A Signature-Based Indexing Method for Efficient Content-Based Retrieval of Relative Temporal Patterns

Edi Winarko and John F. Roddick, *Member*, *IEEE Computer Society*

Abstract—A number of algorithms have been proposed for the discovery of temporal patterns. However, since the number of generated patterns can be large, selecting which patterns to analyze can be nontrivial. There is thus a need for algorithms and tools that can assist in the selection of discovered patterns so that subsequent analysis can be performed in an efficient and, ideally, interactive manner. In this paper, we propose a signature-based indexing method to optimize the storage and retrieval of a large collection of relative temporal patterns.

Index Terms—Content-based data mining queries, organizing temporal patterns, signature-based indexing methods.

1 Introduction

Many rule discovery algorithms in data mining generate a large number of patterns/rules, sometimes even exceeding the size of the underlying database, with only a small fraction being of interest to the user [1]. It is generally understood that interpreting the discovered patterns/rules to gain insight into the domain is an important phase in the knowledge discovery process. However, when there are a large number of generated rules, identifying and analyzing those that are interesting becomes difficult. For example, providing the user with a list of association rules ranked by their confidence and support might not be a good way of organizing the set of rules as this method would overwhelm the user and not all rules with high confidence and support are necessarily interesting for a variety of reasons [2].

Therefore, to be useful, a data mining system must manage the generated rules by offering flexible tools for rule selection. In the case of association rule mining, several approaches for the postprocessing of discovered association rules have been discussed. One approach is to group "similar" rules [3], [4], [5], [6], which works well for a moderate number of rules. However, for a larger number of rules it produces too many clusters. A more flexible approach is to allow the identification of rules that are of special importance to the user through templates or *data mining queries*. This approach can complement the rule-grouping approach and has been used to specify interesting and uninteresting classes of rules (for both association and episodic rules) [7]. The importance of data mining queries has been highlighted by the introduction of the inductive database concept, which allows the user to both query the data and query patterns, rules, and models extracted from these data.

To this end, several query languages with data mining extensions have been proposed, such as Mine-Rule [8], DMQL [9], and OLE DB [10]. These languages are designed to generate the rules from the data rather than allow queries over the discovered rules. However, the data mining query language MSQL can be used not only for rule generation but also for rule querying (using a SelectRules operator) [11]. In addition, Rule-QL has been proposed for querying multiple sets of association rules and utilizes efficient algorithms for processing the queries [12].

While most previous studies are focused on the post-processing of association rules, this paper deals with the postprocessing of temporal patterns [13], [14], [15], an area in which little research to date has been conducted. In particular, we address the problem of efficiently retrieving subsets of a large collection of previously discovered temporal patterns [16], [17]. When processing queries on a small database of temporal patterns, sequential scanning of the patterns followed by straightforward computations of query conditions is adequate. However, as the database grows, this procedure can be too slow, and indexes should be built to speed up the queries. The problem is to determine what types of indexes are suitable for improving the speed of queries involving the content of temporal patterns.

This paper focuses on supporting content-based queries of temporal patterns, as opposed to point- or range-based queries. One example is the *subpattern query*: Given a set of patterns $\mathcal{D}$ and a query pattern q, find the temporal patterns in $\mathcal{D}$ that contain q. For example, we may wish to study further the behavior of a group of rules for which a particular pattern q is already known to be a component. To address this form of query, a signature-based indexing

- *E. Winarko is with Gadjah Mada University, Yogyakarta, Indonesia. E-mail: edwin@ugm.ac.id.*
- *J.F. Roddick is with the School of Informatics and Engineering, Flinders University, PO Box 2100, Adelaide, SA 5001, South Australia. E-mail: roddick@infoeng.flinders.edu.au.*

Manuscript received 7 Apr. 2007; revised 24 Oct. 2007; accepted 6 Dec. 2007; published online 11 Jan. 2008.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number TKDE-2007-04-0135.
Digital Object Identifier no. 10.1109/TKDE.2008.20.

method is proposed that can speed up content-based queries on temporal patterns.

The rest of the paper is organized as follows: Section 2 gives an overview of related work, with the problem described in detail in Section 3. In Section 4, we describe the indexing and retrieval of temporal patterns using the signature-based index. Section 5 presents the result of our experiments. A conclusion and discussion of future works are given in Section 6.

2 Related Work

The temporal patterns described in this paper consist of two components: a set of states and a set of relationships between those states that represent the order of states within the pattern. In order to retrieve such patterns efficiently, any indexing method should deal with both temporal concepts—states and state relationships.

The problem of indexing has been studied in depth in the database literature, (for example, B+ trees [18], R trees [19], etc.). However, studies on set-based indexes that support queries on set-valued attributes (i.e., attributes that are sets of items) are limited [20], [21], [22]. Ishikawa et al. [21] apply the signature file technique to support the processing of queries involving set-valued attributes in OODBs. Two signature file organizations are considered: the sequential signature file and the bit-slice file.

The bit-slice approach still needs to examine every signature in the file but only a part of it. In order to avoid reading every signature in the signature file, the hierarchical file organization uses several levels of signatures. The higher levels perform coarse filtering before the signatures on the lower levels are consulted. Examples of the hierarchical file organization include the S-tree [23] and the SG-tree [24], [25]. The partitioned file organization approach avoids reading every signature by grouping the signatures into several partitions such that all signatures in a given partition possess the same component part, called the *signature key*. The signature key used is usually a substring of the signature. By partitioning the signatures, some of the partitions need not to be searched during the execution of a query so that the number of accesses can be reduced [26], [27], [28].

Helmer and Moerkotte [20] study the performance of four index structures for set-valued attributes (sequential signature files, signature trees, extensible signature hashing, and inverted lists). The indexes are evaluated on three forms of set-valued queries—*equality* queries, *subset* queries, and *superset* queries. It was observed that the inverted file index structure outperformed other index structures for subset and superset queries with respect to query processing time.

Morzy and Zakrzewicz [22] generalize the problem of association rule and item set retrieval as a subset search problem. Two types of queries are examined: first, the retrieval of item sets that contain a given subset of items and, second, the retrieval of rules that contain a given subset of items in their antecedent or consequent. In order to speed up the query processing, a *group bitmap index* is proposed in which the group bitmap key represents a set of items in the database.

These set-based indexing methods do not consider the order of items within the sets, as is required in the case of the indexing and retrieval of sequential patterns. To overcome this limitation, new indexing techniques have been proposed [29], [30], [31], the general idea of which is to convert the sequential patterns into *equivalent sets* that accommodate the ordering of the items. After that, set-based indexing methods [29] can be applied to the equivalent sets. A partitioning technique is proposed to divide large equivalent sets into a collection of smaller subsets so that the probability of collision is reduced.

3 Preliminaries

3.1 Problem Description

Definition 1 (state sequence). *Let S denote the set of all possible states. A state $s \in S$ that holds during a period of time $[b, f)$ is denoted as (b, s, f), where b is the* start time, *and f is the* end time. *The (b, s, f) triple is termed 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 $b_i \leq b_{i+1}$, and $b_i < f_i$.*

Definition 2 (temporal pattern). *Given n state intervals (b_i, s_i, f_i), $1 \leq i \leq n$, a* temporal pattern *of size n is defined by a pair (s, M), where $s : \{1, \ldots, n\} \rightarrow S$ maps index i to the corresponding state, and M is an $n \times n$ matrix whose elements $M[i, j]$ denote the relationship between intervals $[b_i, f_i)$ and $[b_j, f_j)$. The* size *of a temporal pattern α is the number of intervals in the pattern, denoted as $\dim(\alpha)$. If the size of α is n, then α is called an n-*pattern.

If the state intervals within the state sequences have been ordered in increasing index according to their start times, end times, and states, the resulting temporal patterns are considered *normalized*, and only seven out of 13 Allen relationships [32] are required, namely, *before* ($\mathbf{b}$), *meets* ($\mathbf{m}$), *overlaps* ($\mathbf{o}$), *is-finished-by* ($\mathbf{fi}$), *contains* ($\mathbf{c}$), *equals* ($=$), and *starts* ($\mathbf{s}$).[1] For the rest of this paper, let $Rel = \{=, \mathbf{c}, \mathbf{fi}, \mathbf{s}, \mathbf{o}, \mathbf{m}, \mathbf{b}\}$ be the set of these seven relationships. A normalized temporal pattern does not contain temporal extensions because it has been abstracted from the state intervals in a specific state sequence. Fig. 1 shows four normalized temporal patterns defined over a set of states $S = \{A, B, C, D\}$ and a set of interval relations Rel.

Definition 3 (subpattern). *A temporal pattern $\alpha = (s_\alpha, M_\alpha)$ is a subpattern of $\beta = (s_\beta, M_\beta)$, denoted $\alpha \sqsubseteq \beta$, if $\dim(s_\alpha, M_\alpha) \leq \dim(s_\beta, M_\beta)$ and there is an injective mapping $\pi : \{1, \ldots, \dim(s_\alpha, M_\alpha)\} \rightarrow \{1, \ldots, \dim(s_\beta, M_\beta)\}$ such that*

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

Informally, it can be stated that a pattern α is a subpattern of β if α can be obtained from β by removing intervals. As an example, consider the patterns in Fig. 1; p_1 is a subpattern of p_3, but it is not a subpattern of p_4. We can

1. See [17] for a more in-depth discussion of these concepts.

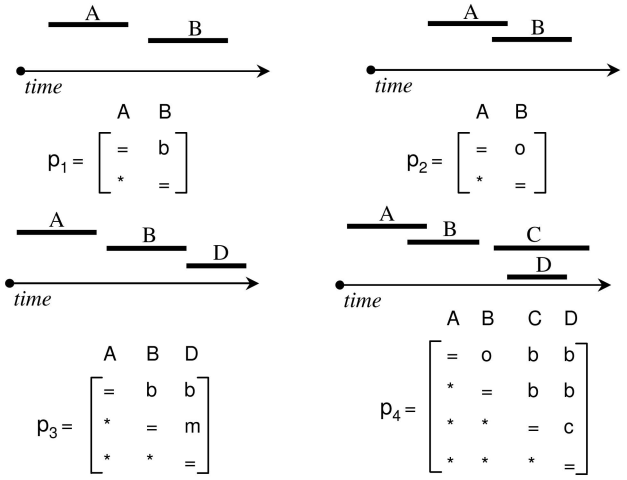

Fig. 1. Example of temporal patterns.

obtain p_1 from p_3 by removing interval state D; on the other hand, removing interval states C and D from p_4 would not result in p_1.

Definition 4 (content-based queries). *Let $\mathcal{D}$ be a temporal pattern database and q be a query pattern. The four forms of content-based queries that this research supports include the following:*

1. ***Subpattern queries.*** *Find those patterns in $\mathcal{D}$ that contain q.*
2. ***Superpattern queries.*** *Find those patterns in $\mathcal{D}$ that are a subpattern of q.*
3. ***Equality queries.*** *Find those patterns in $\mathcal{D}$ equal to q.*
4. **K-nearest subpattern queries**. *Find the k most similar patterns in $\mathcal{D}$ to q.*

Superpattern queries are useful when searching for the characteristic parts of a large pattern, while k-nearest subpattern queries limit the number of patterns generated by subpattern or superpattern queries.

The problem of content-based retrieval of temporal patterns can be formally defined as follows:

Given a database $\mathcal{D}$ of discovered temporal patterns, describe a processing technique that allows the user to find efficiently temporal patterns in $\mathcal{D}$ that satisfy the content-based queries above.

3.2 Temporal Pattern Similarity

To answer k-nearest subpattern queries, a suitable measure of similarity among temporal patterns needs to be defined. To do so, three properties must be considered:

1. Temporal patterns are variable-length objects that cannot be represented in a k-dimensional metric space.
2. Each pattern contains a list of states.
3. Each pattern contains a set of state relationships.

To our knowledge, no similarity measures have been defined for temporal patterns. The closest are the similarity measures proposed by Xiao et al. [33] that measure the similarity of a Web user's access logs. Each user's log is a sequence of pages accessed by the user, which contains information about the pages, the order of pages accessed, and the elapsed time between two page accesses. In our work, some aspects of the similarity measures defined by Xiao et al. [33] have been adopted and extended to temporal patterns.

The similarity measure is defined based on the measure known in the literature as the *Jaccard coefficient* of two sets, which expresses the fraction of elements common to both sets. The Jaccard coefficient is not a metric.[2] Nevertheless, a distance function can be defined in terms of the similarity as $d(A, B) = 1 - sim(A, B)$, and it is easy to show that such a distance function is indeed a metric.

Given two temporal patterns α and β, let S_α and S_β denote a set of states in α and β, respectively. The similarity between patterns α and β is defined as follows:

$$sim(\alpha, \beta) = \frac{|S_c| + |R_c|}{\sqrt{\left(N_\alpha^s + N_\alpha^r\right) \times \left(N_\beta^s + N_\alpha^r\right)}}. \quad (1)$$

In (1), $|S_c| = |S_\alpha \cap S_\beta|$ is the number of common states in α and β, and $|R_c|$ represents the number of common relationships. N_α^s represents the number of states (size) of α, and N_α^r represents the number of relationships in α. For a temporal pattern α of size n, $N_\alpha^s = n$, and $N_\alpha^r = \frac{n(n-1)}{2}$. The value of $sim(\alpha, \beta)$ will be 1 if $\alpha = \beta$ and will be 0 if they do not have common states.

As an example, consider temporal patterns p_2 and p_4 in Fig. 1. $S_{p_2} = \{A, B\}$, and $S_{p_4} = \{A, B, C, D\}$, so $|S_{p_2} \cap S_{p_4}|$ (the number of common states) $= 2$. The patterns only have one common relationship, that is, the relationship (A *before* B). The value of $N_{p_2}^s = 2$, $N_{p_2}^r = 1$, $N_{p_4}^s = 4$, and $N_{p_4}^r = 6$. Therefore, the similarity between patterns p_2 and p_4 can be computed as

$$sim(p_2, p_4) = \frac{2+1}{\sqrt{(2+1) \times (4+6)}} \approx 0.548.$$

By using (1), the similarity matrix of the four temporal patterns in Fig. 1 is

$$SIM = \begin{array}{c} \\ p_1 \\ p_2 \\ p_3 \\ p_4 \end{array} \begin{array}{c} \begin{array}{cccc} p_1 & p_2 & p_3 & p_4 \end{array} \\ \begin{bmatrix} 1 & 0.667 & 0.707 & 0.365 \\ 0.667 & 1 & 0.471 & 0.548 \\ 0.707 & 0.471 & 1 & 0.516 \\ 0.365 & 0.548 & 0.516 & 1 \end{bmatrix} \end{array}.$$

Let $d(x, y)$ be a distance between temporal patterns x and y defined as $d(x, y) = 1 - sim(x, y)$. The distance function d is a metric, because it has the following properties:

1. Nonnegative: $d(x, y) \geq 0$.
2. Symmetry: $d(x, y) = d(y, x)$.
3. Identity: $d(x, y) = 0$ if and only if (iff) $x = y$.
4. Triangle inequality: $d(x, y) \leq d(x, z) + d(z, y)$.

2. Formally, a metric is a function $m(.,.)$ that is nonnegative and symmetric, has the property that $m(x, y) = 0$ *iff* $x = y$, and satisfies the triangle inequality.

4 Signature-Based Index for Retrieval of Temporal Patterns

This section first describes the method used to construct the signature-based index from a collection of temporal patterns and to answer a query using the index. In order to facilitate this discussion, a brief overview of the use of signature files in set retrieval is provided, and the new terminologies used are introduced.

4.1 Signature Files

A *signature* is a bit pattern formed for a data object, which is then stored in the signature file. Signature files were originally proposed in the context of word retrieval in text databases [34]. Recently, they have been used in a wider set of applications, including office automation, hypertext systems, relational and object-oriented databases, and data mining [35].

This brief review of signature files focuses on their use in facilitating the retrieval on set-valued attributes [20], [21], [36]. Given that T and Q denote the target set and query set, respectively, three commonly used set-valued queries are

1. Subset query ($T \supseteq Q$). The target set is a superset of the query set.
2. Superset query ($T \subseteq Q$). The target set is a subset of the query set.
3. Equality query ($T \equiv Q$). The target set is equal to the query set.

An initial target signature is generated for each target set as follows: Each element in a target set is hashed to a bit pattern termed an *element signature*. All element signatures are of bit length F, and exactly b bits are set to "1," where $b < F$. F is termed the *length* of a signature, while b is termed the *weight* of an element's signature. A target signature is obtained by the bitwise union of all element signatures. The pairing of a target signature and an identifier of the object containing that target set is stored in the signature file.

In set retrieval using signature files, set-valued queries are processed in the following way. A query signature is generated from the query set Q (in the same way as the target signature). During the next step, the *filtering step*, each signature in the signature file is examined with the query signature and becomes a *drop* (a candidate that may satisfy the query) if it satisfies a predefined condition as follows:

1. $T \supseteq Q : target\ signature\ \wedge\ query\ signature = query\ signature$.
2. $T \subseteq Q : target\ signature\ \wedge\ query\ signature = target\ signature$.
3. $T \equiv Q : query\ signature = target\ signature$.

In the next step, *false-drop resolution*, each candidate is retrieved and examined to see if it actually satisfies the query condition. Candidates that fail the test are termed *false drops*, while successful candidates are termed *actual drops*. False drops occur due to the collision of element signatures and the superimposed coding method. False drops affect the number of block accesses (I/O time), as well as the processing time used to decide whether a target set should be returned to the user. The main signature file issue is therefore the proper control of false drops.

Kitagawa et al. [36] derived formulas to estimate the probability of false drops. Given that F_d is the false-drop probability, the value of F_d is minimized if the weight of element signature (m) adheres to the following formulas:

$$m_{opt} = \frac{F \ln 2}{|D_t|} \quad (T \supseteq Q), \tag{2a}$$

$$m_{opt} = \frac{F \ln 2}{|D_q|} \quad (T \subseteq Q), \tag{2b}$$

$$m_{opt} = \frac{F \ln 2}{|D_t|} \quad (T \equiv Q), \tag{2c}$$

where $|D_t|$ and $|D_q|$ are the cardinality of the target set and query set, respectively.

A number of approaches have been proposed in signature file organizations. The sequential signature file (SSF) method stores signature/reference pairs sequentially in the signature file. SSF is easy to implement and requires low storage space and update cost; however, during retrieval, a full scan of the signature file is required. To improve retrieval performance, the bit-slice signature file (BSSF) method stores signatures in a columnwise manner [21]. BSSF uses F bit-slice files, one for each bit position of the set signatures. In retrieval, only a subset of the F bit-slice files need to be scanned reducing the search cost. However, the update cost is greater. For example, the insertion of a new set of signatures requires up to F disk accesses. More complex signature file organizations have also been proposed, for example, S-trees [23], signature trees [37], and extendible signature hashing [20]. In this paper, we focus on SSF and BSSF.

4.2 Constructing Signature Files for Temporal Patterns

The signature of a temporal pattern is created by converting the temporal pattern into an *equivalent set* from which the signature is then generated. The idea builds on previous work [29], [30], [31] in which the signature of a sequential pattern is generated by first converting the sequential pattern into its equivalent set. Two functions are required to create the equivalent set of a temporal pattern. The first function is used to map a set of states in the pattern into a set of integers, while the second maps the relationships between states into integers. These two functions are defined in the following:

Definition 5 (state mapping). *Given a set of states S, a* state mapping *function $f(x)$ is a function that transforms a state type $x \in S$ into an integer value, such that $f(x) \neq f(y)$ for $x \neq y$, where $x, y \in S$.*

Let a set of states $S = \{A, B, C, D\}$. An example of a simple state mapping function $f(x)$ can be defined as $f(A) = 1$, $f(B) = 2$, $f(C) = 3$, and $f(D) = 4$. This function maps each state into a unique value.

Definition 6 (relationship mapping). *Given a set of states S and a set of relations Rel, a* relationship mapping $g(x, y, \mathbf{r})$

TABLE 1
Equivalent Sets and Signatures of Temporal Patterns

Pattern	Equivalent set	Signature
p_1	$\{1, 2, 30\}$	0100 0110
p_2	$\{1, 2, 22\}$	0100 0110
p_3	$\{1, 2, 4, 30, 32, 52\}$	0101 0111
p_4	$\{1, 2, 3, 4, 22, 31, 32, 59, 60, 28\}$	1101 1111

is a function that transforms a relationship $(x\ \mathbf{r}\ y)$ *into an integer value, where* $x, y \in S$, *and* $\mathbf{r} \in Rel$.

It is desirable to have a unique mapping for each relationship. However, designing such mapping is not trivial. Therefore, it is sufficient to have a function g such that $g(x, y, \mathbf{r}) \neq g(y, x, \mathbf{r})$ for any $x, y \in S, \mathbf{r} \in Rel$. Consider a function $g(x, y, \mathbf{r}) = h(\mathbf{r}) \cdot f(x) + f(y)$, where f is a state mapping function, and h is a function that maps $\mathbf{r} \in Rel$ into an integer. Let us define $h(\mathbf{r})$ as $h(=) = N$, $h(\mathbf{c}) = 2N$, $h(\mathbf{fi}) = 3N$, $h(\mathbf{s}) = 4N$, $h(\mathbf{o}) = 5N$, $h(\mathbf{m}) = 6N$, and $h(\mathbf{b}) = 7N$, where N is the number of states in S. Using this definition of h, g will have the property that $g(x, y, \mathbf{r}) \neq g(y, x, \mathbf{r})$ for any $x, y \in S$ and $\mathbf{r} \in Rel$. For example, if $S = \{A, B, C, D\}$ and f is defined above, then

$$g(A, B, \mathbf{b}) = (28 \times 1) + 2 = 30 \text{ and}$$
$$g(B, A, \mathbf{b}) = (28 \times 2) + 1 = 57,$$

which results in $g(A, B, \mathbf{b}) \neq g(B, A, \mathbf{b})$.

Having defined mapping functions f and g, the equivalent set of a temporal pattern can be defined using these two functions.

Definition 7 (equivalent set). *Given a temporal pattern* p *of size* k, $S_p = \langle s_1, \ldots, s_k \rangle$ *is the list of states in* p, *and* M_p *is a* $k \times k$ *matrix whose element* $M_p[i, j]$ *denotes the relationship between states* s_i *and* s_j *in* S_p. *The equivalent set of* p, $E(p)$, *is defined as*

$$E(p) = \left(\bigcup_{i=1}^{k} \{f(s_i)\} \right) \bigcup \left(\bigcup_{i=1}^{k-1} \bigcup_{j=i+1}^{k} \{g(s_i, s_j, \mathbf{r})\} \right),$$

where $\mathbf{r} = M_p[i, j]$.

For example, using the mapping functions f and g in the previous example, the equivalent set of patterns p_1 and p_3 (Fig. 1) can be computed as follows:

$$E(p_1) = \{(f(A)\} \cup \{f(B)\} \cup \{(g(A, B, b)\} = \{1, 2, 30\},$$
$$E(p_3) = \{(f(A)\} \cup \{f(B)\} \cup \{f(D)\} \cup \{(g(A, B, b)\} \cup \{(g(A, D, b)\} \cup \{(g(B, D, m)\} = \{1, 2, 4, 30, 32, 52\}.$$

Equivalent sets of the other temporal patterns are shown in the second column of Table 1.

It can be observed that if a temporal pattern is a subpattern of another pattern, then the equivalent set of the first pattern is a subset of the second pattern's equivalent set. For example, p_1 is a subpattern of p_3; therefore, $E(p_1) \subseteq E(p_3)$ (Table 1). This property is formalized in the following.

Property 1. *Given two temporal patterns* p *and* q *and the corresponding equivalent sets* $E(p)$ *and* $E(q)$, *the following properties hold for any two temporal patterns and their equivalent sets:*

1. $p \sqsupseteq q \rightarrow E(p) \supseteq E(q)$.
2. $p \sqsubseteq q \rightarrow E(p) \subseteq E(q)$.
3. $p = q \rightarrow E(p) = E(q)$.

Definition 8 (signature). *The signature of an equivalent set* E, *denoted* $sig(E)$, *is an* F-bit *binary number created by the bitwise union of all element signatures in* E. *Each element signature has an* F-bit *length, and* m-bits *are set to "1."*

For example, given $F = 8$ and $m = 1$, the signature of element $e \in E$ is an 8-bit binary number that can be computed by a hash function $hash(e) = 2^{(e \bmod F)}$. For the set $E(p_3) = \{1, 2, 4, 30, 32, 52\}$, its element signatures are

$$hash(1) = 00000010,\ hash(2) = 00000100,$$
$$hash(4) = 00010000,\ hash(30) = 01000000,$$
$$hash(32) = 00000001,$$

and $hash(52) = 00010000$. The signature of $E(p_3)$ is computed using the bitwise union of all these element signatures, and the resulting signature is "01010111." Using the same method, the signatures of the other temporal patterns are shown in the third column of Table 1.

Property 2. *Given two equivalent sets* $E(p)$ *and* $E(q)$ *and their corresponding signatures* sig_p *and* sig_q, *the signatures of equivalent sets have the following properties:*

1. $E(p) \supseteq E(q) \rightarrow sig_p \wedge sig_q = sig_q$.
2. $E(p) \subseteq E(q) \rightarrow sig_p \wedge sig_q = sig_p$.
3. $E(p) = E(q) \rightarrow sig_p = sig_q$.

Combining Properties 1 and 2, the relations between temporal patterns and their signatures are expressed in the following properties.

Property 3. *Given two temporal patterns* p *and* q *and their corresponding signatures* sig_p *and* sig_q, *these signatures have the following properties:*

1. $p \sqsupseteq q \rightarrow sig_p \wedge sig_q = sig_q$.
2. $p \sqsubseteq q \rightarrow sig_p \wedge sig_q = sig_p$.
3. $p = q \rightarrow sig_p = sig_q$.

As an example, consider temporal patterns p_1 and p_3, where $p_1 \sqsubseteq p_3$. It can be seen in Table 1 that $sig_{p_1} \wedge sig_{p_3} = 01000110 \wedge 01010111 = 01000110$, which is the value of sig_{p_1}.

Using these methods, the signature file of temporal patterns in database $\mathcal{D}$ can be created as follows: For each temporal pattern $p \in \mathcal{D}$, its equivalent set $E(p)$ is calculated, and then, its signature E_p is generated. This signature, together with the temporal pattern identifier (*pid*), is inserted into the signature file. The actual insertion depends on the signature file organization. For example, for SSF, the signature is appended to the end of the file, while for BSSF, each signature bit is appended to the end of the corresponding bit-slice file. Only the signatures are stored in the signature file, while the equivalent sets are only to

facilitate the computation of signatures. This procedure is outlined in Algorithm 4.1.

Algorithm 4.1. Constructing a signature file of temporal patterns
Input: A database $\mathcal{D}$ of temporal patterns
Output: SignatureFile
1: **for** each $p \in \mathcal{D}$ **do**
2: $E(p) = \text{Equivalent\_Set}(p)$
3: $sig_p = \text{Signature}(E(p))$
4: Insert $\langle sig_p, pid_p \rangle$ into SignatureFile
5: **end for**
6: return SignatureFile

4.3 Answering Content-Based Queries Using the Signature File

4.3.1 Subpattern Queries

Given a temporal pattern database $\mathcal{D}$ and a query pattern q, the algorithm for evaluating subpattern queries is called $evaluateSubPattern(\mathcal{D}, q)$, which finds temporal patterns in $\mathcal{D}$ that contain q. If the signatures are stored in SSF, $evaluateSubPattern(\mathcal{D}, q)$, as presented in Algorithm 4.2, is used. The equivalent set $E(q)$ of q is first calculated and, then, a query signature sig_q is formed. Each target signature sig_p in SSF is then examined against the query signature sig_q. If the target signature satisfies the search condition $sig_p \wedge sig_q = sig_q$ (the first property in Property 3), the corresponding temporal pattern becomes a drop, and its identifier is added to the pattern ID (or PID) list. Then, during *false-drop verification*, each drop is checked to determine if it actually satisfies the query condition.

Algorithm 4.2. Pseudocode of evaluateSubPattern using SSF
Input: Temporal pattern database $\mathcal{D}$, a query pattern q
Output: AnswerSet
1: $E(q) = \text{Equivalent\_Set}(q)$
2: $sig_q = \text{Signature}(E(q))$
3: **for** each $\langle sig_p, pid_p \rangle \in SSF$ **do**
4: **if** $sig_p \wedge sig_q = sig_q$ **then**
5: Add pid_p into the PID list
6: **end if**
7: **end for**
8: **for** each pid_p in the PID list **do**
9: Retrieve p from $\mathcal{D}$
10: **if** $p \sqsupseteq q$ **then**
11: Add p into AnswerSet
12: **end if**
13: **end for**
14: return AnswerSet

On the other hand, if the signatures are stored in BSSF, $evaluateSubPattern(\mathcal{D}, q)$ proceeds in a slightly different way, and is shown in Algorithm 4.3. The search condition $sig_p \wedge sig_q = sig_q$ cannot be used, since the target signature sig_p is scattered across bit-slice files. Instead, the bit slices corresponding to the bit positions set to "1" in sig_q are retrieved and, then, a bitwise intersect (bitwise AND) operation is performed on the retrieved bit slices. The corresponding temporal pattern becomes a drop if the resulting bit entry is equal to "1," and its identifier is added to the PID list. The false-drop resolution step is the same as in SSF.

Algorithm 4.3. Pseudocode of evaluateSubPattern using BSSF
Input: Temporal pattern database $\mathcal{D}$, a query pattern q
Output: AnswerSet
1: $E(q) = \text{Equivalent\_Set}(q)$
2: $sig_q = \text{Signature}(E(q))$
3: Retrieve the bit slices corresponding to the bit position set to "1" in sig_q
4: Perform a bitwise intersect operation on the retrieved bit slices
5: **for** each entry where "1" is set in the resulting intersect bit slice **do**
6: Add the corresponding pid_p into the PID list
7: **end for**
8: **for** each pid_p in the PID list **do**
9: Retrieve p from $\mathcal{D}$
10: **if** $p \sqsupseteq q$ **then**
11: Add p into AnswerSet
12: **end if**
13: **end for**
14: return AnswerSet

4.3.2 Superpattern Queries

Let $evaluateSuperPattern(\mathcal{D}, q)$ be the algorithm for evaluating superpattern queries, that is, for finding temporal patterns in $\mathcal{D}$ that are contained in q. If SSF is used, the algorithm is similar to $evaluateSubPattern$ in Algorithm 4.2 except the search condition $sig_p \wedge sig_q = sig_q$ (line 4) is replaced with $sig_p \wedge sig_q = sig_p$, and the query condition $p \sqsupseteq q$ (line 10) is replaced with $p \sqsubseteq q$.

If BSSF is used, the $evaluateSubPattern$ retrieves bit slices corresponding to the bit positions set to "0" in sig_q and performs a bitwise union (bitwise OR) operation on them. The corresponding temporal pattern becomes a drop if the resulting bit entry equals "0." Each drop is then validated with respect to the query condition $p \sqsubseteq q$. The pseudocode of $evaluateSuperPattern$ on BSSF is shown in Algorithm 4.4.

Algorithm 4.4. Pseudocode of evaluateSuperPattern using BSSF
Input: Temporal pattern database $\mathcal{D}$, a query pattern q
Output: AnswerSet
1: $E(q) = \text{Equivalent\_Set}(q)$
2: $sig_q = \text{Signature}(E(q))$
3: Retrieve the bit slices corresponding to the bit position set to "0" in sig_q
4: Perform a bitwise union operation on the retrieved bit slices
5: **for** each entry where "0" is set in the resulting union bit slice **do**
6: add the corresponding pid_p into the PID list
7: **end for**
8: **for** each pid_p in the PID list **do**
9: Retrieve p from $\mathcal{D}$

```
10:     if p ⊑ q then
11:         Add p into AnswerSet
12:     end if
13: end for
14: return AnswerSet
```

4.3.3 Equality Queries

Let $evaluateEquality(\mathcal{D}, q)$ be the algorithm for processing equality queries. Using SSF, the algorithm follows Algorithm 4.2, except that the search condition $sig_p \wedge sig_q = sig_q$ is replaced with $sig_p = sig_q$, and the query condition $p \sqsupseteq q$ is replaced with $p = q$.

When BSSF is used, the algorithm requires access to all bit-slice files, not only part of them. In order to decide if $sig_p = sig_q$, each bit of sig_q must be compared with the corresponding bit of sig_p that is stored in different bit-slice files. This is only possible by accessing all bit-slice files.

4.3.4 K-nearest Subpattern Queries

K-nearest queries are used to limit the number of patterns generated by subpattern queries. Given that a subpattern query generates n temporal patterns containing the query pattern q, the k-nearest subpattern query is used to choose k of n temporal patterns, where $k < n$, that are most similar to q. Using SSF, the query can be processed by modifying Algorithm 4.2 as follows: During false-drop resolution, when a temporal pattern p becomes an actual drop, the similarity between the query patterns q and p is calculated using (1). The k patterns with the largest similarity measures are recorded. The query can be processed using BSSF by modifying Algorithm 4.3 in a similar way.

5 Experiments

To assess the performance of the proposed methods, the *evaluateSubPattern*, $evaluateSuperPattern$, and *evaluateEquality* were implemented on SSF and BSSF signature files. In addition, sequential versions of the methods (SEQ) were also implemented as baseline methods, which process queries by sequentially retrieving the target pattern (without using an index) from the database and comparing it against the query pattern. All programs are written in Java Language. The experiments were conducted on synthetic data sets on a 1.3-GHz Intel Celeron PC with 384 Mbytes of RAM running Windows XP Professional.

The following sections show the performance of $evaluateSubPattern$ and $evaluateSuperPattern$ for processing subpattern and superpattern queries, respectively. Update cost is not considered as it is assumed that the index is only created once after frequent patterns have been generated by a data mining process. Experimental parameters are listed in Table 2.

The temporal pattern database was generated using ARMADA [17] from an interval sequence database D_s containing 10,000 interval sequences ($|D_s|$), with an average length of 10 ($|C|$), and 100 different types of states (N). Using the minimum support of 0.08 percent and the maximum gap of 200, ARMADA generated a set of 106,409 frequent temporal patterns. A temporal pattern database D is populated from this set of temporal patterns.

TABLE 2
Parameters

Symbol	Definition
F	Size of signature (in bits)
m	Weight of a signature element
N	Number of states
$\|D_s\|$	Size of sequence database
$\|C\|$	Average size of sequences
$\|D\|$	Size of temporal pattern database
$\|T\|$	Average size of temporal patterns
Q	Size of a query pattern

First, the size of temporal pattern t was determined randomly from a Poisson distribution with a mean equal to $|T|$. Then, a temporal pattern of size t is randomly picked from the set of frequent temporal patterns and added to the database D.

To guarantee that the evaluated queries do not return an empty result set, query patterns were generated as follows: For subpattern queries, the temporal pattern of size 5 with the highest support in D was selected, then the states from the pattern starting from the last state were individually removed, resulting in a set of five queries. A similar method was performed for superpattern queries by selecting a temporal pattern of size 10 to generate a further set of five queries.

5.1 Effect of Signature Size on the Number of False Drops

This experiment observed how the size of the signature affects the number of false drops in $evaluateSubPattern$ and $evaluateSuperPattern$. It also determined the optimal parameters for each query type in the experimental environment, particularly the values of F and m. As can be seen from (2), the false-drop probability depends on F, m, and the cardinalities of the query set and target set. The value of m was set to 1, and the value of F increased until no further performance improvement could be perceived. The size of the database $|D| = 50,000$, the average size of temporal pattern $|T| = 5$, and the number of states $N = 100$. The size of signature F was varied from 8 to 128 bits. The number of false drops was measured.

Figs. 2a and 2b show the number of false drops for $evaluateSubPattern$ and $evaluateSuperPattern$, respectively. The number of false drops is similar for both SSF and BSSF, since it does not depend on the signature file structures. As can be seen, the number of false drops consistently decreases as the size of the signature increases, and the number of false drops is also influenced by the size of the query pattern. For $evaluateSubPattern$ (Fig. 2a), the larger the query pattern, the lower the number of false drops. Conversely, the larger the query pattern, the higher the number of false drops for $evaluateSuperPattern$ (Fig. 2b). The best recorded performance improvement were achieved with a signature size between 16 and 32 bits, at which point the number of false drops decreases significantly.

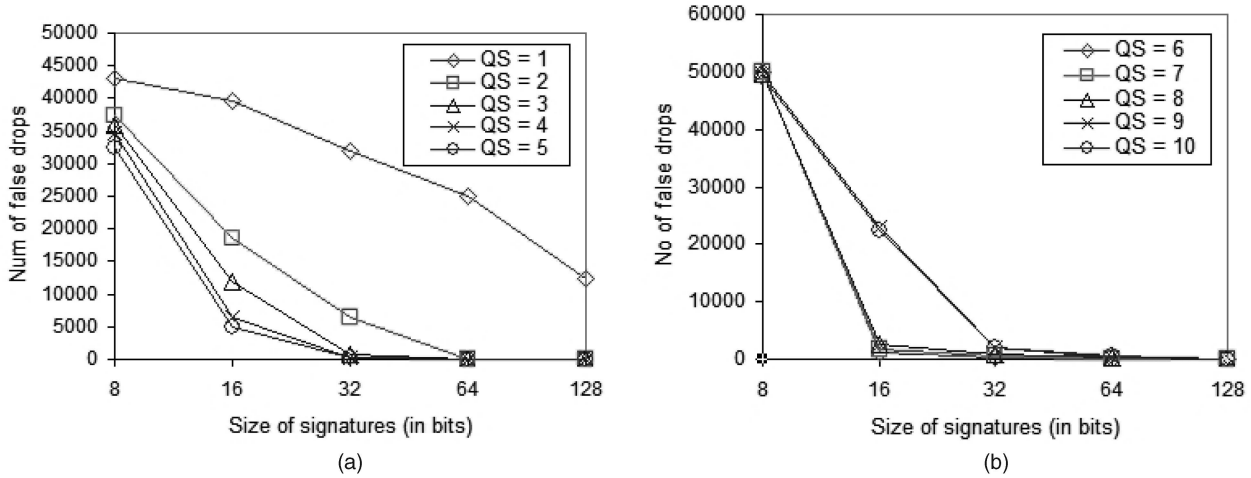

Fig. 2. Effect of signature size on the number of false drops. (a) evaluateSubPattern. (b) evaluateSuperPattern.

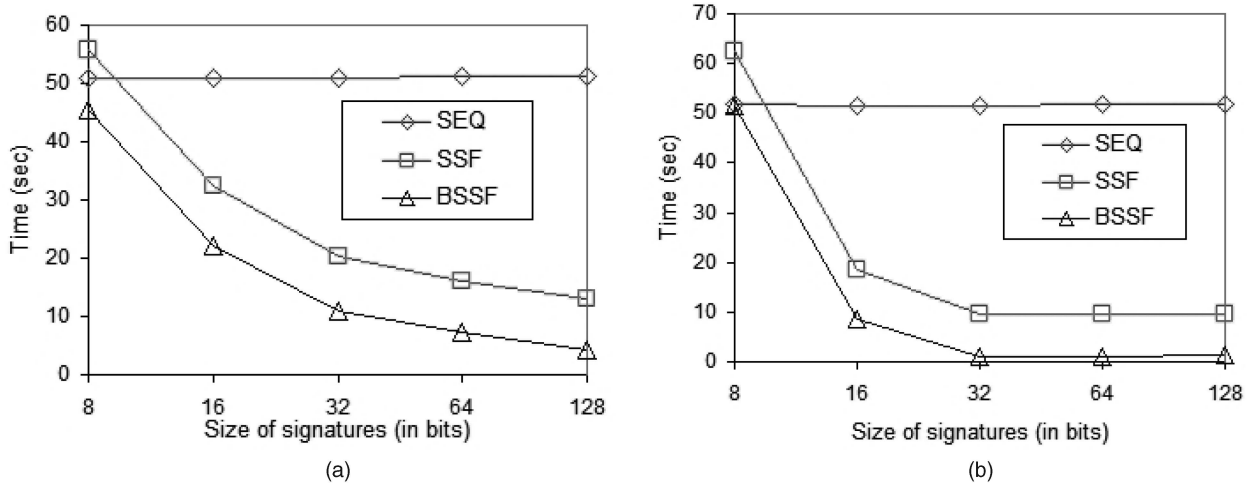

Fig. 3. Effect of signature size on query processing time. (a) evaluateSubPattern. (b) evaluateSuperPattern.

5.2 Effect of Signature Size on Query Processing Time

This experiment used the above data set to compare the relative performance of $evaluateSubPattern$ and $evaluateSuperPattern$ on SEQ, SSF, and BSSF. Each method was run on each of the queries used in the previous experiment. Fig. 3a shows the total time required by $evaluateSubPattern$, while Fig. 3b shows the total time required by $evaluateSuperPattern$.

The figures show that the query processing times of both methods on SSF and BSSF are proportional with the number of false drops from the previous experiments. Both methods gain the best performance improvement when the size of signature is between 16 and 32. Both methods perform better on BSSF. For small values of F, the query processing times of both methods are almost similar on SEQ, SSF, and BSSF. This is because when $F \leq 8$, the number of false drops becomes so high that when SSF and BSSF are used, the algorithms have to retrieve almost all patterns in the database during the verification step. Finally, both methods show a marked improvement on SSF and BSSF over SEQ.

5.3 Effect of Database Size on the Query Processing Time

In order to observe how the methods scale with respect to the database size, five data sets were generated in which $|T| = 5$ and $N = 100$. The size of database $|D|$ was varied from 10,000 to 100,000. Both methods were run on each data set using two values of F (32 and 64 bits). Fig. 5a shows the total time required by $evaluateSubPattern$ on SEQ, SSF 32 bits, SSF 64 bits, BSSF 32 bits, and BSSF 64 bits to process five queries. Fig. 5b shows the total time required by $evaluateSuperPattern$ to process the five queries.

In general, the processing times are proportional to the database size. Both methods remain the slowest on SEQ but show the fastest or the best scaling behavior on BSSF.

5.4 Experiments on Real Data

In addition to using synthetic data sets, we have also performed a series of experiments on a real data set. The data set is the ASL database created by the National Center for Sign Language and Gesture Resources, Boston University, which is available online at http://www.bu.edu/asllrp/. The Sign-Stream database used in this experiment consists of a collection of 730 utterances, where each utterance associates a segment of video with a detailed

TABLE 3
Generated Temporal Patterns from the ASL Database

Pattern size	Number of patterns
1	177
2	8768
3	65178
4	100847
5	32221
6	3239
7	150
Total	210580

transcription. Every utterance can be considered as a state sequence which contains a number of ASL gestural and grammatical fields (e.g., eyebrow raise, head tilt forward, whquestion), each one occurring over a time interval.

A temporal pattern database was generated by running ARMADA on this interval sequence database and setting the minimum support to 1 percent and the minimum gap to 100. The resulting temporal pattern database contains 210,580 frequent temporal patterns, as summarized in Table 3. The temporal pattern database was then used in the experiments to compare the relative performance of $evaluateSubPattern$ and $evaluateSuperPattern$ on SEQ, SSF, and BSSF. Each method was run on each of the queries used in the previous experiment.

Fig. 4a shows the total time required by *evaluateSubPattern*, while Fig. 4b shows the total time required by $evaluateSuperPattern$. The figures show that the query processing times of both methods on SSF and BSSF decrease with the increasing size of signatures. Both methods gain the best performance improvement when the size of signature is between 16 and 32. Both methods show a marked improvement on SSF and BSSF over SEQ but perform the best on BSSF.

6 Conclusion and Future Work

The use of a signature-based index for content-based retrieval of temporal patterns has been presented. The signatures of temporal patterns are created by first converting temporal patterns into equivalent sets and then generating the signatures from the equivalent sets. The study focused on the sequential and BSSF organizations, and a

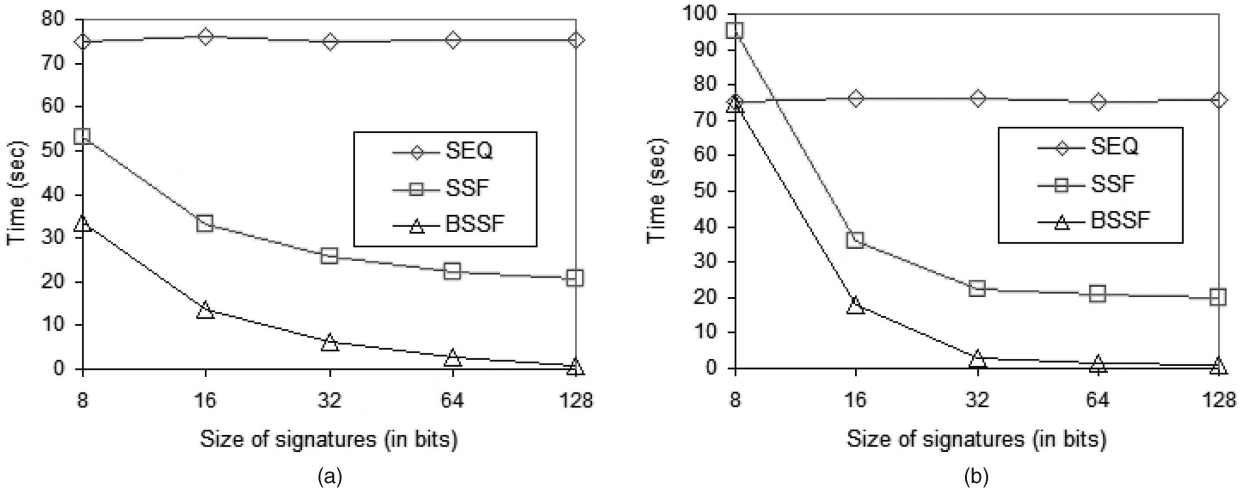

Fig. 4. Effect of signature size on query processing time (ASL database). (a) evaluateSubPattern. (b) evaluateSuperPattern.

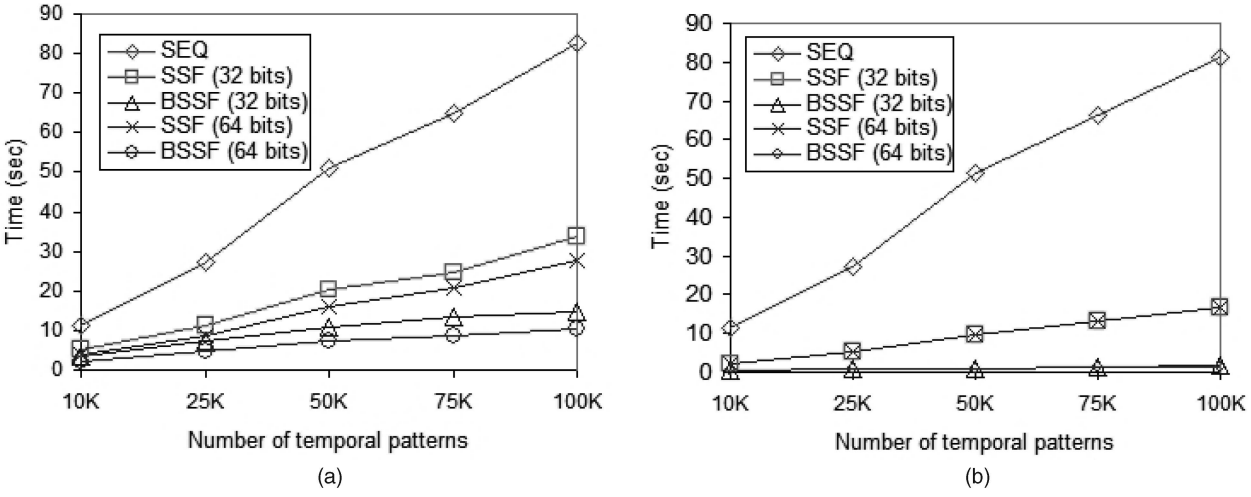

Fig. 5. Effect of database size on query processing time. (a) evaluateSubPattern. (b) evaluateSuperPattern.

series of experiments compared the performance of both signature files in processing subpattern and superpattern queries.

In conclusion, the use of signature files improves the performance of temporal pattern retrieval. The bit-slice signature file performs better than the SSF and is a good choice for content-based retrieval of temporal patterns. This retrieval system is currently being combined with visualization techniques for monitoring the behavior of a single pattern or a group of patterns over time.

ACKNOWLEDGMENTS

Edi Winarko worked on this research as part of his doctoral study at Flinders University, South Australia.

REFERENCES

[1] T. Imielinski and A. Virmani, "Association Rules ... and What's Next? Towards Second Generation Data Mining Systems," *Proc. Second East European Symp. Advances in Databases and Information Systems (ADBIS '98),* pp. 6-25, 1998.

[2] L. Geng and H.J. Hamilton, "Interestingness Measures for Data Mining: A Survey," *ACM Computing Surveys,* vol. 38, no. 3, 2006.

[3] H. Toivonen, M. Klemettinen, P. Ronkainen, K. Hatonen, and H. Mannila, "Pruning and Grouping of Discovered Association Rules," *Proc. ECML Workshop Statistics, Machine Learning, and Knowledge Discovery in Databases,* pp. 47-52, 1995.

[4] B. Lent, A.N. Swami, and J. Widom, "Clustering Association Rules," *Proc. 13th Int'l Conf. Data Eng. (ICDE '97),* W.A. Gray and P.-Å. Larson, eds., pp. 220-231, 1997.

[5] B. Liu, W. Hsu, and Y. Ma, "Pruning and Summarizing the Discovered Associations," *Proc. ACM SIGKDD '99,* pp. 125-134, 1999.

[6] B. Liu, M. Hu, and W. Hsu, "Multi-Level Organization and Summarization of the Discovered Rules," *Proc. ACM SIGKDD,* 2000.

[7] M. Klemettinen, H. Mannila, P. Ronkainen, H. Toivonen, and A. Verkamo, "Finding Interesting Rules from Large Sets of Discovered Association Rules," *Proc. Third Int'l Conf. Information and Knowledge Management (CIKM '94),* N. Adam, B. Bhargava, and Y. Yesha, eds., pp. 401-407, 1994.

[8] R. Meo, G. Psaila, and S. Ceri, "A New SQL-Like Operator for Mining Association Rules," *Proc. 22nd Int'l Conf. Very Large Data Bases (VLDB '96),* M.T. Vijayaramam, A. Buchmann, C. Mohan, and L.N. Sarda, eds., pp. 122-133, 1996.

[9] J. Han, J.Y. Chiang, S. Chee, J. Chen, Q. Chen, S. Cheng, W. Gong, M. Kamber, K. Koperski, G. Liu, Y. Lu, N. Stefanovic, L. Winstone, B.B. Xia, O.R. Zaiane, S. Zhang, and H. Zhu, "DBMiner: A System for Data Mining in Relational Databases and Data Warehouses," *Proc. ACM SIGKDD,* 1996.

[10] A. Netz, S. Chaudhuri, U.M. Fayyad, and J. Bernhardt, "Integrating Data Mining with SQL Databases: OLE DB for Data Mining," *Proc. 17th Int'l Conf. Data Eng. (ICDE'01),* pp. 379-387, 2001.

[11] T. Imielinski and A. Virmani, "MSQL: A Query Language for Database Mining," *J. Data Mining and Knowledge Discovery,* vol. 3, no. 4, pp. 373-408, 1999.

[12] A. Tuzhilin and B. Liu, "Querying Multiple Sets of Discovered Rules," *Proc. ACM SIGKDD '02,* pp. 52-60, 2002.

[13] C.M. Antunes and A.L. Oliveira, "Temporal Data Mining: An Overview," *Proc. ACM SIGKDD Workshop Temporal Data Mining,* pp. 1-13, 2001.

[14] X. Chen and I. Petrounias, "A Framework for Temporal Data Mining," *Proc. Ninth Int'l Conf. Database and Expert Systems Applications (DEXA '98),* pp. 796-805, 1998.

[15] J.F. Roddick and M. Spiliopoulou, "A Survey of Temporal Knowledge Discovery Paradigms and Methods," *IEEE Trans. Knowledge and Data Eng.,* vol. 14, no. 4, pp. 750-767, Mar./Apr. 2002.

[16] F. Höppner, "Learning Temporal Rules from State Sequences," *Proc. IJCAI Workshop Learning from Temporal and Spatial Data,* pp. 25-31, 2001.

[17] E. Winarko and J.F. Roddick, "ARMADA—An Algorithm for Discovering Richer Relative Temporal Association Rules from Interval-Based Data," *Data and Knowledge Eng.,* vol. 63, no. 1, pp. 76-90, 2007.

[18] D. Comer, "The Ubiquitous B-Tree," *Computing Surveys,* vol. 11, no. 2, pp. 121-137, 1979.

[19] A. Guttman, "R-Trees: A Dynamic Index Structure for Spatial Searching," *Proc. ACM SIGMOD '84,* pp. 47-57, 1984.

[20] S. Helmer and G. Moerkotte, "A Performance Study of Four Index Structures for Set-Valued Attributes of Low Cardinality," *VLDB J.,* vol. 12, no. 3, pp. 244-261, 2003.

[21] Y. Ishikawa, H. Kitagawa, and N. Ohbo, "Evaluation of Signature Files as Set Access Facilities in OODBs," *Proc. ACM SIGMOD '93,* P. Buneman and S. Jajodia, eds., pp. 247-256, 1993.

[22] T. Morzy and M. Zakrzewicz, "Group Bitmap Index: A Structure for Association Rules Retrieval," *Proc. ACM SIGKDD '98,* R. Agrawal, P. Stolorz, and G. Piatetsky-Shapiro, eds., pp. 284-288, 1998.

[23] U. Deppisch, "S-Tree: A Dynamic Balanced Signature Index for Office Retrieval," *Proc. ACM SIGIR '86,* pp. 77-87, 1986.

[24] N. Mamoulis, D.W. Cheung, and W. Lian, "Similarity Search in Sets and Categorical Data Using the Signature Tree," *Proc. 19th Int'l Conf. Data Eng. (ICDE '03),* U. Dayal, K. Ramamritham, and T. Vijayaraman, eds., pp. 75-86, 2003.

[25] N. Mamoulis, H. Cao, G. Kollios, M. Hadjieleftheriou, Y. Tao, and D.W. Cheung, "Mining, Indexing, and Querying Historical Spatiotemporal Data," *Proc. ACM SIGKDD '04,* pp. 236-245, 2004.

[26] D.L. Lee and C.-W. Leng, "A Partitioned Signature File Structure for Multiattribute and Text Retrieval," *Proc. Sixth Int'l Conf. Data Eng. (ICDE '90),* pp. 389-397, 1990.

[27] F. Rabitti and P. Zezula, "A Dynamic Signature Technique for Multimedia Databases," *Proc. ACM SIGIR '90,* J.-L. Vidick, ed., pp. 193-210, 1990.

[28] P. Zezula, F. Rabitti, and P. Tiberio, "Dynamic Partitioning of Signature Files," *ACM Trans. Information Systems,* vol. 9, no. 4, pp. 336-367, 1991.

[29] T. Morzy, M. Wojciechowski, and M. Zakrzewicz, "Optimizing Pattern Queries for Web Access Logs," *Proc. Fifth East European Conf. Advances in Databases and Information Systems (ADBIS '01),* pp. 141-154, 2001.

[30] A. Nanopoulos, M. Zakrzewicz, T. Morzy, and Y. Manolopoulos, "Efficient Storage and Querying of Sequential Patterns in Database Systems," *Information and Software Technology,* vol. 45, pp. 23-34, 2003.

[31] M. Zakrzewicz, "Sequential Index Structure for Content-Based Retrieval," *Proc. Fifth Pacific-Asia Conf. Knowledge Discovery and Data Mining (PAKDD '01),* pp. 306-311, 2001.

[32] J. Allen, "Maintaining Knowledge about Temporal Intervals," *Comm. ACM,* vol. 26, no. 11, pp. 832-843, 1983.

[33] J. Xiao, Y. Zhang, X. Jia, and T. Li, "Measuring Similarity of Interests for Clustering Web-Users," *Proc. 12th Australasian Database Conf. (ADC '01),* M. Orlowska and J. Roddick, eds., pp. 107-114, 2001.

[34] C. Faloutsos and S. Christodoulakis, "Signature Files: An Access Method for Documents and Its Analytical Performance Evaluation," *ACM Trans. Office Information Systems,* vol. 2, no. 4, pp. 267-288, 1984.

[35] Y. Chen, "On the General Signature Trees," *Proc. 16th Int'l Conf. Database and Expert Systems Applications (DEXA '05),* pp. 207-219, 2005.

[36] H. Kitagawa, Y. Fukushima, Y. Ishikawa, and N. Ohbo, "Estimation of False Drops in Set-Valued Object Retrieval with Signature Files," *Proc. Fourth Int'l Conf. Foundations of Data Organization and Algorithms (FODO '93),* pp. 146-163, 1993.

[37] Y. Chen, "Building Signature Trees into OODBs," *J. Information Science and Eng.,* vol. 20, no. 2, pp. 275-304, 2004.

[38] J. Yang and M. Hu, "Trajpattern: Mining Sequential Patterns from Imprecise Trajectories of Mobile Objects," *Proc. 10th Int'l Conf. Extending Database Technology (EDBT '06),* pp. 664-681, 2006.

Edi Winarko received the BSc degree in statistics from Gadjahmada University, Indonesia, the MSc degree in computer science from the School of Computing, Queen's University, Canada, and the PhD degree on the discovery and retrieval of temporal rules in interval sequence data from the School of Informatics and Engineering, Flinders University, Australia. He is currently with Gadjah Mada University, Yogyakarta, Indonesia. His current interests include temporal data mining, Web mining, and information retrieval.

John F. Roddick received the BSc(Eng)(Hons) degree from Imperial College London, the MSc degree from Deakin University, and the PhD degree from La Trobe University. He is currently the SACITT chair in information technology and the head of the School of Informatics and Engineering, Flinders University. He joined Flinders University in 2000 after 10 years of being with the University of South Australia and five years of being with the University of Tasmania. This was followed by 10 years of experience in the computing industry as (progressively) a programmer, analyst, project leader, and consultant. He has published more than 100 papers in a number of areas of computing but specializes 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 modeling, specifically in enhanced database systems semantics such as schema evolution and temporal and spatial systems design and use. He is a member of the IEEE Computer Society.