Repetition support and mining cyclic patterns

Ismail H. Toroslu*

Middle East Technical University, Department of Computer Engineering, 06531 Ankara, Turkey

Abstract

For customer transaction database, the support (customer support) of the sequential pattern is defined as the fraction of the customers supporting the sequence. We define new forms of the mining patterns, called cyclic patterns, as an extension to the sequential pattern mining by introducing a new parameter, the repetition support. For customer transaction database, the repetition support specifies the minimum number of repetitions of the patterns in each customer transaction sequence. Repeated patterns can also be viewed as cyclic since the beginning of a sequence will follow the end of the previous occurrence of the same sequence. In this paper, we introduce the repetition support parameter, the cyclic pattern mining problem, describe the related algorithms, and at the end of the paper we give some performance results.

© 2003 Elsevier Ltd. All rights reserved.

Keywords: Data mining; Sequential patterns; Cyclic patterns; Repetition support

1. Introduction

The introduction of the association rule mining (Agrawal, Imielinski, & Swami, 1993) can be considered as the first introduction of the data mining to the database community. Since then, the field of data mining has attracted a lot of researchers from the database community. Several different forms of data mining problems have been investigated, and many different techniques have been developed. Discovering sequential patterns in large databases is one of the most well-known data mining techniques. Sequential pattern mining is first introduced in Agrawal and Srikant (1995) for customer transaction databases. The original form of the sequential pattern mining problem started to get new attention (Garofalakis, Rastogi, & Shim, 1999; Han, Pei, & Yin, 2000; Pei et al., 2001; Zaki, 1998), in addition to the early works on this problem (Agrawal & Srikant, 1995; Srikant & Agrawal, 1996). The roots of the sequential pattern mining problem can be traced back to AI as in Dietterich and Michalski (1985) and string processing as in Wu and Manber (1992). However, it got the attention of the database community after its redefinition in Agrawal and Srikant (1995) as a database problem. Sometimes, sequential pattern mining is treated under more general class of time-series analysis (Han, 1999).

The time-series analysis problem had received more attention from the researchers. Some recent ones are presented in Agrawal, Faloutsos, and Swami (1993), Bettini, Wang, and Jajodia (1998), Han, Dong, and Yin (1999), Han et al. (2000), Huang and Yu (1999) and Mannila, Toivonen, and Verkamo (1997). We extend the very basic form of the sequential patterns to the cyclical patterns. The technique presented in Srikant and Agrawal (1996) also generalizes the sequential pattern mining by considering temporal information on the transactions. However, that work is different than ours, since we have defined our extension not on the temporal information, but directly on the original form of the sequential pattern concept presented in Agrawal and Srikant (1995), by introducing the repetition support and cyclic patterns concepts. In order to define the mining cyclic patterns, we introduced a new parameter, called repetition support in addition to the single support concept in the sequential pattern mining. Our work is also different from Ozden, Ramaswamy, and Silberschatz (1998) since that work deals with association rules.

Addition of this new parameter makes the new mining technique more efficient than sequential pattern mining since it introduces an additional constraint on the patterns.

1 The early version of this work, that only introduces the repetition support and gives the sketch of the cyclically repeated pattern mining problem, is presented in Toroslu and Kantarcıoglu (2001).
searched and hence significantly reduces their numbers. Data mining is basically used for decision support process. Therefore, introducing a new parameter will also give the decision maker more flexibility in obtaining different results. Moreover, it could be easier for the decision maker to interpret the results of the mining process that produces less number of patterns.

The term ‘cyclic patterns’ have been used in data mining to represent patterns repeating themselves within cycles of times as in Bettini et al. (1998). As in classical sequential pattern mining problem, in our cyclic pattern mining problems temporal constraint are not considered. By using the repetition support in addition to the ordinary support concept, it is also possible to define several different forms of repeated cyclic/acyclic patterns. In this paper, cyclic pattern is defined as a repeated sequential pattern satisfying the repetition support. In each repetition, the beginning of a sequence will follow the end of the previous occurrence of the same sequence. Therefore, these repeated sequences are also called cyclically repeated patterns. Also a repeated sequence forms shifted versions of the same sequence when different starting points are chosen in the first occurrence of the sequence. All such patterns can be viewed as related, and together they are called as a family of cyclically repeated patterns.

Consider the sequence ‘CACBACACABBACABA-CAAC’. In this sequence the pattern AAB has two, and the patterns ABA and BAA have three repetitions (possibly interleaved with other characters). Let us also assume the repetition support 3. Even though there are only two repetitions of the sequence AAB in this sequence, AAB is called as a cyclically repeated pattern, supported by the given sequence, since its shifted versions (members of the same family) ABA and BAA has three repetitions satisfying the repetition support.

We can introduce the cyclic pattern mining with an example from a daily stock value changes database. We have selected this domain because of its simplicity and similarity to real life applications. There are also more complicated applications from bioinformatics domain (Rigoutsos & Floratos, 1998; Rigoutsos, Floratos, Parida, Gao, & Platt, 2000).

Daily stock value change database has three attributes: the stock name, the percentage change of its value in terms of its previous day’s value, the date. From this database, it is possible to discover patterns related to stock value increases and decreases. Instead of real percentage values, more abstract concepts like, D (representing more than 10% of the daily value Decrease), I (representing more than 20% of the daily value Increase), and O (Other kinds of changes) are used in order to discover more meaningful patterns. It is also possible to choose another set of abstractions. We assume that the original stock value change database is transformed into a new form including the abstract values instead of the real percentage change values.

In order to search for cyclic patterns, we need to specify a stock support (such as 5%), representing the percentages of stocks supporting this pattern, and the repetition support (such as 5), representing the minimum number of repetitions of the patterns of each stock supporting this pattern. Note that, the repetition support can also be defined as a percentage of the size of the database, but it is easier to define it as a constant.

From the transformed database, a pattern like DDDI can be discovered. This pattern can be interpreted as follows:

It is likely that in a stock market, if the price of a stock ‘decreases more than 10%’ in a day, three times (not necessarily in consecutive days), these three decreases will likely be followed by an ‘increase by more than 20%’ of the stock price (again not necessarily in the consecutive day).

Using the ordinary sequential pattern mining technique, similar kind of rules can also be discovered. However, since in the sequential pattern mining technique, the repetition support is not required, even the stocks having the searched pattern only once in their entire history, are also considered as supporting the pattern. We think such stocks should not be considered in discovering the rules like above. In our approach, the user has to provide an additional parameter in order to introduce a constraint on the minimum number of repetitions of the patterns for each stock. Only the stocks with at least that many repetitions of the patterns, can contribute to the overall stock support value. Similar to the sequential pattern mining technique, in our technique, it is also possible to discover all the patterns with required supports at the same time. For example, another pattern, such as IID, might also be discovered together with the above pattern if it also satisfies the support constraints.

In cyclic pattern mining, in discovering the pattern DDDI, a stock with a pattern DDID repeated at least as many times as the repetition support value also contributes to the stock support, since both patterns are from the same family. If DDID has $N$ repetitions, then DDDI can have either $N - 1$, or $N$, or $N + 1$ repetition. Therefore, in the worst case the pattern that is being searched can be repeated only one less than the repetition support value and still being considered as having sufficient repetitions. Therefore, all the patterns in a family are treated together.

Since the audience of this field is familiar with the customer transaction databases, in the rest of the paper we will also use customer transaction database of Agrawal et al. (1993) and Agrawal and Srikant (1995) as an example. Therefore, instead of the stock support value, which is mentioned above, customer support value is going to be used. Customer transaction database is a relation consisting of the following fields: customer id, transaction time, and items purchased. Itemset is any set of items, which can be purchased. Sequence is an ordered list of itemsets. Other
items can be interleaved with the sequences that are being searched.

The definition of the customer support is also adopted from Agrawal and Srikant (1995). In our technique, customer support of a pattern represents the fraction of customers with that pattern having the required repetition support. Sequential pattern mining is a special case of cyclic mining problem. In order to discover sequential patterns, the repetition support must be chosen as one, in addition to small changes on the algorithms.

The rest of the paper is organized as follows: in Section 2, the main steps of the cyclic pattern mining technique are described. The main difference of the cyclic pattern mining techniques from the sequential pattern mining is in the fourth phase, which is called as repeated sequence phase in our technique, and it corresponds to the sequence phase of the sequential pattern mining technique of Agrawal and Srikant (1995). In Section 3, we describe how the addition of the repetition support changes that phase, especially the candidate generation process. Section 4 describes the overall structure of the repeated sequence phase. In Section 5, some performance results are given, and finally Section 6 presents the conclusions.

2. Sketch of the algorithm

Even though there are a few new algorithms proposed for sequential pattern mining as in Han et al. (2000) and Zaki (1998), we have chosen to use the original classical algorithm of Agrawal and Srikant (1995) since it can easily be adapted for the cyclic pattern mining problem. The main idea of this paper is not only to develop an efficient algorithm for cyclic pattern mining problem, but also to show the efficiency of the cyclic pattern mining as a whole new technique. We claim that this new technique works efficiently than the sequential pattern mining technique and produces more useful results independently of the algorithms chosen to implement it.

Since in our approach the number of patterns discovered is reduced due to the repetition support, it works more efficiently than the sequential pattern mining technique in all steps. Reducing the memory requirement also contributes to the execution time by reducing the page swaps between the main memory and the secondary storage. Except the fourth phase, other phases of our technique are almost the same as the one in Agrawal and Srikant (1995). Below they are briefly defined. The details of these phases of the original sequential pattern mining algorithm can be found in Agrawal and Srikant (1995).

Phase 1: Sort Phase. The database is sorted using customer id as the major key and transaction time as the minor key. There is no change in this phase for the cyclic pattern mining.

Phase 2: Litemset Phase. The set of large item sets, satisfying the given supports, are determined. They are called as litemsets. These itemsets are mapped to contiguous integers after they are determined. The difference between this phase in our algorithms and the one in Agrawal et al. (1993) is that, the litemsets in our technique should also satisfy the repetition supports for each customer. Extending the transformation phase of Agrawal and Srikant (1995) with this additional support constraint is very simple. In order to increment the customer support count, the number of repetitions of the litemsets for each customer is also determined. If the number of repetitions is greater than or equal to the required repetition support then the customer support is increased. As in Agrawal and Srikant (1995), where litemsets correspond to large-1-sequences, the litemsets in our technique also correspond to large-1-cyclic-patterns.

Phase 3: Transformation Phase. This phase is the same as the one in Agrawal and Srikant (1995). The original database is transformed into a database of litemset identifiers by replacing each transaction by the set of all the litemsets in that transaction. Also, the transactions that do not appear in litemsets are removed from the database.

Phase 4: Repeated Sequence Phase. This phase utilizes ‘a priori all’ algorithm defined in Agrawal and Srikant (1995) with some modification. This simply means that through the iterations, using the cyclic patterns generated just in the previous iteration, larger cyclic patterns are generated. At iteration $i$, possible candidates of that iteration, or large-$i$-candidates are generated by using the large-$i$-cyclic-patterns constructed in the previous iteration. These candidates are stored in a hash-tree structure in order to efficiently search each customer transaction sequence, to determine their repetition supports and then, by combining those supports to determine their customer supports. Those candidates having at least the required minimum supports become large-$i$-cyclic-patterns. The hash-tree data structure, which is used to store the candidates, has also been slightly modified. The details of this phase are given in the following sections.

Phase 5: Maximal Phase. Maximal cyclically repeated patterns are determined from the set of all large cyclically repeated patterns. This phase is also very similar to the one in Agrawal and Srikant (1995). Since each large cyclically repeated pattern represents a family of the patterns, just a simple subsequence check among large cyclically repeated patterns is not enough for the containment check. The subsequence check is extended in order to consider the subsequences that can be obtained by wrapping around the patterns.

3. Candidate generation in repeated sequence phase

In this section, we are going to use the following customer transactions sequence, which represents the list of the set of the litemsets obtained after the transformation
phase:

\[\{[1], [2], [2, 3, 4], [3, 4], [1], [2], [2, 3, 4], [3, 4, 1], [1, 4]\}\]

For this example, we assume that the minimum required repetition support is 2. At the first iteration, the supports of large-1-cyclic patterns will be equal to the number of the occurrences of the litemsets. Therefore, large-1-cyclic patterns \([1], [2], \text{and} [3]\) have supports 4, and large-1-cyclic-pattern \([4]\) has a support 5.

When the length of the patterns is 2, there will be only pairs of the form \([a, b]\) and \([b, a]\) in a family. Only one pattern from each family of patterns with the highest support representing that family is kept. Therefore, for our example, we need to consider six patterns, representing all possible families of patterns with four litemsets. Among these patterns, \([1,2], [1,3], [2,3], [2,4], \text{and} [3,4]\) have supports 2, and the pattern \([1,4]\) has a support 3.

When the number of distinct litemsets is \(k\), the maximum number of possible candidates with length \(n\) is \(k!(k - n)!\). Only \(k!(n, (k - n)!\) of them are needed to represent all of them. Therefore, the number of the candidates (and therefore the size of the hash-tables) has also been reduced by the ratio of \(n\) (the length of the candidates) by keeping only one representative from each family of patterns. In sequential pattern mining, all the candidates had to be represented in the hash-table. In this example, there are four distinct litemsets, and eight patterns can represent all 24 possible candidates that can be generated with length three.

Some of the patterns, which are indirectly represented, do not necessarily have to have the required repetition support, and they might have one less repetition support than their representatives. For example the pattern \([3,4,2]\) represented by \([2,3,4]\) has only repetition support 1, but since \([2,3,4]\) (which is the representative of the family) has the required support, \([3,4,2]\) will also be considered for producing larger candidates in the next iteration. No patterns from the families of \([1,4,2]\) and \([1,3,2]\) satisfy the required repetition support 2. All the members of these families have the same support, 1. Therefore, they will be considered as not supported by our transaction sequence. As we mentioned above, among eight patterns representing eight different families, two of them, namely \([1,4,2]\) and \([1,3,2]\) do not have enough support, so they will be ignored for the next iteration. All the others \(([1,2,3], [1,2,4], [1,3,4], [1,4,3], [2,3,4], \text{and} [2,4,3])\) have support 2.

The candidate generation process is different than the one in Agrawal et al. (1993), since only a representative of a family is used to represent all the members of that family. For cyclically repeated patterns, during the third iteration, the hash-tree shown in Fig. 1 is obtained, that contains the candidates of the iteration. The hash-tree in Fig. 1 is generated by using the hash function \(h(k) = k \mod 2\). Therefore, each intermediate node has two pointers. The bucket size of the leaf nodes should be large enough to be able to store all the distinct litemset identifiers with the same hash value in the same node. In this example, the bucket size is two. This size is sufficient for our example, since the litemset identifiers are only between 1 and 4. Therefore, there can be at most two litemsets having the same hash value with this hash function.

In cyclic pattern mining in order to generate the candidates with ‘length \(n + 1\)’ for the next iteration, \(n\) many ‘length \(n - 1\)’ subsequences of each large-\(n\)-patterns are searched for joining. A candidate with ‘length \(n + 1\)’ is obtained by joining two large-\(n\)-patterns found (and therefore supported) in the previous iteration (the iteration \(n\)) by using their common ‘length \(n - 1\)’ subsequences, and then by adding the remaining two litemsets of these two patterns. For the above example, to go from ‘length 3’ to ‘length 4’, all the subsequences of the candidates with ‘length 2’ must be checked. For example, for the pattern \([1,2,3]\), three subsequences, namely \([1,2], [2,3], \text{and} [3,1]\), should be considered. Again notice that, since we consider the patterns as cyclic, the subsequence \([3,1]\) is also included, which is not included in sequential pattern mining.

Since each pattern represents a family of patterns (a pattern with length \(n\) represents \(n\) patterns), and only one instance is used to represent cyclic patterns, the whole family or all instances of the cyclic pattern should be considered in order to determine the candidates of the next iteration. For example the patterns \([1,2,3]\) and \([2,3,4]\) can be used to generate candidates \([2,3,1,4]\) and \([2,3,4,1]\). In this example, the pattern \([2,3,1]\) is from the family represented by \([1,2,3]\). When the large-3-patterns are joined by using their common subsequences, the large-4-candidates shown in Table 1 will be obtained. The patterns \([1,3,2]\) and \([1,4,2]\) are excluded from this table since these patterns do not have sufficient support.

During the generation of the new ‘length \(n + 1\)’ candidates for the next iteration, each new candidate should be checked if a previously generated candidate already represents it, and if so, that candidate is discarded. Among
all these large-4-candidates, assuming they are generated in the order shown in the above table, after removing the ones represented by previously generated candidates, there will be only the six candidates at this iteration (see the first two columns of Table 2).

After this step, the large-4-candidates having at least one large-3-subsequence that does not have enough support should also be removed from the candidate set. Each large-n-candidate has \( n \) large-(\( n - 1 \))-subsequence; some of them are obtained by wrapping around the pattern. Again, for the subsequences, it is sufficient to check whether their representatives have required supports. Actually all of the ‘length \( n - 1 \)’ subsequences (total \( n \) of them) can be obtained, each time by deleting a different itemset from ‘length \( n \)’ candidate. For our example, Table 2 also shows the supports of the candidates from their subsequences.

Since only candidates \([1,2,3,4]\) and \([1,2,4,3]\) have all their subsequences supported in the previous iteration, they will be the only two candidates that will be considered in this iteration. In our example customer sequence, both candidates’ the repetition supports are 2, thus, they both become large-4-cyclically repeated-patterns.

### 4. Repeated sequence phase

We are going to use the sample database in Fig. 2 in order to describe the details of our algorithm. In this database, there are 4 customers and each transaction (items purchased) is represented by a set of capital letters.

The support values are as follows:

- Repetition support = 2 (which means a pattern must be repeated at least twice by a customer in order to be considered as supported).
- Customer support = 3 (which means there must be at least three different customers for the pattern satisfying the given repetition support).

Basically, the customer support is used as in the ordinary sequential pattern mining for eliminating patterns without sufficient supports. However, in determining the customer support, the repetition support of the pattern is also taken into account for each customer. Simply when a customer transaction is processed, the customer support count is incremented only for the patterns that also satisfy the repetition support constraint.

In this database the following items do not have enough customer support with a given repetition support:

- Item E: Customer support = 1 (Only from the customer 1, with repetition support 3.)
- Item F: Customer support = 1 (Only from the customer 3 with repetition support 2. Other insufficient supports from the customers 2 and 4 with repetition support 1 are discarded.)
- Item G: Customer support = 0 (Three insufficient supports form customers 1, 2 and 4 with repetition support 1, and they are all discarded.)

Therefore, items E, F, and G are discarded from the database. After that, the database is transformed into a new form including only large item sets (litemsets). The litemsets are mapped to unique integers. In our original database the following mappings are performed:

\[
\begin{align*}
A & \rightarrow 1 \\
B & \rightarrow 2 \\
C & \rightarrow 3 \\
D & \rightarrow 4 \\
A,B & \rightarrow 5
\end{align*}
\]

Fig. 2 also shows the transformed database. Note that the item number 5 in the transformed database represents actually two original items, which has been determined in the litemset phase of the algorithm. But, since these two items satisfy the support conditions together, they can be treated like a single item in the rest of the computation.

In Fig. 2, each one of the integers in this database represents a litemset with sufficient customer (and therefore

<table>
<thead>
<tr>
<th>Candidate</th>
<th>Represents (discarded)</th>
<th>Subseqs with support</th>
<th>Subseqs w/o support</th>
</tr>
</thead>
<tbody>
<tr>
<td>[1,2,3,4]</td>
<td>[2,3,4,1],[4,1,2,3],[3,4,1,2]</td>
<td>[1,2,3],[2,3,4],[3,4,1,2]</td>
<td>[1,2,3,4]</td>
</tr>
<tr>
<td>[1,2,4,3]</td>
<td>[3,1,2,4],[2,4,3,1],[4,3,1,2]</td>
<td>[1,2,4],[2,4,3],[4,3,1,2]</td>
<td>[1,2,4,3]</td>
</tr>
<tr>
<td>[3,1,4,2]</td>
<td>[2,3,1,4]</td>
<td>[3,1,4],[4,2,3],[2,3,1]</td>
<td>[1,4,2]</td>
</tr>
<tr>
<td>[4,1,3,2]</td>
<td>[2,4,1,3]</td>
<td>[4,1,3],[3,2,4],[2,4,1]</td>
<td>[1,3,2]</td>
</tr>
<tr>
<td>[3,4,2,1]</td>
<td>[2,4,1,3]</td>
<td>[3,4,2],[1,3,4]</td>
<td>[2,1,3]</td>
</tr>
<tr>
<td>[4,3,2,1]</td>
<td>[2,4,1,3]</td>
<td>[4,3,2],[1,3,4]</td>
<td>[2,1,3]</td>
</tr>
</tbody>
</table>

Table 2

<table>
<thead>
<tr>
<th>Large-4-candidates</th>
</tr>
</thead>
<tbody>
<tr>
<td>[1,2,3,4]</td>
</tr>
<tr>
<td>[1,2,4,3]</td>
</tr>
<tr>
<td>[3,1,4,2]</td>
</tr>
<tr>
<td>[4,1,3,2]</td>
</tr>
<tr>
<td>[3,4,2,1]</td>
</tr>
<tr>
<td>[4,3,2,1]</td>
</tr>
</tbody>
</table>
In the repeated sequence phase of our algorithm, similar to the ordinary sequential mining techniques, maximal large cyclically repeated patterns are determined. In order to do this, at each iteration step, larger item sets should be generated with the required supports. In this section the term support is used to represent customer supports satisfying the repetition support conditions. Initially, itemsets 1–4 of Fig. 2 have supports 4 and itemset 5 has a support 3.

In the next iteration, large-2-cyclic-patterns satisfying the both support conditions are determined. Note that many patterns are eliminated since they do not have sufficient customer supports with the required repetition supports. For example, the pattern [1,2] does not have enough repetition support for customer 1 and customer 2 (which is just one), although the same pattern has the repetition support 3 for the customer 3 and customer 4. Although the pattern [1,2] has a repetition support 2 for customer 4, it has been considered to satisfy the repetition support 3, by using the repetition support of another pattern, [2,1]. The pattern [2,1] is the representative of the family since it has the highest support for the family of [1,2] and [2,1]. As a result, pattern [1,2] has a total support 2, and this is less than the required support. Similarly pattern [2,3] will be considered as not supported since its support is also two, and other patterns [1,3] with support 4, and [1,4], [2,4], and [3,4] with supports 3 are considered as supported. After all the patterns with two itemsets having sufficient supports are determined, in the next step, the patterns with three itemsets are determined, and only the pattern [1,4,3] has the required support 3.

The main operations of the repeated sequence phase include insertion of the candidates into the hash-tree, and searching the candidates of the hash-tree in a customer transaction sequence to determine their repetition supports. We have modified the insertion and search procedures of the sequential pattern mining technique of Agrawal et al. (1993) in order to discover cyclically repeated patterns.

Fig. 3 shows the hash-tree insert algorithm. In order to determine the repetition supports of the candidates some additional information must be stored for each candidate, namely the count of the candidate and the ending transaction number of its last occurrence in a currently processed customer transaction. Therefore, during the execution of the sub-procedure add at line 3 of Fig. 3, a pointer from the candidate, which is being inserted into the hash-tree, to an entry of a list called candidate-list has to be generated which will be used to store these two values.

In the new hash-tree-search algorithm (shown in Fig. 4), whenever a candidate is detected in a customer sequence, its repetition support count has to be incremented and its end trans (end transaction numbers) of the last occurrence up to that transaction in a customer sequence has to be recorded. For the hash-tree-search algorithm the following global variables are used:

- candidate-list: a global list keeping the counts of candidates and the end-trans (end transaction numbers) of the last occurrences of the candidates in a customer sequence. Candidates in the hash tree have pointers to the entries of the candidate-list,
† length: a global variable that shows the length of the candidates searched,
† transactions: a global variable representing one customer’s transactions, which is a list of set of itemsets.

Note that, in sequential pattern search, the hash-tree search algorithm should not search the same patterns more than once. This can be done by restricting the recursive calls for only the full-length candidates with counts zero at lines 13 and 20 of the algorithm in Fig. 4.

The modifications made in the hash-tree search algorithm in order to count all the occurrences of the candidates are as follows:

1. The starting transaction number of the partially generated candidate that is being searched is also passed as a parameter to the search algorithm.
2. Whenever the searched pattern is found as a candidate in the hash-tree, its last occurrence ending transaction number is compared to the starting transaction number of the candidate, and if they are equal, the count of that candidate is incremented.

Fig. 3. Hash-tree-insert algorithm: inserts the new candidate (new-cand) into a hash-tree (hash-tree-ptr), starting from the root of the hash-tree (level = 1).

```c
0) hash-tree-insert (hash-tree-ptr, new-cand, level) {
1)   if (hash-tree-ptr.node-type = leaf-node) {
2)     if (not-full(hash-tree-ptr.bucket)) // if there is a space for the new candidate
3)       add (new-cand, hash-tree-ptr.bucket) // add it into the bucket
4)     else { // convert the node to internal node
5)       temp-bucket := hash-tree-ptr.bucket
6)       create-hash-table(hash-tree-ptr) // converts into internal node with hash table
7)       create-children(hash-tree-ptr) // children (leaves) are pointed by hash-table
8)     } // move the current entries and the new-candidate to appropriate children using the
9)     // litemset corresponding to the level number in the hash function
10)    for all cand(i) in temp-bucket do {
11)       hash-tree-ptr := hash-tree-ptr->children(hash(cand(i).litemset(level)));
12)       hash-tree-insert(hash-tree-ptr, cand(i), level+1) }
13)   } // intermediate node
14)   hash-tree-ptr := hash-tree-ptr->children(hash(new-cand.litemset(level)));}
15) } // hash-tree-insert
16) hash-tree-insert(hash-tree-ptr, new-cand, level+1) }
```

Fig. 4. Hash-tree-search algorithm: searches a partially or fully generated candidate (c-cand) in a hash-tree (hash-tree-ptr) using its starting transaction number (start-trans), its current length (current-length), and the current transaction number (current-trans).

```c
0) hash-tree-search (start-trans, hash-tree-ptr, c-candidate, current-length, current-trans) {
1)   if (hash-tree-ptr.node-type = leaf-node) {
2)     if (c-candidate.length = length)
3)       // The function in returns the pointer in candidate-list if c-cand is in the bucket
4)     else if (c-cand.start-trans > candidate-list(c-cand).end-trans { // new occurrence is found
5)       candidate-list(c-cand).count++ // increment count of candidate in candidate-list
6)       candidate-list(c-cand).end-trans := current-trans } // update last occurrence end number
7)     else if (in (prefix(c-candidate,current-length), hash-tree-ptr.bucket)) //if the prefix is in the bucket
8)       for all litemset(i, j) for all i > current-trans // search for the candidate by extending it
9)         for all j : transactions(i).litemsets(j) {
10)        c-candidate := append (c-candidate, litemset(i, j)) // append new litemset to c-cand
11)       } // continue searching the rest in the same leaf node w/o advancing the pointer
12)       hash-tree-search(start-trans, hash-tree-ptr, c-candidate, current-length+1, i ) }
13) } // intermediate node
14) } // for all litemset(i, j) for all i > current-trans // search for the candidate by extending it
15) } // for all j : transactions(i).litemsets(j) {
16) } // continue searching the rest in the hash-tree advancing the pointer
17) } // hash-tree-search(start-trans, hash-tree-ptr, c-candidate, current-length+1, i ) }
```

the currently being searched occurrence to determine whether the currently being searched occurrence had started after the last occurrence had ended (line 5 in Fig. 4). Only if that is the case, the counter of the candidate is incremented, and the ending transaction number of the candidate is updated as the current transaction number (lines 6 and 7 in Fig. 4).

5. Performance results

We have used one of the datasets of Agrawal and Srikant (1995) in our performance evaluations. Since there were not many cyclically repeated patterns in these datasets we have created three new datasets from C10-T2.5-S4-I1.25 (originally has 25,000 customers) by using the first 5000 customers and repeating their transactions 2, 3, and 4 times. These new datasets are called as C10-T2.5-S4-I1.5-2, C10-T2.5-S4-I1.5-3, and C10-T2.5-S4-I1.5-4, respectively. The rest of the parameters in these datasets have not been changed. Our algorithm is applied on these three new datasets with customer support as 1%. Since the main difference of the cyclic pattern mining from the sequential pattern mining is in the repeated sequence phase, we have obtained the execution times of this phase only for the repetition support values from 1 to 5 (Fig. 5). We have also normalized the results by setting the result for the repetition support 1 as 100.

Our performance results show that, searching for cyclically repeated patterns with higher repetition supports significantly decrease the execution time. Since the main reason for this improvement is due to the decrease in the number of candidates searched, similar kind of improvement should be expected from other kind of implementations of these techniques (such as by modifying other sequential pattern mining algorithms for the cyclically repeated pattern mining).

In most cases, the new mining approach discovers much smaller number of patterns than the sequential pattern mining. There are two reasons for this. First, the repetition support introduces an extra constraint for the patterns, and secondly representing the family of patterns with a single pattern reduces the possible number of patterns. This affects all the steps of the mining process by decreasing the number of candidates and by reducing the sizes of hash trees. Due to this, in cyclically repeated pattern mining the memory requirement is also reduced.

6. Conclusions

In this paper, we have extended the sequential data mining technique by adding a new constraint, namely the repetition support. By using this addition, a new mining technique, called cyclic pattern mining is defined. The cyclically repeated pattern mining technique seems to have natural applications, and more efficient than the sequential pattern mining.

We have modified the data structures and the algorithms presented in Agrawal and Srikant (1995) to discover new forms of patterns. We have also introduced the concept of the pattern families as a natural extension to the cyclically repeated pattern mining.

Data mining is basically used in decision-making. Therefore, it might be more desirable to generate smaller number of patterns, so that they can be interpreted easily in a more meaningful way. The additional repetition support also provides more flexibility for the search process. As a result, different forms of constraints can be introduced on the patterns in order to discover different kinds of patterns.

As a future work different forms of cyclic pattern mining techniques might be investigated. For example the requirement for repeating the same pattern can be changed to repeating the patterns of the same family. Or, instead of...
interleaved versions, non-interleaved versions of the cyclic patterns might be searched.

References


Han, J., Pei, J., Mortazavi-Asl, B., Chen, Q., Dayal, U., & Hsu, M.-C. (2000). Freespan: Frequent pattern-projected sequential pattern mining (pp. 355–359). Proceedings of the International Conference on Knowledge Discovery and Data Mining (KDD’00), Boston, MA.

Han, J., Pei, J., & Yin, Y. (2000). Mining frequent patterns without candidate generation (pp. 1–12). Proceedings of the 2000 ACM-SIGMOD International Conference on Management of Data (SIGMOD’00), Dallas, TX.


Pei, J., Han, J., Pinto, H., Chen, Q., Dayal, U., & Hsu, M.-C. (2001). PrefixSpan: Mining sequential patterns efficiently by prefix-projected pattern growth. Proceedings of the 2001 International Conference on Data Engineering (ICDE’01), Heidelberg, Germany, April.


