A Semantics based Approach to Efficient Retrieval of Temporal Patterns

Ritambhra Korpal
Department of Computer Science
University of Pune
Pune, India

Arpita Gopal
Sinhgad Institute of Business Administration and Research
(Affiliated to University of Pune)
Pune, India

ABSTRACT
Temporal data mining unearths patterns from sequential or ordered data. Semantics of these patterns can be different depending on the underlying data, technique used and the purpose of data mining. Patterns included in this paper are taken from three different domains and their structure and semantics are different. First type, which we call temporal patterns, includes a set of states and relationships among the states. Second type, called sequential patterns includes a set of ordered states. Finally, the third type called episodes is a partially ordered set of event types. To index a database of these patterns, Signature based techniques were considered to be a viable option as signatures could accommodate multiple state values as well as the relationship among the states. In this paper we compared the different implementations of signature files when used indexes for database of temporal patterns on various criteria listed in the paper. Further, based on the semantics of sequential patterns and episodes, and the results obtained above, we suggested which implementations would be suitable for databases of other two types of patterns.

General Terms
Indexes, Algorithms, Experimentation, Performance, Measurement, Semantics

Keywords
Temporal Patterns, sequential patterns, episode, Signature Files, Sequential Signature Files, Bit Slice Signature Files, Extendible Signature Hashing, Signature Trees

1. INTRODUCTION
Temporal data mining is concerned with mining of large sequential data sets, i.e. data which is ordered with respect to some index. These data sets could be text, gene sequences, protein sequences, lists of moves in a chess game etc. Here, although there is no notion of time as such, the ordering among the records is very important and is central to the data description/modeling. Temporal patterns are generated whiling mining sequential or ordered data and these can be of different nature depending on the underlying data. Patterns included in this paper are taken from three different domains and their structure and semantics are different. First type, which we call temporal patterns, includes a set of states and relationships among the states. Second type, called sequential patterns, includes a set of ordered states. Finally, the third type called episode, is a partially ordered set of event types. While mining data, a central issue is not only generating patterns but in-depth analysis of generated patterns is essential as well. As the techniques to mine sequential data become more refined, the number of patterns generated becomes very large. Analysing these patterns while they are being generated becomes overwhelming and many interesting and potentially useful patterns may get lost during the process. While many interesting techniques of temporal data mining have been proposed, it has been shown that not all generated patterns are of interest to the user [12], [20]. When the number of generated patterns becomes exceedingly large, post processing of generated patterns becomes essential [3]. For moderate number of patterns, grouping them based on a similarity measure works satisfactorily. But as the number increases, we need different techniques to handle them. One of the approaches could be to store the patterns in a database and later query this database to retrieve the required patterns. Indexes can be used on this database to speed up such queries as the size of the database grows. Signature-based indexing techniques were found to be a viable option as signatures can be generated for each pattern accommodating multiple state values as well as the relationship among the states. There are various implementations of signature files available each with its own strengths and weaknesses. Hence it was imperative to investigate the suitability of each when applied to content-based retrieval of patterns. In this paper, the suitability and limitations of Sequential Signature Files (SSF), Bit-Slice Signature Files (BSSF), Extendible Signature Hashing (ESHF) and Signature Tree (STF) implementations of signature files when applied to database of temporal patterns of were explored. The types of queries considered for efficient response are content-based queries. The content-based queries include equality, sub-pattern and super-pattern queries. These types of queries help to investigate those patterns for which a component of the pattern is already known to us. Based on the semantics of other two types of patterns and results of above study, a suitable implementation was suggested for databases of sequential patterns and episodes.

2. RELATED WORK
Agrawal & Srikant [4] have introduced the problem of finding sequential patterns in large database of customer transactions.
Mannila et al [14] have considered the problem of discovering frequently occurring episodes in a sequence. An episode is a collection of events that occur relatively close to each other in a given partial order.
Edi Winarko et al [7] have studied the application of signature files as an indexing techniques for efficient retrieval of temporal patterns. They have investigated the performance of SSF and BSSF as an indexing technique for efficient retrieval of content-based queries on temporal patterns.
Yangjun Chen and Yibin Chen [21], [22] have proposed a new method to organize signature file into a tree structure, called a signature tree, to speed up the signature file scanning and query

evaluation. They have also analyzed the average time complexity of searching a signature tree and maintenance of signature trees.
Ritambhra and Arpita Gopal [18] have studied the performance of Signature Trees as an indexing technique for efficient retrieval of content-based queries on temporal patterns.
Ritambhra and Arpita Gopal [17] have studied the performance of Extendible Signature Hashing [ESHF] as an indexing technique for efficient retrieval of content-based queries on temporal patterns. They have also compared the performance of ESHF against SSF and BSSF implementations of signature files and have reported that ESHF outperforms the other two implementations.
Y. Ishikawa, H. Kitagawa, and N. Ohbo [23] have applied the signature based indexes for queries involving set-valued attributes in OODBs. They have considered two implementations of signature files: the sequential signature files and bit-slice signature files.
Helmer et all have studied [19] Extendible hashing, proposed by Fagin R, Nievergelt J, Pippenger N, Strong HR [16]. They have modified this indexing method to use signatures instead of hash keys. Theses signature/reference pairs get distributed among various buckets. They investigated the performance of four index structures for set-valued attributes (sequential signature files, signature trees, extendible signature hashing, and inverted lists). The indexes were evaluated on three forms of set-valued queries — equality queries, subset queries, and superset queries.
The comparative studies so far focused mainly on efficient response time. Some other criteria like space overhead, index construction time and scalability were, however, not handled.
Since the set-based indexes do not consider the ordering of items in the sets, whereas ordering is important for indexing and retrieval of temporal patterns, new indexing techniques have been proposed as well [13], [2]. This involves converting the sequential patterns into equivalent sets that accommodate the ordering of items. After that set-based indexing methods can be applied on the equivalent sets.

3. PRELIMINARIES

3.1 Temporal Patterns

The temporal patterns [1] described in this paper consist of two components: a set of state intervals and a set of relationships between those state intervals that represent the order of states within the pattern [9]. These relationships can be before(b), meets(m), overlaps(o), is-finished-by(fi), contains(c) and starts(s) [8]. Fig. 1 shows some temporal patterns defined over set of states S= {A, B, C, D} and set of relationships Rel = { =, b, m, o, fi, c, s}. A pattern β is a subpattern of pattern α, if β can be obtained from α by removing some state intervals.

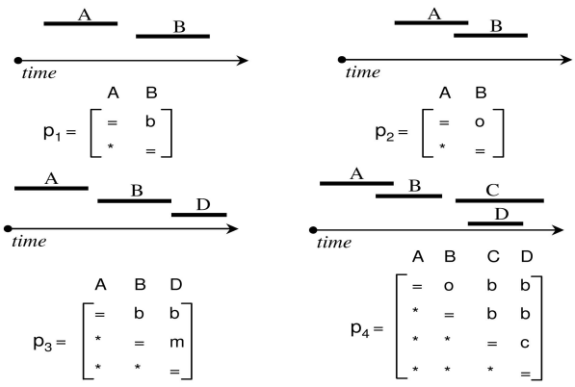

Figure 1. Temporal Patterns

3.2 Sequential Patterns

A sequential pattern is a large and maximal sequence among the set of all large sequences [4]. If a sequence s of itemsets is denoted by ⟨s1 s2..sn⟩, where each sj is an itemset, it is called an n-sequence. E.g ⟨(AB) (F) (BC) (DE)⟩ is a 4-sequence. A sequence a = ⟨a1 a2 . . an⟩ is said to be contained in another sequence b = ⟨b1 b2 . . bm⟩ if there exist integers i1 < i2 < . . < in, such that a1 ⊆ bi1, a2 ⊆ bi2, . . . , an ⊆ bin. That is, an n-sequence a is contained in a m-sequence b if there exists an n-length subsequence in b, in which each itemset contains the corresponding itemsets of a. For example, the sequence ⟨(A)(BC)⟩ is contained in ⟨(AB) (F) (BC) (DE) ⟩ but not in ⟨(BC) (AB) (C) (DE)⟩. Further, a sequence is said to be maximal in a set of sequences, if it is not contained in any other sequence.
Consider an example of a database with 5 customers whose corresponding transaction sequences are as follows:
(1) ⟨*(AB) (ACD) (BE)*⟩,
(2) ⟨*(D) (ABE)*⟩,
(3) ⟨*(A) (BD) (ABEF) (GH)*⟩,
(4) ⟨*(A) (F)*⟩, and
(5) ⟨*(AD) (BEGH) (F)*⟩.
All customer transaction sequences listed above, are maximal in this set of sequences except the sequence of customer 4, which is contained in, transaction sequence of customer 3.

3.3 Episodes

Episodes are patterns that occur sufficiently often in the data presented as a single long sequence [14]. This single *event sequence* is denoted by ⟨ (E1, t1), (E2, t2), ..⟩, where Ei takes values from a finite set of event types E, and ti is an integer denoting the time stamp of the ith event and ∀i, ti ≤ ti+1.

For example, event sequence with 10 events in it can be written as:
⟨(A, 2), (B, 3), (A, 7), (C, 8), (B, 9), (D, 11), (C, 12), (A, 13), (B, 14), (C, 15)⟩

An episode α is defined by a triple (V, ≤, g), where V is a collection of nodes, ≤ is a partial order on V and g : V → E is the mapping that associates each node in the episode with an event type. Thus, an episode is just a partially ordered set of event types. For example, (A → B → C) is a 3-node episode. An episode is said to occur in an event sequence if there exist events in the sequence occurring with exactly in the same order as that prescribed in the episode. For example, in the event sequence

above, the events (A, 2), (B, 3) and (C, 8) constitute an occurrence of the episode (A → B → C).

An episode β is said to be a subepisode of episode α if all the event types in β appear in α as well, and if the partial order among the event types of β is the same as that for the corresponding event types in α. For example, (A → C) is a 2-node subepisode of the serial episode (A → B → C) while (B → A) is not a subepisode.

3.4 Content-based Queries

If D is a database of patterns and q is a query pattern, the content-based queries in this research include the following:

1. Subpattern queries, i.e. those patterns in D that contain q.
2. Superpattern queries, i.e. those patterns in D that are contained in q.
3. Equality queries, i.e. patterns in D that are equal to q.

The problem of selecting a suitable implementation of signature file for efficient content-based retrieval of patterns can then be formally defined as follows:

Given a database D of discovered temporal patterns, compare various implementations of signature-based indexes with respect to various criteria like number of false drops generated, query response time, index construction time, space requirements of the index. Further, based on these results and semantics of sequential patterns and episodes, suggest a suitable implementation of signature file as index for their databases.

4. SIGNATURE BASED INDEXES

4.1 Signature Files

A signature is a superimposed bit pattern generated from the values of the attributes. The target signature is obtained by the bitwise union of all the element signatures of the target set. When applying the technique of superimposed coding, each element of a given set is mapped via a coding function to a bit field of length F, called signature length, where exactly m < F bits are set. These bit fields are superimposed by a bitwise or operation to yield the final signature of the set [5].

If T and Q denote the target set and the query set respectively, commonly used set-valued queries are

a) Subset query ($Q \subseteq T$). The query set is the subset of target set.

b) Superset query ($Q \supseteq T$). The query set is the superset of target set.

c) Equality query ($Q \equiv T$). The query set is equal to the target set.

The important parameters here are the length of the signature denoted as 'F' and the weight of the signature denoted as 'm'.

Retrieval using signature files is based on the inexact filter. They provide a quick test, which discards many of the non-qualifying elements and qualifying elements become drops. This step is called the filtering step. The second step is false-drop resolution. In this step, each drop is retrieved and verified. The drops which do not satisfy the query condition are called the false-drops and those which satisfy the condition are called actual drops. False drops occur due to collision of element signatures and the superimposed coding method used to generate the signatures. The false drops affect the number of block accesses and hence the processing time.

4.2 Sequential Signature Files (SSF)

In a sequential signature file, a set of signature/reference pairs is sequentially stored. When a new signature arrives it is just appended at the end of the file. To retrieve a pattern using SSF, a full scan of the file is done looking for signatures which match with the query signature [21].

4.3 Bit-Slice Signature Files (BSSF)

The signatures in BSSF are stored in a column wise manner. If the length of the signatures is F, then all the signatures will be stored in F files, in each of which one bit per signature for all the signatures is stored. To retrieve using BSSF index, the signatures are checked slice-by-slice rather than each signature to find matching signatures. Only those bit-slices, for which there is a one in the query signature [21] were retrieved.

4.4 Extendible Signature Hashing (ESHF)

In extendible signature hashing for database of temporal patterns [19] signatures are used instead of hash keys. Let sigd be a prefix of signature sig consisting of first d bits.

The index is divided into two parts, the directory and the buckets [16]. A bucket contains tuples consisting of the signature of the data item and reference to the data item. The directory begins with the header that holds the value for the (global) depth d. The directory has 2d entries which are references to the buckets. When searching for a pattern, sigd of the query pattern is determined to find the reference to the bucket where the pattern is to be found. The entries in the directory are not necessarily distinct, so there may be more than one reference in the directory pointing to the same bucket. The local depth d' of a bucket specifies the length of prefix actually used in this bucket, i.e. only the first d' bits of the signatures of all entries in the bucket must be equal.

4.5 Signature Tree (STF)

In a signature tree [21], [22], each path is a signature identifier, as defined below, and is not a continuous piece of bits.

Let si be a signature of length m. si = si[1] si[2]…si[m] where each si[j] ∈ {0,1}, j = (1,…,m). si(j1,…,jh) denotes a sequence of pairs with regard to si: (j1, si[j1]), (j2, si[j2])…(jh, si[jh]), where $1 \le jk \le m$ for k ∈ {1,…,h}.

Signature Identifier: Let S = s1, s2,..., sn be a signature file. Consider si($1 \le i \le n$). If there exists a sequence: j1,...,jh such that for any $k \ne i$ ($1 \le k \le n$), if si(j1,..., jh) ≠ sk(j1,...,jh), then si(j1,..., jh) is an identifier of si with regard to S.

Signature Tree: A signature tree for the signature file S, where si ≠ sj for i ≠ j and |sk| = m for k =1,...,n, is a binary tree T such that

1. For each internal node of T, the left edge is always labeled with 0 and the right edge is labeled with 1.
2. T has n leaves labelled 1,2,...,n, used as pointers to n different positions of s1, s2,..., sn in S.
3. Each internal node v is associated with a number, denoted by sk(v), to tell which bit will be checked.

5. SIGNATURE FILES AS INDEXES FOR DATABASE OF PATTERNS

Since temporal patterns involve sets of states and time ordered relationship among these states, traditional indexes cannot be used. There are three reasons for using signatures to encode

temporal patterns. First, they are of fixed length and hence very convenient for index structures. Second, comparison operators on signatures can be implemented by efficient bit operations. Third, signatures are more space efficient than explicit pattern representation.

5.1 Preserving Ordering among States

To apply any signature-based indexing technique for indexing a database of temporal patterns, the ordering of states and relationships inherent to temporal patterns must be preserved. Thus using the techniques discussed in [13], [2] the temporal pattern is first converted into equivalent sets and then the signatures were generated from there. Table 1 below shows the equivalent sets and signatures generated for temporal patterns shown in Figure1.

TABLE I. 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

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

1. $p \supseteq q \rightarrow sig_p \wedge sig_q = sig_q$

2. $p \subseteq q \rightarrow sig_p \wedge sig_q = sig_p$

3. $p = q \rightarrow sig_p = sig_q$

Here, '$\wedge$' represents bit-wise AND operation. The signature file of temporal patterns in database D can then be created as follows: For each temporal pattern $p \in D$, its equivalent set E(p) is calculated, and then, its signature sigp is generated. This signature together with the temporal pattern identifier (pid) is inserted into the signature file [7, 15, 16].

5.2 Parameters of Investigation

This study aimed at investigating the suitability of signature files as indexes for database of temporal patterns, Since there are various implementations of signature files available, it was natural to be able to rank these implementations for various parameters like the type of queries it is used for, query retrieval time, index construction cost which includes time to build the index as well space requirements if any. The parameters chosen for this study are guided by the following points.

1. Different implementations of signature files used as indexes for a database of temporal patterns were to be investigated for their suitability.

2. The type of queries considered here is content-based queries.

3. The database of temporal patterns is generated by temporal data mining and does not have many deletions and modifications.

4. The insertions are bulk insertions, frequency of which depends on the domain chosen. Thus, when there is a bulk load of new insertions, the index would require to be extensively modified.

Guided by these, the following investigation parameters were decided:

a) Number of false drops generated during subpattern, superpattern and equal pattern queries.

b) Index construction time

c) Query response time i.e. the performance of the index

d) Index size

6. EXPERIMENTAL SETUP

Extensive experiments were conducted to explore various implementations of signatures files as indexes under the criteria chosen. The following sections give the details of these experiments and experimental parameters.

6.1 Hardware/software Platform

All programs were written in C++ Language. The experiments were conducted on a 2-GHz Intel Core 2 Duo PC with 1 GB bytes of RAM running Windows XP Professional.

6.2 Experimental Parameters

Table 2 shows the experimental parameters chosen for this comparative study. During the experiments, the signature size F was varied from 8 to 128, keeping the weight m of the element signatures fixed at 1. Number of states N was chosen to be 100 and Database size was varied from 10,000 to 50,000.

6.3 Generating Data

Database of temporal patterns was generated synthetically as below:

First the size of a pattern was generated using Poisson distribution with mean |T| = 5. The states were picked up randomly from set of states. Then the relationships among the states were picked randomly from the set of relationships. Initially, a database of 10000 patterns was generated this way. To build the index on this database of temporal patterns, each temporal pattern was converted into an equivalent set using state mapping and relationship mapping functions [2],[13].

TABLE II. EXPERIMENTAL PARAMETERS

Symbol	Meaning
F	Signature Size
m	Weight of signature
N	Number of States
|D|	Size of Temporal Patterns Database
|T|	Average size of Temporal Pattern
Q	Size of Query Pattern

6.4 Generating Queries

To ensure a hit for the pattern to be queried, query patterns were generated as follows:

For equality queries, a pattern from the database was picked up randomly. For subpattern queries, the temporal pattern of size 5 was randomly selected, and then the states from the selected pattern starting from the last state were individually removed, resulting in a set of five queries. A similar method was performed

for super pattern queries by selecting a temporal pattern of size 10 to generate a set of five queries.

7. RESULTS

7.1 Index Size

In this experiment, the size of the index for each implementation of signature file was compared on two parameters. First, the signature size was varied, keeping the database size fixed and change in index size was observed (see Figure 2). Also keeping the signature size fixed, database size was varied from 10,000 patterns to 50,000 and change in index size was noted (see Figure 3). It was observed that for SSF, as the signature size is varied, there is no change in the index size and it grows linearly with change in database size.

For BSSF, the index size increased exponentially with the increase in signature size but increased linearly with the increase in database size.In case of ESHF, with the increase in signature size as well as database size, the size of index increased linearly, in a definite pattern.

While index size for these three implementations was only small percentage of size of the database, this size was almost equal to the database size for STF. Thus the space overhead was found to be maximum case of STF amongst all the implementations.

7.2 Index Building Time

In this experiment, the index building time was compared with the increase in signature size as well as increase in database size. It was observed that, increasing the signature size did not significantly impact the index building time in any of the indexing techniques. The index building time increased proportionately with increase in database size across all indexing techniques (see Figure 4).

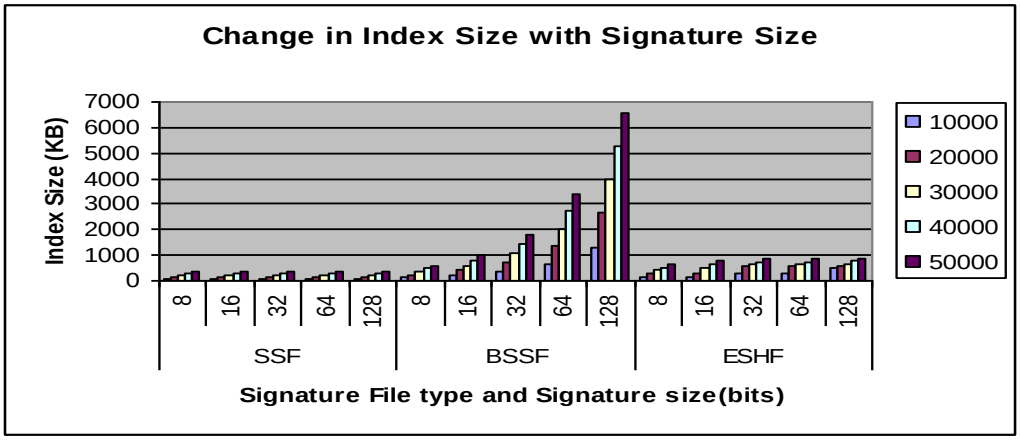

Figure 2. Index Size variation with signature size for different implementations of Signature files

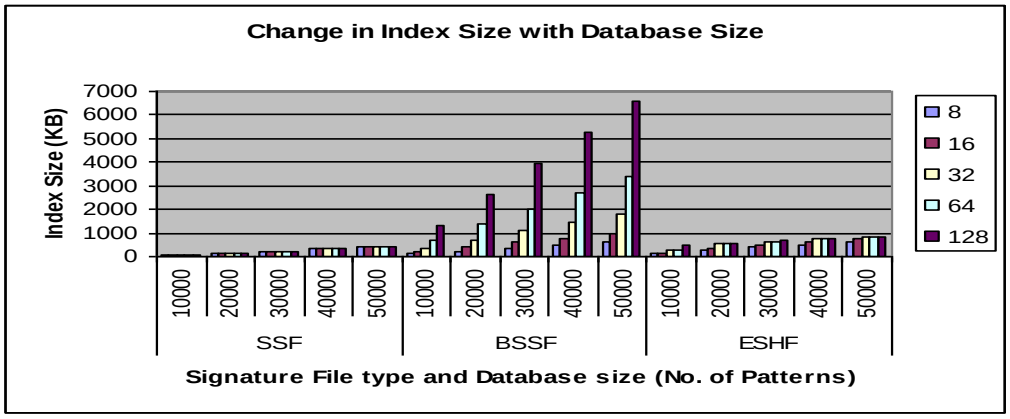

Figure 3. Index Size variation with database size for different implementations of Signature files

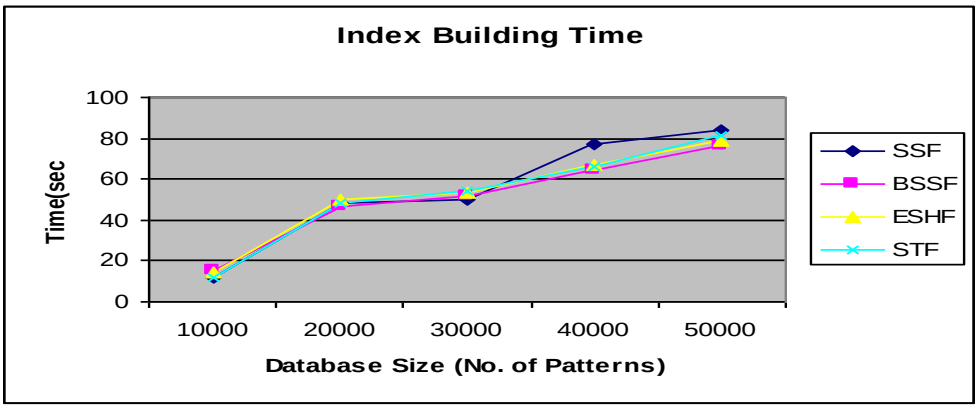

Figure 4. Index building time for different implementations of Signature file

7.3 Number of False Drops

7.3.1 Subpattern Query

In this experiment, the effect of change in signature size on the number of false drops in Sub pattern of queries was observed. The optimal parameters for each query type in the experimental environment were determined. Fixing 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 and the number of false drops was measured. Figure 5 shows the number of false drops for different signature sizes for different implementations of signature files. As can be seen, the number of false drops consistently decreased as the size of the signature increased, for each of the implementations. The number of false drops was also influenced by the size of the query pattern. For SubPattern Query, the larger the query pattern, the lower the number of false drops. Out of all the implementations, the ESHF performed the best for all signature sizes. For this implementation, the number of false drops became almost nil for higher signature size. SSF scored very low in this parameter, with number of false drops almost equal to the number of patterns for lower signature sizes and improving only marginally for higher signature sizes.

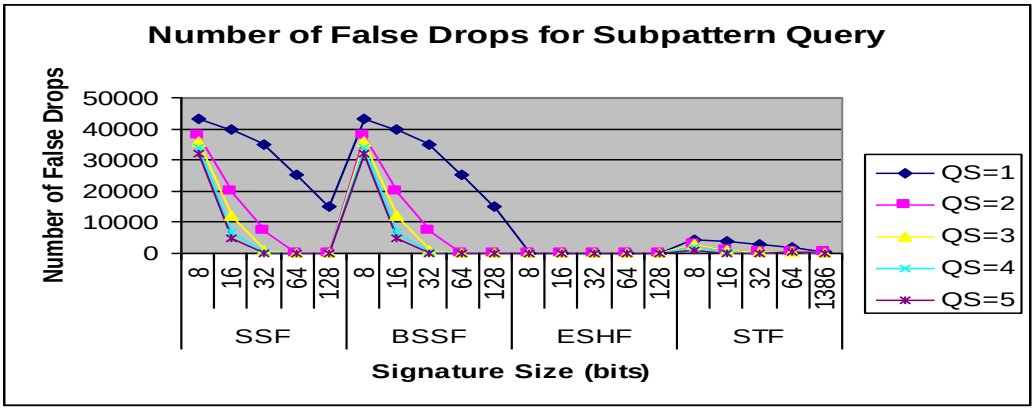

Figure 5. Number of false drops for different implementations of Signature files for subpattern query

7.3.2 Superpattern Query

In this experiment, the effect of change in signature size on the number of false drops in Super Pattern type of queries was observed. The optimal parameters for each query type in the experimental environment were determined. Fixing 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 and the number of false drops was measured. Figure 5 shows the number of false drops for different signature sizes for different implementations of signature files.

For SuperPattern Query, the larger the query pattern, the higher the number of false drops (see Figure 6). In this category also, the number of false drops for ESHF was the lowest as compared to SSF, BSSF and STF. It was also observed that for BSSF and STF, the superpattern search reduces to searching the whole index and tree respectively. This is due to the condition of superpattern queries listed in section 4.5. For STF, since it is build on signature identifiers, the whole tree needs to be traversed to reach to each leaf nodes to get the signatures and then check the condition for superpatterns. Nonetheless, it is better than SSF, because the search here reduces to binary search as compared to sequential search in SSF.

7.4 Query Retrieval Time

This experiment used the above data sets to compare the relative performance of SubPattern Query and Super Pattern Query for different implementations of signature files. Each method was

run on each of the queries used in the previous experiment. It was observed that the query processing times is proportional to the number of false drops from the previous experiments. ESHF index performs better than SSF, BSSF and STF. This is because for smaller size signature size, the number of false drops is so high that almost every pattern gets selected in the database and hence must be retrieved in the verification step. While in ESHF, as shown the previous experiment, the number of false drops reduced considerably and hence the processing time. Thus the retrieval times using SSF or BSSF are much more as compared to STF but ESHF performed the best.

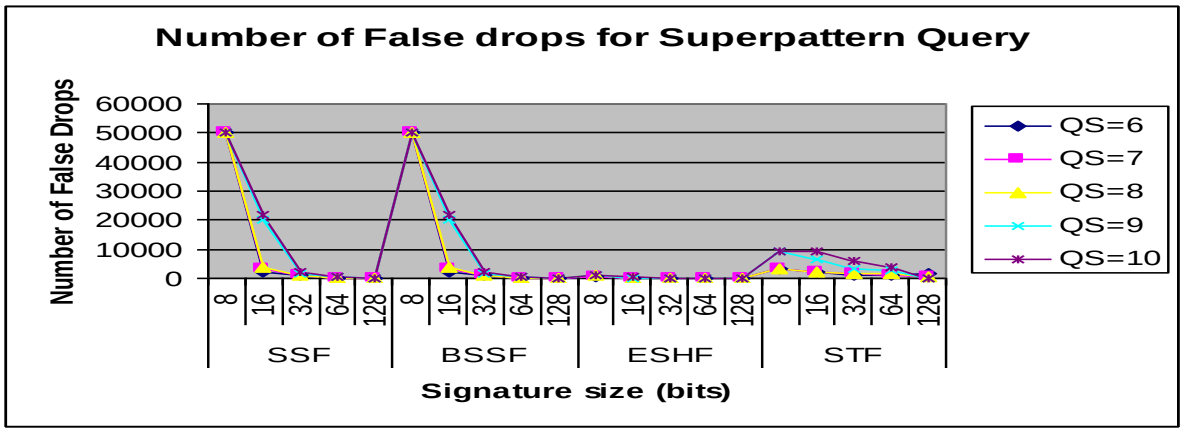

Figure 6. Number of false drops for different implementations of Signature files for superpattern query

8. SEMANTIC ANALYSIS OF OTHER PATTERNS

8.1 Sequential Patterns

As defined in section 3.2, sequential patterns are large and maximal sequences among a set of sequences. Sequential patterns are represented as sequences of itemsets i.e. each itemset is again set of items [4]. Thus a sequential pattern is essentially a large and maximal sequence of sets. Thus to index a database of discovered sequential patterns, we need to use those techniques which are used for indexing set-valued attributes. Signature based techniques are thus suitable to index a database of sequential patterns. Further, here also ordering of sets of items is important and we treat a sequential pattern as ordered sets of sets. Therefore, using the techniques discussed in [13], [2], we will first convert them into equivalent sets to preserve ordering and then apply signature-based indexing techniques. Next we look at the domains where such a data is available. One immediate application is the buying patterns of customers in a super market [4]. We generate sequential patterns from the data available for each customer in their subsequent visits to the supermarket. We analyse this to figure out how the sale deviate and what generally are the peak periods for such deviations. If we get such deviations during a particular period of the year or month over sufficient range of data, it becomes an interesting pattern and can be used to predict sales in the succeeding years. Therefore generally such patterns are used for predictive modelling. Now when we a have database of such sequential patterns, what would the type queries be to retrieve patterns for prediction. After studying many similar systems, we reached to the conclusion that most of the queries are super pattern queries or equality queries. Additionally, since these types of patterns are found in transaction oriented systems, the index should be efficiently scalable, with a proportional increase in storage requirements.

Keeping these semantics in mind and the results shown above, it is suggested that either SSF or ESHF are more suitable for such type of patterns. As can be seen, the storage requirements increase proportional to the increase in database size, the number of false drops is low and both are suitable for all types of content-based queries. For STF and BSSF, the storage requirements are very high and grow very fast with the increase in database size, they would not be suitable. Additionally the inherent nature of these two techniques is more suited for subpattern queries.

8.2 Episodes

As defined in section 3.3, an episode interpretation says that the events in g(V) have to occur in the order described by $\leq$. [14]. Since the episodes are sequence of events and number of events in different episodes may vary, we can use indexing techniques for set-valued attributes. Additionally, the ordering of events in an episode is important. Hence we can use the techniques discussed in [13], [2] to convert the episode into equivalent set, where ordering can be preserved. Thus a database of frequent episodes can be indexed using signature-based indexing techniques. Next, we analyse the domains where the episode discovery may be useful.

The episode discovery was originally applied to analysing alarm streams in a telecommunication network [14]. Another application is the mining of data from assembly lines in manufacturing plants [15]. The analysis of episodes here could help in understanding the hidden correlations between different fault conditions and hence improving the performance and throughputs of the assembly line. In manufacturing plants, frequent episode analysis may help in some process improvements by studying the frequent episodes in one line and comparing them to those in the other. Frequent episode discovery has also been applied to web-navigation logs. Thus we reach to the conclusion that framework of episode discovery falls under descriptive modelling where we are able to summarize a given situation. We now look at the majority of queries be in such situations. From a detailed analysis of applications above, it could concluded that majority of the queries would be subpattern queries or equality queries.

Keeping these semantics in mind, and results shown above it can be suggested that BSSF and STF, which are designed inherently for subpattern queries, would be suitable candidates. Since episode discovery is descriptive modelling, there may not be large changes to the underlying data, thus there is not much change in the database of episodes.

9. CONCLUSION

The paper presented a study of different implementations of signature files as index on a database of temporal patterns for efficient content-based retrieval of temporal patterns. The comparison was done on various parameters like index building time, space overhead, number of false drops and query performance. Each of these parameters was tested on different implementations, for different signature sizes and database sizes. ESHF performed best in all the parameters. However, space overhead becomes erratic as the database size grows or the signature size is increased. While STF shows a large space overhead, SSF was voted out for exceedingly large number of false drops. Both BSSF and STF are found to be not so suitable for superpattern queries. Additionally, STF has the problem of generating skewed trees. Based on these results and semantic analysis of sequential patterns and episodes, we also suggested which implementations would be suitable for databases of these patterns. These suggestions, however, are empirical and have to be verified experimentally. The comparisons were also done on synthetic datasets only. These experiments can also be performed for actual data as a future course of action.

10. REFERENCES

[1] A. Carlson, S. Estepp, M. Fowler. "Temporal Patterns"(AT&T Martin Fowler and Policy Management Systems Corporation, August 1998)

[2] 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.

[3] A. Tuzhilin and B. Liu, "Querying multiple sets of discovered rules," Proc. ACM SIGKDD '02, pp. 52-60, 2002.

[4] Agrawal R, Srikant R 1995 Mining sequential patterns. In *Proc. 11th Int. Conf. on Data Engineering*, (Washington, DC: IEEE Comput. Soc.)

[5] C. Faloutsos and S. Christodoulakis. Signature files: An access method for documents and its analytical performance evaluation. ACM Transactions on Office Informations Systems, 2(4):267– 288, October 1984.

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

[7] E. Winarko and J.F. Roddick, "A signature-based indexing method for efficient content-based retrieval of relative temporal patterns",IEEE Trans. on Knowledge and Data Engineering, VOL. 20, NO. 6, JUNE 2008.

[8] F. Ho¨ppner, "Learning Temporal Rules from State Sequences,"Proc. IJCAI Workshop Learning from Temporal and Spatial Data,pp. 25-31, 2001.

[9] Faloutsos C, Christodoulakis S (1984) Signature files: an access method for documents and its analytical performance evaluation. ACM Trans Office Inform Sys 2(4):267–288

[10] 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.

[11] 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.

[12] L. Geng and H.J. Hamilton, "Interestingness Measures for datamining: A survey," ACM Computing Surveys, vol. 38, no. 3, 2006.

[13] 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.

[14] Mannila H, Toivonen H, Verkamo A I 1997 Discovery of frequent episodes in event sequences. *DataMining Knowledge Discovery* 1: 259–289

[15] Laxman S, Sastry P S, Unnikrishnan K P 2004b Fast algorithms for frequent episode discovery in event sequences. In *Proc. 3rd Workshop on Mining Temporal and Sequential Data*, Seattle, WA.

[16] R. Fagin, J. Nievergelt, N. Pippenger, H.R. Strong, "Extendible hashing – a fast access method for dynamic files. ACM Trans Database Sys 4(3):315–344.

[17] Ritambhra Korpal, Arpita Gopal "Extendible Signature Hashing based Indexing for Efficient Content-based Retrieval of Temporal Patterns" IJCSA Issue 2010, ISSN 0974-0767;178-183

[18] Ritambhra Korpal, Arpita Gopal "Signature Trees as Index for Database of Temporal Patterns", in press.

[19] 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.

[20] 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.

[21] Y. Chen, "Building signature trees into OODBs," J. Information Science and Eng., vol. 20, no. 2, pp. 275-304, 2004.

[22] Y. Chen, Y. Chen, "On the Signature Tree Construction and Analysis, IEEE Trans. On Knowledge and Data Engineering, VOL. 18, NO. 9, SEPTEMBER 2006.

[23] 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.