Content-based Indexing of Sequential Data using Bitmaps for Efficient Retrieval

Dr. Ritambhra Korpal
Department of Computer Science
Savitribai Phule Pune University Pune, India
ritambhra.korpal@unipune.ac.in

Abstract — **While applying machine learning to unearth the underlying abstractions in the data can lead to overwhelmingly large number of detected patterns. Selecting the patterns as they are being generated, which may point to significant trends in the underlying data becomes an arduous task. In order to ensure that relevant patterns can be put to use, they can be stored in a persistent store and can be retrieved later for reference. With huge number of such patterns, the content-based retrieval can be very slow. In this paper, a novel method of indexing is presented for efficient retrieval of selected patterns. The new index structure, proposed in this paper, called Sequence Bitmap, does not require any preprocessing and retains the sequence of states while indexing, which is essential for indexing sequential patterns. Sequence bitmap performed very efficiently as compared to non-indexed content-based retrieval and significant improvements were observed as the database size increased. Variation of index creation time and index size with the increase in database size was also studied.**

Keywords – ***Indexes, Algorithms, Experimentation, Sequential Patterns, Temporal Patterns, Episodes, Sequence Bitmap, Signature based index***

I. INTRODUCTION

Sequential or ordered data sets form a class of datasets where ordering among the records is very important and is central to the data description/modeling. The data can be ordered on any attribute and need not necessarily be temporally ordered. These data sets could be text, gene sequences, protein sequences, lists of moves in a chess game etc. When applying machine learning techniques on such data sets, different patterns get uncovered based on the nature of domain of underlying data. Patterns referred to in this paper are taken from three different domains and their structure and semantics are different. Specifically, *Temporal patterns* include a set of states and temporal relationships among the states are generated from interval-based data. *Sequential patterns*, includes a set of ordered states generated from ordered data like sequence of chess moves or protein sequences. Finally, *Episode*, is a partially ordered set of event types. With the availability of sophisticated hardware and better algorithms, large number of associations/patterns can be discovered, which could be useful in many applications [2], [3]. However, not all generated patterns are of interest to the user [4], [5]. Thus identifying and analysing those that could provide an interesting insight into the underlying data becomes a difficult task. Therefore, these large number of generated patterns needs to be managed effectively so that critically vital patterns do not get overlooked. As the number of generated patterns /rules increases, the system should also provide flexible tools to allow selection of the generated patterns for reviewing. 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 persistent store and later query this store to retrieve the required patterns.

In this paper, the focus is on querying a database of previously discovered patterns from ordered data and improving the response time for content-based queries. Indexes are needed for large databases to make the retrieval more efficient. The indexes which can be built on such data items should be able to handle multiple value attributes as well as retain the ordering among the multiple values of the attribute. Since, it is crucial to not lose ordered nature of the data, traditional indexes cannot be used. Such index structures which can handle multi-valued attributes can be signature–based indexes or bitmaps. Both these index structures do not take into account ordering of elements in the set and hence cannot be directly applied here. The index structure, presented in this paper, is based on the concept of bitmaps and maintains the relative ordering of states in the pattern.

II. RELATED WORK

In our former work, the performance of Extendible Signature Hashing (ESHF) [8] and Signature Trees (STF) [9] as indexing techniques for efficient retrieval of content-based queries on temporal patterns was studied. A comparison of the performance of ESHF and STF against Sequential Signature Files (SSF) and Bit-Slice Signature Files (BSSF) implementations of signature files was carried out. It was observed that ESHF produced minimum number of false drops and hence were found to be most efficient. Another investigative study was conducted to compare these different implementations of signature files with respect to index construction time, space requirements for the index, query response time as well [10]. ESHF performed best in all of these parameters as well. However, space overhead for ESHF became erratic as the database size increased or the signature size increased. E. Winarko and J. F. Roddick [11] studied the application of signature files as an indexing technique for efficient retrieval of temporal patterns. They investigated the performance of SSF and BSSF as an indexing technique for efficient retrieval of content-based queries on temporal patterns.

C.-Y. Chan and Y. E. Ionnidis [12] presented a general framework to study the design space of bitmap indexes for selection queries and examined the disk-space and time characteristics that the various alternative index choices offer. K. Stockinger [13] analysed the behaviour of their bitmap index algorithm against various queries based on different data distributions. They implemented an improved version of one of the most cited bitmap compression algorithms called Byte Aligned Bitmap Compression and adapted it to their bitmap indexes. K. Wu, E. J. Otoo and A. Shoshani [14] presented a comparison of two new word-aligned schemes with some schemes for compressing bitmap indexes.

G.J Scott, M. N. Klaric, C. H. Davis, S. Chi-Ren [15], presented entropy-balanced bitmap (EBB) tree, which exploits the probabilistic nature of bit values in automatically derived shape classes, to efficiently and accurately perform content-based shape retrieval of objects from a large-scale satellite imagery database. T. I. Yamazaki, H. Sato, N. Takahashi, J. Takagi, M. Minami, [16] presented an indexing method using 'offset bitmaps', that can rapidly store slightly disordered time-series sensor data. The offset bitmaps provide pointers that can handle delayed data and so allow it to be managed efficiently. J. P. Yoon [17] proposed a bitmap-indexing scheme in which access control decisions can be sped up. Authorization policies of the form (subject, object, and action) are encoded as bitmaps in the same manner as XML document indexes are constructed. M. Alam, M. Y. Arafat, M. K. U. Iftekhar [18] proposed a new approach of encoded bitmap indexing (EBI) that makes it well defined for most of the selection queries by using association rule mining. Authors used up to n-items pattern selection of query predicate to select the most frequent pattern.

Even though the domains and techniques of applying bitmap indexes have varied with time, to the best of our knowledge, none of these takes into account the sequential nature of the underlying data. The Sequence bitmap, presented in this paper is first such work in this direction, to preserve the sequential nature of the underlying data.

III. PRELIMINARIES

A. *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]. Figure 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 sub-pattern of pattern α, if β can be obtained from α by removing some state intervals. E.g. for patterns in Fig. 1, p1 is a sub-pattern of p3 but not of p4.

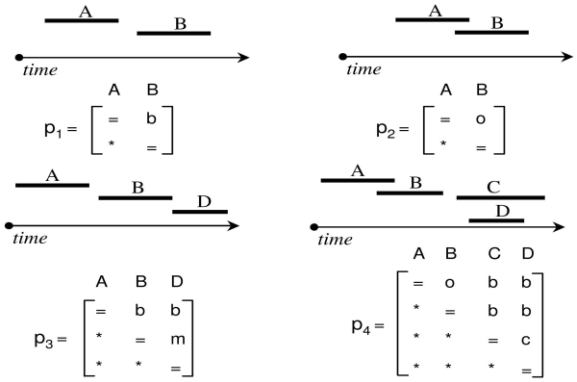


Fig. 1 Temporal patterns

B. *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.

C. *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 $\forall i, t_i \leq t_{i+1}$.

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 sub-episode 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 sub-episode of the serial episode (A → B → C) while (B → A) is not a sub-episode.

D. *Content-based Queries*

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

1. Sub-pattern queries, i.e. those patterns in D that contain q.
2. Sup-erpattern 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 content-based retrieval of ordered patterns can then be formally defined as follows:

Given a database D of discovered Temporal patterns or Sequential patterns or Episodes, describe a processing technique that allows the user to find the required patterns in D that satisfy the content-based queries efficiently.

IV. INDEXING TECHNIQUES FOR MULTIVALUED ATTRIBUTES

A. *Signature Files*

A signature is a superimposed bit pattern generated from the different values of the attribute. Retrieval using signature files is based on the inexact filter and is a two-step process. In the first step, called *filtering step*, a quick test is performed which discards many of the non-qualifying elements. The qualifying elements become *drops*.

The next 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. Thus, the main issue involved in using signature files is the proper control of false drops. Signature files can be implemented as sequential signature files (SSF), Bit Slice Signature Files (BSSF), Extendible Signature Hashing (ESHF), Signature Trees (STF) etc. Each of these implementations has its own merits and demerits when used as indexes.

B. *Bitmaps*

Bitmap indexes are very efficient data structures for querying read-only data [20]. The basic idea is to store one slice of bits per distinct attribute value of a record. So there will be as many bitmaps as the cardinality of the attribute to be indexed, in the record. Each bit of the slice is mapped to a record or a data object. The associated bit is set if the record's attribute value falls in that category. One of the main strengths of bitmaps is that complex logical selection operations can be performed very quickly by means of boolean operations such as AND, OR, or XOR.

If N is the number of data elements and C is the cardinality of the column on which the index is being built, then a basic bitmap index contains total C×N bits in the bitmaps for the given column. As the column cardinality increases, the basic bitmap index requires correspondingly more storage space. In the worst case where each value is distinct, C = N, the total number of bits is N^2.

C. *Sequence Bitmap*

The two different index structures, discussed in section A and B, however, do not take into account the ordering of the attribute values. For ordered patterns, ordering of states is important. Thus, both of these techniques cannot be applied directly. While signature-based indexing can be used for indexing these patterns after some preprocessing has been done to maintain the relative ordering of the state [21], [22]; bitmap indexes cannot be used as they show only presence or absence of a state. The Sequence bitmap proposed in this paper is based on basic bitmap but along with showing presence or absence of states, it also shows relative ordering of states. In this paper, a detailed discussion on implementing Sequence bitmaps, and content-based retrieval applied to temporal patterns in presented. This can be suitably modified when applying to Sequential patterns and Episodes. As defined in Section III, the temporal pattern has a sequence of states and temporal relationship among the states. In Sequence bitmap indexing,

the focus is on the relative ordering of the states in the pattern. When a query is fired, the ordering of the states is matched first from the index and these pattern ids are selected, called *drops*. This is followed by false drops verification where those patterns for which temporal relationship between the states does not match are rejected.

In this index, instead of keeping one bit to show presence or absence of a state, as many bits are kept as are required to show the relative ordering of states. E.g. if there are 4 possible positions, 4 bits are kept for each possible position. So, if a state can appear at 1st position in a pattern, 1st bit is set to 1 for that state in that pattern's bitmap. All position bits 0 indicate absence of that particular state in that pattern. Thus, a bitmap is formed for each state depending on the position at which that state occurs in each of the patterns in the database. Fig. 2 shows a sample database of 10 patterns defined over a set of five possible states K = {A, B, C, D, E}. Fig. 3 shows the Sequence map of this sample database. Thus, for five states and four possible positions of these states in a pattern, 5 × 4 = 20 bits for each pattern are needed. Therefore, if there are 10 patterns, 20 × 10 = 200 bits are needed. Generalizing this, if D is the number of patterns, N is the number of states possible and S is the number of positions available, D × N × S number of bits are required.

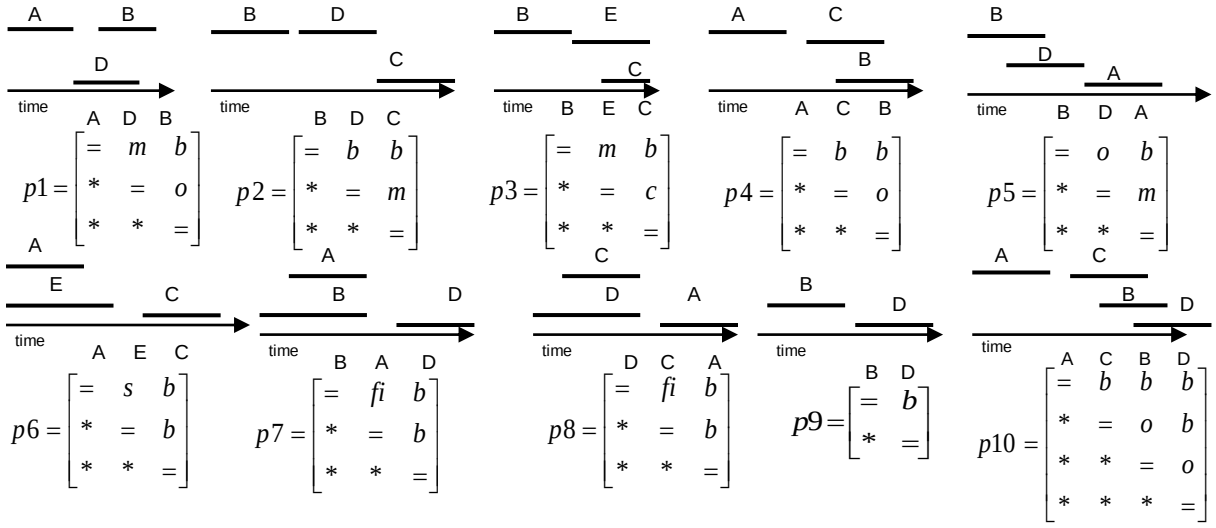


Fig. 2 Sample temporal patterns

	p1	p2	p3	p4	p5	p6	p7	p8	p9	p10
A	0001	0000	0000	0001	0100	0001	0010	0100	0000	0001
B	0100	0001	0001	0100	0001	0000	0001	0000	0001	0100
C	0000	0100	0100	0010	0000	0100	0000	0010	0000	0010
D	0010	0010	0000	0000	0010	0000	0100	0001	0010	1000
E	0000	0000	0010	0000	0000	0010	0000	0000	0000	0000

Fig. 3 Sequence Bitmap for patterns shown in Fig. 2

Important factor here is the number of position (S) considered. In the example, S is taken to be four. This does not, in anyway, restrict the pattern size to be four only. If the size of the pattern is more than S, only first S states are considered to build the index. While matching with the query pattern, only first S states of the query pattern are matched with the corresponding bitmaps of the target patterns. A smaller value of S will lead to a greater number of false drops while a larger value of S will have high space overhead. Hence, an important requirement, when using Sequence bitmaps as indexes, is a thorough understanding of the underlying data which includes knowledge about the approximate number of states possible as well as average size of the generated pattern, so that an optimum value of S can be decided. Nevertheless, this requirement should not deter the application of Sequence bitmaps as indexes, because this information is generally available to the users actively involved in mining activities.

For content-based queries, as against point queries or range queries this Sequence bitmap is very suitable. Retrieval using this bitmap requires as many passes as there are states in the query pattern or S, whichever is smaller. Matching the position of a state in the Sequence bitmap is done by creating a mask for that position. If q is the query pattern, and p is any pattern in the database D,

$$\text{Sequence bitmap(state, p)} \wedge (\text{mask}) = \text{mask} \quad \ldots(1)$$

i.e. the state is present in p and is at the position indicated by the mask.

When a sub-pattern query is fired, for the first state in the query pattern, all those patterns which have matching state in the first position, their pattern-ids are selected. In the subsequent iterations, those pattern-ids which are selected in the previous iteration are checked to match the next states and their position against those in the query pattern using (1). This continues till either the end of

pattern is reached or the available positions (S) are exhausted. Thus, the list of pattern-ids gets refined in each iteration and the final list will be evaluated for false drops.

For a super-pattern query, i.e. to retrieve all those patterns which are sub-patterns of the query pattern, the process is slightly different. All those patterns which match the first state in the query pattern (using (1)), their pattern-ids get selected for further evaluation. Let these patterns be p′. From these, those patterns are retained which satisfy (2) below, to ensure all sub-patterns of the query pattern are selected.

$$\text{Sequence bitmap(state, p}') \wedge (\text{mask}) = \text{mask or 0} \quad \ldots(2)$$

i.e. either the state is present at the position indicated by the mask or it is absent.

The next step is the false drops verification. For each pattern-id selected, the corresponding pattern is brought from the database. States after first S states, if any, and the relationships among all the states are matched with those in the query pattern. If there is a match, the pattern becomes an actual drop otherwise it is a false drop. Since major processing time involves evaluating the selected pattern-ids, the index is efficient if the number of false drops can be minimized.

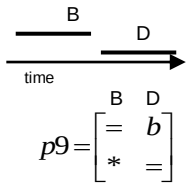


Fig. 4 Query pattern for sub-pattern query

The patterns selected in this pass are p2, p5 and p9. Since there are only two states in the query pattern, the process stops here and the final list of the patterns is p2, p5 and p9, which will be evaluated for relationships' match among the states.

	p1	p2	p3	p4	p5	p6	p7	p8	p9	p10
A	0001	0000	0000	0001	0100	0001	0010	0100	0000	0001
B	0100	0001	0001	0100	0001	0000	0001	0000	0001	0100
C	0000	0100	0100	0010	0000	0100	0000	0010	0000	0010
D	0010	0010	0000	0000	0010	0000	0100	0001	0010	1000
E	0000	0000	0010	0000	0000	0010	0000	0000	0000	0000

Fig. 5 State B present in first position

	p2	p3	p5	p7	p9
A	0000	0000	0100	0010	0000
B	0001	0001	0001	0001	0001
C	0100	0100	0000	0000	0000
D	0010	0000	0010	0100	0010
E	0000	0010	0000	0000	0000

Fig. 6 State D present in second position

Out of these, the number of false drops is 1 as p5 does not satisfy the relationship match criteria.

Let another query pattern on the above data be as in Fig. 7. Let the query type be super-pattern query. From the data it can be seen that this pattern is super-pattern of pattern id p4 and p10. The retrieval using Sequence bitmap is explained below.

The first state in the pattern is A. Select those pattern ids from the map which have A in the first position. The list of pattern ids selected is p1, p4, p6 and p10. (Fig. 8)

State at the second position is C. From the list of pattern ids selected in the first step, those are retained which have a 0 in the second position or have C at the second position (Fig. 9). So, from this list, patterns selected are p1, p4 and p10. p6 is rejected because the position of state C does not match with position of C in query pattern. p1 is retained as it shows absence of state C and hence could be potential sub-pattern of the query pattern. Next state in the query pattern is B.

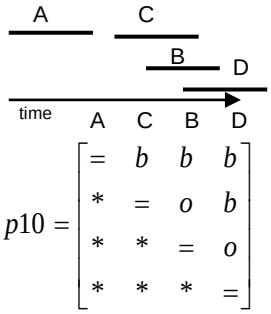


Fig. 7 Query pattern for super-pattern query

	p1	p2	p3	p4	p5	p6	p7	p8	p9	p10
A	0001	0000	0000	0001	0100	0001	0010	0100	0000	0001
B	0100	0001	0001	0100	0001	0000	0001	0000	0001	0100
C	0000	0100	0100	0010	0000	0100	0000	0010	0000	0010
D	0010	0010	0000	0000	0010	0000	0100	0001	0010	1000
E	0000	0000	0010	0000	0000	0010	0000	0000	0000	0000

Fig. 8 State A present in first position

	p1	p4	p6	p10
A	0001	0001	0001	0001
B	0100	0100	0000	0100
C	0000	0010	0100	0010
D	0010	0000	0000	1000
E	0000	0000	0010	0000

Fig. 9 State C present in second position

Following the same logic as above, from p1, p4 and p10, all the patterns are retained. p1 is retained as it shows B at the third position. Last state in the query pattern is D, p1 is rejected as it has D at second position and both p4 and p10 get retained. This becomes the final list for false drops verification. During false drops verification, it is found that both patterns indeed are sub-patterns of query pattern. Hence the number of false drops in this case is zero.

The Sequence bitmap offers the following advantages, as compared to signature based indexes

1. Most of the operations can be at bit level, which makes the processing faster
2. No pre processing on the patterns is required to maintain the ordering of the states

V. SEQUENCE BITMAP AS INDEX FOR DATABASE OF TEMPORAL PATTERNS

A. Creating Sequence Bitmap

Creating a Sequence bitmap for a database of temporal patterns is relatively easy. For each pattern in the database, the states of the pattern are read sequentially. The corresponding bit position is set to 1 in the Sequence bitmap for that pattern for each of the states present in the pattern. If there are more states present in the pattern than the available positions (S) in the bitmap, only first S states of the pattern are considered and their position is indicated in the Sequence bitmap. The remaining states will not be shown in the bitmap. Thus, Sequence bitmap acts as an imprecise filter when used for retrieval. For each query pattern, first S states of the query pattern will be matched against S available positions in the bitmap index created for the database. If these states and their positions match, the corresponding pattern id becomes a drop. These drops will then be verified in the second step called false-drop verification. The selected pattern id (drop) becomes a false drop if the remaining states or the relationship among the states of that pattern id do not match with the query pattern.

Algorithm 5.1.1 lists the pseudo algorithm for building the Sequence bitmap for a database of temporal patterns.

Algorithm 5.1.1. Constructing the Index

Input: Database D of temporal patterns, Available positions S, Number of States possible N
Output: Sequence bitmap file
Method:
1. for each pattern p with pattern-id pid in database D do

2. take first S states of the pattern pid
3. for each state state ∈ S at position j,
set jth bit to 1 in the bitmap (state, pid)
end for
end for
4. Save the map in a file
5. Return the file pointer

B. Answering Sub-pattern Queries

The requirement for sub-pattern queries is to retrieve all those patterns (p) for which the given query pattern (q) is the sub-pattern. First S states of q are taken and first state from there is matched against first state for each patterns p ∈ database (D) using (1). Those patterns are selected for which this state and its position match. Next state of the q is matched with these selected patterns. This process continues till the first S states of q and their positions are matched. The final list of pattern-ids becomes the list of drops. These drops are then verified for actual match.

Given a temporal pattern database D and a query pattern q, FindSubPattern() is used, listed in algorithm 5.2.1, for evaluating sub-patterns.

Algorithm 5.2.1. Pseudocode of FindSubPatterns ()
Input: Database D of temporal patterns, Query Pattern q, Available positions S, Sequence Bitmap for D
Output: Answerset, Falsedrops
Method:
1. select the first S states of q and form the list L
2. let the first state in the list L be state.
3. create the mask for position 1.
4. check the Sequence bitmap of state
for each pattern with id pid in the bitmap of state do
if (mask ∧ (bitmap (state, pid))) is equal to mask then
add pid to the list P
end if
end for
5. for each state u ∈ ordered list {L – state} do
let the position of state u be j.
create the mask for position j
for each pattern with id pid ∈ list P do
if (mask ∧ (bitmap (pid, u))) is equal to mask then
retain pid in the list P
else
P = P – {pid}
end if
end for
end for
6. evaluate list P for false drops
for each pattern with id pid ∈ list P do
retrieve p from D
if p ⊆ q then
add p into AnswerSet
else
increment Falsedrops
end if
end for
7. return Answerset and Falsedrops

C. Answering Super-pattern Queries

The requirement for the super-pattern queries is to retrieve all those patterns which are sub-patterns of the given query pattern (q). In this case, all patterns p ∈ database D, are selected for which the first state of q matches. From this list, those pattern-ids are progressively removed, for which the next states of q do not match. The next states of q may be absent in p, thus making it a potential sub-pattern of q. Given a temporal pattern database D and a query pattern q, FindSuperPattern() is used, listed in algorithm 5.3.1, for evaluating super pattern queries.

Algorithm 5.3.1 : Pseudocode of FindSuperPatterns()
Input: Database D of temporal patterns, Query Pattern q, Available positions S, Sequence Bitmap for D
Output: Answerset, Falsedrops
Method:
1. select the first S states of q and form the list L

2. let the first state in the list L be state.
3. create the mask for position 1.
4. check the Sequence bitmap of state
   for each pattern with id pid in the bitmap of state do
     if (mask ∧ (bitmap (state, pid))) is equal to mask then
       add pid to the list P
     end if
   end for
5. for each state u ∈ ordered list {L – state} do
   let the position of state u be j.
   create the mask for position j
   for each pattern with pid ∈ list P do
     if (mask ∧ (bitmap of (pid , u))) is equal to mask OR
       ((bitmap (pid, u) is equal to 0) then
        retain pid in the list P
       else
            P = P – {pid}
       end if
   end for
  end for
6. evaluate list P for false drops
  for each pattern with id pid ∈ list P do
     retrieve p from D
     if p ⊇ q then
       add p into AnswerSet
     else
       increment Falsedrops
     end if
   end for
7. return Answerset and Falsedrops

D. Answering Equality Queries

The processing for Equality query is similar to Sub pattern query and Algorithm 5.4.1 lists FindEqualPatterns().

Algorithm 5.4.1 : Pseudocode of FindEqualPatterns()
Input: Database D of temporal patterns, Query Pattern q, Available positions S, Sequence Bitmap for D
Output: Answerset, Falsedrops
Method:
1. select the first S states of q and form the list L
2. let the first state in the list L be state.
3. create the mask for position 1.
4. check the Sequence bitmap of state
   for each pattern id pid in the bitmap of state do
     if (mask ∧ (bitmap (state, pid))) is equal to mask then
       add pid to the list P
     end if
   end for
5. for each state u ∈ ordered list {L – state} do
   let the position of state u be j.
   create the mask for position j
   for each pattern with id pid ∈ list P do
       if (mask ∧ (bitmap (pid, u))) is equal to mask then
         retain pid in the list P
       else
         P = P – {pid}
       end if
   end for
  end for
6. Evaluate list P for false drops
  for each pattern with id pid ∈ list P do
     retrieve p from D
     if p ≡ q then
       add p into AnswerSet
     else
       increment Falsedrops

end if
end for
7. return Answerset and Falsedrops

VI. EXPERIMENTS

The performance of these algorithms was assessed by implementing Sequence Bitmap (TBM) as index on the database of temporal patterns. While conducting the experiments, all the experimental parameters were kept same as in [19], as shown in Table 1. The method followed to generate synthetic dataset and query pattern was also same as in [19]. Since the number of false drops is not affected by the hardware/software platform and only depends on the indexing technique, the main comparison criterion chosen was number of false drops. Nevertheless, the time taken to build the index and retrieval time using index and without index was also studied. 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.

A. Number of False Drops

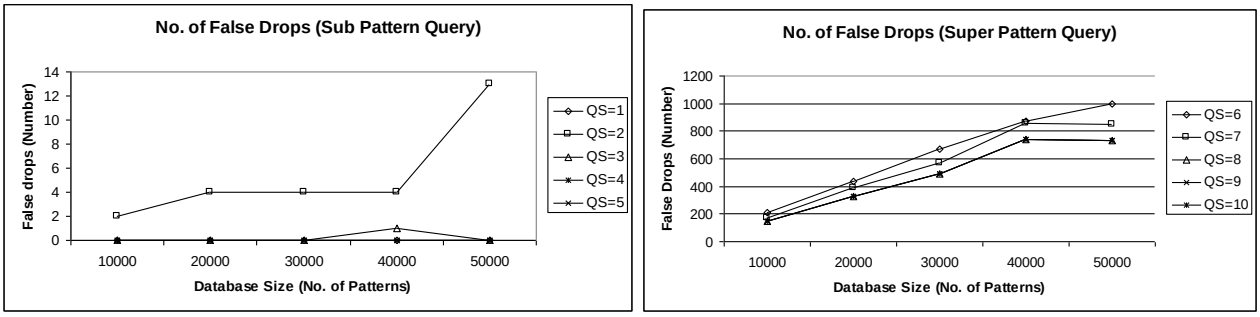
In this experiment, the number of false drops were observed in sub-pattern and super-pattern type of queries. Fixing the available positions S = 8, average size of temporal pattern |T| = 5, and the number of states N = 26, the size of database was varied from |D| = 10000 to |D| = 50000 and the number of false drops was measured. Fig. 10a and 10b show the number of false drops for Sub-Pattern Query and Super-Pattern Query respectively. The number of false drops consistently increased with the increase in database size. The number of false drops is also influenced by the size of the query pattern. For Sub-Pattern Query (Fig. 10a), the number of false drops is almost zero except for query size 2, and increases with the increase in the size of the database. Yet the number of false drops, even in this case also, is of the order of 0.02% of the database size which is negligible compared to size of the database. For Super-Pattern Query, smaller the query pattern, the higher the number of false drops (Fig. 10b), while the number of false drops increases with the increase in database size.

TABLE 1 : PARAMETERS

Symbol	Meaning
N	Number of States
|D|	Size of Temporal Patterns Database
|T|	Average size of Temporal Pattern
Q	Size of Query Pattern
S	Number of available positions

B. Effect of Database Size on Query Processing Time

This experiment used the above data set to compare the relative performance of Sub-Pattern query and Super-Pattern Query on non-indexed retrieval (SEQ) and Sequence Bitmap (TBM) indexed retrieval. In order to observe how the methods scale with respect to the database size, five data sets were generated and the size of database |D| was varied from 10,000 to 50,000. Fig. 11 shows the average of the processing time required by Sub-Pattern Query and Super-Pattern Query. It can be seen that the query processing time is proportional to the database size. As the database size grows, the query processing time grows, as is expected. However, it can be seen that the indexed retrieval performs better than non-indexed retrieval even when the database size is large, as indicated by the slope of the lines. Query processing time grows faster with the increase in database size in case of non-indexed retrieval as compared to the increase in query processing time for indexed retrieval.



(a) (b)

Fig. 10 Effect of database size of false drops using Bitmap. (a) Sub-Pattern Query. (b) Super-Pattern Query

C. Effect of Database Size on Index Creation Time

We wanted to know how the index creation time varies with the increase in the size of the database so as to rule out any erratic behaviour as the database size grows very large. So five datasets were generated and the size of database |D| was varied from 10,000 to 50,000. Fig. 12 shows how index creation time varied with the increase in database size. It can be seen that the index creation time varied linearly with the increase in the size of the database.

D. Effect of Database Size on Space Requirements for the Index

This experiment was conducted to know how the index space varies with the increase in the size of the database. So five datasets were generated and the size of database |D| was varied from 10,000 to 50,000. Fig. 13 shows how the space required for the index varied with the increase in database size. It can be seen that the space required for the index varied linearly with the increase in the size of the database.

VII. CONCLUSION AND FUTUTRE DIRECTIONS

The paper presented Sequence bitmap as an index on a database of temporal patterns for efficient content-based retrieval of temporal patterns. The Sequence bitmap can be similarly created for Sequential patterns and Episodes. The performance of this index was compared against the situation where no indexes are built. As presented, the use of this index considerably reduces the number of false-drops generated as compared to the ones reported in [11]. As a future course of action, the performance of this index will also be evaluated on database of Temporal patterns, Sequential patterns and Episodes generated from real data. The performance of this index built on database of Temporal patterns, Sequential patterns and Episodes will also be compared against each other. This is needed to assess the suitability of the techniques for different semantics and nature of underlying data.

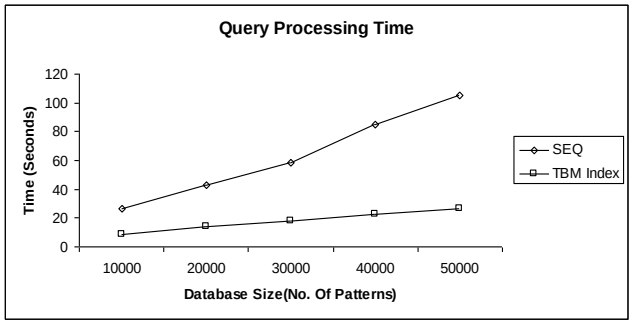


Fig. 11 Query Processing Time for non-indexed retrieval (SEQ) and Indexed Retrieval (TBM Index)

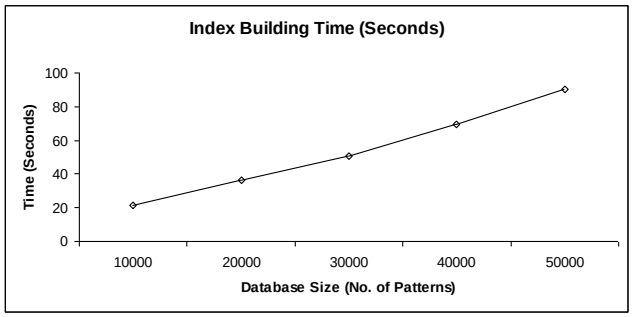


Fig. 12 Index Creation time vs. Database Size

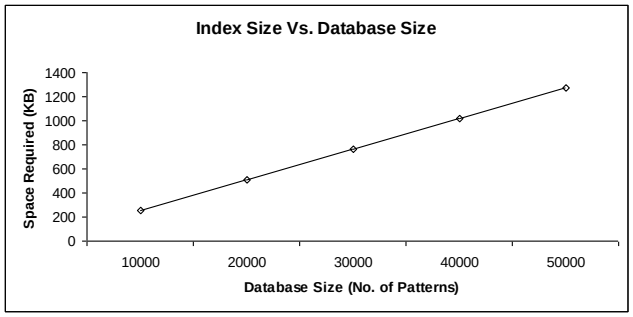


Fig. 13 Index size vs. Database Size

REFERENCES

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

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

[3] 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.

[4] 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.

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

[6] 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.

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

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

[9] R. Korpal, A. Gopal "Signature Trees as Index for Database of Temporal Patterns,"2010 International Conference on Computer and Computational Intelligence (ICCCI 2010), ISBN 978-1-4244-8950-3, V1 171-177

[10] R. Korpal, A. Gopal "A comparative study of signature based indexes for efficient retrieval of temporal patterns", ICWET '11 Proceedings of International Conference & Workshop on Emerging Trends in Technology, ISBN 978-1-4503-0449-8 ACM 2011

[11] 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.

[12] C.-Y. Chan and Y. E. Ionnidis, "Bitmap index design and evaluation," In SIGMOD 1998, pages 355-366. ACM press, 1998.

[13] K. Stockinger. "Bitmap indices for speeding up high-dimensional data analysis," In DEXA 2002. Springer-Verlag, 2002.

[14] K. Wu, E. J. Otoo and A. Shoshani, "A performance comparision of bitmap indexes," Proceedings of the 2001 ACM CIKM Internatiional Conference on Information and Knowledge Management, Atlanta, Georgia, USA, November 5-10, 2001, pages 559-561. ACM, 2001.

[15] G.J Scott, M. N. Klaric, C. H. Davis, S. Chi-Ren, "Entropy-Balanced Bitmap Tree for Shape-Based Object Retrieval From Large-Scale Satellite Imagery Databases IEEE Transactions on Geoscience and Remote Sensing, Volume: 49 , Issue: 5 2011, 1603 - 1616

[16] T. I. Yamazaki, H. Sato, N. Takahashi, J. Takagi, M. Minami, "Efficiently indexing with offset bitmaps for huge sets of slightly disordered sensor data in: Information and Telecommunication Technologies (APSITT), 2010 8th Asia-Pacific Symposium, 15-18 June 2010, pages 1 – 6, Kuching E-ISBN: 78-4-88552-244-4 Print ISBN: 978-1-4244-6413-5

[17] J. P. Yoon, "Presto authorization: a bitmap indexing scheme for high-speed access control to XML documents," IEEE transactions on Knowledge and Data Engineering, Volume: 18 , Issue: 7, 2006 , Page(s): 971 – 987

[18] M. Alam, M. Y. Arafat, M. K. U. Iftekhar, "A new approach of dynamic Encoded Bitmap Indexing Technique based on query history" International Conference on Electrical and Computer Engineering, 2008. ICECE 2008, Pages 974-979.

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

[20] P. O'Neil, D. Quass, "Improved Query Performance with Variant Indexes", Proceedings ACM SIGMOND International Conference on Management of Data, May 1997, Tucson, Arizona, USA

[21] 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.

[22] 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.