Discovering Frequent Event Patterns with Multiple Granularities in Time Sequences

Claudio Bettini, *Member, IEEE*, X. Sean Wang, *Member, IEEE Computer Society*, Sushil Jajodia, *Senior Member, IEEE*, and 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; the goal of the discovery is to find temporal patterns, i.e., instantiations of the variables in the structure that appear frequently 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 testing 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.

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

1 Introduction

A HUGE amount of data is collected every day in the form of event time sequences. Common examples are recordings of different values of stock shares during a day, accesses to a computer via an external network, bank transactions, or events related to malfunctions in an industrial plant. These sequences register events with corresponding values of certain processes, and are 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, or sets 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 monitored process.

Although a lot of work has been done on identifying and using patterns in sequential data (see [1], [11] 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 automatic teller machine transactions, we may want to discover events that are constrained in terms of time granularities such as events occurring in the same day, or events happening within k weeks from a specific one. The system should not simply 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 on providing a formal framework for expressing data mining tasks involving time granularities, and on proposing efficient algorithms for performing such tasks. To this end, we introduce the notion of an *event structure*. 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 a prescribed order, with the second one occurring between four and six hours after the first but within the same business day. We consider data mining tasks where an event structure is given and only some of its variables are instantiated. We examine the event sequence for patterns of events that match the event structure. Based on the frequency of these patterns, we discover the instantiations for the free variables.

To illustrate, assume that we are interested in finding all those events which frequently follow within two business days of a rise of the IBM stock price. To formally model this data mining task, 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 X_1 has to happen within two business days after X_0 happens. The data mining task is now to find all the instantiations of 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 data mining task.

- *C. Bettini is with the Department of Information Science (DSI), University of Milan, Italy. E-mail: bettini@dsi.unimi.it.*
- *X.S. Wang, S. Jajodia, and J.-L. Lin are with the Department of Information and Software Systems Engineering, George Mason University, Fairfax, VA 22030. E-mail: {xywang, jajodia, jllin}@isse.gmu.edu.*

Manuscript received 19 Aug. 1996.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number 104365.

In order to find all the solutions for a given event structure, we first consider the case where each variable is instantiated with a specific event type. We call this a *candidate* instantiation of the event structure. We then scan through the time sequence to see if this candidate instantiation occurs frequently. In order to facilitate this pattern matching process, we introduce the notion of a *timed finite automaton with granularities* (TAG). A TAG is essentially a standard finite automaton with the modification that a set of clocks is associated with the automaton and each transition is conditioned not only by an input symbol, but also by the values of the associated clocks. Clocks of an automaton may be running in different granularities.

To effectively perform data mining, however, we cannot naively consider all candidate instantiations, since the number of such instantiations is exponential in the number of variables. 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.

We consider our algorithms and heuristics as part of a general data mining system which should include, among other subsystems, a user interface. Data mining requests are issued through the user interface and processed by the data mining algorithms. The requests will be in terms of the aforementioned event structures which are the input to the data mining algorithms. In reality, a user usually cannot come up with a request from scratch that involve complicated event structures. Complicated event structures are often given by the user only after the user explores the data set using simpler ones. That is, temporal patterns "evolve" from simple ones to complex ones with a greater number of variables in the event structure and/or tighter temporal constraints. Our algorithms and heuristics are designed, however, to handle complicated as well as simple event structures.

1.1 Related Work

The extended abstract in [5] established the theoretical foundations for this work. Timed finite automata with multiple granularities and reasoning techniques for temporal constraints with multiple granularities are introduced there.

In the artificial intelligence area, a lot of work has been done for discovering patterns in sequence data (see, for example, [9], [11]). In the database context, where input data is usually much larger, the problem has been studied in a number of recent papers [18], [2], [13], [19]. Our work is closest to [13], 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 subpatterns, together with a sliding window method, is shown to provide effective algorithms. Our algorithm essentially follows the same approach, decomposing the given pattern and using the results of discovery for subpatterns to reduce the number of candidates to be considered for the discovery of the whole pattern. In contrast to [13], 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.

In [2], the problem of discovering sequential patterns over large databases of customer transactions is considered. The proposed algorithms generate a data sequence for each customer from the database and search on this set of sequences for a frequent sequential pattern. For example, the algorithms can discover that customers typically rent "Star Wars," then "Empire Strikes Back," and then "Return of the Jedi." Similarly to [13], the strategy of [2] is starting with simple subpatterns (subsequences in this case) and incrementally building longer sequence candidates for the discovery process. While we assume to start directly with a data sequence and not with a database, we consider more complex patterns that include temporal distances (in terms of multiple granularities) between the events in the pattern. This gives rise to the capability, for example, to discover whether the above sequential pattern about "Star Wars" movie rentals is frequent if the three renting transactions need to occur within the same week. A similar extension is actually cited as an interesting research topic in [2]. The need for dealing with multiple time granularities in event sequences is also stressed in [10].

Finally, the work in [18], [19] also deals with the discovery of sequential patterns, but it is significantly different from our work. In [18], the considered patterns are in the form of specific regular expressions with a distance metrics as a dissimilarity measure in comparing two sequences. The proposed approach is mainly tailored to the discovery of patterns in protein databases. We note that the concept of distance used in [18] is essentially an approximation measure, and, hence, it differs from the temporal distance between events specified by our constraints. In [19], a scenario is considered where sequential patterns have previously been discovered and an update is subsequently made to the database. An incremental discovery algorithm is proposed to update the discovery results considering only the affected part of the database.

The temporal constraints with granularities introduced in this paper are closely related to temporal constraint networks and their reasoning problems (e.g., consistency checking) that have been studied mostly in the artificial intelligence area (cf. [8]); 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 [4] for modeling real-time systems and checking their specifications. We extend the automata to ones which have clocks moving according to different time granularities.

The remainder of this paper is organized as follows. 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

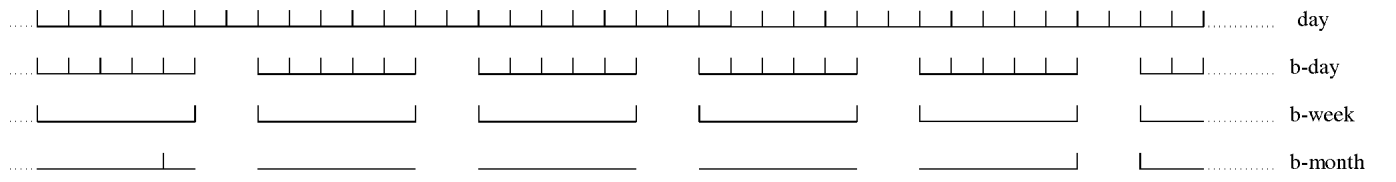


Fig. 1. Three temporal types covering the span of time from February 26 to April 2, 1996, with `day` as the absolute time.

a number of heuristics to be applied in the discovery process. In Section 6, we analyze the costs and effectiveness of the heuristics with the support of experimental results. We conclude the paper in Section 7 with some discussion. In Appendix A, we report on an algorithm for deriving implicit temporal constraints and provide proofs for the results in the paper.

2 Preliminaries

In order to formally define temporal relationships that involve time granularities, we adopt the notion of *temporal type* used in [17] and defined in a more general setting in [6]. 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]) that satisfies the following two conditions 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 $\mu(j) = \emptyset$.

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 ith *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 set of absolute time corresponding to the year 1801, etc. Note that this definition allows temporal types in which ticks are mapped to more than one continuous interval. For example, in Fig. 1, we show 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). 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, if $\exists z'$ (necessarily unique) such that $\nu(z) \subseteq \mu(z')$ then $\lceil z \rceil_{\nu}^{\mu} = z'$, otherwise $\lceil z \rceil_{\nu}^{\mu}$ is undefined. The uniqueness of z' is guaranteed by the monotonicity of temporal types. As an example, $\lceil z \rceil_{\text{second}}^{\text{month}}$ gives the month that includes the second z. Note that while $\lceil z \rceil_{\text{second}}^{\text{month}}$ is always defined, $\lceil z \rceil_{\text{week}}^{\text{month}}$ is undefined if week z falls between two months. Similarly, $\lceil z \rceil_{\text{day}}^{\text{b-day}}$ 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 $\nu =$ `seconds`.

We use the $\lceil \ \rceil_{\nu}^{\mu}$ function to define a natural relationship between temporal types: A temporal type ν is said to be *finer than*, denoted $\preccurlyeq$, a temporal type μ if the function $\lceil z \rceil_{\nu}^{\mu}$ is defined for each nonnegative integer z. For example, `day` $\preccurlyeq$ `week`. It turns out that $\preccurlyeq$ is a partial order, and the set of all temporal types forms a lattice with respect to $\preccurlyeq$ [17].

3 Formalization of the Discovery Problem

Throughout the paper, 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 $\{(E_1, t_1), \ldots, (E_n, t_n)\}$. Intuitively, each event (E, t) appearing in an event sequence σ represents the occurrence of event type E at time t. 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 and n be nonnegative integers with $m \leq n$ and μ be a temporal type. 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, $(t_1, t_2) \in [m, n]\ \mu$ 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) $m \leq (\lceil t_2 \rceil^{\mu} - \lceil t_1 \rceil^{\mu}) \leq n$.*

1. We use the symbol $\mathcal{R}$ to denote the real numbers. We assume that the underlying absolute time is continuous and modeled by the reals. However, the results of this paper still hold if the underlying time is assumed to be discrete.

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_1th and t_2th 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 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 and e_2 satisfy TCG [0, 2] `hour` 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] `month` if e_2 occurs in the month immediately after that 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, 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.*

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. Fig. 2 shows an event structure.

We define two additional concepts based on event structures: a *complex event type* and a *complex event*.

DEFINITION. *Let* $S = (W, A, \Gamma)$ *be an event structure with time granularities. Then a* complex event type derived from S *is* S *with each variable associated with an event type, and a* complex event matching S *is* S *with each variable associated with a distinct event such that the event timestamps satisfy the time constraints in* Γ.

In other words, a complex event type is derived from an event structure by assigning to each variable a (simple) event type, and a complex event is derived from an event structure by assigning to each variable an event so that the time constraints in the event structure are satisfied.

Let $\mathcal{T}$ be a complex event type derived from the event structure $S = (W, A, \Gamma)$. Similar to the notion of an occurrence of a (simple) event type in an event sequence σ, we have the notion of an occurrence of $\mathcal{T}$ in σ. Specifically, let σ' be a subset of σ such that $|\sigma'| = |W|$. Then σ' is said to be an *occurrence* of $\mathcal{T}$ if a complex event matching S can be derived by assigning a distinct event in σ' to each variable in W so that the type of the event is the same as the type assigned to the same variable by $\mathcal{T}$. Furthermore, $\mathcal{T}$ is said to *occur* in σ if there is an occurrence of $\mathcal{T}$ in σ.

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 Fig. 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 five business days after the same rise of the IBM stock and within eight 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* (S, γ, E_0, ρ), where

1) 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 partial mapping which assigns a set of event types to some of the variables* (*except the root*).

An event-discovery problem (S, γ, E_0, ρ) is the problem of finding all complex event types $\mathcal{T}$ such that each $\mathcal{T}$:

1) occurs frequently in the input sequence, and
2) is derived from S by assigning E_0 to the root and a specific event type to each of the other variables.

(The assignments in 2) 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

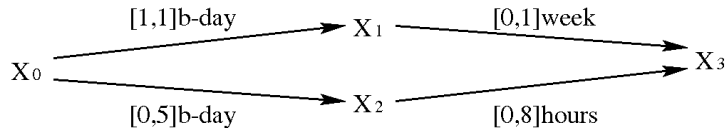


Fig. 2. An event structure.

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* (S, γ, E_0, ρ) *on a given event sequence* σ, *in which* E_0 *occurs at least once, is the set of all complex event types derived from* S, *with the following conditions*:

1) E_0 *is associated with the root of* S *and each event type assigned to a nonroot variable* X *belongs to* $\rho(X)$ *if* $\rho(X)$ *is defined, and*
2) *each complex event type occurs in* σ *with a frequency greater than* γ.

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. (S, 0.8, `IBM-rise`, ρ) is a discovery problem, where S is the structure in Fig. 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 Event Types

In this section, we introduce timed finite automata with granularities (TAGs) for the purpose of finding whether a candidate complex event type 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 [4] 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 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.

DEFINITION. *A* timed automaton with granularities (*TAG*) *is a six-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 nonnegative integer k, $x_\mu \le k$ and $k \le x_\mu$ 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 nonnegative integer,[3] that satisfies the following two conditions:

1) (*Initiation*) $s_0 \in S_0$, and $v_0 = \{(x, 0) \mid x \in C\}$, i.e., all clock values are 0; and
2) (*Consecution*) for each $i \ge 1$, 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}. 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* $TAG_{\mathcal{T}}$ *such that* $\mathcal{T}$ *occurs in an event sequence* σ *iff* $TAG_{\mathcal{T}}$ *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 S is based on a

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.

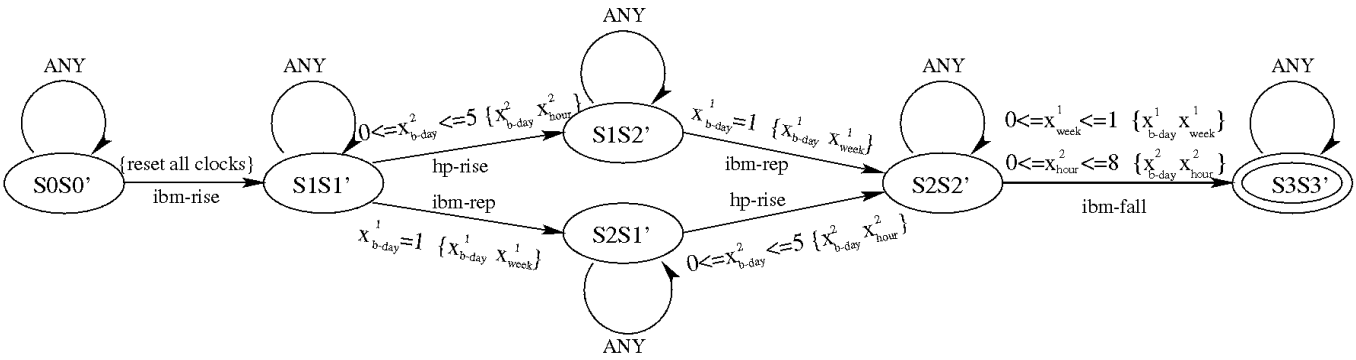


Fig. 3. An example of timed automaton with granularities.

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.[4] A detailed procedure for TAG generation can be found in the Appendix. Fig. 3 shows the TAG corresponding to the complex event type in Example 1.

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 nondeterministic finite automaton (NDFA) (cf. [3, p. 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 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 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 nonroot 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 for 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

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 labeled.

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 Section 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 "subproblem" 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. 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. More specifically, a TCG $[m, n]\ \mu$ between variables X and Y in an event structure is allowed to be converted into $[m', n']\nu$ as long as the following condition is satisfied: For any pair of values x and y assigned to X and Y, respectively, if x and y belong to a solution of S, then they also satisfy $[m', n']\nu$. As an example, consider the event structure with three variables X, Y, and Z with the TCG [0, 0]day assigned to (X, Z) and [0, 0]b-day to (X, Y) as well as (Y, Z). It is clear that we may convert [0, 0]day on (X, Z) to [0, 0]b-day since for any events x and z assigned to X and Z, respectively, if they belong to a solution of the whole structure, these two events must happen within the same *business day*.

In Appendix A, we report an algorithm to derive implicit constraints from a given set of TCGs. The algorithm is based on performing *allowed* conversions among TCGs with different granularities as discussed above, 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 $\mathcal{S} = (W, A, \Gamma)$, it is of practical interest to check if the structure is consistent, i.e., if there exists a complex event that matches $\mathcal{S}$. Indeed, if an event structure is inconsistent, it should be discarded even before the data mining process starts.

Given an input event structure, we apply the approximate polynomial algorithm described in Appendix A to derive implicit constraints. Indeed, if one of these constraints is the "empty" one (unsatisfiable, independently of a given event sequence), the whole event structure is inconsistent.

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 Fig. 2. If a discovery problem is defined on the substructure 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 be the coarsest temporal type such that for each temporal type ν in the constraints and timestamp z in the sequence, 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 timestamp 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.

We proceed as follows: If X_0 is the root, consider all the nonempty 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 Example 2, 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 matching 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.

This technique requires the derivation of implicit constraints. Given an event structure, there are possibly an infinite number of *implicit* TCGs. Intuitively, we want to derive those that give us *more* information about temporal relationships. Formally, a constraint is said to be *tighter* than another if the former implies the latter. We are interested in deriving the tightest possible implicit constraints in all of the granularities appearing in the event structure. In single granularity constraint networks this is usually done applying constraint propagation techniques [8]. However, due to the presence of multiple granularities, these techniques are not directly applicable to our event structures. In [6], we have proposed algorithms to address this problem. Essentially, we partition TCGs in an event structure into groups (each group having TCGs in terms of the same granularity) and apply standard propagation techniques to each group to derive implicit TCGs between nodes that were not directly connected and to tighten existing TCGs. We then apply a conversion procedure to each TCG on each edge,

deriving, for each granularity appearing in the event structure, an implied TCG on the same arc in terms of that granularity. These two steps are repeated until no new TCG is derived. More details on the algorithm are reported in Appendix A.

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 subtype should also occur frequently. (This is similar to [13].) Here by a *subtype* of a complex type $\mathcal{T}$, we mean a complex event type, induced by a subset of variables, such that each occurrence of the subtype can be "extended" to an occurrence of $\mathcal{T}$. However, not every subset of variables of a structure can induce a substructure. For example, consider the event structure in Fig. 2 and let $\mathcal{S}' = (\{X_0, X_3\}, \{(X_0, X_3)\}, \Gamma')$. $\mathcal{S}'$ cannot be an induced substructure, since it is not possible for Γ' to capture precisely the four constraints of that structure. This forces us to consider approximated substructures.

Let $\mathcal{S} = (W, A, \Gamma)$ be an event structure and M the set of all the temporal types 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 propagation algorithm of Appendix A. Then, for each subset W' of W, the *induced approximated substructure 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]`week` and [1,175]`hour`. Note that if a complex event matches $\mathcal{S}$ using events from σ, then there exists a complex event using events from a subsequence σ' of σ that matches the substructure $\mathcal{S}'$.

By using the notion of an approximated substructure, 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 S, except the root X_0, consider the approximated substructure $\mathcal{S}'$ 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-discovery problem (called *induced discovery 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 ([13]) 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 substructure 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 discovery problem of Example 2 with the simple variation that $\rho = \emptyset$, i.e., all nonroot variables are free. $(\mathcal{S}', 0.8,$ `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 substructures that include more than one nonroot variable. For each integer $k = 2, 3, \ldots$, consider all the approximated substructures $\mathcal{S}_k$ induced from the root variable and k other variables in $\mathcal{S}$, where these variables (including the root) form a subchain 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 process results, in practice, in a smaller number of candidate types for induced discovery problems.

6 Effectiveness of the Process and Experimental Results

In this section we motivate the choice of the proposed steps in our strategy by analyzing their costs and effectiveness with the support of experimental results.

As discussed in the introduction (related work), the algorithms and techniques that can be found in the literature cannot be straightforwardly applied to discover patterns specified by temporal quantitative constraints (in terms of multiple granularities) in data sequences. For this reason, we evaluate the cost/effectiveness of the proposed algorithms and heuristics per se, and by comparison with the naive algorithm described in Section 4.3.

The first step (consistency checking) involves applying the approximate algorithm described in Appendix A to the input event structure. The computational complexity of the algorithm is independent from the sequence length, and it is polynomial in terms of the parameters of the event structure [6]. We also conducted experiments to verify the actual behavior of the algorithm depending on the parameters of the event structure [14]. We applied the algorithm to a set of 300 randomly generated event structures with TCG parameters in the range 0 … 100 over eight different granularities. The results show that, in practice, the algorithm is very efficient, since the average number of iterations between the two main steps (each is known to be

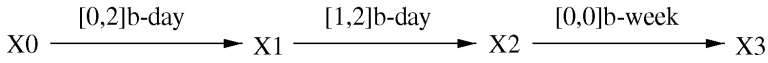


Fig. 4. The event structure used in the experiment.

efficient) is 1.5 for graphs with up to 20 variables, while it is only 1 for graphs with up to six variables.[5] We can conclude that the time spent for this test is negligible compared with the time required for pattern matching in the sequence. On the contrary, if inconsistent structures are not recognized, significant time would be spent searching the sequence for a pattern that would never be found.

Steps 2 through 4 all require scanning the sequence, but it is possible to perform them concurrently so that a single scan is sufficient to conclude steps 2 and 3, and to perform the first pass in step 4. The cost of step 2 is essentially the time to check, for each event in the sequence, if its timestamp is contained in a specific precomputed granularity. This containment test can be efficiently implemented. The benefits of the test largely depend on the considered event sequence and event structure. For example, if the sequence contains events heterogeneously distributed along the time line, while the structure specifies relationships in terms of particular granularities, this step can be very useful, discarding even most of the events in the input sequence and dramatically reducing the discovery time. On the contrary, if *regular* granularities are used in the event structure, or if the occurrences of events in the sequence always fall into the granularities of the event structure, the step becomes useless. Since it is not clear how often these conditions are satisfied, we think that the discovery system should be allowed to switch on and off the application of this step depending on the task at hand.

The cost of step 3 is essentially the time to check, for each reference event in the sequence, the satisfiability of a set of binary constraints between that event and another event in the sequence. In terms of computation time, this is equivalent to running for each constraint a small (two states) timed automata ignoring event types. The benefit is usually significant, since the failure of one of these tests allows one to discard the corresponding reference event and it avoids running on that reference event all the automata corresponding to candidate event types.

The cost/benefit trade-off of step 4 is essentially measured in terms of the number and type of automata that must be run for each reference event. Since this is the crucial step of our discovery process, we conducted extensive experiments to analyze the process behavior.

6.1 Experimental Results on the Discovery Process

In this section, we report some of the experimental results conducted on a real data set. 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 was the closing prices of 439 stocks for 517 trading days during the period between January 3, 1994, and January 11, 1996.[6] For each of the 439 trading companies in the data set, we calculated the price change percentages by using 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 were then partitioned into seven categories: (-∞, −5 percent], (−5 percent, −3 percent], (−3 percent, 0 percent), [0 percent, 0 percent], (0 percent, 3 percent), [3 percent, 5 percent), and [5 percent, ∞). We took each event type as characterizing a specific category of price change for a specific company. The total number of event types in the data set was 2,978 (instead of $3{,}073 = 7 * 439$ since not all of the 439 stocks had price changes in all the seven categories during the period). There were 517 business days in the period, and our event sequence consisted of 181,089 events, with an average of 350 events per business day (instead of 439 events every business day since some stocks started or stopped exchanging during the period).

Fig. 4 shows the event structure S that we used in our experiments. The reference event type for X_0 is the event type corresponding to a drop of the IBM stock of less than 3 percent (i.e., the category (−3 percent, 0 percent)). There are no assignments of event types to variables X_1, X_2, and X_3. The minimum confidence value we used was 0.7 (i.e., the minimum frequency is 70 percent) except for the last experiment where we test the performance of the heuristics under various minimum confidence values. The data mining task was to discover all the combinations of frequent event types E_1, E_2, and E_3 with the constraints that

1) E_1 occurred after E_0 but within the same or the next two business days,
2) E_2 occurred the next business day of E_1 or the business day after, and
3) E_3 occurred after E_2 but in the same business week of E_2.

The choices we made for the reference type and the constraints were arbitrary and the results regarding the performance of our heuristics should apply to other choices.

The machine we used in the experiments was a Digital AlphaServer 2100 5/250, Alpha AXP symmetric multiprocessing (SMP) PCI/EISA-based server, with three 250 MHz CPUs (DECchip 21164 EV5) and four memory boards (each is 512 MB, 60 ns, ECC; total memory is 2,048 MB). The operating system was a Digital UNIX V3.2C.

We started our experiments to see the behavior of pattern matching under a different number of candidate types. We arbitrarily chose 82,088 candidate types derived from the event structure shown in Fig. 4 and performed eight runs against 1/8 to 8/8 of these candidate types. Fig. 5 shows the timing results. It is clear that the execution time is linear with respect to the number of candidate types. (This is no surprise since each candidate type is checked independently in our program. How to exploit the commonalities among candidate types to speed up the pattern matching is a further research issue.) By observing the graph, we found that in this particular implementation, the

5. The theoretical upper bound in [6], while polynomial, is much higher.
6. The complete data file is available from the authors.

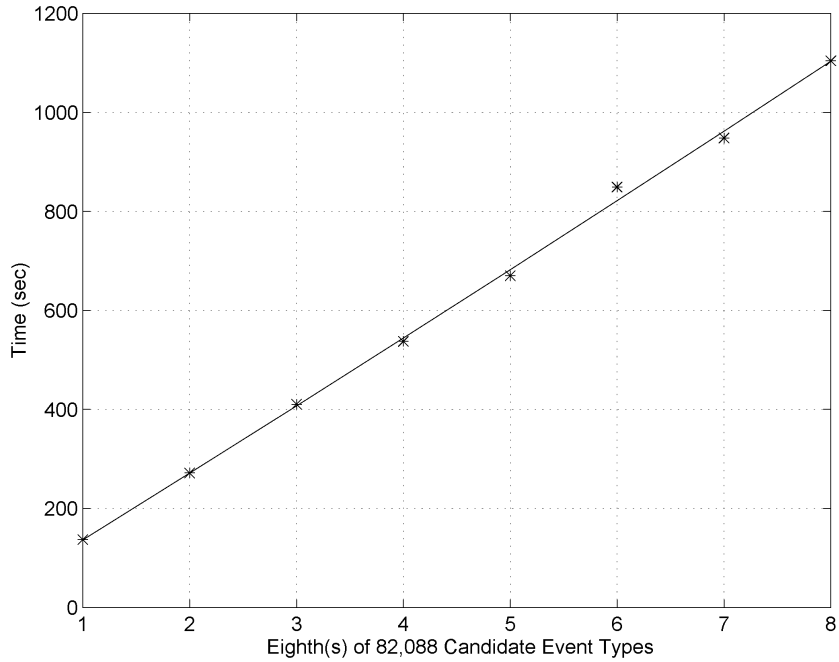


Fig. 5. Timing is linear with respect to the number of candidate event types.

number of candidate types we can handle within a reasonable amount of time, say in five hours of CPU time under our rather powerful environment, is roughly 10 million candidate types. As a reference point, we extrapolated from the graph that using the naive algorithm, which tries all possible $2{,}978^3$ (or roughly 26 *billion*) candidate types, the time needed is more than 10 *years*!

In the next experiment, we focused our attention on the reduction of the candidate event types by using substructures. The experiment was to test whether discovering substructures helps to reduce the number of candidate event types and thus to cut down the total computation time. We display our detailed results in Table 1. The second column of Table 1 shows the induced substructures considered at each stage of our discovery process. We explored six substructures before the original one (shown as stage 7 in the table).[7]

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 nonroot variable ($2{,}978^s$ if s is the number of nonroot variables).

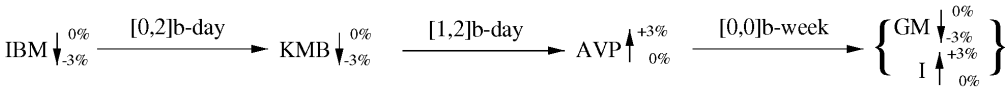
The fourth column shows the number of candidate event types under our heuristics. The basic idea is to use the previous stages to screen out event types (or combination of event types) that are not frequent. By Table 1, the number of candidate event types under our heuristics is much smaller than that under the naive algorithm 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 event types we needed to consider in Stage 4 is 152,456 ($= 1 * 323 * 472$), instead of 8,868,484 ($= 1 * 2{,}978 * 2{,}978$). Thus, we only needed to consider 2 percent of the event types required under the naive algorithm. The number of candidate event types for the original event structure we needed to consider in the last stage was only 82,088, instead of $2.64 * 10^{10}$. The total number of candidate types to be considered using our heuristics was 325,216.

In the experiment, the first three substructures we explored were those with a single nonroot variable. We found frequent event types for each induced substructure. The next stage (Stage 4) was the one with variables X_0, X_1, and X_2. The number of complex event types was 267, while the single event types for X_1 and X_2 were only 59 and 70, respectively. Hence, in stage 5, we only needed to consider as candidate event types 42,480 ($= 1 * 59 * 720$) different event types, instead of 232,560 ($= 1 * 323 * 720$) or even 8,868,484 ($= 1 * 2{,}978 * 2{,}978$). Similarly, we found in stage 5 that the number of event types for X_3 was 587. In stage 6, we only needed to consider those combinations of event types e_2 and e_3 with the condition that there existed e_1 such that (e_1, e_2) was frequent in stage 4 and (e_1, e_3) was frequent in stage 5. We only found 39,258 candidate event types. The number of candidate event types in the last stage was 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 would be considered as a candidate event type if and only if (e_1, e_2) appeared in the result of stage 4, (e_1, e_3) in stage 5, and (e_2, e_3) in stage 6.

7. 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`) labeling 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.

TABLE 1
REDUCTION OF CANDIDATE EVENT TYPES

Stage	Substructure	No. of candidate types		No. of frequent event types	Time (sec.)
		Naive algorithm	Using heuristics		
1	X0 —[0,2]b-day→ X1	2,978	2,978	323	28.3
2	X0 —[1,4]b-day→ X2	2,978	2,978	472	39.3
3	X0 —[1,8]b-day→ X3	2,978	2,978	720	54.6
4	X0 —[0,2]b-day→ X1 —[1,2]b-day→ X2	8,868,484	152,456	267	1751.1
5	X0 —[0,2]b-day→ X1 —[1,6]b-day→ X3	8,868,484	42,480	28,371	590.2
6	X0 —[1,4]b-day→ X2 —[0,0]b-week→ X3	8,868,484	39,258	18,042	516.0
7	X0 —[0,2]b-day→ X1, X1 —[1,2]b-day→ X2, X2 —[0,0]b-week→ X3	$2.64*10^{10}$	82,088	2	1104.5
Total			325,216		4084.0



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%)

Fig. 6. The two frequent event combinations discovered in the experiment.

The fifth column gives the number of (complex) event types discovered which are frequent (with minimum confidence 0.7). These event types were used in later stages to screen out event types as explained above.

Finally, the sixth column gives the number of seconds used in the discovery process for each stage and the total.

Fig. 6 shows the two complex event types found in the last stage.

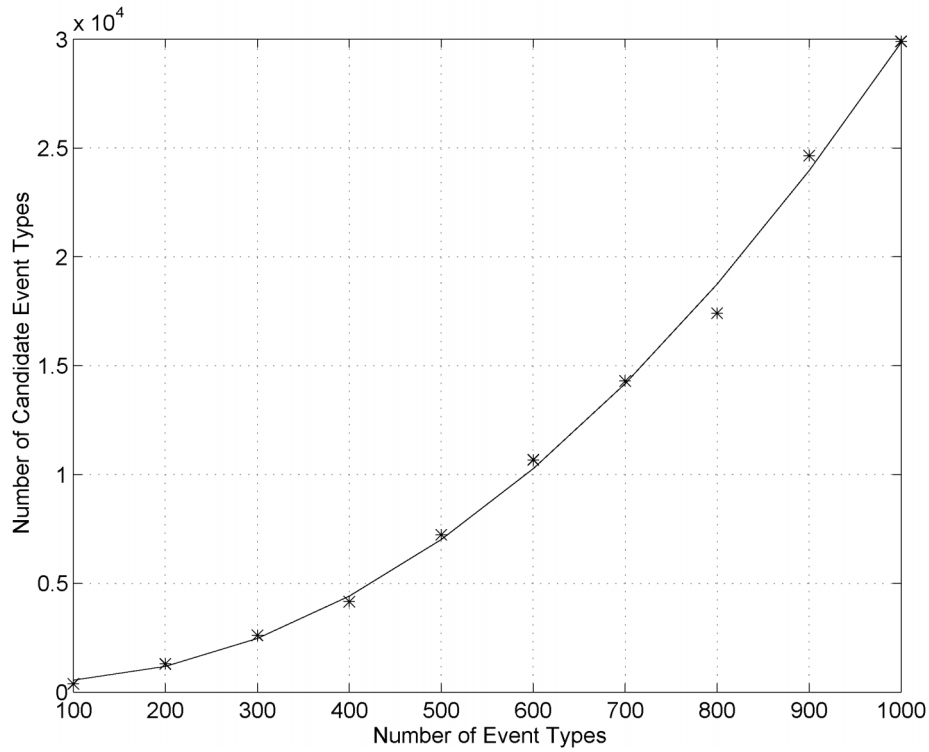


Fig. 7. Growth of candidate event types.

In our last experiment we varied the total number of event types that we have to consider for each variable. We executed 10 runs which assumed, respectively, that each variable can take any one in the given set of 100, 200, …, 1,000 event types. (Thus, under the naive algorithm, we have to consider 100^3, 200^3, …, $1{,}000^3$ combinations of event types, respectively.) The goal of this experiment is to see the behavior of our heuristics with respect to the number of event types in the data set, i.e., "input" event types. Fig. 7 shows the number of candidate event types our algorithm has to consider under different numbers of input event types. A similar curve for the naive algorithm is too steep to be represented in the same graph. Indeed, 10^7 candidate event types have to be considered for 200 input event types. The quadratic fitting curve in Fig. 7 is defined by the equation[8]

$$y = 0.03x^2 - 3.51x + 568.1$$

while the curve describing the behavior of the naive algorithm is $y = x^3$. We observe that

1) there is a significant difference between the numbers of candidate types to be considered, and
2) the growth of the number of candidate types under our heuristics is slower than the growth under the naive algorithm.

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., [15], [12], [7]). The granularities expressible in these languages are all instances of our temporal types. Furthermore, software packages that implement calendars are available [16]. 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 event types 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 optimization of data mining algorithms when an interactive user interface is used. As mentioned in the introduction, we believe that the complex temporal patterns are used only after the user explores the data set using simpler temporal patterns. That is, temporal patterns "evolve" from simple ones to complex ones. Hence, optimization strategies that exploit such an

8. If the cubic fitting is used, the coefficient of the term x^3 is negligible.

evolutionary pattern specification process will be an interesting research topic.

APPENDIX A
DERIVING IMPLICIT CONSTRAINTS WITH GRANULARITIES

We consider here an approximate algorithm for checking consistency and deriving implicit constraints. We proved in [5] that it is NP-hard to decide if an arbitrary event structure is consistent. Hence, it is not likely that the tightest possible implicit constraints can be computed in polynomial time (since an event structure is inconsistent iff each tightest constraint is *false*, i.e., unsatisfiable), and this motivates the choice of an approximate algorithm. The approach we take is called constraint propagation. However, traditional techniques for constraint propagation (e.g., [8]) 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 $[m, n]\mu_1$ of an event structure $\mathcal{S}$ into an implied (i.e., looser) constraint in terms of a granularity μ_2. If we only have granularities—like, e.g., `minute`, `hour`, and `day`—which have fixed conversion factors among them, then the conversion algorithm is trivial. However, if there are types with no fixed conversion factors, incomparable types like `week` and `month`, and/or types with "gaps" like `b-day`, the conversion becomes more complex. For example, it is easy to convert [0, 0]`day` into [0, 23]`hour`, but less trivial from [1, 6]`b-day` into [0, 2]`week`.

In Section 5, we explained that certain conversions are not allowed since no constraints in the target granularity may be implied by the ones in the given event structure. To ensure that only allowed conversions are performed by the algorithm, we impose a condition on the target granularity of the conversion. Given an event structure, we assign to each variable X a temporal type μ_X obtained as the glb[9] of all the temporal types appearing in TCGs involving X. Then, a TCG on variables X and Y can be converted in terms of a target granularity ν only if ν covers a span of time equal or larger than the span of time covered by μ_X and by μ_Y. For example, if μ_X = `b-week` and μ_Y = `week` we can convert a TCG on X and Y, into a TCG in terms of `week` or `month`, but not into a constraint in terms of `week-ends` or `b-day`.

Even when the above condition is guaranteed, an algorithm to perform conversions into *equivalent* constraints does not exist. Indeed, consider a structure with two event variables X and Y with the TCG $[m, n]$`b-day`. Replacing this constraint with any conversion in terms of `seconds` would result in an event structure where the information specifying that the events for X and Y must occur in a business day is lost. That is, the two event structures would not be equivalent. Hence, we are satisfied with a conversion algorithm that allows us to obtain an implied structure in the target granularity.

Several algorithms can be used to perform conversions into implied constraints. The tighter are the resulting constraints, the better is the precision of the algorithm. Different approximations can be obtained depending on the information available on the structure of the granularities and their relationships. We do not provide a specific one here, but we refer the interested reader to [6] where one approximate conversion algorithm can be found.

A.2 An Approximate Algorithm for TCGs Propagation

Let $\mathcal{S} = (W, A, \Gamma)$ be an event structure and M the set of temporal types appearing in Γ. 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 C_μ be the set of all the constraints $X - Y \in [m, n]$, where X, Y are in W and $[m, n]\ \mu \in \Gamma(X, Y)$. Now, the propagation within C_μ is a problem known as the Simple Temporal Problem [8]. We apply the path consistency algorithm [8] 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 C_μ into one in terms of ν and add it into C_ν. 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 matches the given event structure $\mathcal{S} = (W, A, \Gamma)$, then it also matches $\mathcal{S}' = (W, A', \Gamma')$, where A' and Γ' are given by the algorithm (e.g., if $X - Y \in [m, n]$ is in C_μ, then $(Y, X) \in A'$ and $[m,n]\ \mu \in \Gamma'(Y,X)$).

The aforementioned algorithm is an *approximate* propagation for two reasons. First, translation between groups cannot be done precisely since the conversion among granularities is not always precise. 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.

As shown in [6], the proposed algorithm is sound, terminates, and requires time polynomial in the size of the constraint graph.

APPENDIX B
PROOFS

THEOREM 1. *Given a complex event type $\mathcal{T}$, there exists a timed automaton with granularities $TAG_{\mathcal{T}}$ such that $\mathcal{T}$ occurs in an event sequence σ iff $TAG_{\mathcal{T}}$ has an accepting run over σ. This automaton can be constructed by a polynomial-time algorithm.*

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

9. In Section 2, we pointed out that the set of temporal types forms a lattice with respect to the finer-than relation. It follows that for any finite set of temporal types there exists a type that is their greatest lower bound.

INPUT: A complex event type $\mathcal{T} = (\mathcal{S}, \varphi)$, where $\mathcal{S} = (W, A, \Gamma)$ and φ is a mapping assigning to each variable the corresponding event type.

OUTPUT: A TAG such that an event sequence σ is accepted by the TAG iff the complex event 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 $X_1, \ldots, X_{n_l}$ (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 $\wedge_{[m,n]\mu \in \Gamma(X_{j-1}, X_j)}(m \le x_\mu^l \le n)$. 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 $S = \{s^1 \cdots s^k \mid s^l \in S^l \text{ for each } l\}$ 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.[10] 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 $\text{TAG}_{\mathcal{T}}$ 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 nonsimultaneous 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. □

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.*

PROOF. The TAG obtained from a complex event type is nondeterministic. We simulate the nondeterministic TAG using a standard technique presented in [3, 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 nonstarting state to the same state (i.e., loops), labeled with all the event types as input symbols. Hence, it will be nondeterministic, since another outgoing transition labeled with one of the event types will also be present. If a transition from state $S1$ to state $S2$ is labeled with an event type, a constraint $k_1 \le x_\mu \le k_2$, 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

10. 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 labeled.

($|V|$) 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 type 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 $O(|\sigma| * (|S| * min(|\sigma|, (|V| * K)^p))^2)$. □

ACKNOWLEDGMENTS

This work was supported by the National Science Foundation (NSF) under Grant No. IRI–9633541. The work of Claudio Bettini was partially carried out while he was visiting George Mason University. The work of X. Sean Wang and Sushil Jajodia was also supported by NSF Grant No. IRI–9409769 and Grant No. INT–9412507, respectively. The work of Jia-Ling Lin was also partially supported by a George Mason University Doctoral Fellowship Award.

REFERENCES

[1] R. Agrawal, T. Imielinski, and A. Swami, "Database Mining: A Performance Perspective," *IEEE Trans. Knowledge and Data Eng.*, vol. 5, no. 5, pp. 914–925, 1990.

[2] R. Agrawal and R. Srikant, "Mining Sequential Patterns," *Proc. Int'l Conf. Database Eng.*, pp. 3–14, IEEE, 1995.

[3] A.V. Aho, J.E. Hopcroft, and J.D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, 1974.

[4] R. Alur and D.L. Dill, "A Theory of Timed Automata," *Theoretical Computer Science*, vol. 126, pp. 183–235, 1994.

[5] C. Bettini, X. Wang, and S. Jajodia, "Testing Complex Temporal Relationships Involving Multiple Granularities and Its Application to Data Mining," *Proc. PODS 15, ACM SIGACT-SIGMOD-SIGART Symp. Principles of Database Systems*, pp. 68–78, ACM Press, New York, 1996.

[6] 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, vol. 22, nos. 1-2, pp. 29-58.

[7] R. Chandra, A. Segev, and M. Stonebraker, "Implementing Calendars and Temporal Rules in Next Generation Databases," *Proc. Data Eng. Conf.*, 1994.

[8] R. Dechter, I. Meiri, and J. Pearl, "Temporal Constraint Networks," *Artificial Intelligence*, vol. 49, pp. 61–95, 1991.

[9] T.G. Dietterich and R.S. Michalski, "Discovering Patterns in Sequences of Events," *Artificial Intelligence*, vol. 25, pp. 187–232, 1985.

[10] W. Dreyer, A.K. Dittrich, and D. Schmidt, "Research Perspectives for Time Series Management Systems," *SIGMOD Record*, vol. 23, no. 1, pp. 10–15, Mar. 1994.

[11] P. Laird, "Identifying and Using Patterns in Sequential Data," *Proc. Fourth Int'l Workshop Algorithmic Learning Theory*, pp. 1–18, Springer-Verlag, 1993.

[12] B. Leban, D. Mcdonald, and D. Foster, "A Representation for Collections of Temporal Intervals," *Proc. AAAI, Nat'l Conf. Artificial Intelligence*, pp. 367–371, Morgan Kaufmann, Los Altos, Calif., 1986.

[13] H. Mannila, H. Toivonen, and A.I. Verkamo, "Discovering Frequent Episodes in Sequences," extended abstract, *Proc. First Conf. Knowledge Discovery and Data Mining*, AAAI Press, Menlo Park, Calif., pp. 210-215, 1995.

[14] R. Marceca, "Temporal Reasoning with Multiple Granularity Constraint Networks," master's thesis, in Italian, DSI, Univ. of Milan, Italy, 1996.

[15] M. Niezette and J. Stevenne, "An Efficient Symbolic Representation of Periodic Time," *Proc. CIKM*, Baltimore, Md., Nov. 1992.

[16] M.D. Soo, "Multiple Calendar Support for Conventional Database Management Systems," R.T. Snodgrass, ed., *Proc. Workshop Infrastructure for Temporal Databases*, pp. FF1–FF17, June 1993.

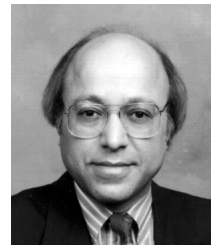
[17] X. Wang, C. Bettini, A. Brodsky, and S. Jajodia, "Logical Design for Temporal Databases with Multiple Granularities," *ACM Trans. Database Systems*, vol. 22, no. 2, pp. 115–170, June 1997.

[18] J.T.-L. Wang, G. Chirn, T.G. Marr, B.A. Shapiro, D. Shasha, and K. Zhang, "Combinatorial Pattern Discovery for Scientific Data: Some Preliminary Results," *Proc. SIGMOD Conf.*, pp. 115–125. ACM Press, May 1994.

[19] K. Wang and J. Tan, "Incremental Discovery of Sequential Patterns," *Proc. Workshop Research Issues on Data Mining and Knowledge Discovery*, in cooperation with ACM-SIGMOD '96 and IRIS/Precarn, Montreal, Quebec, Canada, June 1996.

Claudio Bettini received an MS degree in information sciences in 1987 and a PhD degree in computer science in 1993, both from the University of Milan, Italy. He has been an assistant professor in the Department of Information Science of the University of Milan since 1993. His main research interests include temporal logics, description logics, temporal reasoning in knowledge and databases, and temporal aspects of database security. He has published several conference and journal papers on these topics. He was a visiting researcher at IBM Kingston, New York, in 1988-89 and at George Mason University, Fairfax, Virginia, in 1994-96. Dr. Bettini is a member of the ACM and the IEEE.

X. Sean Wang received his PhD degree in computer science from the University of Southern California in 1992 and, since then, has been an assistant professor in the Department of Information and Software Systems Engineering of George Mason University, Fairfax, Virginia. His main research interests include database theory, query languages and query optimization, databases with sequential data such as temporal and sequence, as well as multidimensional databases. Dr. Wang is a member of the ACM and the IEEE Computer Society. The URL for his web page is http://www.isse.gmu.edu/faculty/xywang.



Sushil Jajodia received his PhD degree from the University of Oregon, Eugene. He is director of the Center for Secure Information Systems and a professor of information and software systems engineering at George Mason University, Fairfax, Virginia. He joined GMU after serving as director of the Database and Expert Systems Program at the National Science Foundation. Before that, he was head of the Database and Distributed Systems Section at the Naval

Research Laboratory, Washington, D.C., and associate professor of computer science and director of graduate studies at the University of Missouri, Columbia. He has also been a visiting professor at the University of Milan, Italy, and at the Isaac Newton Institute for Mathematical Sciences, Cambridge University, England.

Dr. Jajodia's research interests include information security, temporal databases, and replicated databases. He has published more than 200 technical papers in refereed journals and conference proceedings and has edited 10 books, including *Advanced Transaction Models and Architectures* (Kluwer, 1997), *Multimedia Database Systems: Issues and Research Directions* (Springer-Verlag Artificial Intelligence Series, 1996), *Information Security: An Integrated Collection of Essays* (IEEE Computer Society Press, 1995), and *Temporal Databases: Theory, Design, and Implementation* (Benjamin/Cummings, 1993). He received the 1996 Kristian Beckman award from IFIP TC 11 for his contributions to the discipline of information security.

Dr. Jajodia has served in different capacities for various journals and conferences. He is the founding co-editor-in-chief of the *Journal of Computer Security*. He is a member of the editorial board of *IEEE Concurrency* and the *International Journal of Cooperative Information Systems,* and is a contributing editor of *Computer and Communication Security Reviews*. He serves as the program chair of the 1998 IFIP WG 11.5 Working Conference on Integrity and Control in Information Systems and 1998 IFIP WG 11.3 Working Conference on Database Security. He has been named a Golden Core member for his service to the IEEE Computer Society. He is a past chair of the IEEE Computer Society Technical Committee on Data Engineering and the Magazine Advisory Committee. He is a senior member of the IEEE, and a member of the IEEE Computer Society and the Association for Computing Machinery. The URL for his web page is http://www.isse.gmu.edu/~csis/faculty/jajodia.html.

Jia-Ling Lin received her BA degree in management information systems from the National Chengchi University, Taipei, Taiwan, Republic of China, in 1990; and her MS degree in information systems from George Mason University, Fairfax, Virginia, in 1993. She is now a doctoral candidate in information technology and engineering, and a research assistant with the Center for Secure Information Systems at GMU. Her research interests include information security, data mining, and temporal databases. The URL for her web page is http://www.isse.gmu.edu/~jllin.