Discovering Temporal Relationships with Multiple Granularities in Time Sequences*

Technical Report: ISSE-TR-96-06

Claudio Bettini† X. Sean Wang‡ Sushil Jajodia‡ Jia-Ling Lin‡

ABSTRACT

An important usage of time sequences is to discover temporal patterns. The discovery process usually starts with a user-specified skeleton, called an *event structure*, which consists of a number of variables representing events and temporal constraints among these variables; and the goal of the discovery is to find temporal patterns, i.e., instantiations of the variables in the structure, which frequently appear in the time sequence. This paper introduces event structures that have temporal constraints with multiple granularities, defines the pattern discovery problem with these structures, and studies effective algorithms to solve it. The basic components of the algorithms include timed automata with granularities (TAGs) and a number of heuristics. The TAGs are for testing whether a specific temporal pattern, called a *candidate complex event type*, appears frequently in a time sequence. Since there are often a huge number of candidate event types for a usual event structure, heuristics are presented aiming at reducing the number of candidate event types and reducing the time spent by the TAGs to test whether a candidate type does appear frequently in the sequence. These heuristics exploit the information provided by explicit and implicit temporal constraints with granularity in the given event structure. The paper also gives the results of an experiment to show the effectiveness of the heuristics on a real data set.

*A preliminary version of this paper will appear in the 1996 Proceedings of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems.

†Dept. of CS (DSI), University of Milan, via Comelico 39, 20135 Milan, Italy. `bettini@dsi.unimi.it`

‡Dept. of Info.& Software Systems Eng., George Mason University, Fairfax, VA 22030. {`xywang, jajodia, jllin`}`@isse.gmu.edu`

Identifying Key Words:

Data mining, knowledge discovery, time sequences, temporal databases, time granularity, temporal constraints, temporal patterns.

Contact Address:

Prof. X. Sean Wang

ISSE Dept. 4A4, George Mason University,

4400 University Dr., Fairfax, VA 22030

Email: xywang@isse.gmu.edu

Phone: (703) 993-1662 Fax: (703) 993-1638

Contents

List of Figures

1 Introduction

A huge amount of data is collected everyday in the form of event time sequences. Common examples are recording of different values of stock shares during a day, each access to a computer by an external network, bank transactions, or events related to malfunctions in an industrial plant. These sequences represent valuable sources of information not only to search for a particular value or event at a specific time, but also to analyze the frequency of certain events, discover their regularity, or discover set of events related by particular temporal relationships. These types of analyses can be very useful for deriving implicit information from the raw data, and for predicting the future behavior of the process that we are monitoring.

Although a lot of work has been done on identifying and using patterns in sequential data (see [AIS90, Lai93] for an overview), little attention has been paid to the discovery of temporal patterns or relationships that involve multiple granularities. We believe that these relationships are an important aspect of data mining. For example, while analyzing ATM transactions, we want to discover events having quantitative bounds on their distances such as events occurring in the same day, or events happening within k weeks from a specific one. The system should not automatically translate these bounds in terms of a basic granularity since it may change the semantics of the bounds. For example, one day should *not* be translated into 24 hours since 24 hours can overlap across two consecutive days.

In this paper, we focus our attention to providing a formal framework in which to express the data mining tasks involving time granularities and propose efficient algorithms. To this end, we introduce the notion of *event structures*. An event structure is essentially a set of temporal constraints on a set of variables representing events. Each constraint bounds the distance between a pair of events in terms of a time granularity. For example, we can constrain two events to occur in the same business day, but one occurring 4 to 6 hours after the other. We consider data mining tasks where an event structure is given and only some of its variables are instantiated. Possible instances for the other variables have to be discovered based on the frequency on which the corresponding events occur in the event sequence matching the structure.

As a simple example, one may be interested in finding all those events that frequently follow within 2 business days of a rise of the IBM stock price. For this, we set up two variables, X_0 and X_1, where X_0 is instantiated with the event type "rise of the IBM stock" while X_1 is left free. The constraint between X_0 and X_1 is that they have to happen with 2 business days. The discovery task is to find all instantiations of the variable X_1 such that the events assigned to X_1 frequently follow the rise of the IBM stock. Each such instantiation is called a *solution* to the discovery task.

In order to find all the solutions for a given event structure, we first consider the case that we call a *candidate* instantiation of the event structure, where each variable is instantiated to a specific event type. We scan through the time sequence to see if this candidate instantiation occurs frequently. For this pattern recognition problem, we introduce *timed finite automata with granularities* (TAGs) which are essentially standard finite automata with the modification that transitions are conditioned not only by input symbols, but also by the values of the associated clocks. (Clocks may be running in different granularities.)

To effectively perform data mining, however, we cannot naively consider all candidate instantiations, since the number of candidate instantiated structures that we must consider is exponential. We provide algorithms and heuristics that exploit the granularity system and the given constraints to reduce the hypothesis space for the pattern matching task. The global approach offers an effective procedure to discover patterns of events that occur frequently in a sequence satisfying specific temporal relationships.

In terms of related research, our work is closest to [MTV95], where event sequences are searched for frequent patterns of events. These patterns have a simple structure (essentially a partial order) whose total span of time is constrained by a *window* given by the user. The technique of generating candidate patterns from sub-patterns, together with a sliding window method, is shown to provide effective algorithms. In contrast, we consider more complex patterns where events may be in terms of different granularities and windows are given for arbitrary pairs of events in the pattern. The temporal constraints with granularities introduced in this paper are closely related to temporal constraint networks and related problems (e.g., consistency checking) that have been studied mostly in A.I. (cf. [DMP91]); however, these works assume that either constraints involve a single granularity or, if they involve multiple granularities, they are translated into constraints in single granularity before applying the algorithms. We introduce networks of constraints in terms of arbitrary granularities and a new algorithm to solve the related problems. Finally, the TAGs presented here are extensions of the timed automata introduced in [AD94] for modelling real-time systems and checking their specifications. We extend the automata to ones which have clocks "ticking" in different granularities in order to use them to find complex events that have temporal relationships involving multiple granularities.

In Section 2, we begin with a definition of temporal types that formalizes the intuitive notion of time granularities. We formalize the temporal pattern discovery problem in Section 3. In Section 4, we focus on algorithms for discovering patterns from event sequences; and in Section 5, we provide a number of heuristics for effective data mining. In Section 6 we report some results obtained through an experiment on a real data set. We conclude the paper in Section 7 with some discussion.

2 Preliminaries

In order to formally define temporal relationships that involve time granularities, in this section we review the notion of temporal types given in [WBBJ94]. By definition, a *temporal type* is a mapping μ from the set of the positive integers (the *time ticks*) to $2^{\mathcal{R}}$ (the set of absolute time sets[1]) such that for all positive integers i and j with $i < j$, the following two conditions are satisfied:

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 $\mu(j) = \emptyset$.

Property (1) is the *monotonicity* requirement. Property (2) disallows a certain tick of μ to be empty unless all subsequent ticks are empty. The set $\mu(i)$ of reals is said to be the *i-th tick of* μ, or *tick* i *of* μ, or simply *a tick of* μ.

Intuitive temporal types, e.g., `day`, `month`, `week` and `year`, satisfy the above definition. For example, we can define a special temporal type `year` starting from year 1800 as follows: `year`(1) is the set of absolute time (an interval of reals) corresponding to the year 1800, `year`(2) is the

[1]We use the symbol R to denote the real numbers. And we assume the underlying absolute time is continuous and modelled by the reals. However, the results of this paper still hold if the underlying time is assumed to be discrete.

set of absolute time corresponding to the year 1801, and so on. Note that this definition allows temporal types in which ticks are mapped to more than one continuous interval. For example, it is possible to have a temporal type representing business weeks (`b-week`), where a tick of `b-week` is the union of all business days (`b-day`) in a certain week (i.e., excluding all Saturdays, Sundays, and general holidays). See Figure 1.

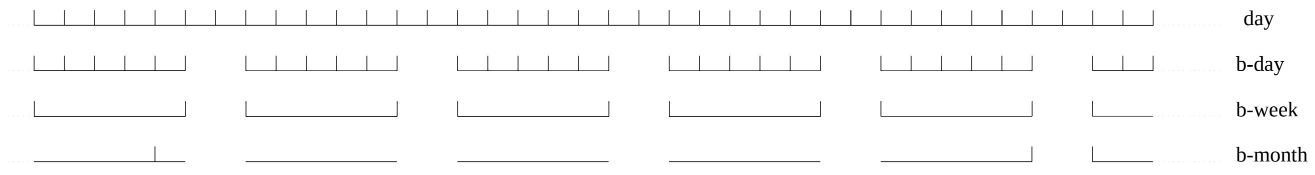

Figure 1: Three temporal types covering the span of time from February 26th till April 2nd 1996, with `day` as the absolute time.

This is a generalization of most previous definitions of temporal types.

When dealing with temporal types, we often need to determine the tick (if any) of a temporal type μ that covers a given tick z of another temporal type ν. For example, we may wish to find the month (an interval of the absolute time) that includes a given week (another interval of the absolute time). Formally, for each positive integer z and temporal types μ and ν, $\lceil z \rceil_{\nu}^{\mu}$ is undefined if $\nu(z) \not\subseteq \mu(z')$ for all z'; otherwise, $\lceil z \rceil_{\nu}^{\mu} = z'$, where z' is the unique positive integer such that $\nu(z) \subseteq \mu(z')$. The uniqueness of z' is guaranteed by the monotonicity of temporal types. As an example, $\lceil z \rceil_{\texttt{second}}^{\texttt{month}}$ gives the month that includes the second z. Note that while $\lceil z \rceil_{\texttt{second}}^{\texttt{month}}$ is always defined, $\lceil z \rceil_{\texttt{week}}^{\texttt{month}}$ is undefined if week z falls between two months. Similarly, $\lceil z \rceil_{\texttt{day}}^{\texttt{b-day}}$ is undefined if day z is a Saturday, Sunday, or a general holiday.

In this paper, all timestamps in an event sequence are assumed to be in terms of a fixed temporal type. In order to simplify the notation, throughout the paper we assume that each event sequence is in terms of `second`, and abbreviate $\lceil z \rceil_{\nu}^{\mu}$ as $\lceil z \rceil^{\mu}$ if $\nu = \texttt{seconds}$.

3 Formalization of the discovery problem

Throughout the paper, we assume a finite set of *event types*. Examples of event types include "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, called the *timestamp* of e, is a positive integer. An *event sequence* is a finite set of events $\{(E_1, t_1), \ldots, (E_n, t_n)\}$. Intuitively, each event (E, t) appearing in an event sequence σ means that event type E occurs at time t. If (E, t) appears in σ, then we will say that E *occurs* in σ. We often write an event sequence as a finite list $(E_1, t_1), \ldots, (E_n, t_n)$, where $t_i \leq t_{i+1}$ for each $i = 1, \ldots, n-1$.

3.1 Temporal constraints with granularities

To model the temporal relationships among events in a sequence, we introduce the notion of a temporal constraint with granularity.

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

Intuitively, for timestamps $t_1 \leq t_2$ (in terms of seconds), t_1 and t_2 satisfy $[m, n]\,\mu$ if there exist ticks $\mu(t'_1)$ and $\mu(t'_2)$ covering, respectively, the t_1-th and t_2-th seconds, and if the difference of the integers t'_1 and t'_2 is between m and n (inclusive).

In the following we say that *a pair of events satisfy a constraint* if the corresponding timestamps do. It is easily seen that the pair of events e_1 and e_2 satisfy TCG $[0, 0]\,\mathtt{day}$ 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 and e_2 satisfy TCG $[0, 2]\,\mathtt{hours}$ if e_2 happens either in the same second as e_1 or within two hours after e_1. Finally, e_1 and e_2 satisfy $[1, 1]\,\mathtt{month}$ if e_2 occurs in the next month after the month in which e_1 occurs.

3.2 Event structures with multiple granularities

We now introduce the notion of an event structure. We assume there is an infinite set of event variables denoted by X, possibly with subscripts, that range over events.

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

Intuitively, an event structure specifies a complex temporal relationship among a number of events, each being assigned to a different variable in W. The set of TCGs assigned to an edge is taken as conjunction. That is, for each TCG in the set assigned to the edge (X_i, X_j), the events assigned to X_i and X_j must satisfy the TCG. The requirement that the temporal relationship graph of an event structure be acyclic is to avoid contradictions, since the timestamps of a set of events must form a linear order. The requirement that there must be a root (i.e., there exists a variable X_0 in W such that for each variable X in W, there is a path from X_0 to X) in the graph is based on our interest in discovering the frequency of a pattern with respect to the occurrences of a specific event type (i.e., the event type that is assigned to the root). See Section 4.

Figure 2 shows an event structure.

$$\begin{tikzcd}
 & X_1 \arrow[dr, "{[0,1]\text{week}}"] & \\
X_0 \arrow[ur, "{[1,1]\text{b-day}}"] \arrow[dr, "{[0,5]\text{b-day}}"'] & & X_3 \\
 & X_2 \arrow[ur, "{[0,8]\text{hours}}"'] &
\end{tikzcd}$$

Figure 2: An event structure.

We define two additional concepts based on the concept of event structures: a *complex event type*, which is an event structure with the event variables instantiated with event types; and a *complex event*, which is a an occurrence of a complex event type. Formally:

Definition Let $\mathcal{S} = (W, A, ,)$ be an event structure with time granularities. Then a *complex event type derived from* $\mathcal{S}$ is a pair $(\mathcal{S}, \varphi)$, where φ is a mapping from W to the event types; a *complex event matching* $\mathcal{S}$ is a pair (σ, α), where σ is an event sequence, and α is a one-to-one correspondence from W to σ such that for each edge (X_i, X_j) in A, if $\alpha(X_i) = (E_i, t_i)$ and $\alpha(X_j) = (E_j, t_j)$, then t_i and t_j satisfy all TCGs in $, (X_i, X_j)$.

A complex event (σ, α) matching $\mathcal{S}$ is an occurrence of a complex event type $(\mathcal{S}, \varphi)$ if for each $X \in W$, $\varphi(X) = E$ implies $\alpha(X) = (E, t)$ for some t. Similar to the notion of an event type occurring in an event sequence, a complex event type $\mathcal{T}$ is said to *occur* in an event sequence σ' if there exists $\sigma \subseteq \sigma'$ such that (σ, α) is an occurrence of $\mathcal{T}$.

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 release of company earnings reports. Consider the event structure depicted in Figure 2. 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. □

3.3 The discovery problem

We are now ready to formally define the discovery problem.

Definition An *event-discovery problem* is a quadruple $(\mathcal{S}, \gamma, E_0, \rho)$, where

(1) $\mathcal{S}$ is an event structure,

(2) γ (the *minimum confidence value*) a real number between 0 and 1 inclusive,

(3) E_0 (the *reference type*) an event type, and

(4) ρ is a mapping which assigns a set of event types to each variable (except the root).

An event-discovery problem $(\mathcal{S}, \gamma, E_0, \rho)$ is to find each complex event type $\mathcal{T}$ that is derived from $\mathcal{S}$ and has E_0 assigned to the root and all other variables assigned with event types agreeing with ρ, such that $\mathcal{T}$ occurs frequently. The frequency here 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. Formally, we have

Definition The *solution* of an event-discovery problem $(\mathcal{S}, \gamma, E_0, \rho)$ on a given event sequence σ, in which E_0 occurs at least once, is the set of all complex event types $\mathcal{T} = (\mathcal{S}, \varphi)$, with the condition that $\varphi(X_0) = E_0$ where X_0 is the root of $\mathcal{S}$ and $\varphi(X) \in \rho(X)$ for all non-root variables X, each of which occurs in σ with a *frequency* greater than γ. The frequency here is defined as the number of times $\mathcal{T}$ occurs for a different occurrence of E_0 (i.e., all the occurrences of $\mathcal{T}$ which use 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 discovery problem, where $\mathcal{S}$ is the structure in Figure 2 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`. □

4 Discovering frequent complex events

In this section, we introduce timed finite automata with granularities (TAGs) for the purpose of finding whether a candidate instantiation of an event structure occurs frequently in an event sequence. TAGs form the basis for our discovery algorithm.

4.1 Timed finite automata with granularities (TAGs)

We now concern ourselves with finding occurrences of a complex event type in an event sequence. In order to do so, we define a variation of the timed automaton [AD94] that we call a *timed automaton with granularities* (TAG).

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 be taken 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.

Definition A *timed automaton with granularities* (TAG) is a 6-tuple $\mathcal{A} = (\Sigma, S, S_0, C, T, F)$, where (1) Σ is a finite set (of *input letters*), (2) S is a finite set (of *states*), (3) $S_0 \subseteq S$ is a set of *start states*, (4) C is a finite set (of *clocks*), each of which has an associated temporal type,[2] (5) $T \subseteq S \times S \times \Sigma \times 2^C \times \Phi(C)$ is a set of *transitions*, and (6) $F \subseteq S$ is a set of *accepting states*.

In (5), $\Phi(C)$ is the set of all the formulas called *clock constraints* defined recursively as follows: For each clock x_μ in C and non-negative integer k, $x_\mu \leq k$ and $k \leq x_\mu$ are formulas in $\Phi(C)$; and any boolean combination of formulas in $\Phi(C)$ is a formula in $\Phi(C)$.

A transition $\langle s, s', e, \lambda, \delta \rangle$ represents a transition from state s to state s' on input symbol e. The set $\lambda \subseteq C$ gives the clocks to be reset (i.e., restart the clock from time 0) with this transition, and δ is a clock constraint over C. Given a TAG $\mathcal{A}$ and an event sequence $\sigma = e_1, \ldots, e_n$, a *run of $\mathcal{A}$ over σ* is a finite sequence of the form

$$\langle s_0, v_0 \rangle \xrightarrow{e_1} \langle s_1, v_1 \rangle \xrightarrow{e_2} \cdots \langle s_{n-1}, v_{n-1} \rangle \xrightarrow{e_n} \langle s_n, v_n \rangle$$

where $s_i \in S$ and v_i is a set of pairs (x, t), with x being a clock in C and t a non-negative integer,[3] that satisfies the following two conditions: (1) (*Initiation*) $s_0 \in S_0$, and $v_0 = \{(x, 0) | x \in C\}$, i.e., all clock values are 0; and (2) (*Consecution*) for each $i \geq 1$, there is a transition in T of the form $\langle s_{i-1}, s_i, e_i, \lambda_i, \delta_i \rangle$ such that δ_i is satisfied by using, for clock x_μ, the value $t + \lceil t_i \rceil^\mu - \lceil t_{i-1} \rceil^\mu$, where (x_μ, t) is in v_{i-1} and t_i and t_{i-1} are the timestamps of e_i and e_{i-1}. For each clock x_μ, if x_μ is in λ_i, then $(x_\mu, 0)$ is in v_i; otherwise, $(x_\mu, t + \lceil t_i \rceil^\mu - \lceil t_{i-1} \rceil^\mu)$ is in v_i assuming (x_μ, t) is in v_{i-1}.

[2]The notation x_μ will be used to denote a clock x whose associated temporal type is μ.

[3]The purpose of v_i is to remember the current time value of each clock.

A run r is an *accepting run* if the last state of r is in the set F. An event sequence σ is *accepted* by a TAG $\mathcal{A}$ if there exists an accepting run of $\mathcal{A}$ over σ.

4.2 Generating TAGs from complex event types

Given a complex event type $\mathcal{T}$, it is possible to derive a corresponding TAG. Formally:

Theorem 1 Given a complex event type $\mathcal{T}$, there exists a timed automaton with granularities $\mathrm{TAG}_{\mathcal{T}}$ such that $\mathcal{T}$ occurs in an event sequence σ iff $\mathrm{TAG}_{\mathcal{T}}$ 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 $\mathcal{T} = (\mathcal{S}, \varphi)$ 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 use the information in φ to change each input symbol X with the event type symbol $\varphi(X)$.[4] A detailed procedure for TAG generation can be found in Appendix. Figure 3 shows the TAG corresponding to the complex event type in Example 1.

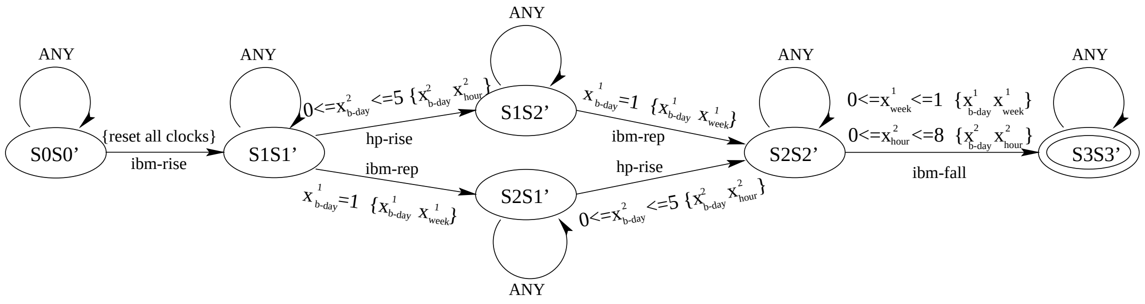
ANY
S0S0'
{reset all clocks}
ibm-rise
S1S1'
$0<=x^2_{b\text{-}day}<=5$ $\{x^2_{b\text{-}day}\ x^2_{hour}\}$
hp-rise
ibm-rep
$x^1_{b\text{-}day}=1$ $\{x^1_{b\text{-}day}\ x^1_{week}\}$
S1S2'
S2S1'
$x^1_{b\text{-}day}=1$ $\{x^1_{b\text{-}day}\ x^1_{week}\}$
ibm-rep
hp-rise
$0<=x^2_{b\text{-}day}<=5$ $\{x^2_{b\text{-}day}\ x^2_{hour}\}$
S2S2'
$0<=x^1_{week}<=1$ $\{x^1_{b\text{-}day}\ x^1_{week}\}$
$0<=x^2_{hour}<=8$ $\{x^2_{b\text{-}day}\ x^2_{hour}\}$
ibm-fall
S3S3'

Figure 3: An example of Timed Automaton with Granularities

[4]The construction would not work if we use the event types instead of the variable symbols from the beginning; indeed we exploit the property that the nodes of $\mathcal{S}$ are all differently labelled.

Theorem 2 Whether an event sequence is accepted by a TAG corresponding to a complex event type can be determined in $O(|\sigma| * (|S| * min(|\sigma|, (|V| * K)^p))^2)$ 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.

The proof basically follows a standard technique for pattern matching using a non-deterministic finite automaton (NDFA) (cf. [AHU74, page 328]). For each input symbol, a new set of states that are reached from the states of the previous step is recorded. (Initially, the set consists of all the start states.) Note however, clock values, in addition to the states, must be recorded. If the graph is just a chain, in the worst case, the number of clock values that we have to record for each state is the minimum between the length of the input sequence and the product of the number of variables in the chain and the maximum range appearing in the constraints. If the graph is not a chain we have to take into account the cross product of the p chains used in the construction of the TAG. Note that, even for reasonably complex event structures, the constant p is very small; hence, $(|V| * K)^p$ is often much smaller than $|\sigma|$.

4.3 A *naive* algorithm

Given the technical tools provided in the previous sections, a naive algorithm for discovering frequent complex events 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 assignment of these event types to the variables. Each of these complex types is called a *candidate complex type* for the event-discovery 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-discovery problem will be derived.

This naive algorithm, however, can be too costly to implement. Assume that the maximum number of event types occurring in the event sequence and in $\rho(X)$ for all X is n, and the number of non-root variables in the event structure is s. Then the time complexity of the algorithm is $O(n^s * |\sigma_{E_0}| * T_{tag})$, where $|\sigma_{E_0}|$ is the number of occurrences of E_0 in σ and T_{tag} is the time complexity of the pattern matching by TAGs. Clearly, if n and s are sufficiently large, the algorithm is rather ineffective.

5 Techniques for an effective discovery process

Our strategy of finding the solutions of event-discovery problems relies on the many optimization opportunities provided by the temporal constraints of the event structures. The strategy can be summarized in the following steps:

1. eliminate inconsistent event structures,
2. reduce the event sequence,
3. reduce the occurrences of the reference event type to be considered,
4. reduce the candidate complex event types, and
5. scan the event sequence, for each candidate complex event type, to find out if the frequency is greater than the minimum confidence value.

The naive algorithm illustrated earlier is applied in the last step (step 5). Several techniques are used in the previous steps to immediately stop the process if an inconsistent event structure is given (1), to reduce the length of the sequence (2), the number of times an automaton has to be started (3), and the number of different automata (4). Although the worst case complexity is the same as the naive one, in practice, the reduction produced by steps 1–4 makes the mining process effective.

While the technical tool used for step 5 is the TAG introduced in Subsection 4.1, steps (1–4) exploit the implicit temporal relationships in the given event structure and a *decomposition* strategy, based on the observation that if a discovery problem has a solution, then part of this solution is a solution also for a "sub-problem" of the considered one.

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.

Definition A TCG $[m', n']\,\nu$ is logically implied by a TCG $[m, n]\,\mu$ if each pair (x, y) satisfying the second constraint, satisfies also the first one.

We allow any conversion from a TCG constraint into one in terms of a different granularity such that the resulting constraint is implied by the starting one. More generally, any conversion should be allowed if the resulting constraint is implied by the global set of constraints in the event structure, i.e., if the derived constraint is between X and Y, then any pair of values (x, y) for X and Y, belonging to a solution of the whole set of constraints must also satisfy the derived constraint. For example, given a single TCG $[1, 2]$ `b-week`, we can convert it into $[3, 18]$ `day` and $[0, 1]$ `month`, while we cannot convert it into $[2, 3]$ `week-end` or $[1, 3]$ `week`, since the resulting constraints are not implied by $[1, 2]$ `b-week`.

In Appendix A we report an algorithm to derive implicit constraints from a given set of TCGs. The algorithm is based on a procedure to perform *allowed* conversions among TCGs with different granularities and on a reasoning process called *constraint propagation* to derive implicit relationships among constraints in the same granularity.

5.1 Recognition of inconsistent event structures

For a given event structure $\mathcal{S} = (W, A, ,)$, it is of practical interest to check if the structure is consistent, i.e., if there exists an complex event that matches $\mathcal{S}$. Indeed, if an event structure is inconsistent, it should be discarded even before the data mining process starts. Unfortunately, however, determining the consistency of event structures turns out to be a difficult problem:

Theorem 3 It is NP-hard to decide if an arbitrary event structure is consistent.

This result suggests to use approximated polynomial algorithms to check consistency of event structures. The algorithm to derive implicit constraints reported in Appendix A is actually one of these approximated algorithms. Indeed, if one of the constraints implied by the given ones and derived by the algorithm is the "empty" one (unsatisfiable, independently of a given event sequence), the whole event structure is inconsistent. We also use additional tests to recognize some cases of inconsistency not recognized by the algorithm.

5.2 Reduction of the event sequence

Regarding Step 2, we give a general rule to reduce the length of the input event sequence by exploiting the granularities. For example, consider the event structure depicted in Figure 2. If a discovery problem is defined on the sub-structure including only variables X_0, X_1, and X_2, the input event sequence can be reduced discarding any event that does not occur in a business-day.

In general, let μ the coarsest temporal type such that for each temporal type ν in the constraints and timestamp in the sequence z, if $\lceil z \rceil^{\nu}$ is defined, then $\lceil z \rceil^{\mu}$ must also be defined, and $\mu(\lceil z \rceil^{\mu}) \subseteq \nu(\lceil z \rceil^{\nu})$. Any event in the sequence whose time occurrence is not included in any tick of μ can be discarded before starting the mining process.

5.3 Reduction of the occurrences of the reference type

Regarding step 3, we give a general rule to determine which of the occurrences of the reference type cannot be the root of a complex event matching the given structure. This is done using the given constraints and the implied ones derived by the conversion and propagation algorithm.

Referring to our example, if no event occurs in the sequence in the next business-day of an `IBM-rise` event, this particular reference event can be discarded (no automata is started for it).

In general, we proceed as follows: If X_0 is the root, consider all the non-empty sets of explicit and implicit constraints on (X_0, X_i), for each $X_i \in W$. Since the constraints are in terms of granularities, for some occurrences of E_0 in the sequence, it is possible that a constraint is unsatisfiable. Referring to our example, if no event occurs in the sequence in the next business-day of an `IBM-rise` event, this particular reference event can be discarded (no automata is started for it). Let N be the number of occurrences of the reference event type in the sequence. Count the occurrences of reference events (instances of X_0) for which one of the constraints is unsatisfiable. These are reference events that are certainly not the root of a complex event satisfying the given event structure. If these occurrences are N' with $N'/N > 1-\gamma$, there cannot be any frequent complex event type satisfying the given event structure and the empty set should be returned to the user. Otherwise ($N'/N \leq 1-\gamma$), we remove these occurrences of E_0 and modify γ into $\gamma' = (\gamma * N)/(N - N')$. γ' is the confidence value required on the new event sequence to have the same solution as for the original confidence value on the original sequence.

5.4 Reduction of the candidate complex event types

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 [MTV95].) 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 complex event that is an 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 2 and let $\mathcal{S}' = (\{X_0, X_3\}, \{(X_0, X_3)\}, , ')$. $\mathcal{S}'$ cannot be an induced sub-structure, since it is not possible for $, '$ to capture precisely the four constraints of that structure. This forces us to consider approximated sub-structures.

Let $\mathcal{S} = (W, A, ,)$ be an event structure and M the set of all the temporal types appearing in , . For each $\mu \in M$, let C_μ be the collection of constraints that we derive at the end of the approximate conversion and propagation algorithm of Appendix A. Then, for each subset W' of W, the *induced approximated sub-structure of* W' is $(W', A', , ')$, 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 $, '(X, Y)$ contains all the constraints in C_μ on (X, Y) for all $\mu \in M$. For example, $, '(X_0, X_3)$ in the previous paragraph contains $[0, 1]$ `week` and $[1, 175]$ `hour`. Note that if a complex event (σ, α) matches $\mathcal{S}$, then there exists a complex event (σ', α') that matches $\mathcal{S}'$, where σ' is a sub-sequence of σ such that $\alpha(X) \in \sigma'$ for each $X \in W'$ and α' is a restriction of α on W'.

By using the notion of an approximated sub-structure, we proceed to reduce candidate event types as follows: Suppose the event-discovery 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., $, '(X_0, X) \neq \emptyset$), consider the event-discovery problem (called *induced discovery problem*) $(\mathcal{S}', \gamma, E_0, \rho')$, where ρ' is a restriction of ρ wrt the variables in $\mathcal{S}'$. The key observation is ([MTV95]) that if no solution to any of these induced discovery 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 discovery problem.

To find the solutions to the induced discovery 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 with E can be "screened out" for further consideration. Consider the discovery 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 discovery 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-discovery 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 discovery problems, the naive algorithm mentioned earlier can be used. Of course, any screened-out candidates from previous induced discovery 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 discovery 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 results, in practice, in a smaller number of candidate types for induced discovery problems.

6 Experimental Results

In this section, we report the experimental results conducted on a real data set. The purpose is to test the heuristics provided in the previous sections. The interpretation and discussion of the significance (or insignificance) of the discovered patterns are out of the scope of this paper.

The data set we gathered is the closing prices of 439 stocks for 517 trading days during the period between January 3, 1994 to January 11, 1996.[5] For each of the 439 trading companies in the data set, we calculate the price change percentages on consecutive trading days (business days). The change percentage for day d is calculated by the formula $(p_d - p_{d-1})/p_{d-1}$, where p_d is the closing price of day d and p_{d-1} is the closing price of the previous trading day.

The price changes are partitioned into 7 categories: (-∞, -5%], (-5%, -3%], (-3%, 0%), [0%, 0%], (0%, 3%), [3%, 5%), and [5%, ∞). We consider each event type as characterizing a specific category of price percentage change for a specific company. Since not all companies started trading on January 3, 1994 and stopped on January 11, 1996, the total number of event types that we consider is 2,978, instead of 3,073 ($= 7 * 439$). There are 517 business days in the said period, and our sequence contains 181,089 events, with an average of 350 events per business day.

Figure 4 shows the event structure $\mathcal{S}$ which we use in our experiment. The reference event

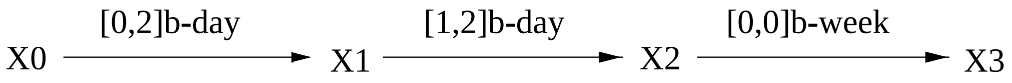

Figure 4: The event structure used in the experiment.

type we use for X_0 is the event type corresponding to a drop of the IBM stock of less than 3% (i.e., the category is (-3%, 0%)). There is no other assignment of event types to variables X_1, X_2 and X_3. The minimum confidence value we use is 0.7, i.e., the minimum frequency is 70%. The data mining task is to discover all the combinations of frequent event types E_1, E_2 and E_3 with the constraints that E_1 occurs after E_0 but within the same or the next two business days, E_2 occurs the next business day of E_1 or the day after, and E_3 occurs after E_2 but in the same business week of E_2. The choices we made for the reference type, the constraints and the minimum confidence value are arbitrary. We expect that the results regarding the performance of our heuristics apply to other choices.

In this experiment, we focus our attention on Step 4 of our strategy, namely reduction of the

[5]The prices are available from the StockMaster at the M.I.T. Artificial Intelligence Laboratory `http://www.ai.mit.edu/stocks.html`.

candidate complex event types by using sub-structures. We display our results in Figure 5. The

Stage	Substructure	No. of candidate types		No. of frequent event types
		Naive algorithm	Using heuristics	
1	$X0 \xrightarrow{[0,2]\text{b-day}} X1$	2,978	2,978	323
2	$X0 \xrightarrow{[1,4]\text{b-day}} X2$	2,978	2,978	472
3	$X0 \xrightarrow{[1,8]\text{b-day}} X3$	2,978	2,978	720
4	$X0 \xrightarrow{[0,2]\text{b-day}} X1 \xrightarrow{[1,2]\text{b-day}} X2$	8,868,484	152,456	267
5	$X0 \xrightarrow{[0,2]\text{b-day}} X1 \xrightarrow{[1,6]\text{b-day}} X3$	8,868,484	42,480	26,771
6	$X0 \xrightarrow{[1,4]\text{b-day}} X2 \xrightarrow{[0,0]\text{b-week}} X3$	8,868,484	3,211	2,637
7	$X0 \xrightarrow{[0,2]\text{b-day}} X1 \xrightarrow[{[1,2]\text{b-day}}]{} X2 \xrightarrow{[0,0]\text{b-week}} X3$	$2.64*10^{10}$	14,561	2

Figure 5: Reduction of candidate event types.

second column of the table in Figure 5 shows the induced sub-structures considered at each stage of our discovery process. We explored six sub-structures before the original one (shown as stage 7 in the table). From the application of the algorithm to derive implicit temporal constraints, the substructures of our example should have an edge from the root to each other variable in the substructure, and two constraints (one for each temporal type in the experiment, namely b-day and b-week) labelling each edge. In the table, for simplicity, we omit some of the edges and one of the two constraints on each edge, since it is easily shown that in this example, for each edge, one constraint (the one shown) implies the other (the one omitted), and some edges are just "redundant", i.e., implied by other edges.

The third column shows the number of candidate event types that we need to consider if the naive algorithm (Section 4.3) is used. The number of candidate event types under the naive algorithm is simply the multiplication of the combinations of candidate event types for each non-root variable ($2,978^s$ if s is the number of non-root variables).

The fourth column shows the number of candidate event types under our heuristics. The basic idea is to use the previous stages to screen out event types (or combination of event types) that are not frequent. Obviously, the number of candidate types under our heuristics is much smaller in the cases of two and three variables. For example, since the number of frequent types for the combination X_0, X_1 and X_2 are, respectively, 1, 323 and 472, it follows that the number of candidate types we need to consider is 152,456 ($= 1 * 323 * 472$), instead of 8,868,484 ($= 1 * 2,978 * 2,978$). Thus, we only need to consider 2% of the event types required under the naive algorithm. The number of candidate types for the original event structure we need to consider in the last stage is only $14,561$, instead of $2.64 * 10^{10}$.

The first three sub-structures we explored were those with a single non-root variable. We find frequent event types for each induced sub-structure. The next stage was the one with variables X_0, X_1 and X_2. The number of complex event types is 267, while the single event types for X_1 and X_2 are only 59 and 70, respectively. Hence, in stage 5, we only need to consider as candidates $42,480$ ($= 1 * 59 * 720$) different temporal types, instead of $232,560$ ($= 1 * 323 * 720$) or even $8,868,484$ ($= 1 * 2978 * 2978$). Stage 5 also finds that the number of event types for X_3 is 587. In stage 6, we only need to consider those combinations of events e_2 and e_3 with the condition that there exists e_1 such that e_1, e_2 is frequent in stage 4 and e_1, e_3 is frequent in stage 5. We only find $3,211$ candidate types. The number of candidate event types in the last stage is calculated by taking all the pairs from stages 4, 5 and 6, and performing a "join". That is, a combination of e_1, e_2 and e_3 will be considered as a candidate only if (i) e_1 and e_2 appear in the result of stage 4, (ii) e_1 and e_3 in stage 5, and (iii) e_2 and e_3 in stage 6.

Finally, the fifth column gives the number of (complex) event types discovered which are frequent (with minimum confidence 0.7). These event types are used in later stages to screen out event types as explained above. Figure 6 show the two complex event types found in the last stage.

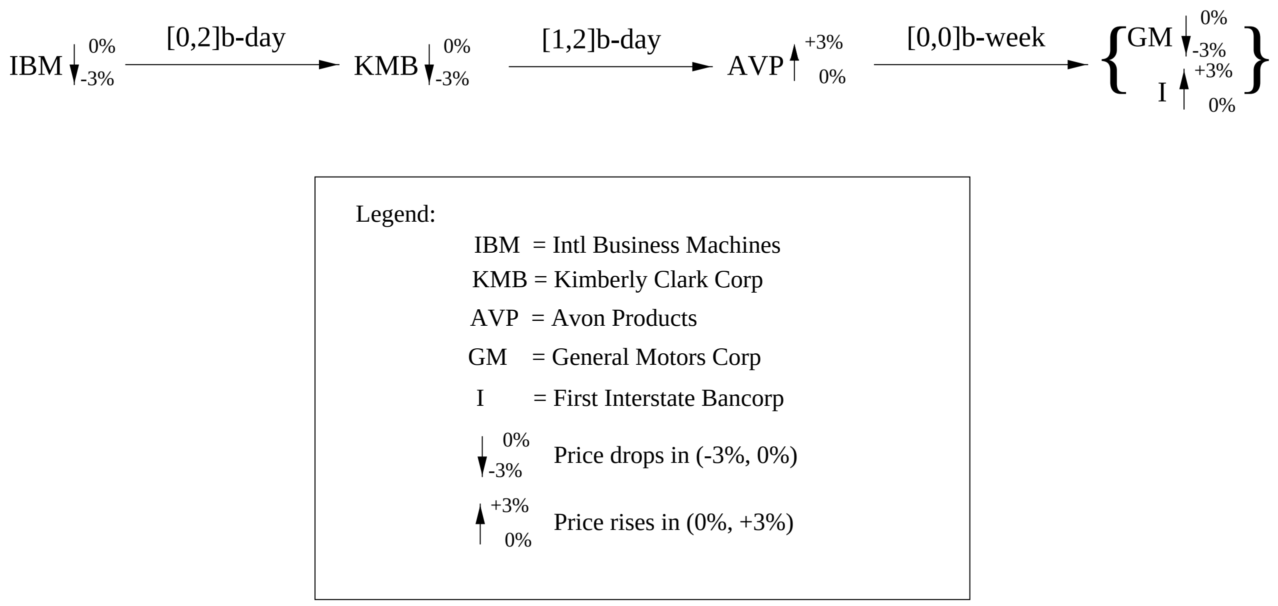
IBM 0% -3%
[0,2]b-day
KMB 0% -3%
[1,2]b-day
AVP +3% 0%
[0,0]b-week
GM 0% -3%
I +3% 0%
Legend:
IBM = Intl Business Machines
KMB = Kimberly Clark Corp
AVP = Avon Products
GM = General Motors Corp
I = First Interstate Bancorp
0% -3% Price drops in (-3%, 0%)
+3% 0% Price rises in (0%, +3%)

Figure 6: The two frequent event combinations discovered in the experiment.

7 Discussion and conclusion

In this paper, we introduced and studied the notion of temporal constraints with granularities and event structures. We also presented a timed automaton with granularities for finding event sequences that match event structures. And lastly, we defined event-discovery problems and provided a practical procedure that exploits the properties of granularities and event structures.

It is important to note that a real system can only treat finite temporal types or infinite temporal types that have finite representations. Hence, a real system can use only a subset of the temporal types that we have defined. Various proposals on representing granularities have appeared in the literature (e.g., [NS92, LMF86, CSS94]). The granularities expressible in these languages are all instances of our temporal types. Furthermore, software packages that implement calendars are available [Soo93]. However, all the algorithms of this paper are implementable in a system using any of the above representations or systems.

The event discovery problem can easily be extended in two different directions. First, the event type E_0 in the event-discovery problem needs not be a "regular" event type. It can be the event type, say, "the beginning of a week." By using this, we can discover complex events such as "what happens in most of the weeks?" Another direction is to include certain constraints on the event types allowed on the variables of an event structure; for example, two or more variables could be constrained to be assigned to the same (or different) event types. We can easily adapt our procedure to accommodate these extensions.

Another research direction is to study the user interface for the pattern discovery. We believe that the complex temporal patterns are arrived at only after the user explores the data set using simpler temporal patterns. That is, temporal patterns "evolve" from simple ones to complex ones. Hence, some friendly user interface is needed to help the users. Also, optimization strategies that exploit such an evolutionary pattern specification process will be an interesting research topic.

References

[AD94] R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.

[AHU74] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The design and analysis of computer algorithms.* Addison-Wesley, 1974.

[AIS90] R. Agrawal, T. Imielinski, and A. Swami. Database mining: A performance perspective. *IEEE Trans. Knowledge and Data Engineering*, 5(5):914–925, 1990.

[BWJ95] C. Bettini, X. Wang, and S. Jajodia. On the relevance of time granularity in data mining. Technical report, ISSE Department, George Mason University, 1995.

[CSS94] R. Chandra, A. Segev, and M. Stonebraker. Implementing calendars and temporal rules in next generation databases. In *Proc. Data Engineering Conference*, 1994.

[DMP91] R. Dechter, I. Meiri, and J. Pearl. Temporal constraint networks. *Artificial Intelligence*, 49, 1991.

[Lai93] P. Laird. Identifying and using patterns in sequential data. In *Algorithmic Learning theory, 4th International Workshop*, pages 1–18. Springer-Verlag, 1993.

[LMF86] B. Leban, D. Mcdonald, and D. Foster. A representation for collections of temporal intervals. In *Proc. AAAI-86*, pages 367–371, 1986.

[MTV95] H. Mannila, H. Toivonen, and A.I. Verkamo. Discovering frequent episodes in sequences (extended abstract). In *1st Conference on Knowledge Discovery and Data Mining*, Montreal, CA, August 1995.

[NS92] M. Niezette and J. Stevenne. An efficient symbolic representation of periodic time. In *Proc. CIKM*, Baltimore, MD, November 1992.

[Soo93] M. D. Soo. Multiple calendar support for conventional database management systems. In R. T. Snodgrass, editor, *Proceedings of the Workshop on an Infrastructure for Temporal Databases*, pages FF1–FF17, June 1993.

[WBBJ94] X. Wang, C. Bettini, A. Brodsky, and S. Jajodia. Logical design for temporal databases with multiple temporal types. Technical report, ISSE Department, George Mason University, 1994.

A Deriving implicit constraints with granularities

We consider here an approximate algorithm for checking consistency and deriving implicit constraints. Given an event structure, there are possibly infinite *implicit* TCGs. Intuitively, we want to derive those that give us *more* information about temporal relationships. Formally, a constraint $[m, n]\,\mu$ is said to be *tighter* than a constraint $[m', n']\,\mu$ if $m \geq m'$ and $n \leq n'$. We are interested in deriving the tightest possible implicit constraints in all of the granularities appearing in the event structure. From the result of Theorem 3 follows that this goal is not likely to be achieved in polynomial time, hence the choice of an approximate algorithm. The approach we take is called constraint propagation. However, traditional techniques for constraint propagation (e.g. [DMP91]) have to be integrated with procedures to convert among TCGs with different granularities.

A.1 Conversion of constraints in different granularities

Consider the problem of converting a constraint $[n, m]\,\mu_1$ into an implied constraint in terms of a granularity μ_2. If we only have a total order of granularities, like e.g. `minute`,`hour`, and `day`, then the conversion algorithm is trivial since fixed conversion factors can be used. However, if incomparable types like `week` and `month`, or types with "gaps" like `b-day` are considered, the conversion becomes more complex. In Figure 7 we propose an algorithm that is sufficiently general to apply to any pair of temporal types, provided that the target type covers a span of time equal or larger than the span of time covered by the source type. For example, we can convert a constraint in terms of `b-week` into a constraint in terms of `week`, `month`, or `b-day`, but not into a constraint in terms of `week-ends`. The above condition on the target type is quite restrictive and could be relaxed, but it ensures that the proposed algorithm performs only allowed conversions.

The algorithm assumes the existence of a primitive type μ (e.g. `second`) in the considered granularity system such that any tick of the other types can be obtained as union of ticks of the primitive type. The algorithm uses an approximation of the relation between each type and the primitive type, and hence, in some cases, it does not give the tightest possible bounds as an output.

To specify the algorithm we first have to introduce some notation. Let $minsize(\mu, k)$ and $maxsize(\mu, k)$ be respectively the minimum and maximum length of k contiguous ticks of μ, expressed in ticks of the primitive type. For example, $minsize(\texttt{month}, 1) = 28$, $maxsize(\texttt{month}, 1) = 31$, and $maxsize(\texttt{b-day}, 2) = 4$ if `day` is used as the primitive type. We also use $mingap(\mu, k)$ to denote the minimal distance in terms of the primitive type between each tick of μ and the kth one after it. Formally,

$mingap(\mu, k) = min_i\{s_i \mid s_i = min(\mu(i+k)) - max(\mu(i))\}$. With respect to $minsize()$, $mingap()$ considers even ticks that have gaps in between. For consistency with standard constraint notation, in the algorithm we represent a TCG $[n, m]\,\mu$ labelling an arc from X to Y in an event structure with the constraint $Y - X \in [m, n]$ in terms of granularity μ. Steps (1) and (3) compute respectively the maximum span of time covered by $m + 1$ ticks of the source type and the minimum span of time between a tick of the source type and its nth successor. This time is measured in terms of the primitive granularity. The three factors in the formulas are used to get a more accurate measurement than using a single conversion factor. Indeed, consider for example the source granularity `month`: the maximal length of a month is 31, but the maximal length of three contiguous months is not $3 * 31$, but 91. $m+1$ is used in the conversion instead of m, since an event could occur at the very beginning of tick i and the other at the very end of tick $i + m$. Their distance is m, but the period that they cover is actually $m + 1$ ticks.

Steps (2) and (4) compute respectively the new maximum and minimum in terms of the target granularity. We use again the trick of considering the length of 1, 2, and 3 contiguous ticks to get a more accurate conversion among granularities. Intuitively, we want to find the minimal "combination" of 1, 2, and 3 ticks of the target unit, that covers the previously computed spans of time. For the maximum we use the minimal length ($minsize()$) of the ticks, since we want to maximize the number of ticks needed to cover the span of time. Analogously, we use the maximal length ($maxsize()$) in the computation of the minimum value.

Note that using 3 as the number of factors for the approximation is just for illustrative purpose. A real implementation should use a variable factor depending on source and target granularities.

A.2 An approximate algorithm for TCGs propagation

Let $\mathcal{S} = (W, A, ,)$ be an event structure and M the set of temporal types appearing in , . The algorithm proceeds as follows. It first partitions TCGs in an event structure into groups, each group having TCGs in the same temporal type. That is, for each μ in M, let $\mathcal{C}_\mu$ be the set of all the constraints $X - Y \in [m, n]$, where X, Y are in W and $[m, n]\,\mu \in , (X, Y)$. Now, the propagation within $\mathcal{C}_\mu$ is a problem known as the Simple Temporal Problem [DMP91]. We apply the path consistency algorithm [DMP91] within each group. Since constraints expressed in a granularity could imply constraints in other granularities, we should try to convert them and add the derived constraints to the corresponding groups. Hence, for each pair of temporal types μ and ν in M such that a conversion is allowed, we convert each constraint in $\mathcal{C}_\mu$ into one in terms of ν and add it into $\mathcal{C}_\nu$. This conversion is done using the procedure reported in

INPUT: a constraint $Y - X \in [m, n]$, a source type μ_1, a target type μ_2 s.t. $\forall i, t\ (t \in \mu_1(i) \rightarrow \exists j\ t \in \mu_2(j))$, values $minsize(\mu_2, k)$, $maxsize(\mu_2, k)$, $mingap(\mu_1, k)$, and $maxsize(\mu_1, k)$ for each $k = 1, 2, 3$.

OUTPUT: a constraint $Y - X \in [\overline{m}, \overline{n}]$ in terms of μ_2 implied by $Y - X \in [m, n]$ in terms of μ_1.

METHOD:

1. Maxspan $= (n+1)$ DIV $3 * maxsize(\mu_1, 3) + ((n+1)$ MOD $3)$ DIV $2 * maxsize(\mu_1, 2) + ((n+1)$ MOD $3)$ MOD $2 * maxsize(\mu_1, 1) - 1$

2. $\overline{n} = min(S) - 1$ where S satisfies the following equations:
   $r_3 * 3 + r_2 * 2 + r_1 = S$
   $r_3 * minsize(\mu_2, 3) + r_2 * minsize(\mu_2, 2) + r_1 * minsize(\mu_2, 1) >$ Maxspan

3. Minspan $= m$ DIV $3 * mingap(\mu_1, 3) + (m$ MOD $3)$ DIV $2 * mingap(\mu_1, 2) + (m$ MOD $3)$ MOD $2 * mingap(\mu_1, 1)$

4. $\overline{m} = min(S) - 1$ where S satisfies the following equations:
   $r_3 * 3 + r_2 * 2 + r_1 = S$
   $r_3 * maxsize(\mu_2, 3) + r_2 * maxsize(\mu_2, 2) + r_1 * maxsize(\mu_2, 1) >$ Minspan

Figure 7: Algorithm for the conversion of constraints

Subsection A.1. The process is repeated with the path consistency algorithm and the conversion, until no new constraints appear in any group.

The above algorithm is *sound* if the converted constraints are logically implied by the original ones. By *sound* we mean that if a complex event (σ, α) matches the given event structure $\mathcal{S} = (W, A, ,)$, then it also matches $\mathcal{S}' = (W, A', , ')$, where A' and $, '$ are given by the algorithm (e.g., if $X - Y \in [m, n]$ is in C_μ, then $(Y, X) \in A'$ and $[m, n]\,\mu \in , '(Y, X)$).

The aforementioned algorithm is an *approximate* propagation for two reasons. First, translation between groups cannot be done precisely since a constraint in one granularity cannot be translated to one in another granularity without losing precision. Second, the set of temporal types we use are only those that appear in the event structure. The algorithm may derive tighter constraints (in the sense of logical implication) if the translation is done more precisely using additional temporal types.

Theorem 4 The proposed algorithm is sound, terminates, and requires time polynomial in the size of the constraint graph.

We note that the NP-hardness of the consistency checking implies that a *complete*, sound algorithm for constraint propagation on event structures is not likely to be polynomial. A *complete* algorithm here is one that always derives the tightest constraints between all pairs of variables. Indeed, if such a polynomial algorithm existed, consistency checking would be polynomial since the tightest constraint between each pair of variables in an inconsistent event structure is "false" (i.e., not satisfiable).

B Proofs

Proof of Theorem 1

We give the procedure for the construction of the timed automata $\text{TAG}_{\mathcal{T}}$ corresponding to a complex event type $\mathcal{T}$.

INPUT: a complex event type $\mathcal{T} = (\mathcal{S}, \varphi)$, where $\mathcal{S} = (W, A, ,)$

OUTPUT: a TAG such that an event sequence σ is accepted by the TAG iff a complex event of type $\mathcal{T}$ occurs in σ.

METHOD:

Step 1. Decompose $\mathcal{S}$ into the minimal number of chains such that (1) each chain starts from the root and ends with a variable having no outgoing arcs, and (2) each arc of the graph is contained in at least one chain.

Step 2. For each chain l with the variables X_1, ..., X_{n_l} (in this order), build a TAG $\mathcal{A}^l = (W, S^l, \{s_0^l\}, C^l, T^l, \{s_{n_l}^l\})$, where $S^l = \{s_0^l, \ldots, s_{n_l}^l\}$, $C^l = \{x_{\mu_1}^l, \ldots, x_{\mu_s}^l\}$ if $\mu_1, \ldots, \mu_s$ are all the temporal types appearing in the constraints of the chain, and T^l consists of the following transitions: (1) $\langle s_0^l, s_1^l, X_1, C^l, true\rangle$, and (2) $\langle s_{j-1}^l, s_j^l, X_j, C^l, \delta_j^l\rangle$ for each $j = 2, \ldots, n_l$, where δ_j^l is the conjunction $\bigwedge_{[m,n]\,\mu\in\Gamma(X_{j-1},X_j)}(m \leq x_\mu^l \leq n)$. Note that different clocks are used for each chain and all the clocks are reset at each transition.

Step 3. Combine all $\mathcal{A}^l$ into a single TAG by using a "cross product" technique as follows: Assume there are k chains. Then the resulting TAG is

$$\mathcal{A} = (W, S, \{s_0^1 \cdots s_0^k\}, C^1 \cup \ldots \cup C^k, T, \{s_{n_1}^1 \ldots s_{n_k}^k\}),$$

where $S = \{s^1 \cdots s^k | s^l \in S^l \text{ for each } l\}$ and T consists of all the transitions $\langle s, s', X, \lambda, \delta\rangle$, with X being in W, that satisfy the following two conditions:

(1) For each chain l, if the corresponding TAG contains a transition $\langle s_{j-1}^l, s_j^l, X, C^l, \delta_j^l\rangle$, then s contains the label s_{j-1}^l, s' contains the label s_j^l, $C^l \subseteq \lambda$, and δ_j^l is a conjunct in δ;

(2) λ and δ are the minimal sets satisfying these requirements.

Step 4. According to the mapping φ substitute each input symbol X in the transitions of the automata obtained in the previous step with the event type symbol $\varphi(X)$. Note that some of the variable symbols can be mapped to the same event type[6]. For each state s in the automaton, we add a reflexive transition $\langle s, s, e, \emptyset, true\rangle$ (i.e., a loop) for each $e \in E$. This last step is to allow the automaton to "skip" events, so that an event sequence is accepted by this final automaton if a subset of its events is accepted by the automaton built at step 3 above.

It is clear from the procedure that the automata can be constructed in polynomial-time. It is also easy to show that if the event sequence σ does not contain simultaneous events, $\mathcal{T}$ occurs in an event sequence σ iff TAG$_\mathcal{T}$ has an accepting run over σ. With a straightforward extension of the above TAG construction and using the event sequence as a set of elements of the form $(E_1, \ldots, E_k, t)$, i.e., all the event types that occur at the same time are combined together, we can eliminate the restriction to non-simultaneous events. The basic idea is to find the "0-length" paths from the TAG built at step 4 in the above procedure and add a transition that (1) the time elapsed must be 0 and (2) the event types that fired this 0-length transition must contain all the event types in this path. For more details, we refer the interested reader to [BWJ95].

[6]The construction would not work if we use the event types instead of the variable symbols from the beginning; indeed we exploit the property that the nodes of the constraint graph are all differently labelled.

Proof of Theorem 2

The TAG obtained from a complex event type is non-deterministic. We simulate the non-deterministic TAG using a standard technique presented in [AHU74, page 328]. The standard simulation of a NDFA by a DFA stores a set of states. For each input symbol, the algorithm scans the states and for each state scanned, consider all possible transitions and generate another set of states. So, for each state, $|S|$ is needed if S is the set of states. Hence, $|S|^2$ time is needed for each input symbol and the complexity of the whole pattern matching is $O(|\sigma| * |S|^2)$ Consider first a simple case for our simulation: the complex event type is representable as a chain. The corresponding automaton will have a transition from each non-starting state to the same state (i.e., loops), labelled with all the event types as input symbols. Hence, it will be non-deterministic, since another outgoing transition labelled with one of the event types will also be present. If a transition from state $S1$ to state $S2$ is labelled with an event type, a constraint $k_1 \leq x_\mu \leq k_2$, and a reset on x_μ, to perform the simulation we have to consider $k_2 - k_1$ new pairs (state, clocks-assignment). If the transition has more than one constraint and clock, the number of new 'states' equals the size of the maximum range in the constraints. Since in our construction from the chain the number of transitions leading to a different state equals the number of variables ($|V|$) in the chain, an upper bound on the number of 'states' that we need to record in the simulation is $|V| * K$ where K is the maximum range in all the constraints of the chain. It is also possible that $|\sigma| < |V| * K$, where $|\sigma|$ is the number of events in the input. Since the different values in clocks are essentially given by the events in the input, in this case the number of 'states' that we need to record in the simulation is $|\sigma|$. If we use the analogy with the standard simulation of NDFA by DFA, for the TAG corresponding to the chain this translates into $O(|\sigma| * (|S| * min(|\sigma|, |V| * K))^2)$ where σ is the event sequence (possibly reduced), and $|S|$ is the number of states in the TAG. Now, let's consider a general complex event type. Our construction obtains the corresponding TAG as a "crossproduct" of the TAGs corresponding to the chains in which the complex event has been decomposed. Hence, the upper bound for the 'states' that we have to record translates to $(|V| * K)^p$, where p is the number of chains in the crossproduct. The global simulation, then, analogously to standard simulation of NDFA would have a complexity upper bound of $O(|\sigma| * (|S| * min(|\sigma|, (|V| * K)^p))^2)$.

Proof of Theorem 3

Given a set of constraints, is there an assignment to the variables that satisfies the constraints? We show that answering this question is at least as hard as solving the SUBSET SUM problem (a knapsack variant). Consider a set of positive integers $n_1, \ldots, n_k$ and s. The SUBSET SUM problem consists in finding a subset such that the sum of its numbers is s. For each instance of this problem we can construct an instance of our problem as follows: Let $W = X_1, \ldots, X_{k+1}, V_1, \ldots, V_k, U_1, \ldots, U_k$. Consider the granularities `n-month` for $n = n_1, \ldots, n_k$ defined by grouping each consecutive n ticks of `month` into a single tick. For each $i = 1, \ldots, k$, create the following two constraints:

$(X_i, X_{i+1}) \in [0, n_i]\,\texttt{month}$.

$(X_1, X_{k+1}) \in [s, s]\,\texttt{month}$.

We now add another set of constraints with the only purpose to rule out all values between 1 and $n_i - 1$ in the first of the above two constraints. For each $i = 1, \ldots, k$ we add:

$(V_i, X_i) \in [0, 0]\,\texttt{n}_\texttt{i}\texttt{-month}$,

$(V_i, X_i) \in [n_i - 1, n_i - 1]\,\texttt{month}$,

$(U_i, X_{i+1}) \in [0, 0]\,\texttt{n}_\texttt{i}\texttt{-month}$.

$(U_i, X_{i+1}) \in [n_i - 1, n_i - 1]\,\texttt{month}$.

This set of constraints implies the disjunctions:

$(X_i, X_{i+1}) \in [0, 0]\,\texttt{month} \vee [n_i, n_i]\,\texttt{month}$ for each $i = 1, \ldots, k$. If an assignment is found to satisfy the whole set of constraints specified above, the distance between the value of X_{i+1} and X_i will be 0 or n_i for each $i = 1, \ldots, k$. The set of indices i for which that value is different from 0 determines the subset of $\{n_1, \ldots, n_k\}$ being a solution of the SUBSET SUM problem. It is also easy to show that if an assignment is not found such a subset does not exist. Since the SUBSET SUM problem is NP-hard, and the transformation in the consistency problem can be done in polynomial time, determining consistency is also NP-hard.

Proof of Theorem 4

Soundness. Trivial.

Termination.

Step 1 (path consistency of an STP) is known to terminate. Step 2 (conversion of constraints) trivially terminates. Hence, we only have to show that we cannot infinitely iterate between steps 1 and 2. Since we do not allow $+\infty$ nor $-\infty$ in the constraints and the constraints form a *dag*, any explicit or implicit constraint between two variables will have only finite (possibly negative) values. Consider $S = \sum_{i=1}^{k}(t_e^i - t_b^i)$ where k is the number of constraints after the first iteration, and t_e^i and t_b^i are respectively the ending and beginning values of constraint i. We show that S is monotonically decreasing at each iteration, and, since it cannot be negative, this means that the algorithm terminates in a finite number of iterations. After the first iteration, the number of constraints does not change any more. Indeed, we'll have $|M|$ constraints, where M is the set of temporal types appearing in the constraints, for each pair of variables. Step 1 results in 'shrinking' the interval of some of the constraints and leaving the rest of them untouched. Step 2 translates each constraint in terms of other granularities. If a constraint C_1 is translated into a constraint C_1' for type μ at iteration j, then, at iteration $j+1$, either C_1 is the same or it has a reduced interval. Based on the translation procedure, this means that C_1' at iteration $j+1$ is either the same or it has a reduced interval. This argument shows that S, the sum of all the interval lengths, can only decrease at each iteration. The condition to continue the iteration of steps 1 and 2, imposing that some new constraint must appear, ensures that the decreasing of S is strictly monotonic.

Complexity.

Step (1) in the worst case takes time $O(n^3 \cdot |M|)$, where n is the number of variables (nodes in the graph) and $|M|$ is the number of temporal types appearing in the explicit constraints. Step (2) in the worst case takes time $O(c \cdot n^2 \cdot |M|)$, where c is the constant time required to translate a constraint from a granularity to an other one. Hence, their combination takes time $O(n^3 \cdot |M|)$. We have shown above that both steps can only reduce the range of values in the constraints. In the worst case, at each iteration (steps 1 + 2), only one constraint range is reduced and it is reduced by only one unit. Thus, the upper bound on the number of iterations is $n^2 \cdot |\mathcal{M}| \cdot w$, where w is the maximal range appearing in the constraints. Hence, the overall worst case complexity is $O(n^5 \cdot |M|^2 \cdot w)$.