$(TD)^2$PaM: A Constraint-Based Algorithm for Mining Temporal Patterns in Transactional Databases

Sílvia Moura Pina and Cláudia Antunes

Dep. Computer Science and Engineering
Instituto Superior Técnico, Technical University of Lisbon
Av Rovisco Pais, 1049-001 Lisboa
{silvia.pina,claudia.antunes}@ist.utl.pt

Abstract. The analysis of frequent behaviors regarding temporal issues begins to achieve some interest, in particular in the area of health care. However, existing approaches tend to ignore the temporal information and only make use of the order among events occurrence. In this paper, we introduce the notion of temporal constraint, and propose three instantiations of it: complete cyclic temporal constraints, partial cyclic temporal constraints and timespan constraints. Additionally, we propose a new algorithm – $(TD)^2$PaM, that together with these constraints, makes possible to focus the pattern mining process on looking for cyclic and timespan patterns. Experimental results reveal the algorithm to be as efficient as its predecessors, and able to discover more informed patterns.

Keywords: Pattern Mining, Temporality, Constraints, Ontologies, Semantic Aspects of Data Mining.

1 Introduction

In modern society, and especially since the explosion of Internet use, we have been accumulating exponentially increasing amounts of data, usually tagged with some time reference. Methods to explore such massive amounts of data have been proposed on the last years, through the field of knowledge discovery and data mining, aiming for extracting new non-trivial information. However, the ability to explore temporal information remains a challenge.

In temporal datasets, a transaction time is attached to each transaction. In this context, hidden information or relationships among the items may be there, which does not necessarily exist or hold throughout the whole time period covered by the dataset, but only in some time intervals. Another feature of temporal data is that they may be of a periodic nature, thus repeating at regular intervals. The existing work in the field of temporal pattern mining presents several critical shortcomings. The most important of these are: (i) not being able to effectively deal with different time granularities; (ii) relying on non-robust and inflexible representations of time.

In this paper, we propose a method to aid the discovery of the regularities among temporal data recorded in transactional databases, namely *cycles* and *lifespans*. This is done within the D^{2}PM framework, through the definition of temporal constraints and a new method able to discover temporal patterns – the (TD)^{2}PaM algorithm (*Temporality in Transactional Domain-Driven Pattern Mining*).

The paper is organized as follows: next, in section 2, we overview the different approaches to temporal data, paying particular attention to the ones dedicated to pattern mining. In section 3, we propose a set of temporal constraints and then describe the new algorithm to deal with it (section 4). Following this, experimental results over real data assess the algorithm's performance and the quality of the discovered patterns. The paper concludes with a critical analysis of the results achieved and some guidelines for future work.

2 Background

Time is a compulsory dimension in any event in the real world, and is usually incorporated into databases by associating a timestamp to the event data [1]. Due to its importance and complexity, time has been a relevant research-topic in the area of Artificial Intelligence, playing a role in areas that span from logical foundations to knowledge-based systems applications. Particularly relevant is the area of *temporal reasoning*, consisting of a formalization of the notion of time and providing a form of representing and reasoning about the temporal aspect of knowledge. The work in this area is vast, and has made important contributions to the representation of time and its impact on the data stored. Of particular importance is the work by Allen [2], where the notions of time point and time interval, as long as a set of relations between temporal entities (points, intervals, or other) were defined.

Another important feature addressed in this field is *time granularity*, which can be defined as the resolution power of the temporal qualification of a statement, and refers to the existence of temporal values or objects at different levels (for example, hours, days, weeks, months) [1]. This involves not merely considering different time units, but moreover considering a layered temporal model, where associations can be formed between different layers through proper operators. Switching from one domain to a finer/ coarser one can alter interpretations of temporal statements.

In this work, we follow the definitions provided in the *Reusable Time Ontology* [3] – a time ontology based on the notion of the time line, and having as central classes *Time Point* and *Time Interval*. Another important concept is the *Time Unit* which encloses the granularity considered for each time point, and consequently for the duration of time intervals. Further down the hierarchy are the concepts of *Convex Time Interval*, which consist of a connected interval on the time line, and *Non Convex Time Interval*, corresponding to non-connected time intervals. A subclass of this is the *Regular Non Convex Time Interval*, which is composed of several convex time intervals corresponding to periodically occurring events. A simplified representation of this ontology is presented in Figure 1.

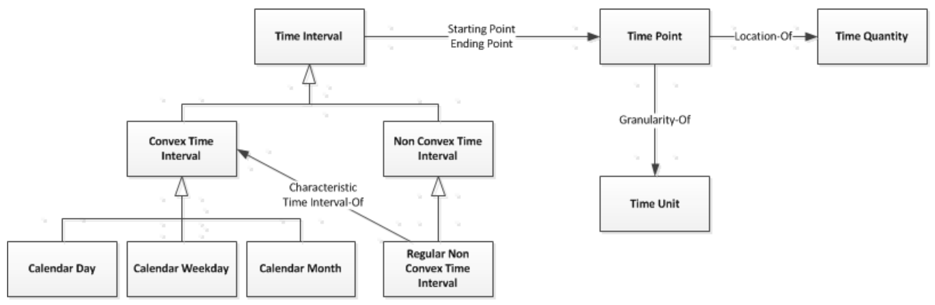

Fig. 1. Simplified representation of the *Reusable Time Ontology*

When talking about time, there are a few fundamental concepts. First *duration*, that refers to the persistence of a property over several time points or over an interval. Second, *order*: *which* expresses the occurrence of one event before or after another one, and can be applied to either time points or time intervals. *Concurrency* and *synchronicity* refer to the closeness or coincidence of two or more temporal events in time, regardless of their relative order. But also, *periodicity* which reveals the repetition of the same event with a given time period, and *evolution* to refer to the changes verified on an event or pattern occurrence over time.

2.1 Temporal Pattern Discovery

In the field of data mining, time has been addressed for decades, but mainly focused on the prediction of time series. In classification problems, time has rarely been considered as a special feature, and most always used as an ordering feature. In terms of pattern mining, time has deserved some attention, in particular in sequential pattern mining, where besides being used as the ordering key, it has also been used to impose constraints on the duration and time-distance among events [4]. Concerning temporal patterns, there are a few approaches, most of all extensions to the sequential pattern discovery ones. These methods usually combine the traditional pattern mining methods with temporal aspects, by using time information to describe the validity, periodicity or change of an association ([5]and [6]).

Besides the discovery of temporal patterns on sequential data, a few methods were proposed for dealing with transactional data, and most of all divide the temporal database into a set of partitions (according to the time granularity considered), and then find frequent temporal itemsets within these partitions. In this context, we can distinguish among the identification of *exhibition periods* or *lifespans* and *periodic patterns*. The first ones, also known as *lifespans*, correspond to the time duration from the partition when an item appears in the transaction database to the partition where the item no longer appears. The *PPM* [7] and the *Twain* [8] are examples of algorithms for this purpose.

On the other hand, *periodic patterns* are patterns that occur in regular periods or durations. In this context, Ozden et al. [9] introduced the idea of *cyclic association rules* – rules that may not hold during all time intervals, but repeat periodically from a fixed time interval until the last one considered. Relaxations to this formulation were proposed, for example through periodicity constraint relaxations specified as wildcards ([10], [11]).

An important extension to those approaches is the incorporation of calendars, as a mean to describe the information about time points and time intervals. In particular, the proposal by Ramaswamy et al. [12] allows users to define time intervals described in the form of a calendar algebraic expression. However, this approach requires that users have some prior knowledge about the temporal patterns he expects to find. In order to avoid reliance on user's prior knowledge, several approaches were proposed ([13], [14]). The approaches conceived for this purpose have, however, some limitations. First, the majority of those works are based on Allen's intervals, which present several drawbacks in terms of the expressivity of time intervals. Second, periodicity as it was defined is unable to capture some real-life concepts such as "the first business day of every month", since the distances between two consecutive business days are not constant. And third, the methods used are static in the sense that the periodicity is almost always defined ahead and then it remains the same throughout the mining process.

In this manner, it is clear that a better representation of time information, which allows for addressing those issues, should contribute significantly for simplifying and complementing the different approaches. Time ontologies are certainly such models.

3 Temporal Constraints and the D^{2}PM Framework

The D^{2}PM framework [15] has the goal of supporting the process of pattern mining with the use of domain knowledge, represented through a domain ontology, and encompasses the definition of pattern mining methods able to discover transactional and sequential patterns, guided by constraints. These constraints are the core of this framework, since they are the responsible for incorporating existing knowledge in the mining algorithm. Moreover, in this context, a constraint is much more than a predicate over the powerset of items, as usual.

Definition 1. In the context of the D^{2}PM framework, a *constraint* is defined as a tuple $C = (\sigma, \mu, \psi, \varphi)$ where σ is the *minimum support threshold*, μ is a *mapping function* that links items in the dataset with the concepts in the ontology, ψ is a predicate called the *equivalence function* that defines the equivalence among items which contribute for the same support, and φ defines the *acceptance function*, specifying the satisfying conditions to consider some pattern valid.

Along with the definition of a generic constraint, a set of pre-defined constraints was suggested: *content constraints*, *structural constraints* and *temporal constraints*. Content constraints are used to limit the items present in discovered patterns, stating that items in the rule should possess some specific characteristic, among those constraints are considered the ones that impose the existence of known relations among the items in the pattern. Structural constraints are defined as content constraints that distinguish the different expressions of patterns and define the constraints that patterns need to satisfy to be considered to be transactional, sequential or structured. Last but not least, temporal constraints focus on the temporal characteristics of events, and aim for introducing dynamic temporal aspects, for example the existence of cycles, or the fact that some transactions are limited to certain time intervals.

3.1 Temporal constraints

We chose to use this framework because it combines pattern mining with domain knowledge, represented through powerful representations models – ontologies. Another strong feature of this framework is its extensibility potential, thus allowing the introduction of different kinds of constraints and algorithms. Next, we present some basic concepts to define temporal constraints in the framework.

Definition 2. Let $L = \{ i_1, \ldots, i_n \}$ be a set of distinct literals, called *items*; a subset of these items is called an *itemset I.* A *transaction* is a tuple $T = (I, t)$, where I is an *itemset* and t a *timestamp*.

The itemset in a transaction corresponds to the set of elements that co-occur at a particular instant of time, denoted by the timestamp.

Definition 3. A transaction $T = (I, t_T)$ is said *to occur in a given time interval t* $= (t_i, t_f)$, if its timestamp is contained in the time interval t, which means that $t_i \leq t_T \leq t_f$.

A temporal constraint is a constraint that aims for introducing time information into the mining process and bases on the main concepts introduced in the reusable time ontology mentioned above.

Definition 4. A *temporal constraint* is a constraint in the D^2PM context, where the mapping function μ maps timestamps to time points at a user-defined time granularity, and the acceptance function φ filters off all the patterns that do not fit into a given time interval.

The first element in the constraint (see Definition 1) consists of the minimum support threshold σ, as defined for constraints in general in D^2PM framework, and so it is not specific to the proposed definition of temporal constraints. Similarly, the equivalence function ψ defines the equivalence among items that determines which items contribute for the same support.

However, some of the functions in the constraint definition are given different responsibilities. First, the mapping function μ has the purpose of mapping the timestamp of each transaction to a time point in the knowledge base, at the granularity chosen. In this manner, the mapping function makes the bridge between the data in the database and the domain knowledge represented in the ontology. Second, the acceptance function φ works as a filter that eliminates all the transactions that do not fit into a given time interval, and only accepting data at the granularity in consideration.

Based on the time ontology, we can further distinguish between two types of temporal constraints: *timespan constraints* and *cyclic temporal constraints.* Cyclic constraints can be further categorized into *complete cyclic temporal constraints* and *partial cyclic temporal constraints*, illustrated in Figure 1.

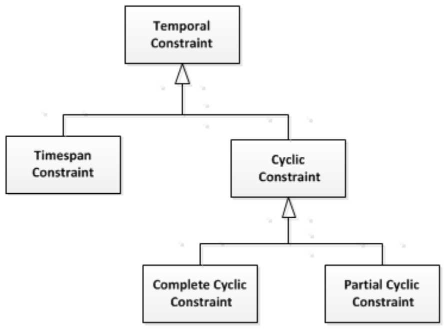

Fig. 2. Hierarchy of temporal constraints

Definition 5. A *timespan constraint* is a temporal constraint whose acceptance function only accepts patterns that occur in convex time intervals.

The timespan temporal constraint considers all patterns that occur in some convex time interval, which means that the pattern repeats over a contiguous partition of time, under the given granularity. Rules for describing items' lifespans [14] should be discovered under timespan constraints.

Definition 6. A *cyclic temporal constraint* is a temporal constraint whose acceptance function only accepts patterns that occur in non-convex time intervals.

Cyclic temporal constraints are useful to restrict the search to patterns that occur periodically, repeating from an initial time point to an ending one. These include both patterns in which every time point contributes to the cyclic behavior (Definition 7), i.e., exhibit full periodicity, and patterns which show a looser kind of periodicity where only some points contribute to it (Definition 8).

Definition 7. A *complete cyclic temporal constraint* is a cyclic temporal constraint where the acceptance function only considers open-ended non-convex time intervals.

Definition 8. A *partial cyclic temporal constraint* is a cyclic temporal constraint where the acceptance function considers non-convex time intervals with a limited duration.

The difference from complete cyclic constraint to partial cyclic constraint has respect to the duration of the time interval considered: while the first one only looks for patterns that occur periodically from a point in time until the last time point in the database, the second considers any non-convex time intervals.

4 (TD)^{2}PaM Algorithm

Using these formulations, the research problem can now be stated as: to define a method for mining temporal patterns in the context of the D^{2}PM framework, that allows for the inclusion of temporal constraints to guide the mining process.

Table 1. Toy problem original dataset

Transactions	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Timestamp	6/1/12 9:01	6/8/12 11:30	6/14/12 12:54	6/15/12 17:34	6/21/12 11:24	7/15/12 18:33	7/13/12 22:55	7/2/12 19:27	7/27/12 23:32	7/1/12 14:34	8/15/12 7:48	8/10/12 12:23	8/23/12 16:54	8/16/12 5:34	8/19/12 4:32	9/15/12 23:39	9/21/12 11:49	9/1/12 19:52	9/28/12 20:34	9/5/12 11:00
Timepoints for C_{Month}	2012-06	2012-06	2012-06	2012-06	2012-06	2012-07	2012-07	2012-07	2012-07	2012-07	2012-08	2012-08	2012-08	2012-08	2012-08	2012-09	2012-09	2012-09	2012-09	2012-09
Time Partition	D_0 (2012-06)					D_1 (2012-07)					D_2 (2012-08)					D_3 (2012-09)				
Items	30 31	30 31 32	30 31 32 34	30 32	31 33	30 31 32	32 33	31 32 33	31 32	33 34	30 31	30 32	31 32 33	31 33	30 31 33	30 32 34	30 32	30 32 33 34	30 31 35	31 33
(30 31)	Support = 3/5					Support = 1/5					Support = 2/5					Support = 1/5				

Table 2. Generated frequent itemsets with their candidate cycles in the toy example

Constraint	*Level*	*Patterns*	*Candidate cycles*
Timespan constraint	1-itemsets	pattern 0: 30 pattern 1: 31 pattern 2: 32 pattern 3: 33	[Cycle(p=1, o=2, d=2)] [Cycle(p=1, o=0, d=4)] [Cycle(p=1, o=0, d=4)] [Cycle(p=1, o=1, d=3)]
	2-itemsets	pattern 4: 31 32	[Cycle(p=1, o=0, d=2)]
Complete cyclic constraint	1-itemsets	pattern 0: 30 pattern 1: 31 pattern 2: 32 pattern 3: 33	[Cycle(p=2, o=0)] [Cycle(p=1, o=0), Cycle(p=2, o=0), Cycle(p=2, o=1)] [Cycle(p=1, o=0), Cycle(p=2, o=0), Cycle(p=2, o=1)] [Cycle(p=2, o=1)]
	2-itemsets	pattern 4: 30 31	[Cycle(p=2, o=0)]

The *Interleaved* algorithm proposed by Ozden et al. [9] is one of the few able to recognize periodic patterns, and so we propose to adapt it. The challenge is to incorporate time constraints deep into its mining process, and at the same time to extend it to deal with other patterns than those that exhibit complete periodicity. In particular, we extend the method to include timespan and partial cyclic constraints defined in the previous section. We refer the resulting method as *(TD)*^{2}PaM, standing for *Temporality in Transactional Domain-Driven Pattern Mining*.

The method devised has two main stages: in a first stage the temporal constraints are instantiated, and according to this, each transaction is assigned to a time partition (i.e., the dataset is partitioned into several time segments, according to the parameters defined in the temporal constraint), and in a second stage the algorithm runs in order to find the temporal patterns.

These stages are described next, using a toy dataset (Table 1) for illustrating its approach. Each transaction is a tuple (Timestamp, Itemset), where the transaction number is not taken into account. Additionally, consider the constraint C_{Month} = (σ= 40%, μ(T=(I,t)) = Year(t):Month(t), ψ, φ), which specifies a calendric constraint, namely Calendar-Month, so the mapping function μ translates each timestamp into the Year(t):Month(t) format.

4.1 Partitioning of the Data

The first stage, the partitioning of the data, receives the dataset file with each line representing one transaction, where one of the fields contains the timestamp and the others contain the items that compose the transaction. In this stage, data is mapped to the ontology and then partitioned into smaller datasets, according to the given time granularity.

In the first step, each transaction timestamp is mapped to the knowledge base (ontology) in use, through the constraint mapping function μ. In particular, from the timestamp in each transaction is created a time point at the granularity specified by the function. For example, the C_{Month} constraint would map all transactions to a specific month, since this is a calendric constraint it would differentiate between months of different years. The row “Time Points for C_{Month}” in Table 1 presents the complete transformation performed by the mapping function, and the dataset after the preprocessing.

Since we are interested in sets of transactions that occur within a given time period, we need to consider the Time Interval class, which by definition includes sets of two or more *Time Points*. This does not mean that instances of *Time Interval* have to correspond to connected intervals; as we have seen before, this class is a generalization and this distinction is made further down the hierarchy.

If the time intervals in consideration correspond to connected intervals, then they are instances of *Convex Time Interval*, such as *Calendar-Month*. This approach is similar to that presented in Ramaswamy et al. [12] for finding cyclic calendric rules. However, this is insufficient for our purposes, since the expressivity of these rules is somewhat limited.

For time intervals that do not correspond to connected intervals, then they are instanced as members of the *Non Convex Time Interval* class, used in the non-calendric mode of instantiation. An important subclass of this is *Regular Non Convex Time Interval*, which represents regularly occurring events and thus represents a cyclic temporal constraint as defined above (Definition 6). So, if we want to consider transactions that occur "every Friday in December" this would be an instance of *Regular Non Convex Time In*terval. This consists of four to five connected intervals, each one representing a Friday, which is in turn an instance of *Convex Time Interval*.

In the second step, the mapped data are partitioned into different sub-datasets, called *time partitions*.

Definition 9. A *time partition* D_i is the set of transactions that occur in the time interval [i x tg, (i+1) x tg], where tg is the time granularity, defined by the μ mapping function in the temporal constraint in use.

Row "Time Partitions" in Table 1 shows the different partitions for the constraint considered. Four partitions are created, one for each month, since transactions span from June to September 2012. So, each transaction, which occurs in the interval defined by the time point, is added to the partition of its corresponding time segment. In order to segment the dataset into different partitions, it is enough to use different temporal constraints, in particular different mapping functions.

4.2 Finding Patterns

In the second stage of the algorithm, the goal is to find complete cyclic, partial cyclic and timespan patterns. After applying the data partitioning, the dataset consists of a list of sets of transactions D_0, ..., D_{n-1}, where D_i is a set of transactions that takes place in a particular time point i, with the time unit within a given granularity.

In the original Ozden et al.'s work [9], the problem of finding all cyclic association rules is solved by the Interleaved algorithm. In this context, a *cycle* is defined as a tuple (*p-period*, *o-offset*) that defines time intervals that start at the *o* time point and that repeat *p* after *p* time points. Given a minimum support threshold σ, a *cyclic pattern* is defined as a tuple (I, ρ), where I is an itemset and ρ is a cycle (p, o), and such I is frequent during the cycle ρ. For example, for a given granularity of hour, the cyclic pattern ({ highGlucoseLevels, insulin } (24, 7)) is discovered if the itemset displays a level of support above minimum threshold repeatedly for the 7-9 a.m. period every 24 hours.

The Interleaved algorithm is Apriori-based and works in two stages: first, the algorithm finds the frequent cyclic itemsets and in a second stage it generates the corresponding cyclic association rules. The algorithm uses three optimization techniques to improve cycle detection: *cycle skipping*, *cycle pruning* and *cycle elimination*.

The strict definition of cyclic patterns only allows for considering periodicity, ignoring duration and the other referred properties. In order to approach them consider the following definitions.

Definition 10. In the D^{2}PM context, a *cyclic pattern* is defined as a tuple *I* (*p*,*o*,*d*), where *d* defines the duration of the pattern, and the periodicity does not have to happen throughout the totality of temporal partitions in consideration, but only partially until the $o+d$th partition.

A particular case of a cyclic pattern is a *complete cyclic pattern*, which corresponds to the one considered by Interleaved, and which *d* corresponds to the total number of partitions. For partial cyclic patterns, the duration parameter *d* corresponds to the number of repetitions of the itemset.

Definition 11. A *timespan pattern* is defined as a temporal pattern, where the cycle is a tuple (*p*,*o*,*d*) defining non-convex time intervals, the period *p* assumes the value 1 and the duration parameter *d* defines the number of time partitions (counted from the offset *o* onwards) in which the pattern is present.

The temporal constraints defined previously are used to target the particular patterns that one aims to discover, in a straightforward way: using a timespan constraint leads to finding timespan patterns, and using a partial or complete cyclic constraint elicits patterns belonging to their corresponding pattern categories.

In order to find this new set of patterns, we propose an adaptation of the Interleaved method that is able to integrate the target temporal constraints in the mining process. We center our method in the discovery of patterns instead of generating association rules. This adaptation, besides allowing the discovery of complete cyclic patterns, contains also an extension for timespan and cyclic patterns with limited duration. The pseudo code for this is shown in Algorithm 1.

Algorithm 1. (TD)^{2}PaM algorithm

```
Input:
-  A temporal constraint C and a sequence $D_0$, ..., $D_{n-1}$, where $D_i$ is a set of itemsets
-  Lmin > 0, Lmax > 0, Minimum level of support $\sigma$
Output:
-  All cyclic, complete cyclic or timespan patterns (I (p, o, d))

Pattern Generation
   $P^1$ = Find-1-ItemsetPatterns()
   For (i = 2 ; ($P^{i-1} \neq$ empty); i++)
      $T^i$ := Find-k-ItemsetCandidatePatterns(i);
      $P^i$ := CheckCandidatePatterns(C, $T^i$)    //Employs optimization techniques

CheckCandidatePatterns:
   For each itemset in Ti:
      Cycle-skipping determines the set of k-itemsets for which support will be
      calculated in each D[i].
      If (Constraint C is a Timespan Constraint)
         If (itemset is frequent in consecutive partitions
            &&  Number of partitions left < minimum duration )
               Add cycle (1, i, duration)
      If (Constraint C is a Partial Cyclic Constraint)
         If (itemset has the minimum support in partition D[i])
            Check successively each (D[i] + period) until ((pattern is
            not frequent in D[i]) or (number of partitions left < period))
            If (number of repetitions of pattern < minimum duration)
               Add cycle (period, i, duration)
```

The proposed algorithm receives as input the temporal constraint, as well as the partitioned dataset, and like Interleaved [9] works in two stages: first, it finds frequent 1-itemsets for each partition (function *Find-1-ItemsetPatterns*); and then, it generates frequent k-itemsets from the (k-1) itemsets for each partition (*Find-k-ItemsetCandidatePatterns*), performing pruning on each step and using the temporal constraint to filter out candidates (*CheckCandidatePatterns*). Indeed, like Interleaved, it follows the candidate generation and test strategy, exploring the anti-monotonicity property: if a j-cycle is frequent (i.e., has support above minimum value), all of the i-itemsets for i<j are also frequent.

In both stages, the candidate procedure is different for complete cyclic constraints on the one hand, and for partial cyclic and timespan constraints on the other. For the first, the complete set of candidate cycles is generated and then progressively eliminated using *cycle elimination*. For partial cyclic constraints, this initial cycle generation is not complete, since each cycle generated begins with duration value set to 0, and the cycle determination is only completed in the *CheckCandidatePatterns* procedure. Finally, for timespan constraints cycles are not generated ahead at all, but only determined also in the *CheckCandidatePatterns* stage.

The procedure for pruning is specified in the temporal constraint, and differs according to the type of constraint received as input. For complete cyclic temporal constraints, the *CheckCandidatePatterns* procedure corresponds to the application of *cycle skipping*, *pruning* and *elimination*. The distinction here is that the temporal constraint is responsible for this checking process which prunes some of the candidates to consider in the rule generation stage.

For partial cyclic and timespan constraints, for each possible period, the pruning procedure (*CheckCandidatePatterns*) checks if there is a repetition with at least duration *d* of a specified value. For example, if the minimum duration is 3, the itemset has to be frequent in at least 3 time partitions to be considered a pattern of interest. For partial cyclic constraints, this involves checking for all possible periods if there is a repetition from p to p time units starting in the partition specified in the offset.

The *CheckCandidatePatterns* procedure, both for timespan and cyclic patterns, only takes advantage of *cycle skipping* and *cycle pruning*. As for cycle elimination, it cannot be applied in these cases, since these are not of cyclic nature in a strict sense (period is assumed to be 1 for timespan patterns, and for cyclic patterns each cycle does not last throughout all the partitions), so for these patterns another optimization strategy is employed. This involves considering a threshold value for the duration parameter, which is used to stop considering a pattern as soon as the number of partitions to process is not enough to achieve a pattern with duration equal or higher than the duration threshold.

We will use the example above to illustrate the algorithm steps for two of the types of constraints considered. In this, the partitioned data serves as input to the algorithm. We use 0.4 as the minimum support threshold, 1 for Lmin and 2 for Lmax. The minimum and maximum length of the period correspond to Lmin and Lmax values.

Table 1 shows how to determine the support for the pattern (30 31), in each partition of the data. According to the support constraint, the pattern only stands in D_0 and D_2. The algorithm receives also as input a temporal constraint that specifies both the granularity and the type of temporal pattern of interest. The candidate patterns generated by the $(TD)^2$PaM procedure in the end of the pattern generation stage are shown in Table 2, with their respective candidate cycles for timespan and complete cyclic constraints.

The algorithm generates all the 1-itemsets and their corresponding possible cycles, which for complete cyclic patterns are: (p=1,o=0), (p=2,o=0) and (p=2,o=1). For itemset 30, observing that this itemset is frequent in all partitions except D_1 allows us to eliminate the first two candidate cycles, maintaining only cycle (p=2,o=0). For itemset 31, we cannot eliminate any cycle because this itemset is frequent in all partitions. So, for the candidate 2-itemset (30 31) we calculate the intersection of the candidate cycles for both 1-itemsets, which is (p=2,o=0) and only need to assess if this cycle is present, and thus calculate support in the corresponding partitions. For partial cyclic constraints, the same process would be applied, with some differences, namely taking into account cycle duration and performing cycle elimination as described above.

In the case of the timespan constraint, the cycles generated for each itemset correspond to contiguous partitions in which the itemset is frequent, and the temporal constraint defines a minimum duration (here we consider at least 2) that is considered to obtain these 1-period timespan cycles. For 1-itemset 30, the only possible timespan comprises the contiguous partitions D_2 and D_3, so the only candidate is (p=1,o=2,d=2), while for 1-itemsets 31 and 32 we have a timespan that includes all the partitions, with cycle (p=1,o=0,d=4). So, for the 2-itemset (30 31) by computing the intersection between both 1-itemset candidate cycles we only need to determine support in D_2 and D_3, and we find that this cycle is not present; for itemset (31 32) we need to verify that the candidate cycle is present, but since the itemset is only frequent in D_0 and D_1, we retain the cycle (p=1,o=0,d=2).

In the next sections the proposed $(TD)^2$PaM method is applied to a real-world dataset, in order to perform its evaluation and discuss some of its implications.

5 Experimental Results

This case study uses a dataset comprised of the listening habits of Lastfm users, and was created using the Last.fm API (http://www.lastfm.com.br/api/tos). It represents the listening habits from nearly 1000 users, spanning a time period of 5 years, and each record contains *user id*, *user gender*, *user country*, *timestamp*, *artist* and *track* listened. From these users, 100.000 records were randomly selected, and used for the analysis of algorithm performance and generation of temporal patterns.

In this section, we analyze the main results achieved by the $(TD)^2$PaM method, for the three types of temporal constraints considered in this work, namely: complete cyclic, partial cyclic and timespan constraints. We also perform tests, for comparison purposes, using the original version of the Interleaved algorithm [9] that allows finding complete cyclic patterns without resourcing to the time ontology used in this work, but performing each data partition in a sequential manner.

This analysis was conducted for several time granularities, and that are representative of the kinds of results that we can achieve. The granularities considered were *year*, *quarter*, *month*, *weekday* and *hour*, but we only present the results for *month* and *hour* granularities, mainly due to lack of space. The granularities chosen to present, raise two different kinds of time intervals. The *month* granularity corresponds to a convex time interval, the *calendar-month*, and maps timestamps to some particular month of a specific year. The *hour* granularity corresponds to a regular non-convex time interval, mapping timestamps to the hour of the day. For example *June 1st, 2013 9:01 am* and *July1st, 2013 9:51 am*, would map to two different time points at the *month* granularity (*June 2013* and *July 2013*), but to only one at the *hour* granularity (*9 am*).

Since the algorithm receives as input the maximum length of the period for the cases of complete and partial temporal constraints, this value also has to be parameterized but not presented. In this work, we use as minimum length the value 1 and as maximum the value given by (number of partitions / 2), rounded down to the nearest integer value, which seems to be a good rule of thumb, but other values could be considered.

The average number spent per pattern, obtained by dividing the total time spent by the number of patterns found is shown in Figure 3. We can see that the charts present the same general trend for all of the types of constraints used and for all time granularities, showing an increase in the average time per pattern when the support increases. This can be attributed to the fact that for higher minimum support values the number of patterns found is greatly reduced, so the average time spent in each one increases. Levels of support above 20% were not evaluated, since no patterns are found in the great majority of cases. Also, for both granularities, the average time spent per pattern in the case of partial patterns is tendentiously higher than for the other types of patterns, for minimum support levels above 1%.

In Figure 4, we present the results concerning the total running time of the algorithm, for each granularity considered. Time values were subjected to a logarithmic transformation such that $t = ln(t+1)$. The lines in the chart present the typical pattern found in pattern mining algorithms, showing an explosion in time (accompanied by an increase in the number of patterns found, as we will see next), for lower support levels (bellow 1%).

The total running time of the algorithm for partial cyclic and timespan patterns is higher across all granularities, when compared to the running time for the complete cyclic patterns. This, in turn, is almost the same as for the original version of the Interleaved algorithm that does not use a time ontology. This result shows that even though the proposed method uses an extra constraint instantiation step, this does not have a deleterious effect on the algorithm performance.

The disparity found between results for the timespan and partial cyclic patterns on the one hand and the complete cyclic patterns on the other can be partially explained by the higher number of patterns found when using timespan constraints, as we will see ahead. For partial cyclic constraints, even though in some cases the number of patterns discovered is not significantly higher than the number of patterns found when using complete cyclic constraints, the higher running time of the algorithm can be attributed to the fact that the number of partitions that we need to check in order to verify if a pattern is present represents a higher cost for the algorithm. Moreover, as seen in Figure 3, the average time spent per pattern seems to be higher for partial patterns than for the other types, especially if we consider higher support values (i.e. above 5%). In contrast, in the case of complete cyclic constraints, if we verify that an itemset is not above minimum support in a given time partition, we instantly eliminate all candidate cycles (*j*, *i* mod *j*), and do not have to count support for this itemset in the corresponding partitions (which is the step of the algorithm that is the most time-consuming).From a quantitative point of view, we will describe the number of patterns found according to different values of minimum support considered, and the maximum pattern length achieved.

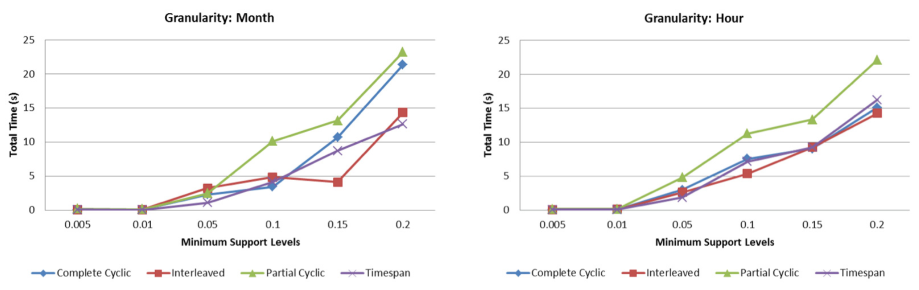

Fig. 3. Average time spent per pattern for (TD)^{2}PaM and Interleaved algorithms for varying support levels

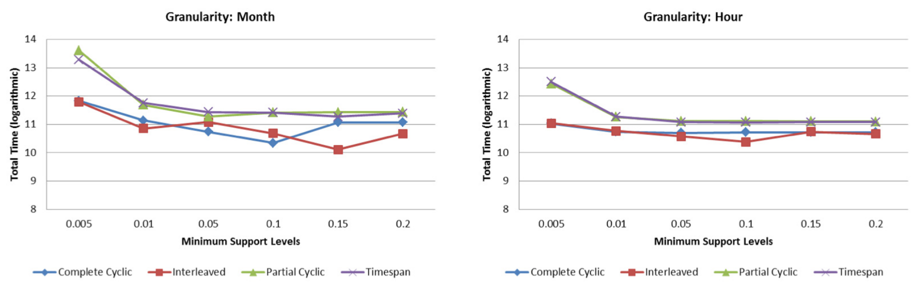

Fig. 4. Total running time for (TD)^{2}PaM and Interleaved algorithms for varying support levels

Figure 5 shows the total number of patterns found for all types of patterns and granularities considered. For levels of minimum support above 10%, we can see that the number of patterns found is extremely reduced, and for the lowest levels (1% and especially for 0.5%), there is a great increase in the number of patterns. This is consistent also both with the results found for pattern mining algorithms in the literature, and with the running time results presented above (and showing an increase in time spent for lower support levels). If we compare both granularities shown in figure 5, we can see this explosion in the number of patterns found for lower support levels and the corresponding decrease in the number of patterns found with higher support.

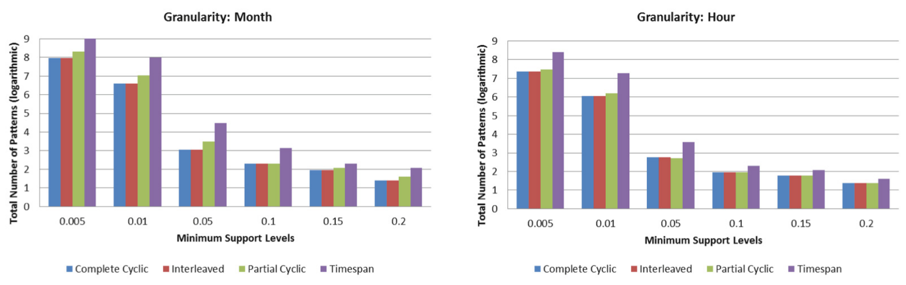

Fig. 5. Total number of patterns generated for $(TD)^2$PaM (in a logarithmic scale) for varying support levels

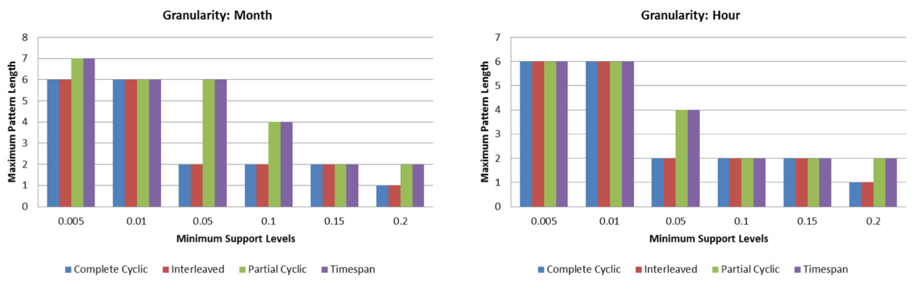

Fig. 6. Maximum pattern length for $(TD)^2$PaM for varying support levels

Figure 6 shows the variation of the maximum pattern length. For the lower values of minimum support, the maximum pattern length achieved is the highest, across all the different granularities. In most cases, the maximum pattern length achieved is the same for all the types of patterns considered, and when it varies (for example in the case of the month granularity for a support of 5%), the maximum length is higher in the case of partial and timespan patterns than for the complete patterns. This result can be merely a reflection of the higher number of patterns found for the first two cases.

Performing a more qualitative analysis of the patterns revealed three different types of patterns: patterns containing only dimensions that refer to the users (containing attributes such as gender and country), patterns referring to artists and/or tracks, and patterns that contain items from both classes of dimensions. These last patterns relate items from the users with the tracks that they listened to. Examples of patterns containing only user dimensions and of patterns containing track dimensions can be seen in Table 3.

Looking at patterns with one item first, for example pattern P1 shows that women listened to tracks on the Lastfm website every month since the beginning of the dataset (the granularity in consideration is month granularity, which is calendric), since this is a complete cyclic pattern and its period is 1 and offset is 0. If we look at pattern P3, we can see that the artist “Blink-182” is listened to for 2 hours (duration) starting at 5 a.m., since the granularity considered is hour (this is a non-calendric granularity, since it aggregates every hour records in the dataset).

Table 3. Illustrative set of temporal patterns and their cycles (p= period, o= offset, d= duration)

	ID	*Patterns*	*Cycles*	*Minimum Support*	*Constraint Type*	*Granularity*
User dimensions	P1	Gender = female	(p=1,o=0)	15%	Complete cyclic	Month
	P2	Gender=male, Country=United Kingdom	(p=2,o=12, d=6) (p=3,o=15,d=3) (p=5,o=13,d=3)	5%	Partial Cyclic	Hour
Track dimensions	P3	Artist=Blink-182	(p=1,o=5,d=2)	1%	Timespan	Hour
	P4	Artist=Daft Punk, Track=Aerodynamic	(p=26,o=9) (p=25,o=10) (p=26,o=10)	1%	Complete cyclic	Month

Table 4. Illustrative set of temporal patterns and their cycles (p= period, o= offset, d= duration), with both items relating to the user and the tracks

ID	*Patterns*	*Cycles*	*Minimum Support*	*Constraint Type*	*Granularity*
P1	*Gender=female, Country=United States, Artist=Elliott Smith*	*(p=4,o=0) (p=3,o=0) (p=4,o=3)*	*0.005*	*Complete cyclic*	*Semester*
P2	*Gender=male, Artist=Queen*	*(p=8,o=3) (p=9,o=2)*	*0.01*	*Complete cyclic*	*Hour*
P3	*Country=United States, Artist=Andrew Bird*	*(p=24,o=1,d=4) (p=26,o=2,d=3) (p=26,o=18,d=3)*	*0.005*	*Partial cyclic*	*Month*
P4	*Gender=female, Country=United States, Artist=Death Cab For Cutie*	*(p=2,o=2,d=4) (p=3,o=3,d=3) (p=7,o=2,d=3)*	*0.005*	*Partial cyclic*	*Hour*
P5	*Gender=female, Artist=The Gathering*	*(p=1, o=2, d=2)*	*0.01*	*Timespan*	*Weekday*
P6	*Gender=female, Country=United States, Artist=Daft Punk*	*(p=1, o=0, d=3)*	*0.005*	*Timespan*	*Weekday*

Patterns P2 and P4 are examples of patterns with more than one item, revealing the co-occurrence of two or more items. In P2, we can see that male users from the United Kingdom listen to tracks with period 2, 3 and 5, respectively starting at 12, 15 and 13 hours (and with the duration pertaining to the number of times this happens during the 24 hours of the day). Even though this information can be hard to grasp at face value, it could be useful for a website that tracks user's listening habits to find such patterns in these periods, for example. In pattern P4, the track "Aerodynamic" is indeed from the artist Daft Punk, so in this case we could discard either artist name or track name from the pattern, or maintain this if intended.

Table 4 shows some illustrative examples of some of the patterns found using this method, which contain items from both user and tracks that they listen to. Looking at some of these patterns, we can see, for example (P2) that males listen to Queen starting at 3 and 2 a.m., with periodicity of 8 and 9 hours. Another example (P6), states that females from the United States listen to Daft Punk music, starting on Sunday (weekday 1, corresponding to the 0 offset), for three days of the week

(Sunday, Monday and Tuesday). So, the patterns presented allow the establishment of relations between both classes of items (user and track dimensions).

We have shown that the proposed method has indeed the potential to find patterns that incorporate the temporal dimension, and could be used as a component of a recommendation system or for advertising targeting purposes, among other applications.

6 Conclusions

The main purpose of this work consisted in developing an integrated and efficient method able to mine expressive temporal patterns in a time-stamped transactional dataset, through the incorporation of temporal constraints in the mining process. In this context, we extended the D^{2}PM framework, with the formalization of temporal constraints, and the proposal and definition of its subtypes, namely timespan and cyclic constraints.

In addition, we defined an efficient algorithm able to deal with these constraints: the (TD)^{2}PaM algorithm. It adapts the Interleaved algorithm [9] for using time constraints defined based on the notions of time represented in a time ontology, and makes use of the same strategies to augment its performance. In this manner, our algorithm allows for the discovery of both lifespans and periodical patterns, according to any time granularity chosen, without any further pre-processing.

According to experimental results over real datasets, is clear that the most consuming step in the algorithm remains the support counting, which leads to the necessity of replace the candidate-based approach with a pattern-growth one. Another interesting line of research would be centered on automatically selecting the granularity levels and other parameters such as maximum period length for the cyclic patterns. To address the granularity choice, one strategy may be adopt the approach followed on mining generalized association rules [16], establishing an hierarchy of time granularities to be considered, and consider all the possibilities.

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

References

1. Mitsa, T.: Temporal Data Mining. Chapman & Hall/CRC (2010)
2. Allen, J.: Maintaining knowledge about temporal intervals. Communications of the ACM 26(11), 832–843 (1983)
3. Zhou, Q., Fikes, R.: A Reusable Time Ontology. In: Press, A. (ed.) AAAI Workshop on Ontologies for the Semantic Web (2002)
4. Srikant, R., Agrawal, R.: Mining Sequential Patterns: Generalizations and Performance Improvements. In: Apers, P.M.G., Bouzeghoub, M., Gardarin, G. (eds.) EDBT 1996. LNCS, vol. 1057, pp. 3–17. Springer, Heidelberg (1996)
5. Chen, Y., Chiang, M., Ko, M.: Discovering time-interval sequential patterns in sequence databases. Expert Systems Applications 25, 343–354 (2003)

6. Mannila, H., Toivonen, H., Verkamo, A.: Discovery of Frequent Episodes in Event Sequences. Data Mining and Knowledge Discovery 1(3), 259–289 (1997)
7. Lee, C., Chen, M., Lin, C.: Progressive partition miner: an efficient algorithm for mining general temporal association rules. IEEE Transaction on Knowledge and Data Engineering 15(4), 1004–1017 (2003)
8. Huang, J., Dai, B., Chen, M.: Twain: Two-End Association Miner with Precise Frequent Exhibition Periods. ACM Transactions on Knowledge Discovery from Data 1(2) (2007)
9. Ozden, B., Ramaswamy, S., Silberschtaz, A.: Cyclic Association Rules. In: International Conference on Data Engineering, pp. 412–421 (1998)
10. Laxman, S., Sastry, P.S.: A survey of temporal data mining. Sadhana 31(2), 173–198 (2006)
11. Han, J., Dong, G., Yin, Y.: Efficient mining of partial periodic patterns in time series database. In: International Conference on Data Engineering, Sydney, Australia, pp. 106–115 (1999)
12. Ramaswamy, S., Mahajan, S., Silberschatz, A.: On the Discovery of Interesting Patterns in Association Rules. In: International Conference on Very Large Databases, New York, USA, pp. 368–379 (1998)
13. Li, Y., Ning, P., Wang, X., Jajodia, S.: Discovering calendar-based temporal association rules. Data Knowledge Engineering 44, 193–218 (2003)
14. Zimbrão, G., Souza, J., Almeida, V.T., Silva, W.: An Algorithm to Discover Calendar-based Temporal Association. In: Workshop on Temporal Data Mining at ACM SIGKDD Int'l Conf on Knowledge Discovery and Data Mining, Edmonton, Alberta, Canada (2002)
15. Antunes, C., Bebiano, T.: Mining Patterns with Domain Knowledge: a case study on multi-language data. In: International Conference on Information Systems, Shanghai, China, pp. 167–172 (2012)
16. Srikant, R., Agrawal, R.: Mining Generalized Association Rules. In: International Conference on Very Large Databases, Zurich, pp. 407–419 (1995)