Pattern Discovery in Temporal Databases: A Temporal Logic Approach[1]

Balaji Padmanabhan and Alexander Tuzhilin
Leonard N. Stern School of Business
New York University
{bpadmana, atuzhili}@stern.nyu.edu

Abstract

The work of Mannila et al. [4] of finding frequent episodes in sequences is extended to finding temporal logic patterns in temporal databases. It is argued that temporal logic provides an appropriate formalism for *expressing* temporal patterns defined over categorical data. It is also proposed to use Temporal Logic Programming as a mechanism for the *discovery* of frequent patterns expressible in temporal logic. It is explained in the paper how frequent temporal patterns can be discovered by constructing temporal logic programs.

Introduction

In this paper, we address the problem of finding interesting patterns in temporal databases [1,2] defined over categorical (symbolic) data. This is an important problem that frequently occurs in various applications such as molecular biology (finding patterns in genetic sequences), telecommunications (finding patterns in network behavior) and financial services (finding patterns in analysts' recommendations of stocks). To address this problem, a language for *expressing* temporal patterns has to be defined and mechanisms to *discover* patterns in temporal databases need to be developed. This is a broad problem, and it has been addressed before in such fields as speech recognition, signal processing, as well as in the KDD field itself. For example, Agrawal et al. [3] provide a *shape definition language*, SDL, for expressing shapes in sequences. In the context of categorical data, the problem has been addressed by Mannila et al. [4] and by the string matching research community [5].

Mannila et al [4] define temporal patterns in sequences of events with *episodes*, where episodes are defined as partially ordered sets of events that can be described by directed acyclic graphs. Given a class of such episodes they describe an efficient algorithm that finds all frequent episodes from that class. Their paper presents an interesting approach that was tested on telecommunications data. However, the episodes defined in [4] have a limited expressive power in specifying temporal patterns. For example, it is unclear how episodes can define such temporal patterns as "event A always occurs until event B occurs" or that "either event A or event B occurs at the same time." In addition, their approach works on sequences, and not on temporal databases (i.e. temporal *predicates* changing over time).

String matching researchers use regular expressions to define patterns on strings of alphabets and develop efficient string matching algorithms. Aho [5] presents a survey of such algorithms. Regular expressions are defined over an alphabet, and therefore this approach works well with strings of symbols but does not generalize to temporal databases, where *predicates* change over time.

In this paper we extend the work of Mannila et al. [4] (finding frequent patterns in temporal categorical data) and propose the use of first-order temporal logic (FOTL) [6] to express patterns in temporal databases (the propositional case constitutes a special case of the first-order case). As an example of how episodes can be expressed in Temporal Logic (TL), the partial order of symbols "A→B→C", defining a *serial* episode of [4], can be expressed as "A *Before* B *and* B *Before* C". In addition, TL can also be used to express patterns such "Hold(Stock) *Until* Bearish_Market_Sentiment", where Hold is a temporal predicate. Moreover, we propose to use TL for discovering temporal patterns by generating *temporal logic programs* (TLP) [7,8,9] for these patterns.

TL provides several important advantages as a mechanism for the specification and discovery of patterns in temporal databases. It is a well-studied, expressive and theoretically sound formalism that has been extensively used in various fields of computer science for dealing with temporal phenomena. TL can be used *both* for the specification of temporal patterns and for their discovery in temporal databases (using TLP techniques described below). Therefore, TL provides a sound framework for integrating the specification and discovery methods.

Preliminaries

We use FOTL [6,10] to express temporal patterns. The syntax of FOTL is obtained from first-order logic by adding temporal operators such as *Since, Until, Next (o),* and *Previous (•)* and some of the derived operators, such as *Always* (□), Future and Past *Sometimes* (◊,♦), *Before, After* and *While*. In addition, we consider *bounded* temporal operators [9,12] $Until_K$, $Since_K$ etc. For example, A$Until_K$B is defined as:

$(D,t) \models A\ Until_K\ B$ iff $\exists m, t \le m \le t+k$, such that $(D,m) \models B$ and $\forall i, t \le i < m, (D,i) \models A$

The semantics of FOTL is defined in terms of *temporal structures*, i.e., predicates changing over time [10]. We assume that time is discrete, linear and bounded, that temporal predicates define a *temporal database* [1,2], and that temporal relations are represented as *event tables* [13],i.e. tables with a single temporal attribute.

A *temporal pattern* is a ground FOTL formula, i.e. a FOTL formula containing only ground atoms and no

[1]This work was supported in part by the NSF under grant IRI-93-18773

variables and quantifiers. A *class* of temporal patterns is defined by a temporal formula, ψ, with one or more variables in it. An *instantiation* of all variables in ψ with specific ground values, defines a temporal pattern. In this paper we adopt the convention of using uppercase alphabet to represent variables in a temporal formula.

Temporal Logic Programming. A TLP program consists of a set of temporal rules of the form BODY $\rightarrow$ HEAD, where various TLP systems make different assumptions about the structure of BODY and HEAD. For example, a rule that "Employees who have been fired from a firm (worked there sometime in the past, but not now) cannot be hired by that firm in the future" can be expressed in an extension of TLP system, *Templog* [7] as:

$$\blacklozenge \text{EMPLOY(firm, person)} \wedge \neg \text{EMPLOY(firm, person)} \rightarrow \Box \neg \text{EMPLOY(firm, person)}$$

Alternatively, as done in $Datalog_{1S}$ [14], we can also express TLP programs in first-order logic using explicit references to time. For example, instead of using the temporal predicate EMPLOY(firm, person), we can use its FOL equivalent EMPLOY(firm, person, time) specifying the employment history of the person over time. Moreover, Templog and the corresponding FOL language Datalog_{1S} are equivalent in their expressive power [8]. We will use an extension of Datalog_{1S} to express TLP programs. Datalog_{1S} will be extended by allowing negation both in the body and in the head of a rule, as done in doubly negated $\text{Datalog}^{\neg *}$ [15], and by allowing comparisons between temporal variables (e.g. $t_1 < t_2$) in the body of Datalog_{1S} rules. We will call the resulting extension $eDatalog_{1S}$. We will adhere to the parallel inflationary semantics of $\text{Datalog}^{\neg *}$ [15] when we define semantics of eDatalog_{1S}. Intuitively, all the eDatalog_{1S} rules are fired in parallel, and if there are conflicts in rules,program execution terminates, as done in $\text{Datalog}^{\neg *}$.

Finding Frequent Temporal Patterns

In this paper we address the problem of finding *frequent* and *most frequent* patterns in temporal databases. For example, if *D* is a temporal database, then we may want to find all the frequent patterns in the class of temporal patterns , $\psi = X_1\ Until_K\ X_2$, where X_1 and X_2 are second-order variables ranging over the predicates in the temporal database *D*. To do this, for any temporal pattern that belongs to the class ψ, we can count the number of time instances for which the pattern holds on *D*. The pattern is then frequent if it exceeds a threshold value *c*.

For example, assume that X_1 is associated with predicate p(X), X_2 with predicate q(X) in *D*, X is instantiated to a_0, and K=5. If a pattern "$p(a_0)\,Until_5\ q(a_0)$" occurs 90 times in *D* and *c* is 70, we conclude that the pattern occurs frequently in *D*. Alternatively, we could have searched for the *most frequent* patterns in *D*, i.e., the patterns having maximal frequency counts in comparison to other patterns in the given class.

In the unrestricted case, the problem of finding the most frequent patterns can be trivially non-interesting. If we consider the (infinite) class comprising of *all* TL formulae, then an example of the most frequent pattern would be "p(a) $\vee$ $\neg$p(a)", where p(X) is a temporal predicate in a temporal database *D*. Therefore, we have to restrict our consideration to certain well-defined classes of temporal patterns for which the problem becomes non-trivial. For example, some classes of temporal patterns can be defined as follows:

1. A single FOTL formula, where the "variables" could be arguments of predicates, but cannot be of second-order, i.e. range over predicates.
2. A parameterized single FOTL formula defines a class of temporal patterns that may differ from each other by some parameters of the temporal operators. In the example discussed above, "K" is a parameter of *Until.*
3. The class of all the temporal patterns defined using only AND and NEXT operators.

Class Defined By A Single FOTL Formula

To illustrate the discovery methods described in this section, consider the following class of temporal patterns defined by the expression

$$\psi(X,Y,Z) = a(X,Y)\ \textit{Until}\ b(Y,Z) \qquad (1)$$

where a and b are temporal predicates from the temporal database *D*. We want to find all possible instantiations for the variables X, Y and Z for which pattern (1) occurs frequently (its frequency is above a certain threshold). To find such instances, we construct a TLP that will identify all the occurrences of temporal patterns that belong to ψ.

Let a(X,Y,T), b(Y,Z,T), and ψ(X,Y,Z,T) be the temporal predicates that appear in (1), but with explicit references to time. Figure 1 illustrates a TLP program, written in eDatalog_{1S}, that computes ψ(X,Y,Z,T) using a distinguished predicate q(X,Y,Z,T).

```
(i) simtime(0)
(ii) simtime(T) → simtime(T+1), ¬ simtime(T)
(iii) simtime(T), a(X,Y,T), ¬a(X,Y,T-1) →
        flag1(X,Y,T), flag2(X,Y,T)
(iv) simtime(T), a(X,Y,T), a(X,Y,T-1) → flag2(X,Y,T)
(v) simtime(T), b(Y,Z,T), flag1(X,Y,T1), flag2(X,Y,T-1),
        (T1 ≤ T2 ≤ T) → q(X,Y,Z,T2)
(vi) simtime(T), ¬a(X,Y,T ), flag1(X,Y,T1),flag2(X,Y,T-1),
        (T1 ≤ T2 ≤ T) → ¬flag2(X,Y,T2), ¬flag1(X,Y,T1)
(vii) b(Y,Z,T) → q(X,Y,Z,T)
```

Figure 1. TLP computing a(X,Y) *Until* b(Y,Z).

To simulate the forward movement in time, the program in Fig.1 uses predicate *simtime* that acts as a system clock, and Rule (ii) advances it forward on a tick-by-tick basis. Rule (iii) sets flag1 when the predicate a(X,Y) is true for the first time instant in a period of time when it holds continuously, and rule (iv) continues to set a new flag (flag2) for all time points in that period. Now when b(Y,Z) is encountered with flag1 and flag2 on, rule (v) sets the distinguished predicate, q(X,Y,Z) to be true for all instances of time from when flag1 was first set *until* the time when b(Y,Z) holds. Rule (vi) resets flags after the *end* of any continuous period of time when a(X,Y) holds.

To find frequent (or most frequent) patterns of the form (1), the program in Fig.1 can easily be extended to incorporate a counter of the number of instances when q(X,Y,Z,T) is true. Note that the use of (first-order) variables in the program facilitates construction of a *single* TLP program to find all instances of temporal patterns in *D* that belong to the class ψ. We can generalize this example into the following theorem, the proof of which can be found in [11]:

Theorem. For any class of temporal patterns specified by a single FOTL formula φ defined on a temporal database *D*, there exists a TLP program PROG with the distinguished predicate *q*, such that for any finite instance of *D*, $q \cong \varphi$, i.e. *q* holds whenever φ holds and vice versa.

The proof of this theorem is by induction on the number of operators in the TL formula φ, and is *constructive*. Also, there is a notion of "safety" of FOTL expressions, without which TLP programs may not terminate. Our TLP programs can be shown to terminate if *D* is finite and the domains of the arguments in predicates are all finite.

Class of Patterns Consisting of AND and NEXT Operators

Here we consider the class of temporal patterns TL{$\wedge$,o} consisting of $\wedge$ and o operators and study two problems of finding the *most* frequent and *all* frequent patterns.

Most Frequent Patterns. Because of the distributivity of $\wedge$ and o operators, any pattern from TL{$\wedge$,o} depending on temporal predicates X_0, X_1, ..., X_n can be converted to the following canonical form (for clarity we omit arguments of the predicates) :

$$X_0 \wedge o^{k1} X_1 \wedge o^{k2} X_2 \ldots \wedge o^{kn} X_n \qquad (2)$$

where $1 \leq k1 \leq \ldots \leq kn$ and for $i \neq j$, $X_i \neq X_j$ when ki= kj. The *most frequent* patterns in this case will be of the form:

$$\psi_{i,j}(K) = X_i \wedge o^K X_j \qquad (3)$$

The TLP that counts the occurrences of all the temporal patterns that belong to $\psi_{a,b}(X,Y,K) = a(X) \wedge o^k b(Y)$ is given in Figure 2, where predicate $count_{a,b}$ (X,Y,K,VAL) specifies how many times (VAL) the pattern $\psi_{a,b}$ (X,Y,K) occurred in the database.

```
(i) simtime(0)
(ii) simtime(T) → simtime(T+1)
(iii) simtime(T), a(X,T), b(Y,T+K), count_{a,b} (X,Y,K,VAL),
   ¬ψ_{a,b} (X,Y,K,T)→ψ_{a,b} (X,Y,K,T),count_{a,b} (X,Y,K,VAL+1)
```

Figure 2. TLP program that simulates a $\wedge$ o^K b.

The program counting the most frequent occurrences of patterns belonging to (3) is obtained by combining the TLP programs presented in Fig. 2 for all possible pairs of predicates a and b and adding the "control" logic selecting the highest values (VAL) in predicate $count_{a,b}$ for various values of a, b, X, Y and K. We would like to point out that the program that finds the most frequent patterns of the form (3) (and hence (2)), does it in *one single* forward "sweep" in time during the execution of the program, and, thus, its complexity is linear in time.

Frequent Patterns. The starting point for finding frequent patterns from TL{$\wedge$,o} is the canonical form (2) of these patterns. Since frequency of patterns is a monotonically decreasing function of n in (2), there is a value of n for which *no* temporal pattern given by (2) is frequent. Therefore, our goal is to find such maximal value of n (N) and also find *all* frequent patterns of the form (2) for n<N.

Unlike the case of the most frequent patterns, we cannot reduce (2) to a simplified expression and will be dealing with the general case of (2). A naive approach would be to find frequent patterns for (2) for successive values of n (until the next value of n does not generate any new frequent patterns) without using the outputs from previous iterations. However, we will use a more efficient algorithm that is based on the idea of generating larger candidate patterns from smaller ones. This method of generating candidate patterns was used effectively by Agrawal et al.[16] and by Mannila et al [4] to mine association rules and episodes respectively.

We will first start with finding frequent patterns for a single predicate p. In other words, the expression (2) is reduced to the case :

$$p(X) \wedge o^{k1} p(X) \wedge o^{k2} p(X) \ldots \wedge o^{kn} p(X) \wedge \ldots \qquad (4)$$

where X is a vector of the attributes of predicate p.

The algorithm, described in Fig 3, iteratively generates frequent patterns for successive values of n in (4) by utilizing the frequent patterns discovered in the previous iteration. T* is the largest value in the time domain (such value exists since the time domain is bounded), $freq_n(X,K_1,..,K_n)$ is the class of all the frequent patterns found at stage n, $count(X,K_1,..,K_n,VAL)$ is a predicate that tracks the frequency count, VAL, and *c* is the threshold frequency value. Predicate $holds(T,X,K_1,...,K_n)$ is used in Rule (iii) to avoid double-counting. Initially, it is set to False for all of its values. The algorithm executes until the saturation point is reached, i.e. until $freq_n = \phi$.

```
n=1 and freq_0 = set of all frequent ground predicates p
repeat
        compute freq_n with TLP program TLP_n ;
        n=n+1;
until (freq_n = φ);
print freq_i , i=1,2,...,n-1
            where the program TLP_n is :
(i) simtime(0)
(ii) simtime(T) ,T ≤ T* → simtime(T+1)
(iii) simtime(T), p(X,T), p(X,T+K_1),...,p(X,T+K_{n-1}),
        freq_{n-1} (X,K_1,...,K_{n-1}), p(X,T+K_n),
        count(X,K_1,..,K_n,VAL), ¬holds(T,X,K_1,...,K_n) →
        holds (T,X,K_1,...,K_n), count(X,K_1,..,K_n, VAL +1)
(iv) simtime(T*+1), count(X,K_1,..,K_n, VAL), (VAL > c)
        → freq_n (X,K_1,...,K_n)
```

Figure 3. Algorithm to find all frequent patterns of form (4)

Note that in rule (iii) in Figure 3, we add the predicate $freq_{n-1}(X,K_1,...,K_{n-1})$ to the body of the rule. This can improve efficiency if a smart rule evaluation strategy is used since, for all values of T, $freq_{n-1}(X,K_1,...,K_{n-1}) \Rightarrow p(X,T),\ p(X,T+K_1),...,p(X,T+K_{n-1})$. Thus, we have to evaluate $p(X,T),\ p(X,T+K_1),...,p(X,T+K_{n-1})$ only for the frequent values of $X,K_1,...,K_{n-1}$ in Rule (iii).

The generalization of the algorithm presented in Figure 3 to the class of patterns of the form (2) that are defined by multiple predicates is straightforward but notationally cumbersome and, thus, we omit it. Furthermore, it leads to the combinatorial explosion in the size of the TLP programs constructed by the algorithm (as a function of the number of predicates in the database). We would like to note that, in contrast to the "most frequent patterns" case, we construct *multiple* TLP programs (one program per iteration) in this algorithm. This means that the complexity of the algorithm is no longer linear in T*.

Experiments and Conclusion

We used TL to express patterns and implemented TLP programs in OPS5 for an application in which different financial analysts rate various stocks in terms of buying, selling, or holding recommendations. In this application, we wanted to find the most frequent recommendation change patterns made by various analysts and the correlation patterns of stock recommendations across different analysts (e.g. frequently, analyst A recommends "buy" for a stock until analyst B recommends "sell"). For example, a recommendation change pattern can be expressed as:

$$\text{Analyst\_Report(analyst,stock,recommendat)} \wedge \neg \circ \text{Analyst\_Report(analyst,stock,recommendat)}$$

In this case we want to find the analysts that changed their recommendations most often for different stocks. The $eDatalog_{1S}$ program that finds such patterns is very similar to the one presented in Figure 2, and we simulated this program in OPS5 on a Sun Sparc20 using an artificially generated data set consisting of 3 analysts, 3 stocks, and 300 days (we assumed that the frequency of recommendations was one day). The performance as a function of the size of the data set is presented in Figure 4.

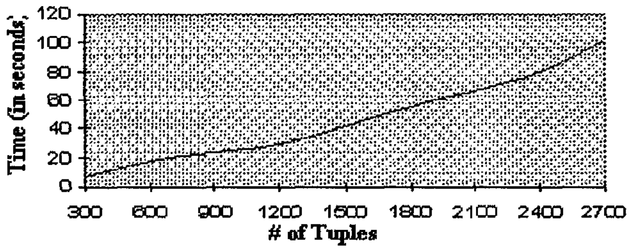

Figure 4. Execution Time Vs. Database Size

These preliminary results demonstrate that the performance of our implementation is quite slow. We attribute this to the following factors. Mainly, our version of OPS5 does not support arithmetic in the body of a rule. Therefore, we simulated expressions of the form P(A,T+1) by creating new predicates and making the OPS5 program bigger and less efficient. Second, the OPS5 interpreter has the serial semantics (one tuple instantiation per recognize-act cycle), and this also slowed the execution of the TLP program. In summary, to make the TLP technology practical for the pattern discovery purposes, there is a need to develop efficient TLP interpreters that support such important features as pseudo-parallel execution of temporal rules and that would provide an efficient support for the temporal dimension.

References

[1] Tansel, A.U., Clifford, J., Gadia, S., Jajodia, S., Segev, A. and Snodgrass, R., 1993. In *Temporal Databases, Theory Design and Implementation.* Benjamin/Cummings.

[2] Clifford J., Tuzhilin A., 1995. *Recent Advances in Temporal Databases.* Springer-Verlag.

[3] Agrawal, R., Giuseppe, P., Edward, W.L., and Zait, M., 1995. Querying Shapes of Histories. In *Proceedings of the 21st VLDB Conference*, pp. 502-514.

[4] Mannila, H., Toivonen, H., and Verkamo, A.I., 1995. Discovering Frequent Episodes in Sequences. In *The First International Conf. on Knowledge Discovery in Databases (KDD-95)*, pp 210-215.

[5] Aho, A.V., 1990. Algorithms for Finding Patterns in Strings. In van Leeuwen, J., ed., *Handbook of Theoretical Comp. Sc., Vol A :Algorithms and Complexity.* Elsevier.

[6] Manna, Z., and Pnueli, A., 1992. Temporal Logic. In *The Temporal Logic of Reactive and Concurrent Systems.* Springer-Verlag.

[7] Abadi, M., and Manna, Z., 1989. Temporal Logic Programming. *J. of Symb. Computation* v8, pp. 277-295.

[8] Baudinet, M., Chomicki, J., and Wolper, P., 1993. Temporal Deductive Databases. Ch13 in [1].

[9] Tuzhilin, A., 1992. SimTL: A Simulation Language Based on Temporal Logic. *Transactions of the Society for Computer Simulation*, 9(2), pp. 87-100.

[10] Kroger, F., 1987. *Temporal Logic of Programs*, volume 8. Springer-Verlag.

[11] Padmanabhan, B., and Tuzhilin, A., 1996. Using Temporal Logic to Find Patterns in Temporal Databases. Working Paper #IS-96-2, Dept. of Info. Systems, NYU.

[12] Koymans, R.,1990. Specifying Real-Time Properties with Metric Temporal Logic. *J. of Real-Time Systems*, v2.

[13] Snodgrass, R.T., 1995. *The TSQL2 Temporal Query Language*. Kluwer.

[14] Chomicki, J., and Imielinski, T., 1988. Temporal Deductive Databases and Infinite Objects. *Proc of PODS.*

[15] Abiteboul, S., and Vianu, V., 1991. Datalog Extensions for Database Queries and Updates. *J. of Computer and System Sciences*, vol. 43, pp. 62-124.

[16] Agrawal, R., Mannila, H., Srikant, R., Toivonen, H. and Verkamo,A.I., 1995. Fast Discovery of Association Rules. In Fayyad, U.M., Piatetsky-Shapiro, G., Smyth, P., and Uthurusamy, R. eds., *Advances in Knowledge Discovery and Data Mining.* AAAI Press.