Finding Periodic Regularities on Sequential Data: Converging, Diverging and Cyclic Patterns

António Barreto

Instituto Superior Técnico – Universidade de Lisboa
Av. Rovisco Pais, 1
1049-001 Lisboa
+351 218 419 407

antonio.barreto@tecnico.ulisboa.pt

Cláudia Antunes

Instituto Superior Técnico – Universidade de Lisboa
Av. Rovisco Pais, 1
1049-001 Lisboa
+351 218 419 407

claudia.antunes@tecnico.ulisboa.pt

ABSTRACT

Temporal information has become one of the most important features when it comes to data analytics. The need to understand the dynamics and evolutionary behaviors of different domains has driven data analytics' processes to make use of the temporal information associated with the data. Therefore, several approaches have been proposed, in the field of Temporal Pattern Mining, in order to use this temporal information to disclose temporal trends that could help in the decision making process. However, there are still significant limitations regarding both the quality of the disclosed information or the efficiency of the processes. In this work we propose a new constraint-based sequential mining method, called *ConstraintPrefixSpan*, for mining three types of periodic regularities: Cyclic, Converging and Diverging. Our experiments on two different datasets show both the quality of patterns found and the efficiency and flexibility of our algorithm to deal with multiple types of regularities.

Keywords

Temporal pattern mining, sequential pattern mining, regularities, periodicities, constraint-based mining.

1. INTRODUCTION

The interesting properties of temporal information and the increasing collection of temporal related data have created the opportunity and need to explore interesting temporal relations that could help in the events prediction.

The field of Temporal Data Mining already provides interesting works regarding the analysis of temporal dimension of the data. Examples of this are the works in [1,2] that focus on the mining of periodicities, i.e., patterns that occur multiple times in a specific time frame, and also the works in [3,4] that introduce the Emerging and Jumping Emerging Patterns as patterns whose support varies across multiple datasets at different points in time. Still, there are some challenges that remain poorly address, even regarding the quality of the disclosed temporal relations, that are not able to represent evolutionary behaviors, or the integration of background knowledge into the mining process,

allowing for a more user oriented approach.

Therefore, we propose a new mining approach, the *ConstraintPrefixSpan*, for mining temporal regularities able to represent evolutionary trends. In particular, we introduce a new type of patterns, the *Converging Patterns*, which extend the existing *Cyclic Patterns* from other works [1,2]. Moreover, we propose a constraint-based methodology that enables the definition of the patterns of interest independently from the mining approach, allowing for a more efficient process and also to focus on user expectations. Our approach is a sequential pattern mining approach, which has been less addressed by the literature and enables the mining of more quality patterns.

The rest of this paper is organized as follows. Section 2 introduces the main concepts required to define our problem and a formal definition of our particular problem. Section 3 introduces our proposed solution, together with the details of the developed algorithm. Finally, in Section 4 we present our experimental results ending with a summary of this paper and a brief reference to the open issues.

2. 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. In this context, we can define an *itemset sequence*, s , as an ordered set of itemsets, say $s = \langle I_1, \dots, I_n \rangle$. If those itemsets have occurred on a sequence of time ordered events, then we can create a *temporal itemset sequence*: a sequence of events, $s = \langle e_1, \dots, e_n \rangle$, where $\forall i \in \mathbb{N} \wedge i < n: e_i.t < e_{i+1}.t$

In this context, a *sequential periodic pattern* is a triple (s, ρ, δ) , where s is an itemset sequence and (ρ, δ) is a cycle, where ρ corresponds to the set periods of the cycle and δ to its duration. The *periods* correspond to the gaps between two consecutive repetitions of s . The *duration* corresponds to the number of occurrences of s .

According to the definition of sequential periodic pattern and considering the set of periods $\rho = (\rho_1, \dots, \rho_n)$, we can define three types of periodic patterns based on the different evolutionary properties of that same set of periods ρ : *sequential converging patterns*, where ρ is defined by a monotonically decreasing function; a *sequential diverging patterns*, where ρ is defined by a monotonically increasing function and *sequential cyclic patterns* where ρ is defined by a constant function.

Considering the previous definitions and the usual definition of itemset subsequences [6], we can say that an itemset sequence S

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the Owner/Author.

Copyright is held by the owner/author(s).

C3S2E '14, Aug 03-05 2014, Montreal, QC, Canada

ACM 978-1-4503-2712-1/14/08.

<http://dx.doi.org/10.1145/2641483.2641519>

contains a sequential periodic pattern (s, ρ, δ) if s is a contiguous subsequence of S , δ times (with $\delta > 2$), and each occurrence of s in S verifies one of the periodic constraints.

The requirement $\delta > 2$ prevents every itemset sequence to be considered as a trivial periodic pattern, with just two occurrences. This guarantees that we have the minimum required number of periods in order to validate the cycles against the defined temporal constraints. Naturally, we may also be interested in cycles with more repetitions (longer cycles). Therefore, we call *minimum cycle* to the minimum number of repetitions that an itemset must present in order to be considered as a periodic pattern.

According to the previous definitions, we define our problem as that of given a database of itemset sequences D , a minimum support threshold σ (as usual in pattern mining), a minimum cycle c , and a periodic constraint C , discover the complete set of frequent sequential periodic patterns that verify σ , c and C .

3. ALGORITHM

The *ConstraintPrefixSpan* is a constraint-based algorithm based on the PrefixSpan algorithm for mining frequent itemset sequences [5]. In line with PrefixSpan our algorithm adopts a divide and conquer strategy, based on the concept of projected databases, which allows for an efficient search space reduction. Also, two other important characteristics of our method are its recursive and breadth-first-search approach, which contributes to the overall performance of the algorithm.

In general, our algorithm follows the same phases as PrefixSpan, though it presents some additional procedures in order to build and verify the periodic constraints defined. Therefore we can consider three main stages of the algorithm: the first, where the 1-item patterns are generated and the recursive phase initiated; the second where the k -item patterns are generated and the third, and final, where the generation of the periodicities and verification of the constraints take place.

3.1 ConstraintPrefixSpan

In the initial phase of the algorithm (Figure 1), given a sequence database DB , and a constraint C , containing the definition of the minimum support value σ , the minimum cycle c , and the periodic constraint that must be satisfied (converging, diverging or cyclic), the algorithm first, constructs the list of 1-item frequent elements (line 2), according to the minimum support value, and in a second phase, for each of those frequent items, it runs the recursive method with the corresponding item's projected database (line 3). When determining the 1-item frequent elements, we additionally verify the minimum cycle constraint for each element. In this way, we guarantee (in an early phase of the process) that the items that do not occur a minimum number of times in the sequences are not considered for the next steps. This is so because, violating this constraint, means no periodic patterns will be generated later on.

Next, in the iteration phase (Figure 1, line 3), the algorithm is

```

1  ConstraintPrefixSpan(Database DB,Constraint C)
2  f_list←findFrequentItems(C, DB)
3  for each b in f_list do
4      L ← L ∪ run(b, C, genProjDB(b, C, DB))
5  return L

```

Figure 1. ConstraintPrefixSpan main function

responsible for executing the recursive method for each of the previous frequent items, with their respective projected databases (Figure 1, line 4). The process of constructing the projected databases is quite similar to the one in the original algorithm [5]. For the 1-item patterns, the projected databases are constructed by mapping all the occurrences of the item in the database sequences. This is done considering only the sequences of the database that contributed to the support of that item and by mapping the offsets of the occurrences of the items in those respective sequences. In this initial phase the entire sequence is swept in order to capture all the occurrences of the elements in the sequence. An example of a projected database for an item 'a' would be $\langle 1:\{0,3,5\}\rangle, \langle 3:\{0,4,8\}\rangle$, 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 offsets corresponding to the positions where 'a' occurs in the sequence. Regarding the structure of the projected databases, we use a simplistic representation by keeping only the reference to the original sequence and the set of offsets, instead of duplicating the original sequences. This allows for a better memory performance management, one of the main drawbacks pointed to projected-databases based methods.

The main function of the algorithm, end with the complete set of periodic patterns L , satisfying the constraint C and generated by the recursive method 'run' (lines 4 and 5).

3.2 Recursive Method

The recursive method (Figure 2) represents the depth-first search strategy of the algorithm and is where, given a frequent pattern α , a constraint C and the projected database for the item α , the new frequent patterns of the form αb are generated. Therefore, and similarly to the main method, in a first step, the new frequent items 'b' are calculated (line 2), now based on the projected database of the previous frequent pattern ' α ', and consequently on less sequences than in the original dataset. This new frequent items must occur in concurrent (to build a new itemset) or contiguous (to build a itemset sequence) positions than the ones of α . Thus, we not allow any gaps between the itemsets of a pattern. Once more, we apply the support and minimum cycle constraint to this process.

Next, in the second step of the recursive method the new frequent patterns β are created (line 4), either as an itemset (αb) or an itemset sequence $(\alpha)b$, and the method is recursively ran with the new projected databases for those same patterns (line 5). Finally, in the last step, and when there are no more frequent items to explore, the pattern's periodicities are generated according to the projected database of the pattern and the defined constraint (line 6).

```

1  run(Pattern  $\alpha$ ,Constraint C,ProjectedDB projDB)
2  f_list←findKFrequentItems(C, projDB)
3  for each b in f_list do
4       $\beta \leftarrow (\alpha b) \parallel (\alpha)b$ 
5      L ← L ∪ run(b,C,genProjDB( $\alpha b$ ,C,projDB))
6  L' ← L ∪ generate_cycles( $\alpha$ , projDB, C)
7  L ← L' ∪ L
8  return L

```

Figure 2. ConstraintPrefixSpan recursive function

3.3 Generate Cycles

Even though the generation of the cycles is dependent on the type of constraint used (Cyclic, Converging or Diverging), the process follows the same line of reasoning shown in Figure 3.

Therefore, the method `GenerateCycles` (Figure 3) is responsible for generating the complete set of cycles (ρ, δ) , for a pattern α , that verify the constraint C . At first, the method generates the set of periods for each sequence of the projected database (line 5), this by performing the difference between the consecutive offsets where the pattern occurs. Having the set of periods, we are able to determine the complete set of cycles that the pattern verifies according to the constraint C (line 6).

The fact that we have the information on the complete set of occurrences of the pattern is important in order to guarantee that we are able to generate the complete set of cycles, given that these periodic type of patterns do not verify the anti-monotonic property of Apriori, i.e., it does not guarantee that if an α pattern does not verify the constraint C , then all its super patterns will also violate. This means that we need to continue processing the new patterns, even though the previous ones didn't conform to the constraint. To overcome this problem, in our approach, we simply project one subsequence, containing all the occurrences of the pattern, for each of the projected database sequences of the pattern. With this, we do not create dependencies between the patterns and we guarantee that we are able to identify all the possible cycles for the super patterns, even though its sub patterns do not verify the constraint.

After determining the set of periods for each sequence of the projected database, we can determine which periodicities are according to the predefined constraint. By analyzing the periods according to the conditions of the periodic constraints, we are able to generate the complete set of valid cycles (line 6). Examples of periodic patterns are: a) $(\{2,2\}, 3)$; b) $(\{2,1\}, 3)$ and c) $(\{1,2\}, 3)$, where $\{2,2\}, \{2,1\}, \{1,2\}$ corresponds to the periods ρ of the cycle and 3 to the duration δ of all cycles. These cycles verify the Cyclic constraint (a)), where the periods remain constant; the Converging constraint (b)), where the periods decrease and the Diverging constraint (c)), where the periods increase. However, data is not so clean and we can have cases where the set of periods do not translate a valid pattern within its entire set. For these cases we need to verify all possible existing sub-cycles. Consider now the example of the set of periods $\{3,3,3,4,5,5,6\}$. This set does not verify any of the periodic

constraints when considering all its periods however, when considering a subsets of those periods, we are able to disclose the following cycles (ρ, δ) : $(\{3,3,3\}, 4)$ and $(\{5,5,5\}, 4)$ for the cyclic constraint $(\{3,3,3,4,5,5,6\})$; $(\{3,4,5\}, 4)$ for the diverging constraint $(\{3,3,3,4,5,5,6\})$ and $(\{5,6\}, 3)$ for the converging one $(\{3,3,3,4,5,5,6\})$.

In the last step, we test the disclosed cycles against the minimum support threshold, given that not all the sequences mapped in the projected databases support the same cycles (line 7). For those who verify the minimum support, we build the final pattern represented by the pattern's itemset sequence and the set of cycles that the pattern verifies (line 8). The computed patterns are recursively returned to the previous iteration until the complete set of patterns is achieved in the main iteration of the algorithm (Figure 1, line 4).

4. EXPERIMENTAL RESULTS

We tested our algorithm in two different datasets from the UCI Repository, the *Household Power (HHP)* and the *EEG* (electroencephalogram) datasets. The first refers to measures regarding the expended power (in kilowatt) at different temporal instants, with a total of 1431 instances (from the years of 2006 to 2010) comprising 1440 measures each one from a set of 101 different values. The second dataset, refers to *EEG* (electroencephalogram) measures in a total of 5568 instances (from all the sensors of 87 patients) having a total of 256 contiguous measures of a set of 11 different values. The evaluated the number of patterns, execution time, maximum cycle duration and pattern size for multiple support threshold values.

Regarding the number of generated patterns, Figure 4 shows that for supports between 30% and 2,5% the growth rate is quite linear, whether for the 1% it grows exponentially. This applies to all the periodic constraints, however with the Cyclic constraint showing more patterns in a regular basis, which means that it is more common to exist cyclic periodicities in the energy consumption behaviors. For the EEG dataset we analyzed the number of cycles generated (Figure 6) and we verified that it follows a more exponential evolution, which allows us to understand, also by the number of cycles, that the unique itemset sequence patterns have multiple cycles.

Regarding the evolution of the execution time we verify that it is consistent with the number of generated patterns, as expected. Figure 5 shows the evolution on the execution time for the HHP dataset, showing again a regular behavior for the initial supports with an increase in the last two measures. Also, in Figure 7, we can see that the execution time per periodic cycle, which shows a decreasing trend for the increasing support values. This is especially important since it guarantees that the time doesn't grow as quickly as the number of patterns increase.

Finally, other interesting measures are related with the properties of the periodic patterns. Considering the number of repetitions of a cycle, or in other words, the duration of the cycle, the results show some interesting characteristics. For both the *Household Power* and *EEG* datasets, the cyclic periodic patterns presented longer cycle than the converging and diverging ones. In the *Power* dataset, the variation of the duration was between 8 and 28 occurrences, which in this case represents a regular consumption of energy during 28 consecutive minutes (when the period is zero). In the *EEG* dataset, the variation of the cyclic patterns was of 27 to 40, which translates the stable measurement of values for 40

```

1  GenerateCycles(Element  $\alpha$ , Constraint  $C$ ,
   ProjectedDB db)
2  cycle_list  $\leftarrow$  empty
3  for each s in db do
   //Pattern periods
5    p_list  $\leftarrow$  s.getPeriods()
   //Generates set of cycles
6    cycle_list  $\leftarrow$  cycle_list  $\cup$  getCycles( $C$ , p_list)
   //The frequent cycles
7  f_cycle  $\leftarrow$  cycle_list.getFrequent()
   //The list of cycles for  $\alpha$ 
8  L  $\leftarrow$  ( $\alpha$ , cycle_list)
9  return L

```

Fig. 3. Generate Cycles method

Fig. 4. Number of generated itemset patterns (*Household Power*)

Fig. 5. Time spent on computation (*Household Power*)

Fig. 6. Number of pattern's cycles (*EEG*)

Fig. 7. Time spent per number of pattern's cycles (*EEG*)

consecutive samples. The variation of the duration in the converging and diverging patterns, on both datasets, stays only between 1 and 3. This may mean that there are not long periods of time where the consumption of the energy increases or decreases continuously. In respect to the size of the patterns, or the itemset sequences, for the second dataset it shows a maximum size of 2 items, which can be explained by the set of items with only 11 different values. In the *Household Power* there is an interesting result, once there are patterns of size 20. These specific patterns represent regularities of 20 minutes, since each sequence item represents one event at a minute granularity.

Generally, the results present consistent performance execution times and also good results for mining datasets with large amounts of cycles in the patterns, demonstrating our efficient approach to overcome the problem of the prefix-anti-monotonicity

5. ACKNOWLEDGMENTS

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

6. REFERENCES

- [1] Pina, S. M. and Antunes, C. (2013). (TD)2PaM: A Constraint-Based Algorithm for Mining Temporal Patterns in Transactional Databases, pages 390–407 Springer.

- [2] Hu, Y. H., & Chiang, I. C. (2011, July). Mining cyclic patterns with multiple minimum repetition supports. In *Fuzzy Systems and Knowledge Discovery (FSKD), 2011 Eighth International Conference on* (Vol. 3, pp. 1545–1549). IEEE
- [3] Dong, G. and Li, J. (1999). Efficient mining of emerging patterns: Discovering trends and differences. *ACM*.
- [4] Bailey, J., Manoukian, T., and Ramamohanarao, K. (2002). *Fast algorithms for mining emerging patterns*, pages 39–50. Springer.
- [5] Pei, J., Pinto, H., Chen, Q., Han, J., Mortazavi-Asl, B., Dayal, U., & Hsu, M. C. (2001, April). Prefixspan: Mining sequential patterns efficiently by prefix-projected pattern growth. In *2001 IEEE 29th International Conference on Data Engineering (ICDE)* (pp. 0215–0215). IEEE Computer Society
- [6] Srikant, R., & Agrawal, R. (1996). Mining sequential patterns: Generalizations and performance improvements (pp. 1–17). Springer Berlin Heidelberg.