Temporal Pattern Mining Using a Time Ontology

Cláudia Antunes

Instituto Superior Técnico / Technical University of Lisbon
Av. Rovisco Pais 1, 1049-001 Lisbon, Portugal
claudia.antunes@dei.ist.utl.pt

Abstract. The analysis of temporal data has deserved a considerable attention, in particular in the analysis of time series. However, the general research on data mining seldom has focused its attention on dealing with the specific attribute – time. The discovery of temporal patterns, that reveal interesting behaviors over time, is one of such cases. In this paper, we propose a new approach to effectively find frequent temporal patterns. Our approach is based on the interleaved algorithm and the use of a time ontology, which precisely defines the main temporal concepts. Based on these concepts is then possible to effectively generate all possible time intervals when frequent patterns could occur, and then count each corresponding support, identifying frequent patterns. The algorithm follows the candidate generation and test philosophy, being able to deal with different time granularities.

Keywords: Pattern mining, Temporal patterns, Cyclic rules, Time ontology, Time granularity, Calendar.

1. Introduction

Time has been one of the most interesting phenomena for men: understanding its relation to observed changes has been a target for thousands of years. Temporal data appears naturally in almost everything in nature, and the improvements on the digital storage capabilities verified in the last decades, has contributed to increase the interest in the discovery of hidden information.

One of the answers to this challenge is the automatic discovery of temporal patterns, and has been mainly performed using the pattern mining approach, either in its pure form or as sequential pattern mining. With these techniques is possible to identify frequent and common behaviors that reveal actions performed at the same time, or as sequences of actions.

However, and despite their relative success, most of the times, these techniques only consider the actions them selves, without paying attention to the instant of time when they are performed. In this manner, it is not possible to understand the impact of time over entities' behavior.

One possibility to surpass this problem is to discover temporal patterns, which mean to find frequent behaviors that occur in particular instants of time, *i.e*, instants of time with certain characteristics. In this paper, we present an extension to

Interleaved algorithm (introduced in [9]) that uses a time ontology to discover *calendric* and *cyclic rules* – rules that reveal those behaviors.

The remainder of this paper is organized as follows. In next section we overview the work on pattern mining that addresses the discovery of temporal patterns, giving a particular attention to the *interleaved algorithm*. In section 3, we describe the reusable time ontology and redefine the problem of temporal pattern mining in this new context. The section ends with the description of the new algorithm – *onto-interleaved*, and the illustration of its execution over a toy-problem. In section 4, some conclusions and directions for future work are discussed.

2. Literature Review

The analysis of temporal data is usually based on the notion of event. An event is a pair (e, t) where e is an event type (usually called an *item*) and t is a positive integer, called the *timestamp* of e [5]. Examples of event types are the items transacted or the actions performed by some entity, or simply some measure over any natural or artificial phenomena.

The discovery of frequent patterns on temporal data has been mainly performed by pattern mining algorithms, either in their original format (like apriori [1]), or in their extension for the discovery of sequential patterns (sequential pattern mining [2]).

In general, those algorithms act incrementally, discovering a pattern with k items, after the discovery of patterns with (k-1) items. This approach allows for the reduction of the search space at each step, making use of the anti-monotonicity property, which states that a pattern only can be frequent if all its sub-patterns are also frequent.

In their original format, pattern mining algorithms aim to discover the sets of items that occur together in a sufficiently large number of events. In this way, the discovered patterns reveal intra-transactional tendencies, and only make possible the prediction of co-occurrences. Usually, pattern mining is used to define association rules, which can be seen as predictive rules for simultaneous outcomes. In contrast, sequential pattern mining allows the discovery of inter-transactional patterns, identifying frequent sequences of sets of items. In this manner, the prediction of future behaviors is possible. When applied to temporal data, these algorithms deal with sequences of events (see [4], for a detailed description).

In general, other temporal issues (beside the sequence of events) have been discarded from the sequential pattern mining process. Actually, most algorithms do not use any temporal constraint. The easy matching of each sequence element with a time instant has lead to a simplification of the problem, avoiding the introduction of other more complex temporal relations, beside its pure sequential nature. The exception to this general frame is the use of sequential constraints, like the imposition that two consecutive elements are closed enough, which means that two consecutive events dist at most a certain number of events (this constraint is known as the *gap constraint*, and is the most used in sequential pattern mining algorithms).

Other works in sequential pattern mining addresses this problem by searching for periodicities in time series [6]. The main idea is to look for the discovery of periodic segments, recognizing that only some segments have cyclic behavior. The discovery

of valid time periods during which association rules hold and the use of items, in general, has also been performed by extending the apriori algorithm, partitioning the initial data in several time periods and then applying the pattern mining process to each partition [8].

Another interesting and more powerful approach is the exploration of constraints over temporal relations, as the ones defined for discovering *cyclic* and *calendric rules*.

2.1. Cyclic Rules and the Interleaved Algorithm

Cyclic rules are association rules that have minimum support and confidence at regular time intervals. In this manner, a cyclic rule can occur only in a small part of the database, corresponding to particular time instants that occur at specific time intervals. In order to discover such rules, it is necessary to search for frequent patterns in a restricted portion of time, since they may occur repeatedly at specific regular time instants but on a small portion of the global time considered. A method to discover such rules is based on the application of any algorithm to obtain the set of traditional rules, and then to detect the cycles behind the rules.

Interleaved Algorithm. A more efficient approach to discover cyclic rules consists on inverting the process: first discover the cyclic patterns and then generate the rules. This algorithm is known as the *interleaved algorithm* [9].

Like any apriori-based algorithms, it has two steps: in the first one, it discovers the set of cyclic large itemsets (*cyclic patterns*) and on the second one, it discovers the cyclic rules.

Since cyclic rules can be generated using the cycles and the support of itemsets, a simple extension of *gen_rules* algorithm is enough (details can be found in [9]). The discovery of cyclic large itemsets is far more interesting, since it explores three new techniques: *cycle-pruning*, *cycle-skipping* and *cycle-elimination*.

A *cycle* is a pair $c=(l, o)$ where the second element o corresponds to the first instant (called a time unit in the original paper) of the cycle and l determines the length of the cycle, corresponding to a multiple of o. Note that the time unit corresponds to a unitary time interval (intervals with a unitary time unit, like a day, a month or a week), and the cycle itself corresponds to a set of consecutive time units, which is itself a time interval with holes.

An itemset has a cycle c, if the itemset occurs in every convex time interval of c. In this manner, an itemset is said to be a large cyclic itemset if its support in some cycle is at least equal to the minimum support required.

Note that a cycle c is multiple of another cycle c' if c' is contained in c. Enunciated techniques explores this definition and the anti-monotonicity property. Additionally, consider a time segment $D[i]$ as the portion of the dataset that occurs in the time interval t_i.

Like apriori, the algorithm works in a candidate generation and test philosophy, generating the k-candidates based on the k-1 cyclic large itemsets. First, it considers as 1-candidates all single items over all possible time intervals, and determines 1-cyclic large itemsets by counting their support and discarding the infrequent ones.

However this support counting can be enhanced by using cycle-skipping and cycle elimination.

Cycle-skipping avoids counting the support for an itemset in time intervals that do not belong to the cycle of the itemset, since if a time interval t_i is not part of a cycle of an itemset, then there is no need to calculate its support in the corresponding time segment $D[i]$. On the other hand, if an itemset is not frequent in some time segment $D[i]$, then it cannot have cycles defined over that time segment. This property allows for the reduction of the cycles that an itemset could have, and it is applied by *cycle-elimination*.

After identifying 1-cyclic large itemsets, k-candidates are generated as in apriori, and using a third technique – *cycle-pruning*, which is based on the property that states if an itemset has a cycle, then any of its subsets has the same cycle. In this manner, k-candidates are just the itemsets generated by joining (k-1)-large itemsets over the cycles that are multiples of the cycles of any of its subsets. In this manner, the number of cycles of an itemset is less than or equal to the number of cycles of any of its subset.

For each k; $k > 1$:

1. If $k = 1$; then all possible cycles are initially assumed to exist for each single itemset. Otherwise (if $k > 1$), cycle-pruning is applied to generate the potential cycles for k-itemsets using the cycles for (k - 1)-itemsets.

// Time segments are processed sequentially.

2. For each time unit t_i:
   - 2.1 Cycle-skipping determines, from the set of candidates cycles for k-itemsets, the set of k-itemsets for which support will be calculated in time segment $D[i]$.
   - 2.2 If a k-itemset X chosen in Step 2.1 does not have the minimum support in time segment $D[i]$; then cycle-elimination is used to discard each cycle C, for which time unit t_i holds, from the set of potential cycles of X.

Fig. 1 Pseudo-code for the *Interleaved* Algorithm [9]

At the end of this procedure, the large itemsets over some cycle are identified, and their support is counted.

2.2. Calendric Rules

Calendric rules are an extension of cyclic rules, and express the rules that are frequent over any time interval.

The interleaved algorithm can be naturally applied in the discovery of calendric rules, without significant changes. In [10], where these rules were defined, it is also demonstrated that with minor changes, the interleaved algorithm is able to deal with different time granularities (such as days, weeks or months). In order to do this, it is only necessary to consider that a time unit that belongs to some time interval specified at a particular time granularity can also belong to another time interval expressed in a

different time granularity. In this manner, only the candidate generation has to consider multiple granularities.

The new extension of the algorithm deals with calendars, as defined in [7]. A *calendar* is a set of time intervals. Details about the use of calendars can be found in [3].

3. Mining Temporal Patterns with Reusable Time Ontology

With the advances in the area of knowledge representation and management, nowadays it is simpler to deal with specific notions of certain concepts. For example calendars can be precisely defined recurring to the concepts described in a time ontology.

Before presenting the extension of the interleaved algorithm, we describe the Reusable Time Ontology and state the problem of mining temporal patterns.

3.1. Reusable Time Ontology

The Reusable Time Ontology [11] is based on the notion of a time line, with time being continuous and linear, and defines the main temporal concepts.

The core concepts are *Time Point* and *Time Interval*, but include the concepts of *Time Quantity*, *Time Unit* and *Time Granularity*, as shown in Fig. 2 (represented in UML, and with the relations labeled with their names; inheritance is represented by empty arrows).

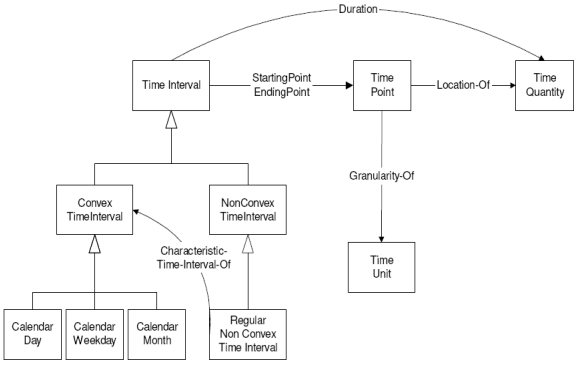

Fig. 2. Reusable Time Ontology

A *Time Point* represents a specific time position on the timeline, which can be viewed as a particular moment. A *Time Interval* corresponds to a time-period that occurs between two time points (the *Starting Point* and the *Ending Point*). Another way to see time intervals is to consider them as approximations to time points [7], accepting some uncertainty on their value. This means that a time interval may represent partial information about the location of a time point. Time intervals are the

most central concept for temporal pattern mining, since they can represent both the lifespan of items and periods of interest.

This ontology defines two classes of time intervals, convex and non-convex ones. The first class corresponds to connected intervals in the time line and the second is a disjoint and complete decomposition of convex time intervals, which correspond to non-connected time intervals, *i.e.* with "holes" in them. A special kind of non-convex time interval is the *Regular Non Convex Time Interval*, which is composed of several convex time intervals for representing regularly recurring events.

Moreover, a time point with a certain level of granularity is a single time point with the uncertainty that it may be anywhere in a certain time interval. For example, time point "June1st,2007" with day granularity is a single time point that can be any point within the convex time interval starting midnight of May31st,2007, and ending midnight of June1st ,2007. The function *Granularity-Of* receives a *Time Point* and returns a *Time Granularity* instance, which may have an arbitrary number of instances, representing the different levels of granularity since every time quantity can correspond to a level of granularity. The ontology already have some pre-defined instances for commonly used time units: *Year-Granularity*, *Month-Granularity*, *Day-Granularity*, *Hour-Granularity*, *Minute-Granularity* and *Second-Granularity*, and we extended it to include *Week-Granularity* and *Season-Granularity*. Fig. 3 presents the granularity graph that shows the relations among different granularities and the conversion from a time granularity to another more abstract. Note that for the time point "June1st,2007", defined at the *Day-Granularity*, the results of applying functions *Month-of*, *Year-of*, *Season-of* and *Week-Day-of* are *June2007*, 2007, *Spring* and *Friday*, respectively. Additionally, note that all these abstractions could be seen as time points at different granularities or as time intervals, all convex except the two last ones, which correspond to regular non-convex ones.

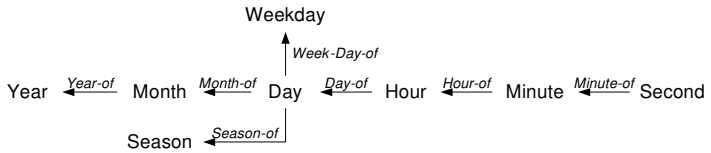

Fig. 3. The granularity graph

Furthermore, the ontology defines a *Time Quantity* as an amount of time that is represented by a real number and a *Time Unit*. Note that the relation *location-of* determines the position of a specific time point in the timeline. Finally, *Time units* correspond to the granularities of time, such as year, month, day, hour, and so on. Note that there is a correspondence between *Time units* and *Time granularities*, achieved by the relation *Time-Unit-Of*. For example *Year-Granularity* has *Year* as its *Time-Unit-Of* function value.

The ontology also provides a set of predefined *Convex Time Intervals*, for different days, weekdays and months – *Calendar Day*, *Calendar Week* and *Calendar Month*, respectively. Additionally, *Calendar-Day-1* and *Calendar-Day-31* are subclasses of *Calendar Day*, with instances *June1st,2007* and *June30th,2007*, respectively. In the same manner, *Calendar-June* is a subclass of *Calendar Month*, and *June,2007* is one of instances.

Besides, it is easily extended to incorporate more sub-classes of convex time intervals, such as *Calendar Year*, *Calendar Decade*, *Weekend*, *Working Day, Calendar Season*, and all the other temporal abstractions of interest for some domain.

It is important to note that in order to deal with time intervals similar to "*every Monday in June*" we have to use a regular non convex time interval, whose characteristic time interval corresponds to a *Calendar-Monday*.

Beside the relations represented graphically, the ontology also defines three important predicates *before*, *after* and *equal*, defined over two time points. They correspond to "<", ">" and "=" operators in the timeline, respectively.

Based on this predicates, it is easy to define the predicates *belong* and its counterpart *contains*, that verifies if a time point occurs in a time interval.

Definition 1. A time point t_i *belongs* to a convex time interval $\mathcal{T}$, with starting point t_0 and ending point t_n, if t_i is before or equal to t_n and t_i is after or equal to t_0. In the same conditions, $\mathcal{T}$ is said to *contain* t_i.

For non-convex time intervals those predicates can also be applied:

Definition 2. A time point t_i *belongs* to a non-convex time interval $\mathcal{T}$, if t_i belongs to any of $\mathcal{T}$ convex time intervals.

Other important feature, is the definition of functions like *Year-Of*, *Month-Of*, *Day-Of*, *Week-Day-Of*, *Hour-Of*, *Minute-Of* and *Second-Of*, that can be derived by the *Location-Of* relation, but can be very useful for dealing efficiently with temporal data. Similar functions are *Day-Of-Month* or *Season-Of* that can be added to the ontology. Note that by using these functions it is easy to generate all the abstractions of a specific *Time Point*.

3.2. Problem Statement

Like for discovering calendric rules, the extension of the interleaved algorithm to use time intervals, instead of calendars, is straightforward. However, before presenting the algorithm, it is important to present a clear statement of the problem of discovering temporal patterns in the context of the reusable time ontology, and to relate the concepts used in previous works with the concepts defined in this new context.

Consider a set of items $\mathcal{I}$, an *itemset* $\mathcal{A}$ as a subset of $\mathcal{I}$ ($\mathcal{A} \subseteq \mathcal{I}$), and an *event* $e=(\mathcal{A}, t)$ as an itemset that occurred at a particular time point t, called its *timestamp*.

Definition 3. An itemset $\mathcal{A}$ *occurs* on a time point t if there is an event $e=(\mathcal{B}, t)$ where $\mathcal{A}$ is contained in the event itemset $\mathcal{B}$ ($\mathcal{A} \subseteq \mathcal{B}$). An itemset $\mathcal{A}$ *occurs* on a time interval $\mathcal{T}$ if there is an event $e=(\mathcal{B}, t)$ that contains the itemset $\mathcal{A}$ and its timestamp t belongs to $\mathcal{T}$.

Based on this notion, we can address the repetition of some itemset over an entire time interval, as required in cyclic and calendric rules.

Definition 4. An itemset A *occurs repeatedly over* a time interval T if A occurs at every time point of T.

Additionally, consider a *dataset* $\mathcal{D}$ as a set of events and a *data segment* $\mathcal{D}_T$ as the portion of events of $\mathcal{D}$ that occur on the time interval T. (Note that data segments correspond to time segments defined in [9]).

Definition 5. Given a dataset $\mathcal{D}$ and a real number σ (with $\sigma \in]0,1]$), a *temporal pattern* is an itemset that occurs repeatedly over a particular time interval T, at least in $\sigma \times 100\%$ of the events in the data segment $\mathcal{D}_T$.

Using, these notions is now possible to state the problem of mining temporal patterns:

Definition 6. Given a dataset $\mathcal{D}$ and a real number σ (with $\sigma \in]0,1]$), find all temporal patterns existent over all possible time intervals defined in the time scope of the dataset.

Note that the most interesting temporal patterns correspond to itemsets that occurs repeatedly over regular non-convex time intervals. This kind of time interval presents some time regularities like the ones described before as cycles.

3.3. Onto-Interleaved Algorithm

In the context of the reusable time ontology and the problem of mining temporal patterns, we can redefine the interleaved algorithm. This new extension is called *onto-interleaved* algorithm, and its pseudo-code is presented in Fig. 4.

```
Procedure Onto-Interleaved(Dataset D, double σ,
                           Convex Time Interval T)
   I←{single itemsets in D}
   TP←{all time points in T}
   L1←discoverL1(I, TP, σ)
   k←2
   while Lk-1≠∅ do
      Ck←join(Lk-1)
      Lk←supportPruning(Ck,σ)
      k++
   return ∪k Lk
```

Fig. 4. Pseudo-code for *onto-interleaved* algorithm

As expected, the algorithm receives the dataset $\mathcal{D}$ and minimum support threshold σ as parameters. Additionally, it receives the time interval of interest T, the time span where it will look for temporal patterns. Note that this time interval can be simply the convex time interval with the first and last timestamps of the dataset as starting and ending points, respectively.

The first step is naturally the discovery of frequent temporal patterns with 1 item (procedure *discover*L_1 described in detain in Fig. 5). This procedure receives an itemset for each item in the dataset ($\mathcal{I}$), and the set of time points t that belong to $\mathcal{T}$ (elements of $\mathcal{TP}$). Since all possible abstractions should be tested, the procedure invokes the *genAllTimeGranularitiesFor* function to generate all possible abstractions for each time point, and creates a candidate for each combination of itemsets and time intervals.

For example, according to the granularity graph on Fig. 3, the application of *genAllTimeGranularitiesFor* to *Jun1st2007* returns {*Jun1st2007, Jun2007*, *2007*, *Friday*, *Spring*}.

```
Procedure discoverL1 (Set of Itemsets I, Set of Time Points TP, double σ)
    for each t∈ TP do
        T_i←genAllTimeGranularitiesFor (t)
        for each t_i∈ T_i do
            T←T ∪ new Time Interval (t_i)
    C_1← I×T
    L_1←supportPruning (C_1,σ)
return L_1
```

Fig. 5. Pseudo-code for *discover*L_1 procedure

After creating all possible candidates (combinations of single itemsets with possible time intervals), their support is counted, and infrequent ones are discarded.

The discovery of the rest of the patterns consists on combining frequent single itemsets that share a same time interval. For example if $\mathcal{A}$ and $\mathcal{B}$ are frequent over the interval *Jun30th2007*, then ($\mathcal{A}$,$\mathcal{B}$) can also be frequent on that time interval. However, if $\mathcal{A}$ and $\mathcal{B}$ are frequent over disjoint time intervals, then ($\mathcal{A}$,$\mathcal{B}$) cannot be frequent and its support count can be avoided. This technique corresponds to the *pruning* (described as *cycle-prune* before), and is used in the *join* procedure (in Fig. 4) for generating k-candidates.

After identifying k-candidates, a second technique is applied to reduce the time spent on counting the support of candidates – the *skipping* technique. It is used in *supportPruning* procedure in Fig. 4, and avoids counting the support of a candidate over some data segments. If an itemset $\mathcal{A}$ is not frequent in the time interval $\mathcal{T}$, then it is not necessary to count its support in the corresponding data segment $\mathcal{D}_\mathcal{T}$. Note that the reduction of the search space (the number of dataset events to check) contributes significantly to reduce the time spent in counting the support. The algorithm terminates its execution when there is no candidate to test.

The generation of rules is done in the same manner as described in [9], and is not addressed here.

3.4.Illustrative Example

In order to exemplify the usage of the *onto-interleaved* algorithm, consider the set of events constituted by ({*a*,*b*,*c*,*d*},*Jun1st07*), ({*a*,*b*,*c*},*Jun8th07*), ({*a*,*b*},*Jun2nd07*), ({*a*,*c*},*Jun2nd07*) and a support of 75%. Note the convex time interval entering as parameter in the onto-interleaved algorithm will be [Jun1st2007, Jun8th2007]. Table 1 shows its execution, presenting the candidates, their support and the corresponding frequent ones.

Table 1. Onto-interleaved execution with 75% of support

C_1	Sup	L_1	C_2	Sup	L_2
(*a*,Jun1st07)	100%	(*a*,Jun1st07)	(*ab*,Jun1st07)	100%	(*ab*,Jun1st07)
(*b*,Jun1st07)	100%	(*b*,Jun1st07)	(*ac*,Jun1st07)	100%	(*ac*,Jun1st07)
(*c*,Jun1st07)	100%	(*c*,Jun1st07)	(*ad*,Jun1st07)	100%	(*ad*,Jun1st07)
(*d*,Jun1st07)	100%	(*d*,Jun1st07)	(*bc*,Jun1st07)	100%	(*bc*,Jun1st07)
(*a*,Jun2nd07)	100%	(*a*,Jun2nd07)	(*bd*,Jun1st07)	100%	(*bd*,Jun1st07)
(*b*,Jun2nd07)	50%	(*a*,Jun8th07)	(*cd*,Jun1st07)	100%	(*cd*,Jun1st07)
(*c*,Jun2nd07)	50%	(*b*,Jun8th07)	(*ab*,Jun8th07)	100%	(*ab*,Jun8th07)
(*d*,Jun2nd07)	0%	(*c*,Jun8th07)	(*ac*,Jun8th07)	100%	(*ac*,Jun8th07)
(*a*,Jun8th07)	100%	(*a*,Jun07)	(*bc*,Jun8th07)	100%	(*bc*,Jun8th07)
(*b*,Jun8th07)	100%	(*b*,Jun07)	(*ab*,Jun07)	75%	(*ab*,Jun07)
(*c*,Jun8th07)	100%	(*c*,Jun07)	(*ac*,Jun07)	75%	(*ac*,Jun07)
(*d*,Jun8th07)	0%	(*a*,07)	(*bc*,Jun07)	50%	(*ab*,07)
(*a*,Jun07)	100%	(*b*,07)	(*ab*,07)	75%	(*ac*,07)
(*b*,Jun07)	75%	(*c*,07)	(*ac*,07)	75%	(*ab*,*Spring*)
(*c*,Jun07)	75%	(*a*,*Spring*)	(*bc*,07)	50%	(*ac*,*Spring*)
(*d*,Jun07)	25%	(*b*,*Spring*)	(*ab*,*Spring*)	75%	(*ab*,*Friday*)
(*a*,07)	100%	(*c*,*Spring*)	(*ac*,*Spring*)	75%	(*ac*,*Friday*)
(*b*,07)	75%	(*a*,*Friday*)	(*bc*,*Spring*)	50%	(*bc*,*Friday*)
(*c*,07)	75%	(*b*,*Friday*)	(*ab*,*Friday*)	100%	
(*d*,07)	25%	(*c*,*Friday*)	(*ac*,*Friday*)	100%	
(*a*,*Spring*)	100%	(*a*,*Saturday*)	(*bc*,*Friday*)	100%	
(*b*,*Spring*)	75%				
(*c*,*Spring*)	75%				
(*d*,*Spring*)	25%				
(*a*,*Friday*)	100%				
(*b*,*Friday*)	100%				
(*c*,*Friday*)	100%				
(*d*,*Friday*)	50%				
(*a*,*Saturday*)	100%				
(*b*,*Saturday*)	50%				
(*c*,*Saturday*)	50%				
(*d*,*Saturday*)	0%				

C_3	Sup	L_3	C_4	Sup	L_4
(*abc*,Jun1st07)	100%	(*abc*,Jun1st07)	(*abcd*,Jun1st07)	100%	(*abcd*,Jun1st07)
(*abd*,Jun1st07)	100%	(*abd*,Jun1st07)			
(*bcd*,Jun1st07)	100%	(*bcd*,Jun1st07)			
(*abc*,Jun8th07)	100%	(*abc*,Jun8th07)			
(*abc*,*Friday*)	100%	(*abc*,*Friday*)			

At the first step, candidates are the pairs achieved by combining the set of items {*a*,*b*,*c*,*d*} with the set of possible time intervals. Note that the procedure *genAllTimeGranularitiesFor* will be called for *Jun1st07*, *Jun2nd07* and *Jun8th07*, generating the set {*Jun1st07*, *Jun2nd07*, *Jun8th07*, *Friday*, *Saturday*, *Jun07*, *2007*, *Spring*}. In support counting, it is possible to verify that for example (*a*, *Jun1st07*) has a support of 100%, since it occurs in every event with timestamp *Jun1st07*, and (*b*, *Jun2nd07*) is not frequent given it occurs in only one of two events on *Jun2nd07*.

After the first step, candidates are generated using the same method as the join procedure in apriori, but only considering itemsets that share the same (*k*-1)-prefix and occur in overlapping time intervals (*pruning* as described in section 3.3). For example, (*ab*,*Jun1st07*) is a candidate but (*ab*, *Jun2nd07*) is not, since (*b*, *Jun2nd07*) is not frequent. Note that anti-monotonicity can also be applied. For example, it is not necessary to count the support for (*abd*,*Jun8th07*) given (*bd*,*Jun8th07*) is not frequent.

With the third optimization, achieved through the use of the skipping technique, the support counting is restricted to the data segments corresponding to specific time intervals. For example, for counting the support of (*a*, *Jun1st07*) we only need to check the event ({*a*,*b*,*c*,*d*},*Jun1st07*).

4. Conclusions

Temporal pattern mining has been seldom applied on temporal data, with time series the only exception to this scenario. Indeed, there are just a few approaches to identify temporal patterns over non-numeric data, with. Though the interleaved algorithm is one of the most promising ones, to our knowledge it has not been applied to real problems. One of the possible reasons for this failure is requiring that users define calendars of interest, and possibly the difficulties on implementing efficient versions of the algorithm.

In this paper, we try to surpass these difficulties by redefining the problem in a simpler context, based on the usage of a time ontology that should contain all the time intervals of interest for some problem domain. In this manner, users have their job facilitated, and limited to choose the time intervals to consider. In this new context, is then possible to define a simple algorithm, easier to implement and to improve.

The next step, is naturally to develop more efficient algorithms, for example, following other pattern mining approaches than the candidate generation and test one.

References

1. Agrawal, R., and Srikant, R. Fast Algorithms for Mining Association Rules in Large Databases. In *Proc. Int'l Conf. Very Large Data Bases* (1994) pp. 487–499.
2. R. Agrawal and R. Srikant, "Mining Sequential Patterns", in *Proc. Int'l Conf. Data Engineering* (ICDE 95), pp. 3-14, 1995.
3. Allen, J.F., "Maintaining Knowledge about Temporal Intervals", in *Communications of the ACM*, vol. 26, nr. 11, 1983.

4. Antunes, C. *Pattern Mining over Nominal Event Sequences using Constraint Relaxations*. Ph.D. Thesis, Instituto Superior Técnico, Lisboa, Portugal, January 2005.

5. Bettini, C., Jajodia,S. and Wang, S.X, *Time Granularities in Databases, Data Mining and Temporal Reasoning*, Springer, 2000.

6. Han, J., G. Dong and Y. Yin, "Efficient Mining of Partial Periodic Patterns in Time Series Database", in Proc. Int'l Conf. Data Engineering (ICDE 99), pp. 106-115, 1999.

7. Hayes, P. *A Catalog of Temporal Theories*, Technical Report UIUC-BI-AI-96-01, University of Illinois, 1995.

8. Lee, H., C.R. Lin, M.S. Chen, "On Mining General Temporal Association Rules in a Publication Database", in *Proc. Int'l Conf. Data Mining* (ICDM01), pp. 337-344, 2001.

9. Özden, B., Ramaswamy, S., and Silberschatz, A. Cyclic Association Rules. In *Proceedings of the Fourteenth International Conference on Data Engineering*, (Feb. 1998)

10. Ramaswamy, S., Mahajan, S., and Silberschatz, A. On the Discovery of Interesting Patterns in Association Rules. In *Proceedings of the 14th Very Large Data Bases Conference*, New York, USA, 1998.

11. Zhou, Q., and Fikes, R. *A Reusable Time Ontology*. in Semantic Web Workshop at AAAI'02. AAAI Press. 2002.