Discovering Temporal Patterns for Interval-based Events

Po-shan Kam and Ada Wai-chee Fu

Department of Computer Science and Engineering
The Chinese University of Hong Kong
{pskam,adafu}@cse.cuhk.edu.hk

1 Introduction

In many daily transactions, the time when an event takes place is known and stored in databases. Examples range from sales records, stock exchange, patient records, to scientific databases in geophysics and astronomy. Such databases incorporate the concept of time which describes when an event starts and ends as historical records [9]. The temporal nature of data provides us with a better understanding of the trend or pattern over time. In market-basket data, we can have a pattern like "75% of customers buy peanuts when butter *starts* to be in big sales and *before* bread is sold out". We observe that there may be some correlations among peanuts, butter and bread so that we can have better planning for marketing strategy. Knowledge discovery in temporal databases thus catches the attention of researchers [8,4].

Previous work for knowledge discovery in temporal data includes [2,6,5]. These techniques only treat data as series in chronological order and mainly support point-based events. Therefore, the physical ordering of events would be quite simple and the expressive power in specifying temporal relations such as *during, overlaps*, etc. is limited. When we deal with temporal data for events which last over a period of time, besides parallel and serial ordering of event sequences, we may find more interesting temporal patterns. For instance, patterns like "event A occurs *during* the time event B happens" or "event A's occurrence time *overlaps* with that of event B and both of these events happen *before* event C appears" cannot be expressed as simple sequential orders. On the other hand, temporal logic [7] is suggested to be used for expressing temporal patterns defined over categorical data. Temporal operators such as *since, until* and *next* are used. We may have patterns like "event A always occurs *until* event B appears". Simple ordering of events is considered.

In this paper, we consider interval-based events where the duration of events is expressed in terms of endpoint values, and these are used to form temporal constraint in the discovery process. We introduce the notion of temporal representation which is capable of expressing the relationships between interval-based events. We develop new methods for finding such interesting patterns.

2 Problem Statement

Suppose we are given a temporal database $\mathcal{D}$, which stores a list of clinical records. Each record contains a person-id, name of disease and a pair of ordered time points t_s and t_e where $t_s \leq t_e$ and both are integers. They are the **start time** and **end time**, indicating the period during which the patient contracted certain kind of disease.

A record may have other attributes, for simplicity, we consider here a single temporal attribute and denote it as an **event**. We assume a set $\mathbf{E}$ of event types. An event E has an associated time of occurrence and it is specified by a triple (A, t_s, t_e), where $A \in \mathbf{E}$ is an event type and t_s and t_e are the start time and the end time, respectively. We also use $E.t_s$ and $E.t_e$ to indicate these times.

We assume that each event is associated with one person. A **sequence** of events is defined as a list of events where each event is associated with the same person such that for person j, we have the following sequence s_j, where $s_j = \langle (A_1, t_{s_1}, t_{e_1}), (A_2, t_{s_2}, t_{e_2}) ... (A_n, t_{s_n}, t_{e_n}) \rangle$. The events are ordered by the end times where $t_{e_i} \leq t_{e_{i+1}}$ for all $i = 1, ..., n-1$.

Relation	Relation for Inverse	Pictorial Example	Endpoints Constraint
X before Y	Y after X	X Y	$X.t_e < Y.t_s$
X equal Y	Y equal X	X Y	$X.t_s = Y.t_s$ $X.t_e = Y.t_e$
X meets Y	Y met-by X	X Y	$X.t_e = Y.t_s$
X overlaps Y	Y overlapped-by X	X Y	$X.t_s < Y.t_s$ $X.t_e > Y.t_s$ $X.t_e < Y.t_e$
X during Y	Y contains X	X Y	$X.t_s > Y.t_s$ $X.t_e < Y.t_e$
X starts Y	Y started-by X	X Y	$X.t_s = Y.t_s$ $X.t_e < Y.t_e$
X finishes Y	Y finished-by X	X Y	$X.t_s > Y.t_s$ $X.t_e = Y.t_e$

Fig. 1. The thirteen possible relationships between two intervals X and Y

Relationship among time intervals can be described in different ways. **Allen's taxonomy of temporal relationships** [3] is adopted to describe the basic relationships among events and summarized in Figure 1. The relations between intervals can be expressed in terms of relations between their endpoints, we call these the *endpoint constraints*. For instance, consider the sequence $\langle E_1$=(A,5,10), E_2 =(B,8,12)$\rangle$, we have "A overlaps B" since $E_1.t_s < E_2.t_s$, $E_1.t_e > E_2.t_s$ and $E_1.t_e < E_2.t_e$.

These thirteen relationships can be used to express any relationship held between two intervals and they provide a basis for the description of temporal patterns. Some of the relations are mirror image of the other, for example, "X overlaps Y" implies the same relation as "Y is overlapped-by X". We only focus on seven primitive temporal relations with the order of items preserved. The seven relations are shown in the shaded area in Figure 1. Let us call this set of seven temporal relations **Rel**. We obtain binary temporal predicates if we consider two events only. To express complex relation among more events in a sequence, we have the following definitions:

Definition 1. *A* **temporal pattern** *is defined recursively as follows:*

- *If E is a single event type in* **E**, *then E is a temporal pattern, it is also called an* **atomic pattern**.
- *If X and Y are two temporal patterns, and $rel \in$* **Rel**, *then $(X\ rel\ Y)$ is also a temporal pattern. This is a* **composite pattern**.

The **size** *of a temporal pattern is the number of atomic patterns in the pattern.*

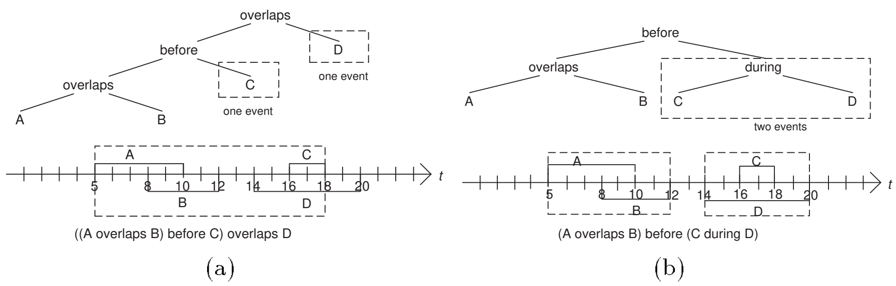
Definition 2. *An atomic temporal pattern X has a* **mapping** *in a sequence S if we can find an event E of type X in the sequence. We denote this mapping by $\mathcal{M}(X,S) = \{E\}$. We associate a time duration to the mapping as follows: $\mathcal{M}(X,S).t_s = E.t_s$, $\mathcal{M}(X,S).t_e = E.t_e$ We say that X* **holds** *in S.*

A composite pattern $(X\ rel\ Y)$ in which Y is an atomic pattern and $rel \in$ **Rel** *has a* **mapping** *$\mathcal{M}((X\ rel\ Y),S)$ in a sequence S if X has a mapping $\mathcal{M}(X,S)$ in S and we can find an event $E \notin \mathcal{M}(X,S)$ of type Y in S to be mapped as $\mathcal{M}(Y,S)$ such that if we consider some imaginary event Z with start time of $\mathcal{M}(X,S).t_s$ and end time of $\mathcal{M}(X,S).t_e$, then $Z\ rel\ E$ is true.*

In this case, $\mathcal{M}(X\ rel\ Y,S) = \mathcal{M}(X,S) \cup \{E\}$. We say that the relation $(X\ rel\ Y)$ **holds** *in S. The mapping $\mathcal{M}((X\ rel\ Y),S)$ has an associated time interval given by*

$$\mathcal{M}((X\ rel\ Y),S).t_s = min\{\mathcal{M}(X,S).t_s, \mathcal{M}(Y,S).t_s\}$$
$$\mathcal{M}((X\ rel\ Y),S).t_e = \mathcal{M}(Y,S).t_e$$

In the above mapping of a composite pattern in a sequence, **union** of two time intervals takes place. For example, for the sequence shown in Figure 2(a), we obtain a composite pattern "(A overlaps B) before C" with the resultant interval being [5,18].



(a) (b)

Fig. 2. Different temporal pattern representation

We can see that in Figure 2, two different composite patterns "((A overlaps B) before C) overlaps D" and "(A overlaps B) before (C during D)" hold in the sequence by considering different ways of combinations. The number of such possible descriptions is exponential in the sequence size. We restrict our interest to temporal patterns of the form $((...(A_1\ rel_1\ A_2)\ rel_2\ A_3)...\ rel_{k-1}\ A_k)$. We call these the $A1$ **temporal pattern**.

There are two main reasons for this restriction: (1) We believe that the temporal relations give some insight into causal relationships. As such, when a few events have happened, together they may become the cause of a following event. (2) Discovering all possible patterns can be computationally inhibitive and also the amount of results to the user can be overwhelming. We have implemented the mechanism to discover temporal pattern obtained by appending composite pattern of size two at a time as shown in Figure 2(b). Experimental results show that even a small extension in this way would lead to a much increase in computational cost.

Moreover, user can specify the maximum length of time interval that is of interest, known as the **window-size**, *win_size*. The intuition is that if some events do not happen close enough to each other, we would not be interested to find any correlation between them. If w is a given window-size, a temporal pattern P **holds within** w in a sequence S if there is a mapping $\mathcal{M}(P,S)$ such that $\mathcal{M}(P,S).t_e - \mathcal{M}(P,S).t_s \leq w$.

Definition 3. *Let A_i, $i = 1, ..., k$ be a bag of k event types in* $\mathbf{E}$, *$rel_i \in$* $\mathbf{Rel}$, *i = 1,..,k-1, a* ***k*-item** *has the form $\{\{A_1, A_2, ...A_k\}, \{rel_1, rel_2, ...rel_{k-1}\}, \mathcal{P}\}$, where $\mathcal{P}$ is a temporal pattern in terms of the events types $A_1, A_2, ..., A_k$ and the relations $rel_1, ..., rel_{k-1}$, and $k \geq 1$.*

Given a window-size w, let M be a subset of the set of events in a sequence S, M **supports** an k-item $\{\{A_1, A_2, ...A_k\}, \{rel_1, rel_2, ...rel_{k-1}\}, \mathcal{P}\}$ if M is a mapping for the temporal pattern $\mathcal{P}$ and $M.t_e - M.t_s \leq w$.

The **support** of a temporal pattern $\mathcal{P}$ in a set of sequences $\mathcal{D}$ is defined as

$$support(\mathcal{P}, \mathcal{D}) = \frac{|\{M \subseteq S | S \in \mathcal{D}, M\ supports\ \mathcal{P}\}|}{|\mathcal{D}|}$$

The **support** of a k-item is the support of the temporal pattern in the k-item.

A **large *k*-item** is a k-item having support greater than a threshold, namely *min_sup* provided by users, that is, $support(\mathcal{P}, \mathcal{D}) \geq min\_sup$. Our aim is to find the large k-items for all k.

3 Mining A1 Temporal Patterns

Here we propose a method for mining the frequent $A1$ temporal patterns as shown in Figure 2(a). Consider a patient database shown in Table 1. Each tuple contains a person-id, the disease contracted by the patient and the duration of the disease. The database can be used to find if some diseases are likely to cause

person-id	disease	start	end
1	A	5	10
1	B	8	12
1	D	14	20
1	C	16	18
1	B	17	22
2	A	8	14
2	B	9	15
2	D	16	22
2	C	19	21
3	A	12	20
3	B	22	25
3	D	28	32
3	C	29	31

Original database

person-id	seq-list
1	(A,5,10),(B,8,12),(D,14,20), (C,16,18),(B,17,22)
2	(A,8,14),(B,9,15), (D,16,22),(C,19,21)
3	(A,12,20),(B,22,25), (D,28,32),(C,29,31)

seq_list

item	A	B	D	C
	(1,5,10)	(1,8,12)	(1,14,20)	(1,16,18)
	(2,8,14)	(1,17,22)	(2,16,22)	(2,19,21)
	(3,12,20)	(2,9,15)	(3,28,32)	(3,29,31)
		(3,22,25)		

item_list

Table 1. Transform the database as *seq_list* and *item_list*

some other diseases and their temporal relations. It is assumed that the *min_sup* is 66% and the *win_size* is set to be 100 time units. We use a different layout of event sequence that used in finding sequential pattern [2]. Instead of transforming the original database into a list of sequences, the *seq_list*, where each sequence corresponds to one person, we use an *item_list* to represent the temporal data. Each event is associated with a list of person-id, start time and end time (pid, t_s, t_e). Tables 1 illustrates the differences between the two approaches. Note that each item is an atomic or composite pattern and the t_s and t_e in the *item_list* indicates an time interval for this pattern. Similar idea of transforming the database into the form which we called *item_list* is used in [10].

A basic strategic similar to the Apriori-gen approach [1] is used. With *seq_list*, we need to scan the database once in each iteration. The *item_list* approach avoids this problem since it enables us to count support by direct composition of the lists. The size of these lists and number of candidates shrink as the sequence length increases, which facilitates fast computation. This motivates us to choose *item_list* format to store the large k-items for efficient support counting.

The general structure of the method for the mining process is mainly divided into two phases: (1) **Candidate Generation** and (2) **Large k-items Generation**. It takes multiple passes over the data. In each pass, we start with a seed set of large items which is found in the previous pass. We use the seed set for generating new potentially large items, called candidates, C_k, by adding one atomic item to elements in the seed set. In the second phase, for each candidate, we examine the L_{k-1} and L_1 *item_lists* and determine the temporal relations between the composite pattern and atomic pattern that have sufficient supports. We then generate new large k-items and obtain the L_k *item_list* by merging the composite and atomic patterns with the temporal relation. The algorithm terminates when we cannot find any large k-items after the end of the current pass.

Candidate Generation:

The logical form of the candidates is shown in an example in Table 2. The candidate generation to obtain C_k from L_{k-1} is done by adding one large 1-item each time. We generate the candidates by combining the events in atomic patterns with those in composite patterns.

item	count	item	count	item	count
A overlaps B	2	A before B	2	C during D	3
(1,5,12)		(1,5,22)		(1,14,20)	
(2,8,15)		(3,12,25)		(2,16,22)	
				(3,28,32)	

Partial large 2-items

item 1	item 2
A overlaps B	C
A overlaps B	D
A before B	C
A before B	D
⋮	⋮

The 3-candidates

Table 2. Partial large 2-items and the 3-candidates

In each pass, we generate C_k from L_{k-1}. We prune out some irrelevant candidates as follows: let b_i be the $k-1$-item with $((..(a_{i1} rel_{r1} a_{i2})...rel_{rk-2} a_{ik-1})$ where $rel_{ri} \in$ Rel if $(a_{i1} rel_{ij} a_j) \in L_2$ or $(a_{ik-1} rel_{ij} a_j) \in L_2$ for any rel_{ij} then take the candidate element $\{b_i, a_j\}$. For instance, for the 2-item with a pattern of "C during D", we aim to find any temporal relation between the 2-item and event A. In the previous pass, we have found that no pattern of the form "C *rel* A" or "D *rel* A" is large, we therefore exclude the possibility of having the candidate of "(C during D)" with "A" in the generation of large 3-item.

Large k-Items Generation:

This phase is further divided into two subphases. They are the support counting phase and generation of large k-items. First, we need to find the supports for the candidates generated. We determine the number of sequences that support each temporal relation between the composite patterns in each candidate. We compare the endpoint values of elements in L_{k-1} and L_1 and determine if any temporal relation holds between the composite and atomic patterns. Large k-items are formed if their support is greater than the threshold given.

To facilitate efficient counting, we use a hash tree to store L_1 and also the relevant part of the 1-*item_list* and a hash tree to store L_{k-1} and part of the $(k-1)$-*item_list*. We use the value of the event as a key for hashing in the hash tree for L_1. For the hash tree for composite patterns, we use the values of all the events included and the temporal relations together to form a key by simple concatenation. The leaf nodes of the hash tree corresponds to some large k-item I and it also records the mappings for the pattern in I. The mappings are stored in the form of *item_list*, with the person-ids, start times and end times. For each candidate, we use the composite pattern and event as search keys and find from the hash trees the corresponding L_{k-1} and L_1 items.

During the search in the hash tree, hashing is done in the internal nodes until we reach the leaf nodes where we perform exact matching for the composite or atomic pattern. We consider a pattern P for a large $(k-1)$-item and a pattern P' for a large 1-item. We can identify any temporal relation that holds between a mapping for P and a mapping in P', since the start times and end times are recorded in the corresponding hash trees. If some composite pattern is found, the count for the candidate with respect to the specific temporal pattern is increased by 1. The counts are kept as auxiliary information to the table for the candidate items. There are seven counts for each candidate, one for each of the temporal relationships.

The second subphase is to form large k-items. Table 2 shows a partial set of the large 2-items. After identifying any temporal relation between the items in C_k, we generate L_k from L_{k-1} and L_1 with the corresponding temporal relation. Each new item in L_k is a composite pattern and is used for the next pass. The resultant interval is obtained from the union of two intervals of $(k-1)$-item and 1-item. For instance, as shown in Table 2, the start time and end time of the mapping of composite pattern "A overlaps B" are {[5,12], [8,15]}.

4 Mining Other Temporal Patterns

In the previous section, in mining the A1 temporal pattern, we generate large k-items by adding one atomic pattern in L_1 at a time. Here we consider a slightly more complex form of temporal patterns which we call A2 temporal pattern. The A2 temporal pattern is defined recursively as the following: (1) a temporal pattern of size 2 is an A2 temporal pattern. E.g. "A before B". (2) if X is an A2 temporal pattern, and Y is a temporal pattern of size 2, then $(X\ rel\ Y)$ where $rel \in \mathbf{Rel}$ is also an A2 temporal pattern. Example of such a composition is shown in Figure 2(b). The patterns we generate are in even number of events, i.e. $2k$-items. We therefore focus on temporal relations among events by adding one 2-item each time. We shall refer to the method in Section 3 as the **first method** and the method described in this section as the **second method**. The second method works similarly as the previous method except for the candidate generation phase and the formation of large $2k$-items.

Candidate Generation:

In the formation of C_2, the process is the same as before. Next we start to generate C_{2k}, where $k \geq 2$, we examine L_{2k-2} and L_2 and use compositions of the elements in the two *item_list* of L_{2k-2} and L_2. When we prune any irrelevant candidates in this phase, we need to consider two newly added atomic patterns this time, say a_{j1} and a_{j2}, where $a_{j1} rel_j a_{j2} \in L_2$. The two added items can be combined with a composite pattern, say b_i, where $b_i \in L_{2k-2}$ if both of the following conditions hold:

1. there is a relation between the leftmost atomic pattern of b_i and at least one of a_{j1} and a_{j2}.
2. there is a relation between the rightmost atomic pattern of b_i and a_{j1} or a_{j2}.

Large $2k$-Items Generation:

We also divide this into two phases namely support counting and the generation of large $2k$-items. In general, the second method works in the same manner as the first method in that we generate incrementally larger $2k$-items by combining the two composite patterns of a $2k$-candidate. The difference is that we shall use the *item_list* of L_{2k-2} and L_2.

5 Performance Evaluation

To evaluate the performance of the proposed methods over a large range of data, we conducted several experiments on an UltraSparc 5/270 workstation with 520MB of main memory. The two methods in Sections 3 and 4 are written in C. We consider synthetic data in an application domain of a medical database same as that of the given example. For each person we record a sequence of clinical records stating the different diseases contracted.

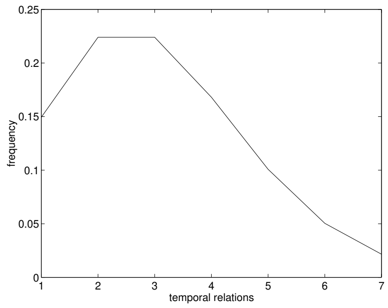
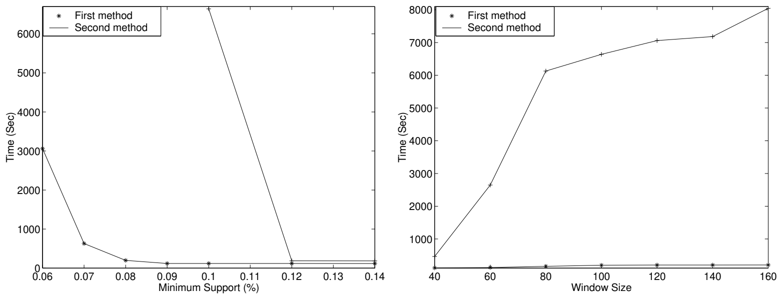


Fig. 3. Distribution of temporal relations between events

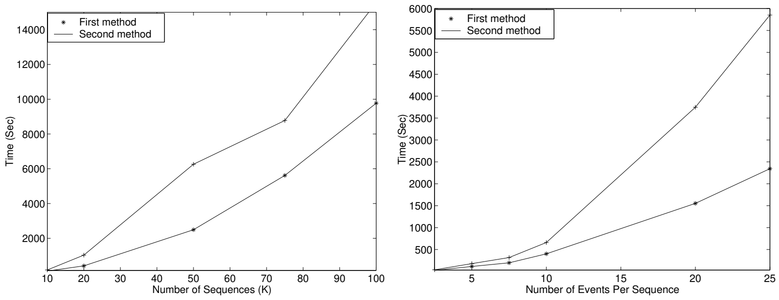
The synthetic data generation program took four parameters. They are number of sequences ($|D|$), average number of events per large item ($|P|$), number of maximal potentially large item, (N_S), and number of event types (N). The data generation model was based on the one used for mining sequential pattern [2] with some modification to model the patient database. We generated each large sequence by first picking a number of events from a Poisson distribution with mean μ equal to $|P|$ and we chose the event types randomly. We picked a temporal relation between events and formed a pattern. Temporal relations are chosen from the set **Rel**, according to a distribution shown in Figure 3. We used the values of {1,2,3,4,5,6,7} to represent *starts, overlaps, before, during, finishes, meets, equal* respectively. The distribution was determined arbitrarily by our intuitive expectation of the likeliness of the relations. Each person was then assigned a potentially large sequence. The time interval of each event followed an exponential distribution with mean μ equal to 5. For each sequence, the time where the first event took place was chosen randomly within the time interval [0,500] of time units. The following events of the item then followed the temporal relation held between events. We generated the dataset by setting N_S=2000, N=1000, $|D|$=10K and $|P|$=5 with 1MB of data. We studied the effect of different values of *min_sup*, *win_size*, number of sequences and events per sequence for the two methods.

First, we studied the effect of minimum support on the processing time. We used 7 values of minimum support (*min_sup*) as shown in Figure 4(a), and 100 time units for window size (*win_size*) for the test. It shows the execution time decreases when the minimum support increases for both methods. As the support threshold increases, less large items are generated and hence the size of the candidate set in each iteration decreases dramatically. Thus less time is required for support counting and hash tree searching of large items. Comparing the two methods, the second method needs much more time than that of the first method. This is due to large amount of computation time in pruning the candidates as the addition of two atomic patterns are considered instead of one.



(a) (b)

Fig. 4. Variation on minimum support and window size



(a) (b)

Fig. 5. Scale-up: Number of sequences & Number of events per sequence

We then studied the effect of window size on the processing time. We chose the values of min_sup being 0.0008 for the first method and 0.001 for the second method. In Figure 4(b), we can see that when the window size increases, the execution time increases for both methods. This is because more sequences are included and the time for support counting phase increases. Also the number of iterations increases and which also requires a longer execution time.

Our next target is to consider the scale-up effects. Figure 5(a) shows how both methods scale up as the number of sequences is increased ten-fold, ranging from 10K to 100K and with $min\_sup = 0.0025$ for both methods. The execution time for both methods increase with increasing number of sequences.

Finally we studied the scale-up as we increase the average number of events per sequence. The number of sequences used is 10K and kept constant. We vary the average number of events per sequence from 2.5 to 25. Figure 5(b) shows the scalability results of the two methods. We set $min\_sup = 0.0025$ for both methods. From the figure, the execution time increases rapidly with the increasing number of events per sequence.

In conclusion, we propose methods for discovering interesting temporal patterns for interval-based events. A set of experiments has been conducted to demonstrate the overall performance of the methods. From experiments, we find that the computation time required for the first pattern is much more acceptable.

References

1. R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proceedings of the 20th VLDB Conference*, pages 487–499, 1994.
2. R. Agrawal and R. Srikant. Mining sequential patterns. In *11th International Conf. on Data Engineering*, pages 3–14, March 1995.
3. J. F. Allen. Maintaining knowledge about temporal intervals. In *Communications of the ACM 26(11)*, pages 832–843, 1983.
4. X. Chen and I. Petrounias. An architecture for temporal data mining. In *IEE Colloqium on Knowledge Discovery and Data Mining*, pages 8/1–8/4, 1998.
5. H. Mannila and H. Toivonen. Discovering generalized episodes using minimal occurrences. In *2nd International Conf. on Knowledge Discovery and Data Mining*, pages 146–151, August 1996.
6. H. Mannila, H. Toivonen, and A. I. Verkamo. Discovering frequent episodes in sequences. In *1st International Conf. on Knowledge Discovery and Data Mining*, pages 210–215, August 1995.
7. B. Padmanabhan and A. Tuzhilin. Pattern discovery in temporal databases: A temporal logic approach. In *2nd International Conf. on Knowledge Discovery and Data Mining*, pages 351–354, August 1996.
8. M. H. Saraee and B. Theodoulidis. Knowledge discovery in temporal databases. In *IEE Colloquium on Digest No. 1995/021(A)*, pages 1/1–1/4, 1995.
9. A. Tansel, J. Clifford, S. Gadia, S. Jajodia, A. Segev, and R. Snodgrass. *Temporal Databases: Theory, Design, and Implementation*. Benjamin/Cummings, 1993.
10. M. J. Zaki. Fast mining of sequential patterns in very large databases. Technical report, Technical Report 668 of the Department of Computer Science, University of Rochester, Nov 1997.