Mining Temporal Relationships with Multiple Granularities in Time Sequences*

Claudio Bettini
DSI Department, University of Milano
via Comelico 39
20135 Milano, Italy
bettini@dsi.unimi.it

X. Sean Wang and Sushil Jajodia
Department of Information and Software Systems Engineering
George Mason University
Fairfax, VA 22030
{xywang,jajodia}@isse.gmu.edu

1 Introduction

Discovering sequential relationships in a time sequence is often important to many application domains, including financial, manufacturing, etc. Indeed, a lot of work has been done in this area (see [1, 6] for an overview). An important aspect for such a discovery process is, however, largely missing from the literature: namely discovering temporal patterns or relationships that involve multiple time granularities. This paper reports the progress in this front. A more detailed study can be found in [4].

In this paper, we focus on algorithms for discovering sequential relationships when a rough pattern of relationships is given. The rough pattern (which we term "event structure") specifies what sort of relationships a user is interested in. For example, a user may be interested in "which pairs of events occur frequently one week after another". The algorithms will find the instances that fit the event structure. They can also be used when more information is filled in the structure, as in in the following example from the stock market domain: "what kind of events frequently follow within 5 trading-days from the occurrence of a rise in IBM stocks, when this is preceded, one week before, by the fall of another hi-tech company stock". These algorithms form a core component for a data mining environment.

We view the actual data mining process as being done interactively through a user interface. Data mining requests (in terms of event structures) are issued through the user interface and processed using the algorithms. Initially, the user will issue a request with a simple structure. Complicated structures may be given by the user

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

*This work was supported by the National Science Foundation (NSF) under the grant IRI–9633541. The work of Wang and Jajodia was also supported by the NSF grants IRI–9409769 and INT–9412507, respectively. The work of Bettini was partially carried out while visiting George Mason University.

after s/he explores the data set using simpler ones. That is, event structures "evolve" from simpler ones to more complex ones. Our algorithms are designed, however, to handle complicated as well as simple event structures.

The remainder of this paper is organized as follows. In Section 2, we precisely define what we mean by "mining temporal relationships" via the concepts of event structures and their solutions. In Section 3, we present an algorithm for finding frequent events that fit the given event structures. This algorithm may be too time consuming when implemented directly. In Section 4, we provide a powerful heuristic that dramatically reduces the computation time. The heuristic takes advantage of our theoretical study of the event structures. We conclude with Section 5.

2 The event-mining problem

A number of notions must be introduced to formalize the mining problem. We assume that there is a finite set of *event types*. Examples of event types are "deposit to an account" or "price increase of a specific stock". We use the symbol E, possibly with subscripts, to denote event types. An *event* is a pair $e = (E, t)$, where E is an event type and t is a positive integer, called the *timestamp* of e. An *event sequence* is a finite set of events. Intuitively, each event (E, t) appearing in an event sequence σ represents the occurrence of event type E at time t.

To model the temporal relationships among events in a sequence, we first specify what we mean by a time granularity, following [8, 5], and then introduce the notion of a temporal constraint with granularity.

We assume an underlying time domain $(\mathcal{T}, \leq)$ where $\mathcal{T}$ is an infinite set of time instants and $\leq$ a total order. A *granularity* is a mapping μ from the set of the positive integers (used as index) to subsets of the time domain such that for all positive integers i and j with $i < j$: (1) $\mu(i) \neq \emptyset \wedge \mu(j) \neq \emptyset$ implies that each number in $\mu(i)$ is less than all the numbers in $\mu(j)$, and (2) $\mu(i) = \emptyset$ implies $\mu(j) = \emptyset$.

Each set $\mu(i)$, if non-empty, is called a *granule* of μ. Property (1) says that granules do not overlap and that the order on indexes follows the order on the corresponding granules. Property (2) says that the subset of the index corresponding to the granules forms a set of contiguous integers. Intuitive granularities, e.g., `day`, `month`, `week` and `year`, satisfy the above definition. For example, we can define a special granularity `year` starting from year 1800 as follows: `year`(1) is the subset of the time domain (an interval of reals) corresponding to the year 1800, `year`(2) is the subset corresponding to the year 1801, and so on. The above definition allows to model also more complex granularities like `leap-year`, `business-week`, or `business-day` (in the following abbreviated in `b-day`). In this paper, all timestamps in an event sequence are assumed to be in terms of a fixed granularity, and in particular we assume that granularity to be `second`.

We can now define a temporal constraint with granularity.

Definition 1: Let m and n be non-negative integers with $m \leq n$ and μ be a granularity. A *temporal constraint with granularity* (TCG) $[m, n]\,\mu$ is the binary relation on positive integers defined as follows: For positive integers t_1 and t_2, (t_1, t_2) satisfies $[m, n]\,\mu$ iff (1) $t_1 \leq t_2$, (2) $\lceil t_1 \rceil^\mu$ and $\lceil t_2 \rceil^\mu$ are both defined, and (3) $m \leq (\lceil t_2 \rceil^\mu - \lceil t_1 \rceil^\mu) \leq n$.

In the above, $\lceil k \rceil^\mu$ denotes the index i of the granule of μ which contains the k^{th} second, if it exists, otherwise denotes undefined. Note that a set of these constraints cannot be translated into an equivalent set of constraints in terms of the same granularity [5]. For example, we should *not* translate $[0, 0]$ `day` into $[0, 23]$ `hour` since 23 hours can overlap across two consecutive days.

In the following we say that *a pair of events satisfies a constraint* if the corresponding timestamps do. It is easily seen that the pair of events (e_1, e_2) satisfies TCG $[0, 0]$ `day` if events e_1 and e_2 happen within the same day but e_2 does not happen earlier than e_1. Similarly, (e_1, e_2) satisfies TCG $[0, 2]$ `hours` if e_2 happens either in the same second as e_1 or within two hours after e_1. Finally, (e_1, e_2) satisfies $[1, 1]$ `month` if e_2 occurs in the month immediately after that in which e_1 occurs.

Hence, TCGs are used as temporal binary constraints on events and can be combined in a complex temporal relationship among several events, each being assigned to a different variable.

Definition 2: An *event structure (with granularities)* is a rooted directed acyclic graph (W, A, Γ), where W is a finite set of event variables, $A \subseteq W \times W$ and Γ is a mapping from A to the finite sets of TCGs.

The set of TCGs assigned to an edge is taken as conjunction. Figure 1 shows an event structure.

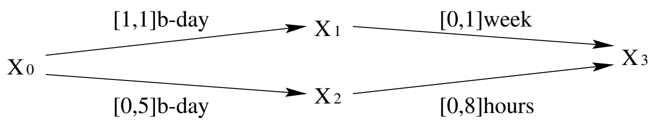

Figure 1: An event structure.

Given an event structure $\mathcal{S} = (W, A, \Gamma)$, we call *complex event type derived from* $\mathcal{S}$ the structure $\mathcal{S}$ with each variable associated with a specific event type, and we call *complex event matching* $\mathcal{S}$ the structure $\mathcal{S}$ with each variable associated with a distinct event such that the event timestamps satisfy the time constraints in Γ.

Example 1: Assume an event sequence that records stock-price fluctuations (rise and fall) every 15 minutes (this sequence can be derived from the sequence of stock prices) as well as the time of the releases of company earnings reports. Consider the event structure depicted in Figure 1. If we assign the event types for X_0, X_1, X_2 and X_3 to be `IBM-rise`, `IBM-earnings-report`, `HP-rise`, and `IBM-fall`, respectively, we have a complex event type. This complex event type describes that the IBM earnings were reported one business day after the IBM stock rose, and in the same or the next week the IBM stock fell; while the HP stock rose within 5 business days after the same rise of the IBM stock and within 8 hours before the same fall of the IBM stock.

If $\mathcal{T}$ is a complex event type derived from an event structure having k variables, and σ' is a subset of an event sequence σ such that $|\sigma'| = k$, then σ' is said to be an *occurrence* of $\mathcal{T}$ if a complex event matching that event structure can be derived by assigning a distinct event in σ' to each variable so that the type of the event is the same as the type assigned to the same variable by $\mathcal{T}$.

We are now ready to formally define the mining problem.

Definition 3: An *event-mining problem* is a quadruple $(\mathcal{S}, \gamma, E_0, \rho)$, where $\mathcal{S}$ is an event structure, γ (the *minimum confidence value*) a real number between 0 and 1 inclusive, E_0 (the *reference type*) an event type, and ρ is a partial mapping which assigns a set of event types to some of the variables (except the root).

An event-mining problem $(\mathcal{S}, \gamma, E_0, \rho)$ is the problem of finding all complex event types $\mathcal{T}$ such that each $\mathcal{T}$ (i) occurs frequently in the input sequence and (ii) is derived from $\mathcal{S}$ by assigning E_0 to the root and a specific event type to each of the other variables. (The assignments in (ii) must respect the restriction stated in ρ.) The frequency is calculated against the number of occurrences of E_0. This is intuitively sound: If we want to say that event type E frequently happens one day after IBM stock falls, then we need to use the events corresponding to falls of IBM stock as a reference to count the frequency of E. We are not interested in an "absolute" frequency, but only in frequency relative to some event type.

The *solution* of an event-mining problem $(\mathcal{S}, \gamma, E_0, \rho)$ on a given event sequence σ is the set of all complex event types derived from $\mathcal{S}$, with the following conditions: (i) E_0 is associated with the root of $\mathcal{S}$ and each event type assigned to a non-root variable X belongs to $\rho(X)$ if $\rho(X)$ is defined, and (ii) each complex event type occurs in σ with a *frequency* greater than γ.[1]

[1] The frequency here is defined as the number of times the complex event type occurs for a different occurrence of E_0 (i.e., all the occurrences using the same occurrence of E_0 for the root are counted as one) divided by the number of times E_0 occurs.

Example 2: $(\mathcal{S}, 0.8, \texttt{IBM-rise}, \rho)$ is a mining problem, where $\mathcal{S}$ is the structure in Figure 1 and ρ assigns X_3 to IBM-fall and assigns all other variables to all the possible event types. Intuitively, we want to discover what happens between a rise and fall of IBM stocks, looking at particular windows of time. The complex event type described in Example 1 where X_1 and X_2 are assigned respectively to IBM-earnings-report and HP-rise will belong to the solution of this problem if it occurs in the input sequence with a frequency greater than 0.8 with respect to the occurrences of IBM-rise.

3 Finding frequent complex event types

In [4] we introduced a variation of the timed automaton [3] that we call a *timed automaton with granularities* (TAG). Through the use of TAGs, we can find occurrences of a complex event type in an event sequence. This ability forms the basis for finding a solution to an event-mining problem.

A TAG is essentially an automaton that recognizes words. However, there is a timing information associated with the symbols of the words signifying the time when the symbol arrives at the automaton. When a timed automaton makes a transition, the choice of the next state depends not only on the input symbol read, but also on values in the clocks which are maintained by the automaton and each of which is "ticking" in terms of a specific time granularity. A clock can be set to zero by any transition and, at any instant, the reading of the clock equals the time (in terms of the granularity of the clock) that has elapsed since the last time it was reset. A constraint on the clock values is associated with any transition, so that the transition can occur only if the current values of the clocks satisfy the constraint. It is then possible to constrain, for example, that a transition fires only if the current value of a clock, say in terms of week, reveals that the current time is in the next week with respect to the previous value of the clock. We do not report here the formal definition of TAGs and related notions, which can be found in [4]. An example of TAG is reported in Figure 2.

To achieve our goal, we must be able, given a complex event type $\mathcal{T}$, to derive a corresponding TAG. This is guaranteed by the following theorem ([4]).

Theorem 4: Given a complex event type $\mathcal{T}$, there exists a timed automaton with granularities $\text{TAG}_{\mathcal{T}}$ such that $\mathcal{T}$ occurs in an event sequence σ iff $\text{TAG}_{\mathcal{T}}$ has an accepting run over σ. This automaton can be constructed by a polynomial-time algorithm.

The technique we use to derive the TAG corresponding to a complex event type derived from $\mathcal{S}$ is based on a decomposition of $\mathcal{S}$ into chains from the root to terminal nodes. For each chain we build a simple TAG where each transition has as input symbol the variable corresponding to a node in $\mathcal{S}$ (starting from the root), and clock constraints for the same transition correspond to the TCGs associated with the edge leading to that node. Then, we combine the resulting TAGs into a single TAG using a "cross-product" technique and we add transitions to allow the skipping of events. Finally, we change each input symbol X with the corresponding event type. A detailed procedure for TAG generation can be found in [4]. Figure 2 shows the TAG corresponding to the complex event type in Example 1.

We have also found a bound on the time needed to determine whether an event sequence is accepted by a TAG corresponding to a certain complex event type. This time is $O(|\sigma| * (|S| * min(|\sigma|, (|V| * K)^p))^2)$ time, where $|S|$ is the number of states in the TAG, $|\sigma|$ is the number of events in the input sequence, $|V|$ is the number of variables in the longest chain used in the construction of the automata, K is the size of the maximum range appearing in the constraints, and p is the number of chains used in the construction of the automata. Note that, even for reasonably complex event structures, the constant p is very small; hence, $(|V| * K)^p$ is often much smaller than $|\sigma|$.

Since we now have a tool to check occurrences of complex event types in a time sequence, a naive algorithm for discovering frequent complex event types can proceed as follows: Consider all the event types that occur in the given event sequence, and consider all the complex types derived from the given event structure, one from each

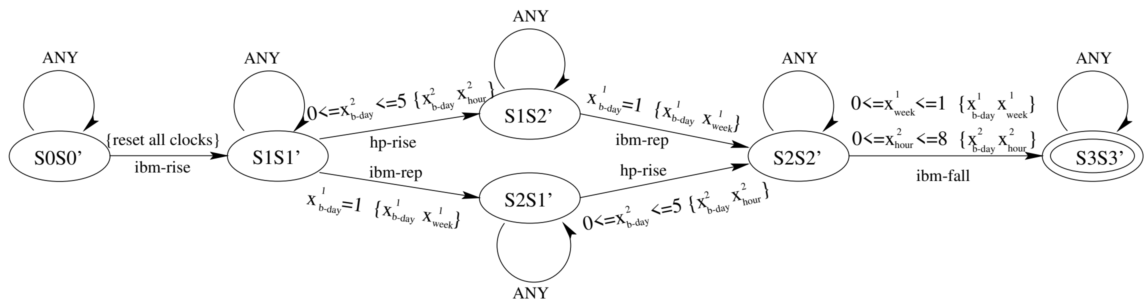

Figure 2: An example of Timed Automaton with Granularities

assignment of these event types to the variables. Each of these complex types is called a *candidate complex type* for the event-mining problem. For each candidate complex type, start the corresponding TAG at every occurrence of E_0. That is, for each occurrence of E_0 in the event sequence, use the rest of the event sequence (starting from the position where E_0 occurs) as the input to one copy of the TAG. By counting the number of TAGs reaching a final state, versus the number of occurrences of E_0, all the solutions of the event-mining problem will be derived.

This naive algorithm, however, can be too costly to implement. Indeed, the number of candidate types to be checked is exponential in the number of event types occurring in the event sequence and in $\rho(X)$ for all X, with the exponent being the number of non-root variables in the event structure. In the worst case, this implies an exponential number of TAGs, each one to be run for each occurrence of the reference event type in the sequence.

4 Achieving effectiveness by temporal reasoning and decomposition

Our strategy for finding the solutions of event-mining problems relies on the many optimization opportunities provided by the temporal constraints of the event structures. These optimizations consist in (1) identifying the possible inconsistencies in the given event structure before starting the process, (2) reducing the length of the sequence, (3) the number of times an automaton has to be started, and (4) the number of different automata to be started. The naive algorithm illustrated earlier is applied only after these steps.

Steps (1–4) exploit the implicit temporal relationships in the given event structure and a *decomposition* strategy, based on the observation that if an event-mining problem has a solution, then part of this solution is a solution also for the corresponding "sub-problem".

To derive implicit relationships, we must be able to convert TCGs from one granularity to another, not necessarily obtaining equivalent constraints, but *logically implied* ones. However, for an arbitrarily given TCG$_1$ and a granularity μ, it is not always possible to find a TCG$_2$ in terms of μ such that it is logically implied by TCG$_1$, i.e., any pair of events satisfying TCG$_1$ also satisfy TCG$_2$. For example, $[m, n]$ `b-day` is not implied by $[0, 0]$ `day` no matter what m and n are. The reason is that $[0, 0]$`day` is satisfied by any two events that happen during the same day, whether the day is a business day or a weekend day.

In our framework, we allow a conversion of a TCG in an event structure into another TCG if the resulting constraint is implied by the set of all the TCGs in the event structure.

In [5, 4] the interested reader can find approximate polynomial algorithms for conversion of TCGs and for the derivation of implicit TCGs in the event structure. Since there are possibly infinite implicit TCGs, the algorithm's goal is to approximates the *tightest* ones. Referring to the event structure of Figure 1 the algorithm would derive, among others, TCG $[0, 127]$ `hour` between X_1 and X_3, and TCG $[0, 1]$ `week` between X_0 and X_3. Consider the last one: this is the tightest in terms of `week` between those two variables, since no solution to the temporal constraints can lead to a distance less than 0, nor greater than 1.

When an unsatisfiable TCG is derived from an event structure by the algorithm, we can avoid starting the mining process, since this means that an inconsistent structure specification has been given. Implicit TCGs are also the basis for steps 2-4. In this paper we limit the discussion to step 4 which is probably the most effective of these optimization techniques. We refer to [4] for a detailed description of 2 and 3.

The basic idea of step 4 is as follows: If a complex event type occurs frequently, then any of its sub-type should also occur frequently. (This is similar to [7].) Here by a *sub-type* of a complex type $\mathcal{T}$, we mean a complex event type, induced by a subset of variables, such that each occurrence of the sub-type can be "extended" to an occurrence of $\mathcal{T}$. However, not every subset of variables of a structure can induce a sub-structure. For example, consider the event structure in Figure 1 and let $\mathcal{S}' = (\{X_0, X_3\}, \{(X_0, X_3)\}, \Gamma')$. $\mathcal{S}'$ cannot be an induced sub-structure, since it is not possible for Γ' to capture precisely the four constraints of that structure. This forces us to consider approximated sub-structures.

Let $\mathcal{S} = (W, A, \Gamma)$ be an event structure and M the set of all the granularities appearing in Γ. For each $\mu \in M$, let C_μ be the collection of constraints that we derive at the end of the approximate algorithm for the derivation of implicit constraints. Then, for each subset W' of W, the *induced approximated sub-structure of* W' is (W', A', Γ'), where A' consists of all pairs $(X, Y) \subseteq W' \times W'$ such that there is a path from X to Y in $\mathcal{S}$ and there is at least a constraint (original or derived) on (X, Y). For each $(X, Y) \in A'$, the set $\Gamma'(X, Y)$ contains all the constraints in C_μ on (X, Y) for all $\mu \in M$. For example, $\Gamma'(X_0, X_3)$ in the previous paragraph contains $[0, 1]\,\texttt{week}$ and $[1, 175]\,\texttt{hour}$. Note that if a complex event matches $\mathcal{S}$ using events from σ, then there exists a complex event using events from a sub-sequence σ' of σ that matches the sub-structure $\mathcal{S}'$.

By using the notion of an approximated sub-structure, we proceed to reduce candidate event types as follows: Suppose the event-mining problem is $(\mathcal{S}, \gamma, E_0, \rho)$. For each variable X appearing in $\mathcal{S}$, except the root X_0, consider the approximated sub-structure $\mathcal{S}'$ induced from X_0 and X (i.e., two variables). If there is a relationship between X_0 and X (i.e., $\Gamma'(X_0, X) \neq \emptyset$), consider the event-mining problem (called *induced mining problem*) $(\mathcal{S}', \gamma, E_0, \rho')$, where ρ' is a restriction of ρ with respect to the variables in $\mathcal{S}'$. The key observation is ([7]) that if no solution to any of these induced mining problems assigns event type E to X, then there is no need to consider any candidate complex type that assigns E to X. This reduces the number of candidate event types for the original mining problem.

To find the solutions to the induced mining problems is rather straightforward and simple in time complexity. Indeed, the induced sub-structure gives the distance from the root to the variable (in effect, two distances, namely the minimum distance and the maximum distance). For each occurrence of E_0, this distance translates into a window, i.e., a period of time during which the event for X must appear. If the frequency (i.e., the number of windows in which the event occurs divided by the total number of these windows) an event type E occurs is less than or equal to γ, then any candidate complex type with X assigned to E can be "screened out" for further consideration. Consider the mining problem of Example 2 with the simple variation that $\rho = \emptyset$, i.e. all non-root variables are free. $(\mathcal{S}', 0.8, \texttt{IBM-rise}, \emptyset)$ is one of its induced mining problems. $\Gamma'(X_0, X_3)$, through the constraints reported above, identifies a window for X_3 for each occurrence of `IBM-rise`. It is easy to screen out all candidate event types for X_3 that have a frequency of occurrence in these windows less than 0.8.

The above idea can easily be extended to consider induced approximated sub-structures that include more than one non-root variable. For each integer $k = 2, 3, \ldots$, consider all the approximated sub-structures $\mathcal{S}_k$ induced from the root variable and k other variables in $\mathcal{S}$, where these variables (including the root) form a sub-chain in $\mathcal{S}$ (i.e., they are all on a particular path from the root to a particular leaf), and $\mathcal{S}_k$, considering the derived constraints, forms a connected graph. We now find the solutions to the induced event-mining problem $(\mathcal{S}_k, \gamma, E_0, \rho_k)$. Again, if no solution assigns an event type E to a variable X, then any candidate complex type that has this assignment is screened out. To find the solutions to these induced mining problems, the naive algorithm mentioned earlier can be used. Of course, any screened-out candidates from previous induced mining problems should not be considered any further. This means that if in a previous step only k event types have been assigned to variable X as a solution of a mining problem, if the current problem involves variable X, we consider only candidates within those k event types. This process can be extended to event types assigned to combinations of variables. This process results, in practice, in a smaller number of candidate types for induced mining problems.

5 Conclusion

In this paper, we briefly discussed the process of mining sequential relationships that involve multiple granularities. We first precisely formulated the problem and then developed an effective procedure. We have conducted several experiments (see [4]) which show that the heuristics we introduce are especially powerful in reducing the number of candidate event types.

References

[1] R. Agrawal, T. Imielinski, and A. Swami. Database mining: A performance perspective. *IEEE Trans. Knowledge and Data Engineering*, 5(5):914–925, 1990.

[2] R. Agrawal and R. Srikant. Mining sequential patterns. In *International Conference on Database Engineering*, pp. 3–14. IEEE, 1995.

[3] R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.

[4] C. Bettini, X. S. Wang, S. Jajodia, and J. Lin. Discovering Frequent Event Patterns With Multiple Granularities in Time Sequences. *IEEE Trans. Knowledge and Data Engineering*, to appear.

[5] C. Bettini, X. Wang, and S. Jajodia. A General Framework for Time Granularity and its Application to Temporal Reasoning. *Annals of Mathematics and Artificial Intelligence*, Baltzer Science Publishers, 1998.

[6] P. Laird. Identifying and using patterns in sequential data. In *Algorithmic Learning theory, 4th International Workshop*, pp. 1–18. Springer-Verlag, 1993.

[7] H. Mannila, H. Toivonen, and A.I. Verkamo. Discovering frequent episodes in sequences (extended abstract). In *Proc. 1st Conference on Knowledge Discovery and Data Mining*, AAAI Press, Menlo Park, pp. 210-215, 1995.

[8] X. Wang, C. Bettini, A. Brodsky, and S. Jajodia. Logical design for temporal databases with multiple granularities. *ACM Transactions of Database Systems*, 22(2):115–170, June 1997.

[9] J. Tsong-Li Wang, G. Chirn, T.G. Marr, B.A. Shapiro, D. Shasha, and K. Zhang. Combinatorial pattern discovery for scientific data: Some preliminary results. In *Proc. of SIGMOD Conference*, pages 115–125. ACM Press, May 1994.

[10] K. Wang and J. Tan. Incremental discovery of sequential patterns. In *Proc. of Workshop on Research Issues on Data Mining and Knowledge Discovery, in cooperation with ACM-SIGMOD'96 and IRIS/Precarn*, Montreal, Canada, June 1996.