Available online at www.sciencedirect.com

ScienceDirect

Data & Knowledge Engineering 63 (2007) 76–90

DATA & KNOWLEDGE ENGINEERING

www.elsevier.com/locate/datak

ARMADA – An algorithm for discovering richer relative temporal association rules from interval-based data

Edi Winarko, John F. Roddick [*]

School of Informatics and Engineering, Flinders University, P.O. Box 2100, Adelaide, South Australia 5001, Australia

Received 13 October 2006; received in revised form 13 October 2006; accepted 13 October 2006
Available online 17 November 2006

Abstract

Temporal association rule mining promises the ability to discover time-dependent correlations or patterns between events in large volumes of data. To date, most temporal data mining research has focused on events existing at a point in time rather than over a temporal interval. In comparison to static rules, mining with respect to time points provides semantically richer rules. However, accommodating temporal intervals offers rules that are richer still. In this paper we outline a new algorithm, ARMADA, to discover frequent temporal patterns and to generate richer interval-based temporal association rules. In addition, we introduce a maximum gap time constraint that can be used to get rid of insignificant patterns and rules so that the number of generated patterns and rules can be reduced. Synthetic datasets are utilized to assess the performance of the algorithm.

Keywords: Relative temporal data mining; Interval data; Temporal association rules; ARMADA

1. Introduction

Temporal data mining can be defined as the search for interesting correlations or patterns in large sets of temporal data. Temporal data mining has the capability to discover patterns or rules which might be overlooked when the temporal component is ignored or treated as a simple numeric attribute [1]. A large volume of research has therefore been focused on temporal data mining to discover temporal rules such as sequential patterns [2], episodes [3], temporal association rules [4,5] and inter-transaction association rules [6]. However, almost all of these studies have been focused on data that are stamped with, and interpreted as, time points, whereas intervals, and their relationships, have been largely overlooked.

Böhlen et al. [7] argue that for some applications events are better treated as intervals rather than time points. For example, consider a medical database, in which a patient's treatment is regarded each time as

[*] Corresponding author. Tel.: +61 8 8201 5611; fax: +61 8 8201 2904.
E-mail addresses: edi.winarko@infoeng.flinders.edu.au (E. Winarko), roddick@infoeng.flinders.edu.au (J.F. Roddick).

0169-023X/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.datak.2006.10.009

an event time-stamped with a time point, indicating the time of the treatment. While useful, it could be advantageous to interpret as an interval, the period between the first and last occurrences of the treatment.

In this paper, we consider the problem of finding temporal rules from interval-based data. We can consider that we have a collection of entities, for example in-patients in a hospital database, the activities of which are recorded as a sequence of states, where each state is associated with an interval, indicating the period of its occurrence. As stated elsewhere [1] there is a risk of losing valuable knowledge if the temporal constraints are not flexible enough. For example, while there may be a low-level association between two symptoms, one may often immediately precede the other enabling the former to act as an alert for the latter. The richer the temporal relationships used, the more likely the association will meet the specified confidence thresholds.

Mining temporal rules from such data is undoubtedly more complex and requires a different approach from mining patterns from point-based data, such as mining sequential patterns or episodes. An interval has duration and therefore the generated patterns have different semantics than simply *before* and *after*. Allen's temporal interval logic (and its extensions) [8–10] are commonly used to describe the relationships among intervals.

In this paper, therefore, we propose a new algorithm, ARMADA, for discovering temporal patterns from interval-based data. First, we extend the MEMISP (*MEM*ory *I*ndexing for *S*equential *P*attern mining) algorithm [11] to mine frequent temporal patterns. We choose to extend the MEMISP algorithm because it is more efficient than both GSP[1] [12] and PrefixSpan[2] [13] algorithms in finding sequential patterns from transactional databases [11]. Similar to the MEMISP algorithm, our algorithm requires one database scan and does not require candidate generation or database projection. When the database is too large to fit into memory, the algorithm divides the database into several partitions and mines each partition. A second pass of the database is then required to validate the true patterns in the database. Following this, we generate temporal rules, which we term *richer temporal association rules*, from the frequent patterns.

Additionally, we include a *maximum gap* time constraint that can be used to remove insignificant patterns, which in turn can reduce the number of frequent patterns and temporal association rules generated by the algorithm. The paper discusses the result of our experiments using synthetic datasets.

The remainder of the paper is organised as follows. Section 2 discusses previous research related to our work. Section 3 describes our temporal association rule mining problem. The proposed algorithm, ARMADA, for mining richer temporal association rules from interval-based data is explained in Section 4. Section 5 discusses the *maximum gap* time constraint. Section 6 discusses the results of our experiments. Our conclusions and areas for further research are presented in Section 7.

2. Related work

There is some previous work on the discovery of temporal patterns from interval-based data. Villafane et al. [14] propose a technique to discover containment relationships from interval time series. While existing techniques consider time series as point-based events, this paper treats time series as interval-based events. One of the applications of containment relationships is the medical field where containment relationships among diseases can be discovered. For example, we may discover that during a flu infection, a certain strain of skin-borne bacteria is present. The containment rules discussed, however, are constrained to the Allen relations *contain* or *during*.

Kam and Fu [15] consider the discovery of temporal patterns for interval-based events stored in a temporal database. They use Allen's interval operators to formulate patterns. The rules are restricted to so called *A1 patterns*, that only allow concatenation of operators on the right hand side. The patterns are mined with the Apriori-like algorithm [16]. The algorithm transforms the original database into vertical data format, as used in SPADE algorithm [17].

[1] An Apriori based algorithm.
[2] An FP-Growth based algorithm.

The problem of discovering temporal patterns and rules from a state sequence is presented in [18]. Suppose s is a state, b is the *start-time* of the state, and f is the *end-time* of a state, a state sequence is defined a series of triples defining state intervals $(b_1,s_1,f_1),(b_2,s_2,f_2),(b_3,s_3,f_3),\ldots,(b_n,s_n,f_n)$, where $b_i \leqslant b_{i+1}$ and $b_i < f_i$. A temporal pattern is defined as a set of states together with their interval relationships. These relationships are represented as a square matrix R whose elements $R[i,j]$ denote the relationship between state intervals i and j. To discover the patterns, the algorithm is based on the Apriori algorithm [16] and is designed to work with a single sequence of states. After all frequent temporal patterns are found, the temporal rule $X \mapsto Y$ is generated from every pair (X, Y) of frequent temporal patterns where X is a subpattern of Y.

Our work is closely related to the work discussed in [15,18]. Our data model is similar to the one described in [15], while our definition of temporal patterns follows the one presented in [18]. However, ARMADA uses memory-based indexing, instead of Apriori-based algorithm.

3. Problem statement

Definition 1. Given a temporal database $D = \{t_1 \ldots t_n\}$, each record t_i consists of a *client-id*, a *temporal attribute*, a *start-time*, and an *end-time*, where *start-time* < *end-time*. We assume that the interval between the *start-time* and *end-time*, indicating the interval during which the record values are valid, is a relatively short interval (as compared to the total period under analysis). Each *client-id* can be associated with more than one record.

In most databases, several temporal attributes can be recorded. Each of these attributes represents a different temporal dimension of the data. For example, in a medical database the date of birth of a patient, the dates of medical examinations, the dates of important medical incidents and other dates concerning different facts of the evolution of the health of a patient can be recorded [19]. In these cases, we can choose one or more temporal attributes as our target in the mining process.

Definition 2. Let S denote the set of all possible states. A state $s \in S$ that holds during a period of time $[b,f)$[3] is denoted as (b,s,f), where b is the *start-time* and f is the *end-time*. The (b,s,f) is called a *state interval*. A *state sequence* on S is a series of triples defining state intervals $\langle(b_1,s_1,f_1),(b_2,s_2,f_2),\ldots,(b_n,s_n,f_n)\rangle$, where $b_i \leqslant b_{i+1}$ and $b_i < f_i$.

If all records in the database D with the same *client-id* are grouped together and ordered by increasing *start-time*, the database can be transformed into a collection of state sequences. Each state sequence is called the client state sequence (or *client sequence* for short). As a result, the database D can be viewed as a collection of such client sequences.

Definition 3. If s is a single state type in S, then s is a temporal pattern, denoted as $\langle s\rangle$.

Definition 4. Given n state intervals (b_i,s_i,f_i), $1 \leqslant i \leqslant n$, a *temporal pattern* of size $n > 1$ is defined by a pair $(\mathfrak{s}, M)$, where $\mathfrak{s} : \{1,\ldots,n\} \to S$ maps index i to the corresponding state, and M is an $n \times n$ matrix whose elements $M[i,j]$ denotes the relationship between intervals $[b_i,f_i)$ and $[b_j,f_j)$. The number of intervals in the temporal pattern p is denoted as $\dim(p)$. If $\dim(p) = k$, then p is called a k-pattern.

As for Höppner [18], we use *normalized* temporal patterns in which the state intervals within the patterns are ordered in increasing index according to their start times, end times, and states. Thus, the normalized temporal patterns only require seven relations out of 13 relations listed in [8], namely, *before* (b), *meets* (m), *overlaps* (o), *is-finished-by* (fi), *contains* (c), *equals* (=), and *starts* (s), as shown in Fig. 1. The first five relationships are when the start times differ. In this case, the ordering is based on the start times. If both intervals are identical, we use the order on the states so that we have *A equals B*, instead of *B equals A*. If the start times are the same and the end times are different, the ordering is based on the end times.

[3] As for most temporal databases, we assume the begin time is inclusive but the end time is not.

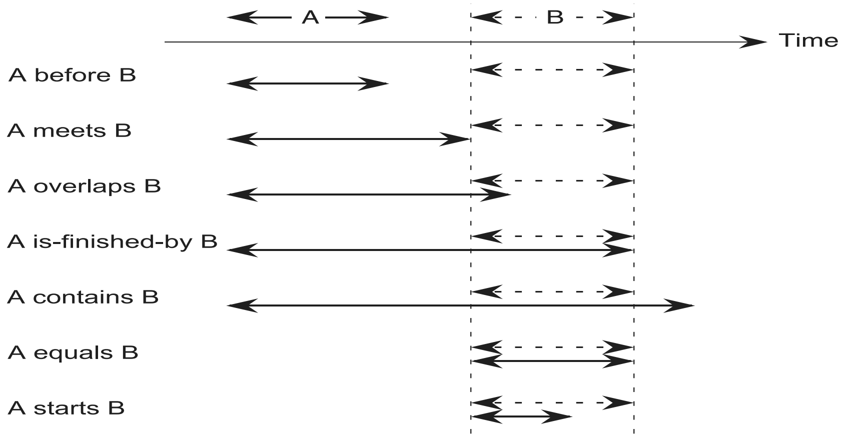


Fig. 1. Seven relations in normalized temporal patterns.

Definition 5. A temporal pattern $\alpha = (\mathfrak{s}_\alpha, M_\alpha)$ is a subpattern of $\beta = (\mathfrak{s}_\beta, M_\beta)$, denoted $\alpha \sqsubseteq \beta$, if $\dim(\mathfrak{s}_\alpha, M_\alpha) \leqslant \dim(\mathfrak{s}_\beta, M_\beta)$ and there is an injective mapping $\pi : \{1, \ldots, \dim(\mathfrak{s}_\alpha, M_\alpha)\} \to \{1, \ldots, \dim(\mathfrak{s}_\beta, M_\beta)\}$ such that

$$\forall i,\ j \in \{1, \ldots, \dim(\mathfrak{s}_\alpha, M_\alpha)\} : \mathfrak{s}_\alpha(i) = \mathfrak{s}_\beta(\pi(i)) \wedge M_\alpha[i,j] = M_\beta[\pi(i), \pi(j)]$$

Informally, it can be stated that a pattern α is a subpattern of β if α can be obtained by removing intervals from β. As an example consider the three temporal patterns in Fig. 2.[4] A pattern p_1 is a subpattern of p_2, but it is not a subpattern of p_3. We can obtain p_1 from p_2 by removing an interval state D, on the other hand, removing interval states C and D from p_3 would not result in p_1.

Definition 6. A client sequence α in D supports a pattern $p = (\mathfrak{s}_p, M_p)$ if $(\mathfrak{s}_p, M_p) \sqsubseteq (\mathfrak{s}_\alpha, M_\alpha)$, where $(\mathfrak{s}_\alpha, M_\alpha)$ is a pattern that represents the relationships between intervals in the client sequence. The support of a pattern p is defined as $\sigma(p) = \frac{|D_p|}{|D|}$, where $|D_p|$ is the number of client sequences that support the pattern p, and $|D|$ is the number of client sequences in the database D.

Definition 7. Given a minimum support *minsup*, a pattern is called *frequent* if its support is greater than or equal to *minsup*.

As an example, suppose we are given a temporal database D, which stores a list of clinical records, as shown in Fig. 3. Each record contains a *patient-id*, a *disease-code* and a pair of ordered time points, indicating the period during which the patient exhibited a given disease. Records in the database have been sorted on the *patient-id*, the *start-time*, the *end-time*, and the *disease-code*. The last column in the table is used to visualize the relative position of state intervals in each patient. The database D contains four client sequences (one for each *patient-id*), namely, cs_1, cs_2, cs_3, and cs_4. Using a *minsup* of 40%, the frequent temporal patterns are shown in Table 1.

Definition 8. A richer temporal association rule is an expression $X \Rightarrow Y$, where X and Y are frequent temporal patterns such that $X \sqsubseteq Y$ (X is a subpattern of Y). The confidence of a richer temporal association rule $X \Rightarrow Y$ is defined as

$$conf(X \Rightarrow Y) = \frac{\sigma(Y)}{\sigma(X)}$$

[4] For brevity, we do not put labels on the rows of the matrix because they are always similar to the column labels.

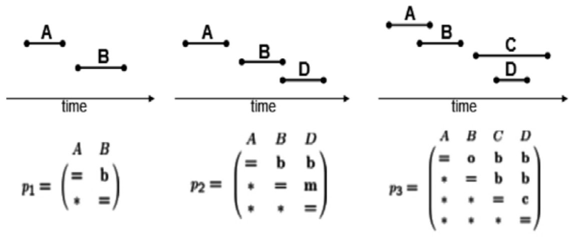


Fig. 2. Example of temporal patterns.

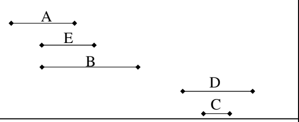
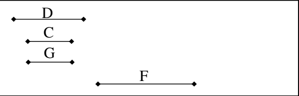
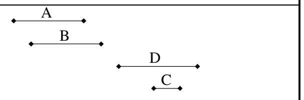
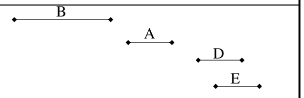
Pat-id	Disease Code	Start Time	End Time	Relative Position
1	A	2	7	
1	E	5	10	
1	B	5	12	
1	D	16	22	
1	C	18	20	
2	D	8	14	
2	C	10	13	
2	G	10	13	
2	F	15	22	
3	A	6	12	
3	B	7	13	
3	D	14	20	
3	C	17	19	
4	B	8	16	
4	A	18	21	
4	D	24	27	
4	E	25	28	

Fig. 3. Database example.

Given a temporal database D, the problem of mining richer temporal association rules is to generate all richer temporal association rules that have confidence greater than or equal to the user-specified minimum confidence *minconf*.

We are interested in generating the *forward* rules, that is, the rules that are used for predicting the future rather than in the past. As an example, from a frequent pattern $X = \begin{pmatrix} A & B \\ = & o \\ * & = \end{pmatrix}$ and a temporal pattern $Y = \begin{pmatrix} A & B & D \\ = & o & b \\ * & = & b \\ * & * & = \end{pmatrix}$ (see Table 1), we can get a rule $X \Rightarrow Y$ with the confidence of 100%. The rule can be interpreted as *if A overlaps B occurs, then it is highly likely that A before D and B before D will also occur.*

4. ARMADA – mining richer temporal association rules

As in mining association rules [16], the problem of mining richer temporal association rules can be decomposed into two subproblems: first, to find all frequent temporal patterns that have support above minimum support and, second, to generate the rules from the frequent patterns. In Section 4.1, we describe ARMADA, our proposed algorithm for discovering frequent temporal patterns, assuming that the database fits into memory. Section 4.2 outlines the method for discovering frequent temporal patterns from large databases that do not fit in memory. Our method to generate the rules is given in Section 4.3.

Table 1
Frequent temporal patterns

1-Patterns	(A) $(\sigma = 75\%)$, (B) $(\sigma = 75\%)$, (C) $(\sigma = 75\%)$, (D) $(\sigma = 100\%)$, (E) $(\sigma = 50\%)$
2-Patterns	$\begin{pmatrix} A & B \\ = & o \\ * & = \end{pmatrix} (\sigma = 50\%)$, $\begin{pmatrix} A & D \\ = & b \\ * & = \end{pmatrix} (\sigma = 75\%)$, $\begin{pmatrix} A & C \\ = & b \\ * & = \end{pmatrix} (\sigma = 50\%)$, $\begin{pmatrix} B & D \\ = & b \\ * & = \end{pmatrix} (\sigma = 75\%)$, $\begin{pmatrix} B & C \\ = & b \\ * & = \end{pmatrix} (\sigma = 50\%)$, $\begin{pmatrix} D & C \\ = & c \\ * & = \end{pmatrix} (\sigma = 75\%)$
3-Patterns	$\begin{pmatrix} A & B & D \\ = & o & b \\ * & = & b \\ * & * & = \end{pmatrix} (\sigma = 50\%)$, $\begin{pmatrix} A & B & C \\ = & o & b \\ * & = & b \\ * & * & = \end{pmatrix} (\sigma = 50\%)$, $\begin{pmatrix} B & D & C \\ = & b & b \\ * & = & c \\ * & * & = \end{pmatrix} (\sigma = 50\%)$, $\begin{pmatrix} A & D & C \\ = & b & b \\ * & = & c \\ * & * & = \end{pmatrix} (\sigma = 50\%)$
4-Patterns	$\begin{pmatrix} A & B & D & C \\ = & o & b & b \\ * & = & b & b \\ * & * & = & c \\ * & * & * & = \end{pmatrix} (\sigma = 50\%)$

4.1. Discovering frequent temporal patterns

ARMADA discovers frequent temporal patterns in three steps. First, the algorithm reads the database into memory. While reading the database, it counts the support of each state and generates frequent 1-patterns. The algorithm then constructs an index set for each frequent 1-pattern and finds frequent patterns using the state sequences indicated by elements of the index set. Finally, using a recursive *find-then-index* strategy, the algorithm discovers all temporal patterns from the in-memory database. Each of these steps is described in the following sections. Pseudo-code of ARMADA is shown in Algorithm 1. To illustrate it, we use the algorithm to discover frequent patterns from the example database shown in Fig. 3 and a *minsup* of 40%.

Definition 9. Given a pattern ρ, where $\dim(\rho) = n$, and a frequent state s in the database, a pattern ρ' of size $(n+1)$ can be formed by adding the s as a new element to ρ and setting the relationships between s and each element of ρ. The frequent state s is called *stem* of the pattern ρ' and ρ is the *prefix pattern* (*prefix* for short) of ρ'.

Algorithm 1: Pseudo-code for ARMADA

```
INPUT: a temporal database $D$, $minsup$
OUTPUT: all frequent normalized temporal patterns
 1:  read $D$ into $MDB$ (in-memory database) to find all frequent states
 2:  for each frequent state $s$ do
 3:     form a pattern $\rho = \langle s \rangle$, output $\rho$
 4:     construct $\rho$-$idx$ = $CreateIndexSet(s, \langle\,\rangle, MDB)$
 5:     call $MineIndexSet(\rho, \rho\text{-}idx)$
 6:  end for
```

4.1.1. Step 1 – reading the database into memory

In this first step, the algorithm reads the database D into memory, which will be referred to as MDB hereafter. While reading each client sequence from the database, the algorithm computes the support count of every state, then finds the set of all frequent states. From the example database, the algorithm finds frequent states $\langle A \rangle$ ($\sigma = 75\%$), $\langle B \rangle$ ($\sigma = 75\%$), $\langle C \rangle$ ($\sigma = 75\%$), $\langle D \rangle$ ($\sigma = 100\%$), and $\langle E \rangle$ ($\sigma = 50\%$). The state A is supported by three client sequences, i.e., the client sequences cs_1, cs_3, and cs_4. Each of these states will become a frequent patterns of size 1 (see Table 1).

Notice that the set of all frequent patterns can be grouped into several groups such that the patterns within a group share the same prefix. For example, from the set of frequent 1-patterns found in this step, the set of all frequent patterns can be grouped into five groups according to the five prefix patterns: $\langle A \rangle$, $\langle B \rangle$, $\langle C \rangle$, $\langle D \rangle$, and $\langle E \rangle$. Each group of frequent patterns then can be mined by constructing corresponding index set and mine each recursively, as shown in the following steps.

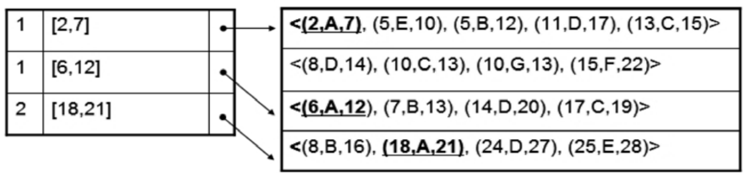
4.1.2. Step 2 – constructing the index set

Let ρ' be a pattern formed by combining a prefix pattern ρ and a stem s. An index set ρ'-*idx* is a collection of client sequences that contains a pattern ρ'. Each element of the index set contains three fields, namely, *ptr_cs*, *a_intv*, and *pos*. The *ptr_cs* is a pointer to the client sequence, *a_intv* is a list of intervals in the client sequence which generates a pattern ρ', and *pos* is the first occurring position of s in the client sequence with respect to ρ. The pseudo-code for constructing the index set is shown in Algorithm 2. The third parameter in the algorithm, *range-set*, is a set of client sequences for indexing, whose value is either MDB or an index set.

Algorithm 2: Pseudo-code for constructing index set

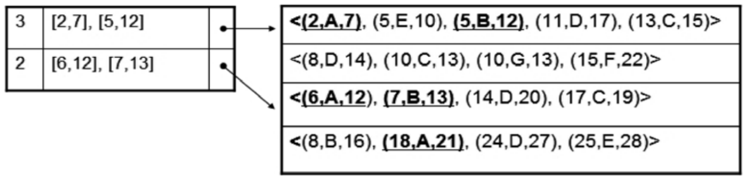
```
1: // Construct the index set $\rho'$-idx
2: // $\rho'$ is a pattern formed by combining $\rho$ and $s$
3: // range-set is a set of client sequences for indexing
4: CreateIndexSet($s, \rho$, range-set):
5: for each client sequence cs in range-set do
6:     if range-set = MDB then
7:        start-pos = 0
8:     else
9:        start-pos = pos
10:    end if
11:    for pos = (start-pos+1) to |cs| do
12:       if stem state s is first found at position pos in cs then
13:          insert (ptr_cs, a_intv, pos) to the index set $\rho'$-idx, where ptr_cs points to cs
14:       end if
15:    end for
16: end for
17: return index set $\rho'$-idx
```

From the running example, in order to find the frequent patterns with prefix $\langle A\rangle$ we construct the index set $\langle A\rangle$-*idx*. The index set is created by calling *CreateIndexSet*($A, \langle\rangle, MDB$) and is shown in Fig. 4(a). As we can see, the index set $\langle A\rangle$-*idx* contains a set of client sequences that support $\langle A\rangle$. The value of *pos* of an index element pointing to cs_1 is set to 1 because, with respect to the current prefix $\rho = \langle\rangle$, in cs_1 a stem A is found at position 1. Analogously, in cs_3 and cs_4, a stem A is found at positions 1 and 2, respectively. The *a_intv* contains an interval corresponding to a pattern $\rho = \langle A\rangle$. Note that cs_2 is not pointed to by any pointer in the index set because it does not contains a stem A (w.r.t prefix $\rho = \langle\rangle$).



1 [2,7]
1 [6,12]
2 [18,21]
<(2,A,7), (5,E,10), (5,B,12), (11,D,17), (13,C,15)>
<(8,D,14), (10,C,13), (10,G,13), (15,F,22)>
<(6,A,12), (7,B,13), (14,D,20), (17,C,19)>
<(8,B,16), (18,A,21), (24,D,27), (25,E,28)>

(a) Index Set <A>-idx



3 [2,7], [5,12]
2 [6,12], [7,13]
<(2,A,7), (5,E,10), (5,B,12), (11,D,17), (13,C,15)>
<(8,D,14), (10,C,13), (10,G,13), (15,F,22)>
<(6,A,12), (7,B,13), (14,D,20), (17,C,19)>
<(8,B,16), (18,A,21), (24,D,27), (25,E,28)>

(b) Index Set $\begin{array}{cc} A & B \\ \left(\begin{matrix} = & o \\ * & = \end{matrix}\right) \end{array}$-idx

Fig. 4. Examples of index sets.

4.1.3. Step 3 – mining patterns from the index set

Suppose we have an index set ρ-*idx*. The goal of mining the index set ρ-*idx* is to find stems with respect to prefix ρ. Any state in the indexed client sequences whose position is larger than the value of *pos* could be a potential stem (with respect to ρ). Thus, for every client sequences in ρ-*idx*, the algorithm increases the support count of such state by one. Afterward, the algorithm determines which of the states are frequent and become stems. Each of these stem will be combined with the prefix ρ to generate a frequent pattern ρ'. Then, recursively, the index set ρ'-*idx* is constructed and mined until no more stem can be found. The pseudo-code for mining an index set is shown in Algorithm 3.

Algorithm 3: Pseudo-code for mining index set

```
1: // Mine patterns from an index set $\rho$-idx
2: MineIndexSet($\rho, \rho$-idx):
3: for each cs pointed by index elements of $\rho$-idx do
4:     for $pos = pos + 1$ to $|cs|$ in cs do
5:         $count(s) = count(s) + 1$, where s is a potential stem state
6:     end for
7: end for
8: find $S =$ the set of stems s
9: for each stem state $s \in S$ do
10:    output the pattern $\rho'$ by combining prefix $\rho$ and stem s
11:    call CreateIndexSet($s, \rho, \rho$-idx) // to construct the index set $\rho'$-idx
12:    call MineIndexSet($\rho', \rho'$-idx) // to mine patterns with index set $\rho'$-idx
13: end for
```

Continuing our example, after obtaining $\langle A\rangle$-*idx*, we mine it to find all stems with respect to prefix $\langle A\rangle$, by calling *MineIndexSet*($\langle A\rangle, \langle A\rangle$-*idx*). We process each client sequence in the index set, checking any state interval appearing after the *pos* position. The first element of $\langle A\rangle$-*idx*, which points to cs_1, has the value of *pos* 1. Thus we only focus on the interval states occurring after position 1. As a result, we increase the support count of a potential stem E for a potential pattern $p2_E = \begin{pmatrix} A & E \\ = & o \\ * & = \end{pmatrix}$ by one. There are also potential stems B for a pattern $p2_B$, D for a pattern $p2_D$, and C for a pattern $p2_C$, where $p2_B = \begin{pmatrix} A & B \\ = & o \\ * & = \end{pmatrix}$, $p2_D = \begin{pmatrix} A & D \\ = & b \\ * & = \end{pmatrix}$, and $p2_C = \begin{pmatrix} A & C \\ = & b \\ * & = \end{pmatrix}$, respectively.

Using the same process, we perform the support count for the states occurring after positions 1 and 2 at the client sequences cs_3 and cs_4, respectively. After validating the support counts, we obtain stems B ($\sigma = 50\%$), C ($\sigma = 50\%$), and D ($\sigma = 75\%$) (w.r.t. prefix $\langle A\rangle$) to form patterns $p2_B$, $p2_C$, and $p2_D$, respectively.

Let ρ' be a pattern formed by combining prefix $\rho = \langle A\rangle$ and $s \in \{B, C, D\}$. The next process is to construct index set ρ'-*idx* and mine it, recursively, for each $s \in \{B, C, D\}$. We proceed by taking prefix $\rho = \langle A\rangle$ and $s = B$ to obtain a pattern $\rho' = \begin{pmatrix} A & B \\ = & o \\ * & = \end{pmatrix}$. We call *CreateIndexSet*($B, \langle A\rangle, \langle A\rangle$-*idx*) to construct the index set ρ'-*idx*. Note that the value of the third parameter, *range-set*, is the index set $\langle A\rangle$-*idx*, instead of *MDB*. It means that in creating ρ'-*idx*, we only need to consider the set of client sequences in $\langle A\rangle$-*idx* (i.e., cs_1, cs_3, and cs_4), rather than all client sequences in *MDB*. The resulting index set is shown in Fig. 4(b). With respect to prefix $\langle A\rangle$, a stem B is at position 3 in cs_1 and 2 in cs_3. We store these values at the field *pos* of the index set. The interval values of a state B in cs_1 and cs_3 are added to the array *a_intv*. There is no entry created for cs_4 because it does not support a pattern ρ'. After creating ρ'-*idx*, the index set $\langle A\rangle$-*idx* is not discarded but it is stored for later use. We mine the index set ρ'-idx, and find stems C and D that will form patterns $p3_C = \begin{pmatrix} A & B & C \\ = & o & b \\ * & = & b \\ * & * & = \end{pmatrix}$ and $p3_D = \begin{pmatrix} A & B & D \\ = & o & b \\ * & = & b \\ * & * & = \end{pmatrix}$, respectively.

Now we have to continue the recursive process on prefix $\rho = \begin{pmatrix} A & B \\ = & o \\ * & = \end{pmatrix}$ and stem $s \in \{C, D\}$. Taking the prefix ρ and a stem C, we output a pattern $\rho' = p3_C$, and construct ρ'-idx. We mine ρ'-idx but cannot find stems, so we stop this process. Next, we continue with prefix ρ and a stem D. We output a pattern $\rho' = p3_D$, then create and mine ρ'-idx to find a stem C.

Continue the recursive process by taking prefix $\rho = p3_D$ and a stem C, we output a pattern ρ' and create an index set ρ'-idx, where $\rho' = \begin{pmatrix} A & B & D & C \\ = & o & b & b \\ * & = & b & b \\ * & * & = & c \\ * & * & * & = \end{pmatrix}$. The mining of ρ'-idx finds no more stems. Since we cannot continue the recursive process, we repeat the process by taking prefix $\rho = \langle A \rangle$ and a stem s, where $s \in \{C, D\}$. This process will generate patterns: $\begin{pmatrix} A & C \\ = & b \\ * & = \end{pmatrix}$, $\begin{pmatrix} A & D \\ = & b \\ * & = \end{pmatrix}$, and $\begin{pmatrix} A & D & C \\ = & b & b \\ * & = & c \\ * & * & = \end{pmatrix}$. At this stage, we have finished the mining process of a stem A with prefix $\rho = \langle\rangle$. All frequent patterns can be discovered by continuing the mining process on stems B, C, D, and E with prefix $\rho = \langle\rangle$.

4.2. Handling large databases

The above algorithm only works if the database fits into memory. If the database is too large to fit into memory, the frequent temporal patterns are discovered by *partition-and-validation* technique, as shown in Algorithm 4. First, the database is partitioned so that each partition can be processed in memory by ARMADA. In order to be frequent in the database, a temporal pattern has to be frequent in at least one partition. Therefore, we can obtain the set of potential frequent patterns by collecting the discovered patterns after running ARMADA on these partitions. The next step is the validation step, in which the actual frequent patterns can be identified through support counting against the data sequences with only one extra database pass. Therefore, ARMADA requires two passes of database scan to mine large databases that do not fit into memory.

Algorithm 4: Using ARMADA to process large databases

```
Input: a database $D$, $minsup$
Output: a set of frequent temporal patterns $F$
1: for each partition $D_i \subset D$ do
2:    $F_i = ARMADA\_Gen(D_i, minsup)$
3: end for
4: $C = \cup_n F_i$
5: for each sequences $s \in D$
6:    increment support count of all $c \in C$ supported by $s$
7: end for
8: $F = \{c \in C | sup(c) \geqslant minsup\}$
```

4.3. Generating temporal association rules

After all frequent temporal patterns have been discovered, we can generate temporal association rules as follows. If Y is a frequent n-pattern, where $n > 1$, and $S = \langle s_1, s_2, \ldots, s_n \rangle$ is a list of states in Y (S has been ordered in increasing index according to the order within Y), then we will find all subpatterns X of Y which have ordered list of states $S_i = \langle s_1, s_2, \ldots, s_i \rangle$, $i = 1, 2, \ldots, (n-1)$. Therefore, for each frequent n-pattern Y, there are at most $(n-1)$ subpatterns X of Y. For every such subpattern X, we generate a rule of the form $X \Rightarrow Y$ if its confidence is greater than or equal to *minconf*.

The process of finding X starts by taking $i = (n-1)$ so that X has a list of states $S_{n-1} = \langle s_1, s_2, \ldots, s_{n-1} \rangle$. If $X \Rightarrow Y$ has enough confidence, we will generate this rule and continue to test the next subpattern. If $X \Rightarrow Y$ does not have enough confidence, we do not have to check for X where $i < (n-1)$ because the generated rule will have a lower confidence.

5. Maximum gap time constraint

As with all temporal data mining algorithms, ARMADA can easily generate very large numbers of frequent temporal patterns. One of the reasons is that in the above model, time gaps between intervals in the temporal patterns are not specified so that some uninteresting patterns are likely to appear. As an example, consider the database in Fig. 3, in which without specifying the maximum gap, we find a temporal pattern (*A before C*) is frequent with the support of 50% (see Table 1). However, this pattern may be insignificant because the time gap between states A and C is too wide. We therefore introduce a *maximum gap* time constraint in the mining to reduce the number of generated patterns and reinforce the significance of mining results.

Definition 10. Let $\alpha = \langle (b_1, s_1, f_1), (b_2, s_2, f_2), \ldots, (b_n, s_n, f_n) \rangle$, where $b_i \leqslant b_{i+1}$ and $b_i < f_i$, be a client sequence. The time gap between state intervals i and j, for $i < j$, is defined as $gap(i,j) = b_j - f_i$. The maximum gap of the client sequence α is defined as $\delta(\alpha) = \max\{gap(i,j) | i < j,\ i = 1, \ldots, (n-1) \text{ and } j = 2, \ldots, n\}$.

Definition 11. Given a user specified *maxgap*, a client sequence α supports a pattern $p = (\mathfrak{s}_p, M_p)$ if $(\mathfrak{s}_p, M_p)$ is a subpattern of $(\mathfrak{s}_\alpha, M_\alpha)$, and the maximum gap of state intervals that take part in the pattern p is less than or equal to *maxgap*.

As an example, consider a client sequence of *patient-id* 1 in Fig. 3. Originally this client sequence supports a pattern (*B before D*). If we add the constraint by taking *maxgap* = 2, the pattern is no longer supported by the client sequence because the maximum gap of state intervals involved in the pattern is bigger than *maxgap* (the gap between B and D is equal to 4). This is also the case for a client sequence of *patient-id* 4.

Note that if the value of *maxgap* is a positive integer, the constraint only affects intervals that have temporal relation *before*. If we set *maxgap* = ∞, we get the original model as described in Section 3, where there is no time constraint specified.

Definition 12. Given the *minsup* and the *maxgap*, a temporal pattern is called frequent if its support is greater than or equal to *minsup*.

To mine frequent patterns with the maximum gap constraint, we use the algorithm discussed in Section 4.1 but modify it so that when searching for stems it also checks the gap between intervals. The method to generate temporal rules from frequent pattern described in Section 4.3 is still applicable because the property that if a pattern p is frequent then so all its subpatterns still holds.

6. Experiment results

To assess the performance of our proposed algorithm for discovering frequent temporal patterns, we conducted several experiments on the synthetic datasets. ARMADA was implemented in Java on a 2.4 GHz Athlon PC with 512MB of RAM running Windows 2000 Professional.

The synthetic data generation program takes five parameters, namely, the number of client sequences ($|D|$), average size of client sequences ($|C|$), number of maximal potentially frequent temporal patterns (N_P), average size of potentially frequent temporal patterns ($|P|$), and number of states (N). The data generation model is based on the one used for mining association rules [16] with some modification to model the temporal database.

We first created a random pool of potentially frequent patterns that were used in the generation of client sequences. The number of potentially frequent patterns was N_P. A frequent pattern was generated by first picking the size of the pattern (the number of states in the pattern) from a Poisson distribution with mean equal to $|P|$. Then, we chose the state types randomly (from N state types). We selected the temporal relationships between consecutive states randomly and formed a pattern. Since we used normalized temporal patterns (as described in Section 3), the temporal relations were chosen from the set {*before, meets, overlaps, is-finished-by, contains, starts, equal*}. Each state in the pattern was then assigned an interval value according to its temporal relation with the state that comes before it. The interval value of the first state in the pattern was chosen randomly. If the pattern contains two similar consecutive states, its temporal relation was set to *before*. After all potentially frequent patterns are generated, we generated $|D|$ client sequences. Each client sequence was generated by first determining its size, which was picked from a Poisson distribution with mean equal to $|C|$. Then, each client sequence was assigned a series of potentially frequent patterns.

We conducted four sets of experiments, by varying the minimum supports, the maximum gaps, the number of states, and the size of databases (number of sequences). In each set of experiments, we recorded the processing times of the algorithm as well as the number of generated frequent patterns.

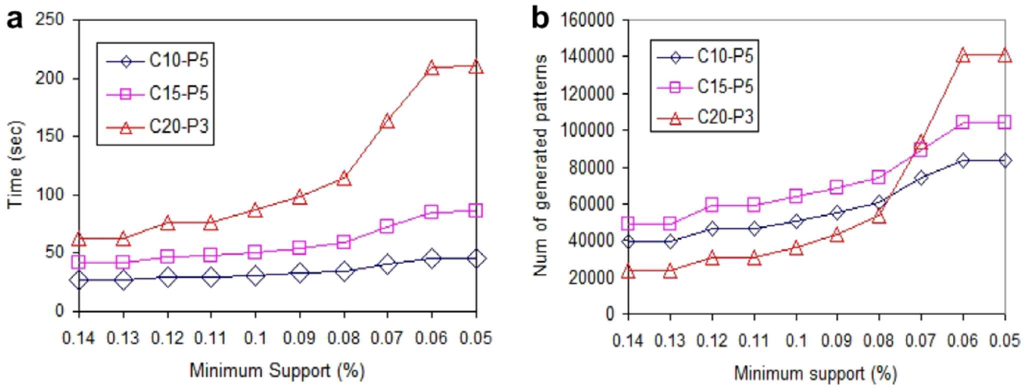


Fig. 5. Effect of decreasing minimum support.

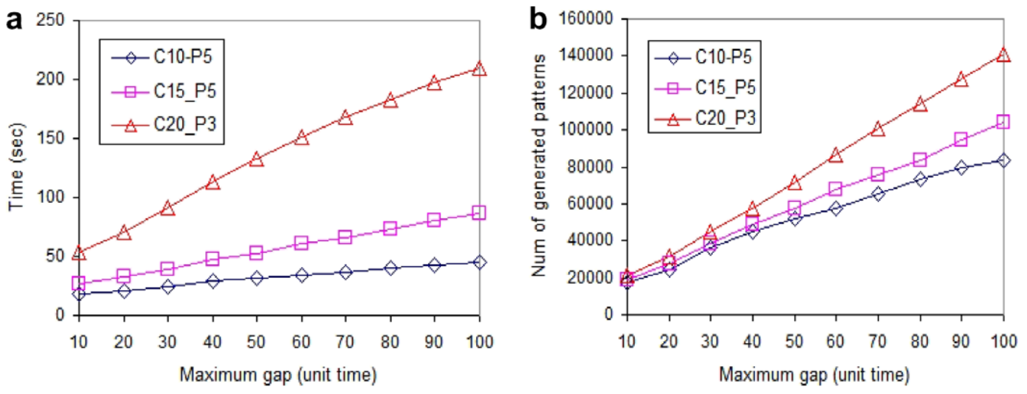


Fig. 6. Effect of increasing maximum gap.

First we investigated the effect of varying minimum support on the processing times. We generated three datasets by setting $N = 1000$ and $N_P = 2000$. We chose three values of $|C|$: 10, 15, and 20, and two values of $|P|$: 3 and 5. The number of client sequences $|D|$ was set to 10,000. We varied the values of minimum support from 0.05% to 0.14%, and set the value of maximum gap to 100 time units. Fig. 5 shows the number of generated patterns and the processing times as the values of minimum support decreases. As expected, as the minimum support decreases, the processing times increase for all datasets (Fig. 5(a)). This is because as we decrease the minimum support, the number of generated patterns increases (Fig. 5(b)), resulting in increasing processing times. The dataset with longer sequences requires more processing times compared to that with shorter sequences. Consider the dataset *C20-P3* (Fig. 5(a)), even though its generated patterns are not always the highest, its processing times are the highest for all values of minimum support.

The second set of experiments looked into the effect of varying maximum gap on the processing times. We used the above datasets. We set the value of minimum support to 0.05% but varied the values of maximum gap from 10 to 100 time units. As shown in Fig. 6(a), the processing times increase as we increase the values of maximum gap. Similar to the previous experiments, when the maximum gap increases, more frequent patterns will be generated (Fig. 6(b)), which resulting in increasing processing times. The dataset with longer sequences requires more processing times compared to that with shorter sequences.

For the third set of experiments, we created two sets of datasets. The first set has $|C| = 10$ and $|P| = 5$, while the second set has $|C| = 15$ and $|P| = 5$. We kept the database size constant at $|D| = 50{,}000$ and the value of $N_P = 2000$. We set the values of minimum support and maximum gap to 0.05% and 100, respectively. Fig. 7 shows the processing times and the number of generated patterns when the number of states is increased from

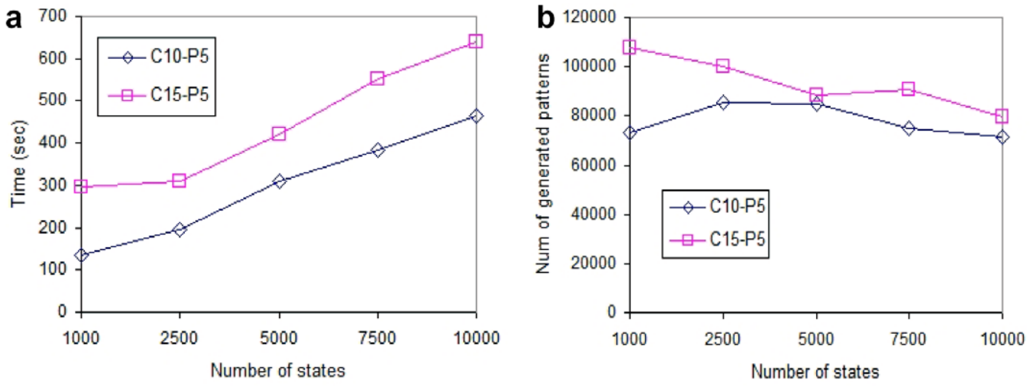


Fig. 7. Effect of increasing number of states.

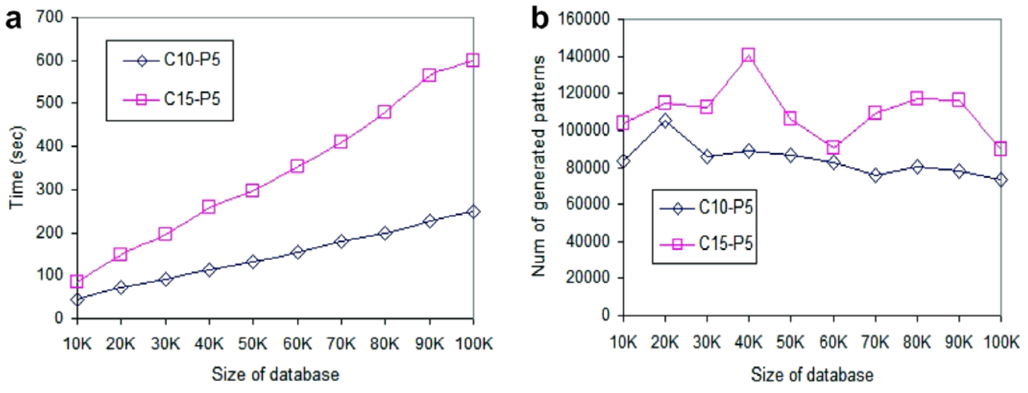


Fig. 8. Effect of increasing database size.

1000 to 10,000. It can be seen in Fig. 7(a) that the processing times increase as the number of states is increased. However, the number of generated patterns tends to decrease as the number of states increases (Fig. 7(a)).

The last set of experiments investigate how the algorithm scales up as the size of the database increases. We created two sets of datasets, the first one has $|C| = 10$ and $|P| = 5$, while the second one has $|C| = 15$ and $|P| = 5$. All datasets have the values of $N_P = 2000$ and $N = 1000$. We set the minimum support to 0.05% and the maximum gap to 100. Fig. 8(a) shows the algorithm scales up linearly as the size of databases increases from $10K$ to $100K$, regardless of fluctuation on the number of generated patterns.

7. Conclusion and future work

In this paper we have studied the discovery of richer temporal association rules from interval-based data. We have proposed a new algorithm, ARMADA, by extending MEMISP, an existing algorithm for mining sequential patterns, to discover the frequent temporal patterns. ARMADA is illustrated using an example and the method to generate richer temporal association rules from the frequent temporal patterns is described. In addition, we have proposed a maximum time constraint to reduce the number of patterns generated by the algorithm. To assess the performance of the algorithm, we conducted experiments on synthetic datasets, varying the value of minimum supports, the size of maximum gaps, the number of client sequences, and the number of states.

In summary, we can say that the proposed ARMADA algorithm looks promising as a method for discovering patterns and rules from interval-based data. It reads the database only once, except for large databases described in Section 4.2. ARMADA does not require candidate generation, it utilizes a simple index advancing to grow longer temporal patterns from the shorter frequent ones. In the process of growing the patterns, ARMADA only considers those client sequences indicated by current index set, instead of searching on every client sequences in the database. It is true that we need the storage to store the index set, in addition to the memory allocated for the database. However, the size of index set is getting smaller as the prefix pattern to create the index set getting longer.

Future work to be undertaken by the authors includes an application to real-world problem domains and enhancements to the interface to facilitate the discovery of temporal patterns and rules directly from relational temporal databases. Ideally, a temporal mining algorithms should understand all types of relationship and convention, thus other work that could be considered is the link between relative relationships and the use of either accepted calendars (for example, the work of Hamilton and Randall [20] and others) and/or mixing references to relative time with those of absolute time.

References

[1] J.F. Roddick, M. Spiliopoulou, A survey of temporal knowledge discovery paradigms and methods, IEEE Transactions on Knowledge and Data Engineering 14 (4) (2002) 750–767.
[2] R. Agrawal, R. Srikant, Mining sequential patterns, in: P.S. Yu, A.S.P. Chen (Eds.), Proceedings of the 11th International Conference on Data Engineering (ICDE'95), IEE Computer Society Press, Taipei, Taiwan, 1995, pp. 3–14.
[3] H. Mannila, H. Toivonen, Discovering generalised episodes using minimal occurrences, in: E. Simoudis, J. Han, U. Fayyad (Eds.), Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining (KDD'96), AAAI Press, Portland, Oregon, 1996, pp. 146–151.
[4] Y. Li, P. Ning, X.S. Wang, S. Jajodia, Discovering calendar-based temporal association rules, in: Proceedings of the 8th International Symposium on Temporal Representation and Reasoning, 2001, pp. 111–118.
[5] B. Ozden, S. Ramaswamy, A. Silberschatz, Cyclic association rules, in: Proceedings of the 14th International Conference on Data Engineering (ICDE'98), IEE Computer Society Press, Orlando, Florida, USA, 1998, pp. 412–421.
[6] H. Lu, L. Feng, J. Han, Beyond intratransaction association analysis: mining multidimensional intertransaction association rules, ACM Transactions on Information Systems 18 (4) (2000) 423–454.
[7] M.H. Böhlen, R. Busatto, C.S. Jensen, Point-versus interval-based temporal data models, in: Proceedings of the 14th International Conference on Data Engineering (ICDE'98), IEE Computer Society Press, Orlando, Florida, USA, 1998, pp. 192–200.
[8] J. Allen, Maintaining knowledge about temporal intervals, Communications of the ACM 26 (11) (1983) 832–843.
[9] C. Freksa, Temporal reasoning based on semi-intervals, Artificial Intelligence 54 (1992) 199–227.
[10] J.F. Roddick, C.H. Mooney, Linear temporal sequences and their interpretation using midpoint relationships, IEEE Transactions on Knowledge and Data Engineering 17 (1) (2005) 133–135.

[11] M.Y. Lin, S.Y. Lee, Fast discovery of sequential patterns by memory indexing, in: Proceedings of the 4th International Conference on Data Warehousing and Knowledge Discovery (DaWaK'02), Aix-en-Provence, France, 2002, pp. 150–160.
[12] R. Srikant, R. Agrawal, Mining sequential patterns: Generalizations and performance improvements, in: Proceedings of the 5th International Conference on Extending Database Technology, Avinon, France, 1996, pp. 3–17.
[13] J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, M.-C. Hsu, PrefixSpan: Mining sequential patterns efficiently by prefix projected pattern growth, in: Proceedings of the 17th International Conference on Data Engineering (ICDE'01), IEEE Computer Society Press, Heidelberg, Germany, 2001, pp. 215–224.
[14] R. Villafane, K.A. Hua, D. Tran, B. Maulik, Mining interval time series, in: Data Warehousing and Knowledge Discovery, 1999, pp. 318–330.
[15] P.-S. Kam, A.W.-C. Fu, Discovering temporal patterns for interval-based events, in: Y. Kambayashi, M.K. Mohania, A.M. Tjoa (Eds.), Proceedings of the 2nd International Conference on Data Warehousing and Knowledge Discovery (DaWaK'00), vol. 1874, Springer, London, UK, 2000, pp. 317–326.
[16] R. Agrawal, R. Srikant, Fast algorithms for mining association rules, in: Proceedings of the 20th International Conference on Very Large Data Bases, 1994, pp. 487–499.
[17] M.J. Zaki, SPADE: An efficient algorithm for mining frequent sequences, Machine Learning Journal 42 (1/2) (2001) 31–60.
[18] F. Höppner, Learning temporal rules from state sequence, in: Proceedings of IJCAI Workshop on Learning from Temporal and Spatial Data, Seattle, USA, 2001, pp. 25–31.
[19] G. Koundourakis, B. Theodoulidis, Association rules and evolution in time, in: Proceedings of Methods and Applications of Artificial Intelligence, Second Hellenic Conference on AI, SETN 2002, Thessaloniki, Greece, 2002, pp. 261–272.
[20] H. Hamilton, D. Randall, Data mining with calendar attributes, in: J.F. Roddick, K. Hornsby (Eds.), International Workshop on Temporal, Spatial and Spatio-Temporal Data Mining, TSDM2000, vol. 2007 of LNAI, Springer, Lyon, France, 2000.

Edi Winarko received his BSc in Statistics from Gadjahmada University, Indonesia and an MSc in Computer Science from School of Computing, Queen's University, Canada. He is currently a Ph.D. candidate at the School of Informatics and Engineering, Flinders University, Australia. His current interests include temporal data mining, web mining, and information retrieval.

John Roddick is currently the SACITT Chair in Information Technology and Associate Head (Research) in the School of Informatics and Engineering at Flinders University. He joined Flinders in 2000 after 10 years at the University of South Australia and 5 years at the University of Tasmania. This followed 10 years experience in the computing industry as (progressively) a programmer, analyst, project leader and consultant. Prof. Roddick has published over 100 papers in a number of areas of computing but specialises in the fields of Data Mining and Knowledge Discovery – specifically in temporal and spatial data mining and as applied to medical and health data, and Conceptual Modelling – specifically in enhanced database systems semantics such as schema evolution and temporal and spatial systems design and use. He holds a PhD from La Trobe University, an MSc from Deakin University and a BSc (Eng) (Hons) from Imperial College, London.