TDPrefixGrowth: an algorithm for mining multiple sequential periodic regularities

António Barreto
antonio.barreto@tecnico.ulisboa.pt

Instituto Superior Técnico, Universidade de Lisboa, Lisboa, Portugal

November 2014

Abstract

The dynamic and evolutionary settings of most domains created a challenge for data analytics’ processes. Therefore, the use of temporal information, in order to disclose interesting evolutionary behaviors, has become one of the most important features in Pattern Mining techniques. In particular, several approaches to Temporal Pattern Mining, such as the Emerging and Cyclic patterns, have been proposed with the aim of disclosing different types of evolutionary patterns. However, existing solutions still lack on quality temporal patterns, able to represent interesting dynamic behaviors, and also efficient mining methods able to meet user’s expectations. In this paper, besides defining the problem of mining Cyclic, Converging and Diverging sequential pattern, we propose a new sequential constraint-based approach, named TDPrefixGrowth, for solving it. We also propose an optimization to our initial algorithm, together with two additional variations of the original patterns, the Arithmetic and Geometric Progressions. Our experiments on two different datasets show the efficiency of our algorithm, when compared to other sequential pattern mining approaches, and also its flexibility for dealing with different types of temporal patterns and multiple constraints.

Keywords: Temporal Pattern Mining, Periodicities, Cyclic, Emerging, Sequential Patterns

1. Introduction

Data collection processes have been implemented, in most domains, with the aim of supporting decision making processes with more quality information. Therefore, the increase in data collection has allowed for the contiguous recording of the events and, consequently, for the analysis of this data at multiple points in time. Furthermore, this analysis became interesting as a way to understand the evolutionary properties of the data and understand trends or behaviors that could help predicting future events.

Sequential Pattern Mining (SPM) and Temporal Pattern Mining (TPM) are two of the most relevant streams of research that seek the use of temporal data in order to disclose interesting temporal related information. The first, was introduced with the aim of mining frequent subsequence patterns from a set of sequences, which allowed for the understanding of the frequent order among the events, even though no temporal association could be made. GSP, SPAM, PrefixSpan and GenPrefixSpan are examples of some approaches proposed in this field [10, 4, 8, 3]. Even though SPM approaches already introduce interesting temporal related information (order), they are not able to represent a pattern’s evolution throughout time. TPM techniques, on the other hand, aim for mining temporal relations among the patterns of a database. The works of Emerging and Cyclic Patterns [6, 9], MMR and MRS represent patterns with a particular behavior/evolution in time, allowing for a better understanding of their evolution [7, 11]. Still, these works lack on two main aspects: efficiency and data models. First, the nature of most of temporal patterns does not verify the antimonotonic property of the Apriori algorithm [1], which requires algorithms to have additional computational resources in order to disclose these particular patterns. Second, most of existing approaches make use of transactional data models, which consider the database as a single population instead of a set of different instances, leading to the existence of false positives among the results.

According to these main challenges and drawbacks of existing approaches, we propose a new problem in TPM for mining new temporal patterns with regular periodicities. In particular, we define
three main types of patterns: Cyclic, Converging and Diverging, that represent different evolutionary trends. Our approach adopts a sequential data model and a constraint-base approach that allows for the discovery of those temporal patterns and also for the specification of particular criteria in an early phase of the mining process.

The rest of this paper is organized as follows. In Section 2 we provide a brief overview over the most relevant work in Sequential and Temporal Pattern Mining, with focus on the problem of mining periodic regularities. Later, in Section 3 we introduce the major concepts of our approach (data model and types of patterns) with a formalization of our particular problem. The algorithm proposed and its optimizations are presented in Section 4 followed by experimental results that discuss the algorithm’s performance. Finally, some conclusions and future are presented (Section 5).

2. Background

Pattern Mining focus on the mining of frequent patterns in a specific dataset, aiming for the discovery of relevant information able to better represent the domain and support decision making. Most of Pattern Mining approaches make use of static datasets, i.e., datasets that correspond to a particular instant in time and do not differentiate, or make use, of any temporal information. This raises an issue regarding the validity of the information found, because there is no guarantee that the patterns found at a particular point in time (in the past) would remain constant in the future and, therefore, we are not able to understand the evolution of the domain.

Sequential Pattern Mining (SPM), was one of the first fields of research that, somehow, approached this particular problem. SPM approaches introduced the notion of order among the items of a pattern, which already gives an idea of the natural order for the occurrence of the events. Also, depending on the type of patterns and mining techniques, several approaches have been proposed [3] [4] [10] [8]. PrefixSpan [8], has been recognized as one of the most efficient approaches for SPM, much due to the use of projected-databases that allows for a better search space reduction and, therefore, better memory and computational resources management.

Temporal Pattern Mining focus on the use of temporal information in order to find interesting temporal relations among the patterns of a database, mitigating the previously introduced limitations.

In line with SPM, different approaches have been proposed based on different data structures or temporal relations of interest. Examples of this heterogeneity are the works on the mining of temporal association rules, patterns in the form of \( X \rightarrow Y \) where, \( X \) and \( Y \) are sets of items that are highly correlated. In these works, the relations among the items might: i) be considered valid in a specific time window [2], ii) reflect the variation of the attributes along time and according to different operators [13] or iii) represent the correlation among the evolution of numerical attributes [12]. Moreover, [3] [5] introduced a new type of temporal patterns, namely Emerging and Jumping Emerging Patterns, as patterns that represent a variation on their support across different datasets collected at different time points. These particular types of patterns, allow the analysis of the temporal evolution of the patterns in terms of their frequency, i.e., understand if the same pattern is more or less frequent in one time point when compared to the other. More recently, the works in [9] [7] [11] address the mining of regularities and periodic patterns as a way to understand evolutionary trends, being those, the most relevant and similar approaches to our work.

In the first work [9], the authors propose a new algorithm, named (TD)2PaM, to mine regularities (cyclic) over transactional data. This approach considers three types of patterns: complete cyclic, partial cyclic and lifespan, which differ according to three main properties: cycle period, cycle duration and initial cycle instant. Based on a transactional data model, this approach applies a database partitioning mechanism according to its different temporal instants. After this partitioning, the mining algorithm is ran over each individual partition and later, the periodicities are determined as a post-processing step. This technique has performance limitations given that the number of existing partitions influences the number of times that the mining algorithm is ran, and the periodicities cannot be determined at the same time, which requires additional processing and, consequently, an increase of the overall running time. Still, the representation of the patterns according to their period, duration and offset (first time instant where the pattern occurs) allows for a better analysis on the type of evolution of the patterns and, consequently, their expected behavior in future time points.

The work in [7] is an approach for mining patterns with multiple repetitions within a specific time window. This work is an extension of the
work in [11], through the adoption of a sequential data model and a PrefixSpan based algorithm which, contrary to the work of [7], allows for more informative patterns (best representative of the dataset) and for a unique process that doesn’t require any post-processing phase to determine the regularities of the patterns. However, the disclosed patterns are represented only according to a minimum repetition value, i.e., they must occur at least a minimum number of times in the sequences of the database. Even though this can represent the regularities of the data, it does not specify how these regularities are spaced along time, i.e., if they are cyclic (stable periodicity), or show a different evolution. On the other hand, and even though there is no comparisons available between the two approaches, we believe that the pattern-growth based approach in [7] is able to perform better that the Apriori based from [9].

Considering the previous analysis of the work in SPM and TPM, we understand that the mining of regularities still lacks in efficient solutions able to disclose quality temporal patterns. Most of existing solutions are based on transactional data models and some, such as [9], which requires post-processing steps in order to disclose the patterns of interest. Others, do not allow for the specification of any criteria in order to guide the mining process and focus in particular temporal patterns, which together with a poor representation of the temporal patterns do not provide quality patterns able to represent temporal regularities of interest. Therefore, we propose a new approach for mining periodic regularities that combines both the informative representation of patterns in [9] and the efficient data model and mining solution in [7].

3. Problem Statement

Let an item be an element from an alphabet Σ. An itemset I is an ordered set of items, that occur simultaneously, for example in the same transaction. An event e, is a tuple (I, t), where e.I represents its itemset and e.t its time point.

Accordingly, we can define an itemset sequence, s, as an ordered set of itemsets, say \( s = \langle I_1, \ldots, I_n \rangle \). If those itemsets have occurred on a sequence of time ordered events, than we can create a temporal itemset sequence \( S = \langle e_{t_1}, \ldots, e_{t_n} \rangle \), where \( \forall i \in \mathbb{N} \land i \leq n, e_{t_i} < e_{t_{i+1}} \). Table 1 shows an example of four temporal itemset sequences where \( (A : 1) \) corresponds to an event where \( 'A' \) is the itemset and \( 't' \) is the time point. Further on, we will only refer to itemset sequences assuming that their itemsets are ordered in time. Considering the adopted data model, we can define our temporal patterns according to the following definition:

**Definition 1.** A Sequential Periodic Pattern is a triple \((s, \phi, \delta)\), where \( s \) is an itemset sequence and \( \phi : \mathbb{N}^* \rightarrow \mathbb{N} \) is a cycle, where \( \phi \) is a function from \( \mathbb{N}^* \rightarrow \mathbb{N} \), which defines the set of periods of the cycle - the gaps between the consecutive repetitions of \( s \); and \( \delta \) the duration - the number of occurrences of \( s \).

As we can understand from the previous definition, the \( \phi \) function allows for the definition of multiple types of regularities, as it defines the evolution of the periodicities of a pattern. Thus, we will consider three main types of periodic patterns: Cyclic, Converging and Diverging.

**Definition 2.** A Cyclic Pattern is a sequential periodic pattern with the periods defined as a constant function, i.e., for every point \( i \in \mathbb{N}^* \), \( \rho_i = \rho_{i+1} \).

For example, \( \{(a)(b), 3, 5\} \) represents a sequential pattern where \((a)(b)\) occurs in every \( 3 \) elements, for \( 5 \) times.

**Definition 3.** A Converging Pattern is a sequential periodic pattern with the periods defined by any monotonically decreasing function, i.e., for every point \( i \in \mathbb{N}^* \), \( \rho_i > \rho_{i+1} \).

For example, \( \{(a)(b), \{3, 2, 1\}, 4\} \) represent a sequential pattern where \((a)(b)\) occurs in every \( 3, 2 \) and \( 1 \) elements, for \( 4 \) times.

**Definition 4.** A Diverging Pattern is a sequential periodic pattern with the periods defined by any monotonically increasing function, i.e., for every point \( i \in \mathbb{N}^* \), \( \rho_i < \rho_{i+1} \).

For example, \( \{(a)(b), \{1, 2, 3\}, 4\} \) represent a sequential pattern where \((a)(b)\) occurs in every \( 1, 2 \) and \( 3 \) elements, for \( 4 \) times.

*Figure 1* shows the behavior of the previous patterns, considering a finite timeline. In order to allow for the introduction of background knowledge into the mining algorithm, we have defined a set of constraints that define the previous patterns. Accordingly, we also introduced two specializations of the Converging and Diverging patterns, based on particular \( \phi \) functions.

<table>
<thead>
<tr>
<th>S.id</th>
<th>Itemset Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>id1</td>
<td>ab(1)cb(2)dc(6)</td>
</tr>
<tr>
<td>id2</td>
<td>bc(1)dc(2)cd(6)</td>
</tr>
<tr>
<td>id3</td>
<td>df(1)cf(2)d(6)</td>
</tr>
<tr>
<td>id4</td>
<td>ef(1)a(2)d(6)</td>
</tr>
</tbody>
</table>

Table 1: Temporal Itemset Sequence Example Dataset.
Definition 5. A Sequential Periodic Constraint \( C \) is a tuple \( (\varrho, c) \), where \( \varrho \) is a function that defines the evolution of the periods of the constraint and \( c \) is a minimum cycle value. The notion of minimum cycle corresponds to the minimum number of occurrences of an itemset in a sequence, in order to be considered as a periodic pattern. We defined the default value of \( c > 2 \) (can be changed according to user expectations), which guarantees that all patterns have at least 3 occurrences in the database sequences, allowing for the analysis of at least two periods in time and the definition of the type of periodicity. In line with this, we can introduce the Arithmetic and Geometric constraints that represent particular Converging and Diverging patterns.

Definition 6. An Arithmetic Constraint is Sequential Periodic Constraint \( (\varrho, c) \), where \( \varrho \) defines an arithmetic progression of the form \( a_n = a_1 + (n-1)d \), with \( n \) the number of terms of the sequence, \( a_n \) and \( a_1 \) the values for the \( n \)th and first term of the sequence and finally \( d \) the common difference between the terms of the sequence, verifying \( d \in \mathbb{Z} \). \( c \) corresponds to the minimum cycle constraint.

Definition 7. A Geometric Constraint is Sequential Periodic Constraint \( C = (\varrho, c) \), where \( \varrho \) defines a geometric progression of the form \( a_n = a_1 \cdot r^{n-1} \), with \( n \) the number of terms of the sequence, \( a_n \) and \( a_1 \) the values for the \( n \)th and first term of the sequence and finally \( r \) the common ratio between the terms of the sequence, verifying \( r \in \mathbb{R} \). \( c \) corresponds to the minimum cycle constraint.

Finally, we can clearly define our problem as, given a minimum support threshold \( \theta \) (as usual in pattern mining) and a sequential periodic constraint \( C = (\varrho, c) \), discover the complete set of sequential periodic patterns that verify both \( \theta \) and \( C \), i.e., both \( C.\varrho \) and \( C.c \).

4. TDPrefixGrowth

In this section we will introduce the TDPrefixGrowth algorithm, standing for Temporal Driven Prefix Growth, for mining sequential periodic patterns from a sequence database. Our approach is an extension of the PrefixSpan algorithm, adopting the both the divide and conquer and the use of projected databases strategies of this algorithm.

The first step, responsible for the initialization of the algorithm, determines the 1-length frequent items and initializes the recursive step based on those frequent items. In the recursive step (second) the algorithm generates all the k-length frequent items, until no more patterns can be found. In this phase, we consider the third and last step where the patterns periodicities are determined and the constraints are verified. Next, we detail each one of these steps.

4.1 Main Function

In the initial step (Algorithm 1), the algorithm receives the complete sequence database \( DB \) and a constraint \( C \), previously defined according to the Definition 5. Based on the sequence database and the defined constraint, first, the algorithm generates all the 1-item frequent elements (line 2). This process already deviates from the usual methods once, we not only consider the support constraint but also the minimum cycle constraint.

The support constraint, similarly to the original method, is used to guarantee that the items are frequent in the dataset, ensuring that they occur in a significant number of sequences. However, our patterns require the items to occur multiple times in the same sequence, which cannot be verified by the support constraint. Therefore, considering the minimum cycle constraint \( c \) or duration constraint.

Algorithm 1 : TDPrefixGrowth Main Function

1: function TDPrefixSpan(Database DB, Constraint C, Support \( \theta \))
2: \( f\text{list} \leftarrow \text{findFrequentItems}(C, DB, \theta) \)
3: for all \( b \) in \( f\text{list} \) do
4: \( pdb \leftarrow \text{genProjDB}(b, C, DB) \)
5: \( L \leftarrow L \cup \text{run}(b, C, \theta, pdb) \)
6: end for
7: return L
8: end function
δ, we guarantee that the frequent items have also multiple occurrences in the same sequence. This is important in order to reduce the search space in an early phase of the algorithm, by discarding the items and sequences that do not show any regularity.

After the generation of the 1-item frequent elements, in the original approach, the algorithm initializes the recursive method (line 5), where the patterns regularities are discovered and the periodic constraints verified. This process is initialized with the projected databases for each of the frequent items, one of the main features of our approach. On the other hand, on the optimized approach, we guarantee that the projected databases of the 1-item frequent elements are generated before the recursive step takes place. With this approach we are able to make use of all the 1-item projected databases in the generation of all the k-item patterns, which allows for an improvement on the search space reduction when mining the new patterns.

For the example of Table 1, if we consider a minimum support \( \theta = 25\% \) (items must occur in at least one database sequence) and a minimum cycle \( c = 3 \) (items must occur at least three times in a sequence, or \( \delta = 3 \)), we would obtain the set of frequent items \{a, c, d, e, f\}. In this particular example, the item \( b \) is the only one which does not verify the conditions required. Even though it verifies the support measure, appearing in 75% of the sequences, it only has a maximum of two occurrences within the same sequence, violating the minimum cycle constraint.

### Algorithm 2: Recursive method main function

```plaintext
1: function run(Pattern α, Constraint C, Support θ, ProjectedDB projDB)
2:     f_list ← findFreqKItems(C, θ, projDB)
3:     for all b in f_list do
4:         β ← ab
5:         pdb ← genProjDB(β, C, projDB)
6:         L ← L ∪ run(β, C, θ, pdb)
7:     end for
8:     L' ← L ∪ generate_cycles(α, projDB, C)
9:     L ∪ L'
10: return L
11: end function
```

#### 4.2 Recursive Step and Projected Databases

The recursive step of the algorithm initiates after the definition of the 1-item frequent elements, as shown in Algorithm 1 line 5. In this step, the algorithm is responsible for the generation of the k-sequence patterns (either simple itemsets or itemset sequences) and the periodicities of those patterns. In order to do so, the algorithm uses a depth first search strategy where it first, finds the new frequent elements (line 2) (now based on the projected database of the pattern \( α \)), constructs the new projected database for the new pattern \( ab \) (line 5) (based on the projected database of \( α \) and \( b \)) and, finally, generates the periodicities of the pattern according to the constraint \( C \) (line 8). Note that, in the original approach the generation of the projected database of \( ab \) would only consider the \( α \)-projected database, as the \( b \)-projected database is not created at this time. Using both projected databases helps reducing the search space more efficiently.

The process of constructing the prefix-projected databases deviates significantly from the one in the original PrefixSpan algorithm. Still, it follows the same principle that is projecting the sequences postfixes having as prefix the frequent itemset in analysis. Considering the result of the findFrequentItems method, that generates the set of frequent items in the database, the genProjDB method will create the corresponding prefix-projected database for each of those elements. Therefore, the overall method consists on mapping all the itemset occurrences in the database sequences, which requires an iteration over all the sequences of the original database. This overall idea differs from the PrefixSpan algorithm as this one only projects the first occurrence of the itemset in the sequence. In our approach the mapping of all the itemset occurrences allows us to test the projected sequences against the periodic constraint, verifying once more which sequences contribute to the possible generated patterns.

According to this idea, the projected databases will be composed by the set of pairs that represent the original sequence in the database and the corresponding offsets of the itemset occurrences in that sequence. Here it is important to notice that the notion of ‘offsets’ do not correspond to the temporal information associated to the events but, to the position where the events occur in the sequence. Taking as example the data from Table 1, the frequent item \( a \), its projected databases will be composed by the set of pairs: \(< 1 : \{0,3,5\} >, < 3 : \{0,4,8\} >\), where the first identifiers \{1,3\} correspond to the unique
Identifiers of the sequences in the original database and the set of integers \(\{0, 3, 5\}, \{0, 4, 8\}\) to the corresponding offsets (positions) where \(a\) occurs in the sequences. Also, the sequences with \(id = \{2, 4\}\) are not mapped into \(a\)-projected database as their corresponding offsets \(\{4\}, \{1\}\) do not satisfy the minimum cycle constraint.

This specific representation of the projected databases, using only the reference to the database sequence (id) and the set of offsets of the items, allows for an overall better memory performance than if mapping the postfix sequences themselves. This comes as an important feature once the memory usage due to the construction of projected databases is one of the main drawbacks pointed to as an important feature once the memory usage due to the construction of projected databases is one of the main drawbacks pointed to as an important feature once the memory usage due to the construction of projected databases is one of the main drawbacks pointed to as an important feature once the memory usage due to the construction of projected databases is one of the main drawbacks pointed to as an important feature once the memory usage due to the construction of projected databases is one of the main drawbacks pointed to.

### 4.3 Periodicities

The periodicities of the patterns are determined according to the patterns’ projected databases and the predefined constraint given as input in the initial step of the algorithm. This process must guarantee the generation of the complete set of periodicities for the patterns, i.e., all the possible cycles \(\phi = (\alpha, \delta)\) according to the constraint.

**Algorithm 3**: Generate Cycles method

1. **function** `GENERATECYCLES(Pattern \(\alpha\), Constraint \(C\), ProjectedDB \(projDB\))`
2. \(cycles \leftarrow \text{empty}\)
3. **for all** \(s\) in \(projDB\) **do**
4. \(p\text{list} \leftarrow s\text{.getPeriods()}\)
5. \(cycles \leftarrow cycles \cup \text{getCycles}(C, p\text{list})\)
6. **end for**
7. \(f\text{.cycle} \leftarrow cycles\text{.getFrequent()}\)
8. \(L \leftarrow (\alpha, f\text{.cycle})\)
9. **return** \(L\)
10. **end function**

The method for generating the cycles of a pattern \(\alpha\) is described in Algorithm 3. For each sequence mapped in \(\alpha\)-projected database, we determine the set of periods of the pattern, corresponding to the difference between the consecutive offsets mapped in the projected database (Algorithm 3 line 6). For example, considering the projected database for the pattern \(a = (\langle 1 : \{0, 3, 5\}, < 3 : \{0, 4, 8\}\rangle\), the result for the periodicities would be the set \(\langle 1 : \{3, 2\}, < 3 : \{4, 4\}\rangle\), where \(\{1, 3\}\) correspond to the sequence \(id\) and the corresponding \(\{3, 2\}\) and \(\{4, 4\}\) to the difference between the consecutive offsets where \(a\) occurs (periods).

From the resulting set of periods of each sequence, it is possible to determine the set of existing cycles according to the specific constraint \(C\) (Algorithm 3 line 6). This process is of particular relevance since it determines the ability to generate the complete set of existing cycles and, therefore, the complete set of existing patterns. To do so, we need to have the complete set of occurrences of the pattern in every sequence of the database, which according to the nature of the periodic patterns does not allow for the application of the anti-monotonic property. As we have seen before, if we apply this specific property when mining the cycles for an \(ab\) pattern, we would only consider the sequences and offsets that contributed to the cycles of the pattern \(\alpha\), which would create a loss of information for the mining of the \(ab\) cycles. Thus, in our particular problem, we cannot restrict the cycles of a pattern \(ab\) to the ones of its super pattern \(\alpha\), which creates significant efficiency challenges, since we do not benefit from the search space reduction of the anti-monotonic property.

Considering these limitations, our strategy does not make the \((k+1)\)-sequence patterns dependent on \(k\)-sequence patterns’ cycles. In our approach we simply project one subsequence, containing all the occurrences of the pattern, for each sequence in the projected database. Therefore, the ability to generate the complete set of cycles of a pattern is only dependent on the analysis of the offsets mapped in the pattern projected database, without any restriction from the previous generated patterns. From here, we are able to improve both memory (keeping the projected databases small) and computational time (reducing the number of possibilities) requirements.

Having the complete set of occurrences of the pattern for each sequence of the database, guarantees the generation of the complete set of periods for those sequences and later the generation of the complete set of cycles according to the constraint.
In order to evaluate the performance of the proposed solution, we adopted an experimental approach based on the analysis of a dataset, the Household Power dataset from the UCI Repository [https://archive.ics.uci.edu/ml/]. This dataset corresponds to electric power consumption measurements, in one household, with a minute sampling rate over a period of almost 4 years (2006 to 2010). This dataset has a total of 9 attributes, three of which we consider for our case study (Date, Time and Global Active Power). The Global Active Power attribute corresponds to the household global minute-averaged active power (in kilowatt). Thus, in order to discover interesting temporal patterns, we pre-processed the data to display the measures at minute granularity. Our dataset consist in 1431 records, corresponding to unique Date attribute values, each of those records with a total of 1440 electric measures (Global Active Power). This corresponds to the minute measures for each day (24 hours * 60 minutes). Also, considering the nature of the attributes values (at a thousandth magnitude order), we simplified the alphabet by rounding the values to a tenth magnitude order, ending with a total of 101 different values.

As we can see, from both figures (Figure 2 and 3), the evolution on the number of patterns is consistent to most of the results from pattern mining techniques, with a pattern explosion for low support values. Also, the evolutions are similar for the different constraints applied (Converging, Diverging and Cyclic) with a significant increase for the transition between the 2.5% and 1% support measure. The most interesting result demonstrated by the charts, is the relation between the number of unique itemset patterns and periodic patterns. For example, for 1% of support, we verify a difference of around 2000 patterns for the Cyclic constraint and 3000 for the Converging and Diverging constraints, which translates the fact that the existing patterns have multiple periodicities associated. In average, for the global set of supports, we verified a relation of 10 periodic patters per itemset pattern, with a maximum of 18 to 1 for the Converging and Diverging constraints at the support 2.5%.

On the execution time analysis, we have also considered an analysis based on different support measures and different constraints. For the overall execution time, we can verify some similarities according to the evolution on the number of disclosed patterns. From Figure 4 we verify an increase on the execution time along the decreasing support measures. This result was expected due to the amount of computations required, which increases for low support values. In these cases there are more frequent items that can generate new patterns and therefore, additional structures and computations must be performed.

5. Experimental Results

Back to the Algorithm [3] after determining the periods and according to the specified constrain \( C \), the cycles for each sequence are determined (Algorithm [3] line 5). This method performs according to the constraint defined, though it relies in the same basic methodology: given a set of periods, the method iterates over those same periods and extracts all the unique cycles in the form of \( \phi = (\rho, \delta) \), whose \( \rho \) verifies the constraint \( C \). In Table 3 we represent the set of cycles generated for the set of periods \( \{3,2,1,1,4,8\} \) according to the different constraints, representing the function \( \phi \) through the set of periods it produces.

In this process, it is important to notice that we generate all the possible cycles by spanning the set of periods only once, which is much more efficient than generating and spanning the periods for multiple sequences. Also, we generate all the possible sub-cycles from a cycle, meaning that all the cycles having \( \delta > \) minimum cycle constraint, are able to generate shorter cycles. Considering the cycle \( \{(3,2,1,4)\} \) from the example of Table 3 it is still possible to generate cycles with \( \delta = 3 \), such as \( \{(3,2),3\} \) and \( \{(2,1),3\} \), both verifying the same constraint. This is also important for the support counting of the existing cycles, once not all the sequences share the same cycles for the pattern and, therefore, we need to guarantee that all the cycles are considered. After the generation of the cycles for the sequences in the projected database, we determine the frequent cycles and later add them to the structure of discovered patterns (Algorithm [3] line 7 and 8).

Table 3: Cycle generation example for multiple constraints

<table>
<thead>
<tr>
<th>Periods</th>
<th>Constraint</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>{3,2,1,1,4,8}</td>
<td>Cyclic</td>
<td>( ({1,1,1}, 4) ); ( ({1,1}, 3) )</td>
</tr>
<tr>
<td></td>
<td>Converging</td>
<td>( ({3,2,1}, 4) ); ( ({3,2}, 3) ); ( ({2,1}, 3) )</td>
</tr>
<tr>
<td></td>
<td>Diverging</td>
<td>( ({1,4,8}, 4) ); ( ({1,4}, 3) ); ( ({4,8}, 3) )</td>
</tr>
</tbody>
</table>

datasets/Individual+household+electric+power+consumption). This dataset corresponds to electric power consumption measurements, in one household, with a minute sampling rate over a period of almost 4 years (2006 to 2010). This dataset has a total of 9 attributes, three of which we consider for our case study (Date, Time and Global Active Power). The Global Active Power attribute corresponds to the household global minute-averaged active power (in kilowatt). Thus, in order to discover interesting temporal patterns, we pre-processed the data to display the measures at minute granularity. Our dataset consist in 1431 records, corresponding to unique Date attribute values, each of those records with a total of 1440 electric measures (Global Active Power). This corresponds to the minute measures for each day (24 hours * 60 minutes). Also, considering the nature of the attributes values (at a thousandth magnitude order), we simplified the alphabet by rounding the values to a tenth magnitude order, ending with a total of 101 different values.
When comparing the different constraints applied, we verify that the Cyclic stands out from the Converging and Diverging constraints. This can also be explained, mainly, by the difference on the number of itemset patterns, represented in Figure 2. Still, the evolution for the different supports seems to follow a more linear trend when compared to the number of disclosed patterns. If we consider the exponential growth on the number of patterns at 1% support, it would be expected to verify a similar trend in the execution time, however, even though an increase is verified, it follows the same trend as in previous supports.

Regarding the memory consumption evaluation, we verify a significant deviation for the different constraint approaches. Figure 5 shows the average memory consumption verified according to the different support values. We can verify that there is a significant gap between the Cyclic and the Converging and Diverging constraints, these last showing very similar results. This is in line with the number of disclosed patterns, which is consistently higher for the Cyclic constraint. Moreover, the duration of the cyclic patterns showed in average a duration higher than the converging and diverging patterns, which supports the need for more computational requirements regarding memory consumption (more complex projected databases). On the Execution time distribution (Figure 7), we analyzed the impact of three main functions of the algorithm: Frequent Item generation, Projected Databases construction and generation of the patterns’ periodicities. The results for the Cyclic constraint show that, for higher support values, the constraint validation becomes the major phase of the process. This is due to the number of validations required when compared to the number of patterns generated, showing the complexity of this particular phase. Yet, as the support decreases, the phases tend to balance and, for the lowest supports, the behavior reverses, with the projected database phase turning into the most time consuming phase. In this experiment, the constraint validation phase reached 60% of the total execution time (30% support) and the
projected databases a maximum of 57% of the total time (1% support). Therefore, we can understand that the projected database construction phase is costly, and our optimization efforts helped addressing its performance by reducing the number of sequences that needed to be generated in each projected database.

Finally, Figure 6 shows a comparison between the original TDPrefixGrowth algorithm and its optimized approach, together with a comparison with an implementation of the GenPrefixSpan algorithm from [3]. It is important to refer that the GenPrefixSpan approach was instantiated using a gap value of zero, which corresponds to the mining of contiguous sequential patterns that do not have any gap in between the occurrence of the elements. This is so because those are the most similar patterns according to our approach. Moreover, in this particular test, we have reduced the minimum support constraint to less than 1%, so we can compare the evolution of the different approaches in more detail.

The results in Figure 6 show that all the approaches tested follow a similar evolution behavior regarding the execution time. In this test, we verify that, when compared to the GenPrefixSpan algorithm, our optimized approach is able to perform in less time, while the original one performs slightly slower. The results of the original and the optimized approach confirm the importance of the projected databases’ process and how it affects the overall algorithm’s performance. With the optimized approach, we were able to achieve a performance improvement of 33% (relatively to the original approach), corresponding to an average of 81 seconds in the total running time.

The GenPrefixSpan [3], in turn, is able to achieve better results for higher supports, which can be explained by the extra computational requirements of our algorithm in order to verify the periodic constraints applied. However, for lower support values, the large number of generated patterns (in the GenPrefixSpan) overcomes the additional computational requirements (in the TDPrefixGrowth), making the first to perform worse. Still, we can consider this difference to be stronger, if we consider the extra post-processing requirements in order to disclose our specific type of patterns. The GenPrefixSpan algorithm is only able to find the frequent sequences of the database, without any kind of periodic information.

6. Conclusions
The ability to disclose and understand temporal trends is currently one of the major challenges in Data Mining and still lacks on efficient solutions and quality results. The value of this type of information, that translates the evolution of the domain over time, introduced the complexity of the temporal dimension into usual pattern mining techniques. Therefore, several approaches have been proposed though, with significant limitations regarding performance or the quality of the information disclosed.

Following this, in this paper we proposed a new problem formulation aiming for quality patterns, able to represent solid information regarding the evolution of the data, and also proposed an efficient and flexible solution that allows for the focus on user expectations. We proposed a new sequential algorithm for mining sequential periodic patterns (TDPrefixGrowth) and also two new types of patterns, namely, converging and diverging patterns. Sequential periodic patterns, such as...
the defined ones, are able to represent two major temporal properties: regularity and periodicity. Regularity represents the ability of a pattern to reoccur in multiple instants of time, i.e, if it is frequently repeated at different points in time. This particular property supports the evidence that states that it is likely for those patterns to occur in a future instant in time. The second property, the periodicity, translates the frequency of the occurrences of a pattern, i.e, the time that goes from one occurrence to another. From this property we are able to understand the evolution of the occurrences of the pattern and support whether it is becoming more or less frequent. Together, these properties allow for quality information that supports temporal evolutions/trends and for a more precise prediction of future events. Moreover, our approach allows for the definition of specific criteria that pattern should verify. Examples are the duration of the pattern (number of occurrences of the pattern in the considered time window), type of evolution (strict, arithmetic or geometric) and type of periodicity (cyclic, converging diverging).

The conducted experiments show that our approach is efficient for the mining of the different types of constraints proposed and is in line with the results of other approaches such as the GenPrefixSpan algorithm. We have also verified that our implemented optimization reduced the complexity of the projected database process and enabled the overall performance to improve. Still, there are some issues that should be explored. First the adaptation of the algorithm to make use of the data temporal information when mining the patterns (instead of the offsets in the sequences) and, finally, some improvements on the cycle generation phase, in order to benefit from the periodicities already disclosed in earlier phases of the algorithm.

Acknowledgements
This work is supported by Fundação para a Ciência e Tecnologia under research project D2PM (PTDC/EIA- EIA/110074/2009).

References