data is quantized, the period length of the periodic patterns, and the size of the increment part to the original database. We synthesize controlled data based on the timed sales transactions data. Using a synthesized data set with a variable number of quantization levels, Fig. 9 shows that the difference between the performance of the ES algorithm and that of the ME algorithm is not affected much as the number of quantization levels varies. Using synthesized data sets, each of which has the same size but a different value of the period length, Fig. 10 shows an increase trend in the time with the increase in the period length. A similar behavior is shown in Table 3. The drops in the time that occur frequently prove that the performance depends mainly on the tree size of the original database. Fig. 11 shows an expected increase in the time with the increase of the size of the increment part to the database. The three figures show that the ES algorithm always outperforms the ME algorithm.

We study the performance of the block incremental algorithm for different block sizes, while varying the confidence threshold, with a constant period size (equals to 7 for the power consumption database and 24 for the timed sales transaction data). Fig. 12 and Fig. 13 give the comparison results using the two databases, respectively. They illustrate that increasing the block size reduces the probability of frequent patterns and, hence, decreases the execution time. In Fig. 12, notice that there is a drop in execution time that is encountered at confidence threshold 60 percent for all the algorithms. The reason is that the number of frequent patterns in the added segments happens to be zero from this point on. Note also that there is a sudden increase in execution time that happens at confidence threshold 75 percent for the smallest two block sizes. In this case, the root of the max-subpattern tree gets updated, which implies that the max-subpattern tree is modified significantly, as described in Section 3.2. This results in an increase in execution time.

We study the performance of Algorithm ES while considering the inverted lists approach, discussed in Sections 3.2 and 3.3, to avoid the one scan over the database. Since the inverted lists approach is useful only when this scan is needed, i.e., when there are inserted letters in $C_{\max}$ as discussed earlier, we compare the performance of the algorithm when this scan is actually needed. Table 4 illustrates that, in those particular cases, the inverted lists approach outperforms the approach of scanning the database with average speedup of 24 percent. Of course, this is at the expense of the additional space occupied by the inverted lists.

Finally, we study the performance of the proposed algorithm for the merge mining operation. Using the timed sales transactions database, Table 5 shows the results of merge mining two time-series for two specific stores versus mining the combined time-series. The results show that we achieve a high speedup for merge mining (column 4) over mining the combined time-series (column 5). Note that merge mining assumes that the two time-series were mined previously. Yet, the results show that the total time for

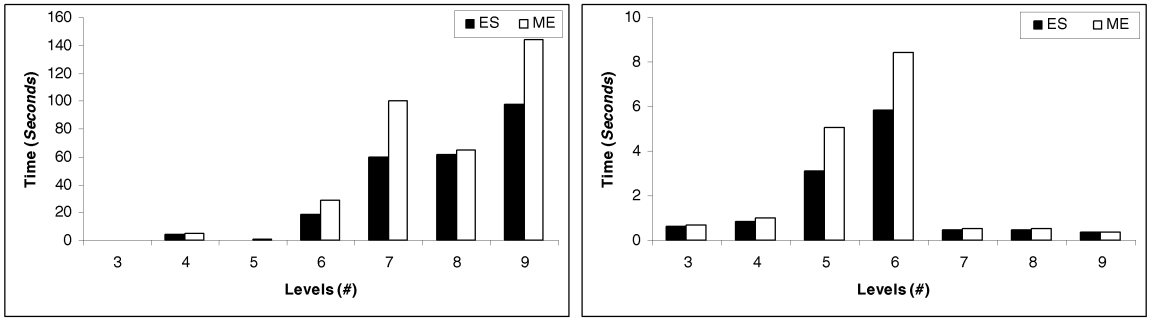

Fig. 9. Time comparison with respect to the number of quantization levels. (a) 10 percent confidence threshold and (b) 50 percent confidence threshold.

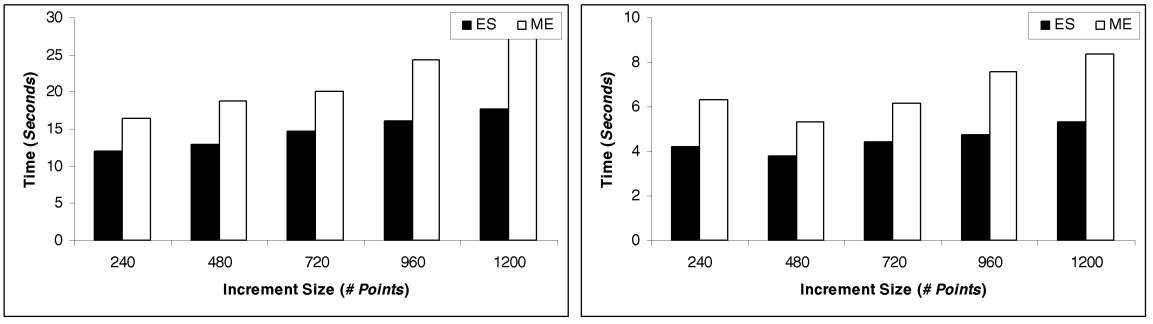

Fig. 11. Time comparison with respect to the increment size. (a) 10 percent confidence threshold and (b) 50 percent confidence threshold.

mining each time-series, and then for applying the merge mining algorithm, is again lower than the time needed for mining the combined time-series. We believe that this happens because of the increased size of the combined time-series.

7 Conclusion

Various new algorithms are proposed for the incremental mining problem, which also prove to fit the online mining problem. The performance analysis shows that the *entire-segments incremental mining* algorithm is the best among the proposed incremental algorithms. The inverted lists approach is very expensive, although it saves time as it avoids rescanning the database.

We define a new problem of *merge mining* and propose an algorithm for solving it in the context of partial periodic patterns in time-series databases. Performance analysis shows that merge mining is a promising problem to be investigated more in the context of other types of mining patterns and other types of databases.

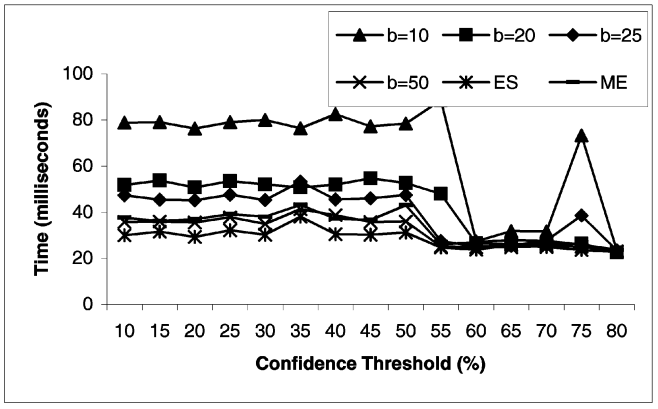

Fig. 12. Time comparison for different sizes of block incremental mining (power consumption database).

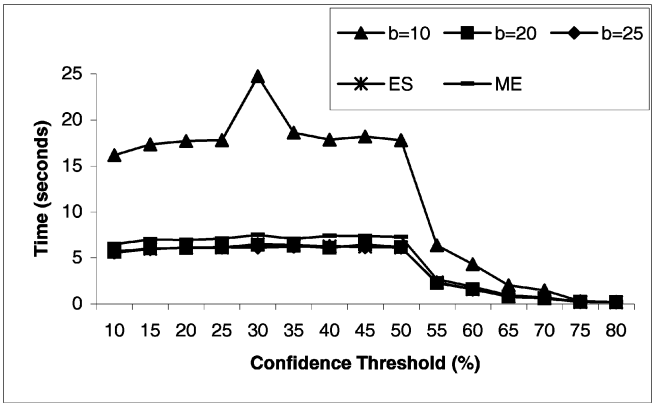

Fig. 13. Time comparison for different sizes of block incremental mining (timed sales transactions database).

Acknowledgments

This work has been supported in part by the US National Science Foundation under grants IIS-0093116, EIA-9972883, and IIS-0209120, the ARO/EPRI under Grant W08333-02, the Purdue Research Foundation, the NAVSEA/NSWC-Crane, jointly with the Purdue University Center for Sensing Science and Technology under the Integrated Detection of Hazardous Materials (IDHM) Program, and by grants from NCR and Wal-Mart.

References

[1] R. Agrawal, C. Faloutsos, and A. Swami, "Efficient Similarity Search in Sequence Databases," *Proc. Fourth Int'l Conf. Foundations of Data Organization and Algorithms,* 1993.

[2] R. Agrawal, K. Lin, H. Sawhney, and K. Shim, "Fast Similarity Search in the Presence of Noise, Scaling, and Translation in Time-Series Databases," *Proc. 21st Int'l Conf. Very Large Databases,* 1995.

[3] R. Agrawal and G. Psaila, "Active Data Mining," *Proc. First Int'l Conf. Knowledge Discovery and Data Mining,* 1995.

[4] R. Agrawal, G. Psaila, E. Wimmers, and M. Zait, "Querying Shapes of Histories," *Proc. 21st Int'l Conf. Very Large Databases,* 1995.

TABLE 4
Comparison Results with Respect to the Inverted Lists Approach (Power Consumption Database)

Confidence Threshold (%)	Entire-Segments Incremental Mining Time (*milliseconds*)	
	Inverted Lists	Database Scanning
57	51.66	66.14
58	49.22	65.74
59	49.38	65.78
60	49.38	65.52
61	50.54	65.72
62	49.34	65.68

TABLE 5
Comparison Results Regarding the Merge Mining Algorithm (Timed Sales Transactions Database)

Confidence Threshold (%)	Mining Time (1st time-series) (seconds)	Mining Time (2nd time-series) (seconds)	Merge Mining Time (seconds)	Mining Time (combined time-series) (seconds)
50	22.63	16.75	15.1	67.87
55	22.68	13.27	9.98	55.28
60	22.85	12.59	11.42	60.94
65	26.76	12.34	9.41	58.15
70	23.09	11.95	10.10	66.60

[5] R. Agrawal and R. Srikant, "Fast Algorithms for Mining Association Rules," *Proc. 20th Int'l Conf. Very Large Databases,* 1994.

[6] R. Agrawal and R. Srikant, "Mining Sequential Patterns," *Proc. 11th Int'l Conf. Data Eng.,* 1995.

[7] C. Bettini, X. Wang, S. Jajodia, and J. Lin, "Discovering Frequent Event Patterns with Multiple Granularities in Time Sequences," *IEEE Trans. Knowledge and Data Eng.,* vol. 10, no. 2, pp. 222-237, Mar./Apr. 1998.

[8] K. Chan and A. Fu, "Efficient Time-Series Matching by Wavelets," *Proc. 15th Int'l Conf. Data Eng.,* 1999.

[9] D. Cheung, J. Han, V. Ng, and C. Wong, "Maintenance of Discovered Association Rules in Large Databases: An Incremental Updating Technique," *Proc. 12th Int'l Conf. Data Eng.,* 1996.

[10] M. Ester, H. Kriegel, J. Sander, M. Wimmer, and X. Xu, "Incremental Clustering for Mining in a Data Warehousing Environment," *Proc. 24th Int'l Conf. Very Large Databases,* 1998.

[11] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos, "Fast Subsequence Matching in Time-Series Databases," *Proc. ACM SIGMOD Int'l Conf. Management of Data,* 1994.

[12] *Advances in Knowledge Discovery and Data Mining,* U. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, eds. AAAI/MIT Press, 1996.

[13] R. Feldman, Y. Aumann, A. Amir, and H. Mannila, "Efficient Algorithms for Discovering Frequent Sets in Incremental Databases," *Proc. SIGMOD Workshop Data Mining and Knowledge Discovery,* 1997.

[14] V. Ganti, J. Gehrke, and R. Ramakrishnan, "DEMON: Mining and Monitoring Evolving Data," *Proc. 16th Int'l Conf. Data Eng.,* 2000.

[15] M. Garofalakis, R. Rastogi, and K. Shim, "SPIRIT: Sequential Pattern Mining with Regular Expression Constraints," *Proc. 25th Int'l Conf. Very Large Databases,* 1999.

[16] J. Gehrke, V. Ganti, R. Ramakrishnan, and W.-Y. Loh, "BOAT: Optimistic Decision Tree Construction," *Proc. ACM SIGMOD Int'l Conf. Management of Data,* 1999.

[17] J. Han, G. Dong, and Y. Yin, "Efficient Mining of Partial Periodic Patterns in Time Series Databases," *Proc. 15th Int'l Conf. Data Eng.,* 1999.

[18] J. Han, W. Gong, and Y. Yin, "Mining Segment-Wise Periodic Patterns in Time-Related Databases," *Proc. Fourth Int'l Conf. Knowledge Discovery and Data Mining,* 1998.

[19] C. Hidber, "Online Association Rule Mining," *Proc. ACM SIGMOD Int'l Conf. Management of Data,* 1999.

[20] H. Mannila, H. Toivonen, and A. Verkamo, "Discovering Frequent Episodes in Sequences," *Proc. First Int'l Conf. Knowledge Discovery and Data Mining,* 1995.

[21] J. Quinlan, "Induction of Decision Trees," *Machine Learning,* vol. 1, pp. 81-106, 1986.

[22] D. Rafiei, "On Similarity-Based Queries for Time-Series Data," *Proc. 15th Int'l Conf. Data Eng.,* 1999.

[23] R. Srikant and R. Agrawal, "Mining Sequential Patterns: Generalizations and Performance Improvements," *Proc. Fifth Int'l Conf. Extending Database Technology,* 1996.

[24] S. Thomas, S. Bodagala, K. Alsabti, and S. Ranka, "An Efficient Algorithm for the Incremental Updation of Association Rules in Large Databases," *Proc.Third Int'l Conf. Knowledge Discovery and Data Mining,* 1997.

[25] P. Utgoff, "ID5: An Incremental ID3," *Proc. Fifth Int'l Conf. Machine Learning,* pp. 107-120, 1988.

[26] K. Wang and J. Tan, "Incremental Discovery of Sequential Patterns," *Proc. SIGMOD Data Mining Workshop Research Issues on Data Mining and Knowledge Discovery,* 1996.

Walid G. Aref is an associate professor of computer science at Purdue University. His research interests are in developing database technologies for emerging applications, e.g., spatial, multimedia, genomics, and sensor-based databases. He is also interested in indexing, data mining, scalable media servers, and geographic information systems (GIS). Professor Aref's research has been supported by the US National Science Foundation, DoN, Purdue Research Foundation, CERIAS, Panasonic, and Microsoft Corp. In 2001, he received the CAREER Award from the US National Science Foundation. He is a member of the ACM and the IEEE.

Mohamed G. Elfeky received the BSc and MSc degrees in computer science from Alexandria University, Egypt, in 1996 and 1999, respectively. He is pursuing a PhD degree in computer science at Purdue University. His current research interests include data mining, data quality, and object-orientation.

Ahmed K. Elmagarmid received the BSc degree from the University of Dayton in 1977, and the MS and PhD degrees from the Ohio State University in 1980 and 1985, respectively. He is a chief scientist in the Office of Strategy and Technology at Hewlett-Packard (HP). He is responsible for software strategy coming out of the corporate CTO office. As chief scientist in the Office of Strategy and Technology, he contributes to cross company roadmap initiatives and serves on the technology council for HP. He works closely with the business units to identify areas of leverage in the software directions for HP. He was director of the Indiana Center for Database Systems and the Indiana Telemedicine Incubator. He is on leave from Purdue University where he serves as a professor of computer science. He also served on the faculty of The Pennsylvania State University and the University of Padua. He has worked on long term consulting engagements with Harris Commercial Systems, IBM, Bellcore, Telcordia, MDL, UniSql, MCC, CSC, DoD, the Padua Chamber of Commerce, and the Italian Government. He received a US National Science Foundation Presidential Young Investigator award from President Ronald Reagan, and distinguished alumni awards from Ohio State University and the University of Dayton in 1988, 1993, and 1995, respectively. Dr. Elmagarmid is the editor-in-chief of the *Distributed and Parallel Databases: An International Journal*, editor of the *IEEE Transactions on Knowledge and Data Engineering*, *Information Sciences Journal*, *Journal of Communication Systems*, and editor of the book series on *Advances in Database Systems*. He has written six books and more than 150 papers in database systems. He is a senior member of the IEEE and the IEEE Computer Society.