Summarizing Event Sequence Database into Compact Big Sequence

Mosab Hassaan
Faculty of Science, Benha University, Egypt

***Abstract*—Detecting the core structure of a database is one of the most objective of data mining. Many methods do so, in pattern set mining, by mining a small set of patterns that together summarize the dataset in efficient way. The better of these patterns, the more effective summarization of the database. Most of these methods are based on the Minimum Description Length principle. Here, we focus on the event sequence database. In this paper, rather than mining a small set of significant patterns, we propose a novel method to summarize the event sequence dataset by constructing compact big sequence namely, BigSeq. BigSeq conserves all characteristics of the original event sequences. It is constructed in efficient way via the longest common subsequence and the novel definition of the compatible event set. The experimental results show that BigSeq method outperforms the state-of-the-art methods such as Gokrimp with respect to compression ratio, total response time, and number of detected patterns.**

***Keywords*—Sequence data; compressing patterns mining; minimum description length**

I. Introduction

Detecting the key patterns from a database is one of the main objectives of data mining. There are many studies for mining all patterns that satisfy some constraints (such patterns may be frequent patterns as in PrefixSpan [11], CM-SPADE [19], PRISM [15], and [20], or closed patterns as in [24][7][14][2], or maximal patterns as in [3][23]). Rather than mining all patterns, existing methods mining a set of patterns that is significat for summarizing the dataset. There are many methods to define this significant patterns. One of these methods is the Minimum Description Length (MDL) principle [21][12][6][22] which has proven to be particularly the winner one.It is based on the insight that any regularity in the dataset can be used to compress the dataset. Note that, the more we can compress, the more regularity we have found. More details about MDL are described in next section.

For itemsets data, Krimp [13] is based on MDL principle. For sequence data, the authors of SeqKrimp [8][9], Gokrimp [8][9], and SQS [18] used MDL principle to compress the sequence data. More details about these algorithms are illustrated in the related work section (Section III).

In this paper, we focus on the event sequence data. Our objective is to search for a summary of the given event data sequences. The size of this summary must be very small compared to the size of the event sequence dataset. Also this summary must converse the all characteristics of the original event sequences. The existing methods mine a significant patterns that compress the dataset well. Some of these methods generate the sequential patterns as a first phase. Then the significant patterns are selected with respect to MDL as a second phase. Note that the significant patterns is only a small subset of the set of all the sequential patterns and the process of mining all sequential patterns is very expensive process. Therefore, the other existing methods devise some effective pruning methods to prune the ineffective parts of the search space that do not contain any significant pattern. Unfortunately, the process of the pruning the ineffective parts of the search space consumes more time if it not used efficient techniques.

Contribution. From above, all existing methods apply the mining process to search for the significant patterns. In contrast, our proposed method donot apply the mining process. Instead of, all event sequences in the dataset are merged into only one compact big sequence. In other words, our proposed method detects only one significant pattern which is the compact big sequence. Note that, the detected big sequence must be compact as much as possible. Therefore, we introduce an efficient method for constructing the big sequence to reduce the size of big sequence as much as possible. The construction method is based on the longest common subsequence and the novel definition of the compatible event set. Our compact big sequence converses the all characteristics of the original event sequences via preserving the order of events as in the dataset and also associating with each event in the big sequence a list of sequence ids that contains this event. To confine the larger size of the lists of sequence ids, we can represent them as sets of bit-vectors. Here, the consecutive zeros in the sets of bits-vector are compressed in efficient way.

Organization. This paper is organized as follows. Section II defines the preliminary concepts. Section III presents the related work. Section IV presents our proposed algorithm. Section VI reports the experimental results. Finally, Section VII concludes the paper.

II. Preliminary Concepts

Let $E = \{e_1, e_2, ..., e_m\}$ be a set of m distinct events. Event sequence $S =< u_1, u_2, ..., u_l >$ over E is ordered list such that $u_i \in E$. Event sequence $W = \{w_1, w_2, ..., w_h\}$ is subsequence of the event sequence S if there are h integers $(j_1, j_2, ..., j_h)$ such that $1 \leq j_1 < j_2 < ... < j_h \leq l$ and $w_1 = s_{j_1}$, $w_2 = s_{j_2}$,... ,$w_h = s_{j_h}$. Event sequence with length l is called an l-sequence. Event sequence database $\mathcal{D}$ = $\{S_1, S_2, ..., S_n\}$ is a set of event sequences where $|\mathcal{D}|$ = n. For example, consider Table I which contains an example of event sequence database $\mathcal{D}$ with $|\mathcal{D}|$ = 8. The sequence S_5 = $ABCB$ is subsequence of the sequence S_1 = $ABCBC$

($S_5 \sqsubseteq S_1$). Also we can said S_1 is supersequence of S_5.

TABLE I. EVENT SEQUENCE DATABASE, $\mathcal{D}$

Sid	Sequence
S_1	$ABCBC$
S_2	$ABAA$
S_3	$CABAC$
S_4	CAC
S_5	$ABCB$
S_6	$CBAC$
S_7	$BCAB$
S_8	$ACBBA$

Definition 2.1: **Longest Common Subsequence.** Given two event sequences X and Y, the longest common subsequence between X and Y denoted as $lcs(X, Y)$ is a longest sequence Z that is a subsequence of both X and Y.

Problem Definition: Given event sequence database $\mathcal{D}$, the objective is to find a summary, $\mathcal{S}$ of $\mathcal{D}$ such that $\mathcal{S}$ conserves all characteristics of $\mathcal{D}$ and the size of $\mathcal{S}$ is sharply less than the size of $\mathcal{D}$. ($|\mathcal{S}| << |\mathcal{D}|$).

III. RELATED WORK

In the beginning, we discuss the minimum description length in details as follows.

The Minimum Description Length

The minimum description length (MDL) principle [21][12][6][22] widely used in text compression. It used as a method for selecting a set of compressive patterns. If these patterns are used as a dictionary then we have {the potential} to maximally compress the dataset into a compact pattern encoding. In other words, these patterns represent the dataset in efficient way. Unfortunately, the process of selecting such patterns that based on MDL is NP-hard problem. Given a set of models $\mathcal{M}$, the MDL principle states that the winner model $M \in \mathcal{M}$ for the dataset $\mathcal{D}$ is the best model that provides the lossless compression. Formally, we optimize $Len(\mathcal{D}, M) = Len(M) + Len(\mathcal{D} \setminus M)$ where $Len(M)$ is the length in bits of the description of $\mathcal{M}$ and $Len(\mathcal{D} \setminus M)$ is the length in bits of the dataset when compressed with model M. MDL was applied to detect compressed frequent patterns from itemsets and sequences data. In next sections, we discuss the algorithms that based on MDL.

For itemsets data, there is algorithm called Krimp [13] that based on MDL principle. This algorithm is effective in solving the redundancy issue in the descriptive pattern mining. For sequence data, the authors of SeqKrimp [8][9] used MDL principle to compress the sequence data. This algorithm contains two steps. The first step generates the sequential patterns as candidates by using existing sequence mining method. The second step greedily checks the candidate set to find the useful patterns which together minimizes the description length. The SeqKrimp algorithm has two main disadvantages which are the process of generating the candidates is expensive and the patterns that do not belong to candidate set have no chance to be selected even if they have ability to minimizes the description length.

The authors of Gokrimp [8][9] mine a set of non-redundant sequential patterns that compress the sequence data using the MDL principle. GoKrimp do not generate candidates as in SeqKrimp. Instead of, it directly mines compressed useful patterns by greedily extending a pattern until no additional compression benefit added. To taming the hardness of the checks for additional compression benefit of an extension, Gokrimp proposed a dependency test which only selects related events for extending a given pattern.

As in GoKrimp, SQS [18] also directly mines the compressed patterns from the sequence dataset. The patterns are constructed iteratively. In each iteration, the pattern is selected if it achieves the largest MDL gain among the possible patterns. Note that, each iteration requires at least one scan of the sequence dataset.

IV. PROPOSED ALGORITHM

The method is based on the observation that the most event sequences in real dataset share the same subsequences. To avoid the overhead of duplicated computations, we propose big sequence method that merges all event sequences in the dataset into one big sequence abbreviated as BigSeq. The construction method of BigSeq is one of main operations in our algorithm. BigSeq must be compact and efficient. At the same time, it must conserve all characteristics of the original dataset.

To construct compact BigSeq, we should propose an efficient method to reduce the size of BigSeq as much as possible. Thus, we will propose a new efficient method to construct BigSeq. Next we discuss the steps of the construction method on the sequence dataset of Table I.

First, we select any sequence S in the sequence database, $\mathcal{D}$ (see Table I) as initial value of BigSeq. Suppose we selected the first sequence $S_1 \in \mathcal{D}$. Then the BigSeq is $ABCBC$. As we will see, some events will be inserted into the current BigSeq to generate the final BigSeq. Therefore, we set a temporary index for each event in the current BigSeq as follows. The temporary indices of events in the current BigSeq will be $i_1 i_2 i_3 i_4 i_5$ with $i_1 << i_2 << i_3 << i_4 << i_5$. We can assume the following $i_j = i_{j-1} + (j-1).\epsilon$ with $2 \leq j \leq 5$ and $\epsilon \geq 1$. For example, $i_3 = i_2 + 2\epsilon$ After generating the final BigSeq, we will set the actual value for each temporary index, i_j.

Second, for each remaining sequence S' in $\mathcal{D}$, compute the longest common subsequence between S' and BigSeq, namely $LCS(S', BigSeq)$. After that we store the positions in BigSeq for each event that belong to LCS and store also the remaining events in S', that do not belong to $LCS(S', BigSeq)$ (Note that these remaining events will be further inserted in BigSeq). For example, let S' be the sixth

sequence, $S' = S_6 = CBAC$. We have $LCS(S', BigSeq)$ = $LCS(CBAC, ABCBC)$ = CBC. The positions of the three events (C, B, and C) of $LCS(S', BigSeq)$ in BigSeq are i_3, i_4, and i_5, respectively. We store these positions. Also, we store the remaining event, A, in S' that does not belong to $LCS(S', BigSeq)$. This remaining event, A, will be further inserted in BigSeq. The remaining events of each remaining sequence must be inserted in the correct position in the BigSeq. Therefore, we will associate with each remaining event e_r a range of positions in BigSeq. We expect that e_r will fall within this range in BigSeq. We call this range an **E**xpected **R**ange of **P**ositions for event e_r, namely $ERP(e_r)$. Recall let $S' = S_6 = CBAC$ then we have only one remaining event A. The position of the event A in S' falls between the positions of two events B and C. Note that these two events (B and C) belong to $LCS(S', BigSeq)$ and their positions in BigSeq are i_4 and i_5. Therefore, we have $ERP(A) =]i_4, i_5[$. As a consequence, we should insert the remaining event A in BigSeq at a new position between i_4 and i_5. Table II shows the expected range of positions for each remaining event e_r, $ERP(e_r)$.

Finally, indeed, we do not insert each remaining event in BigSeq instead of we cluster the remaining events into compatible event sets. After that we insert only one **rep**resntative **e**vent, e_{rep}, for each compatible event set into BigSeq at a specific position p. This position p must belong to the expected range of positions of every event in the compatible event set of e_{rep}. See next definition of compatible event set and see next example.

Definition 4.1: **Compatiable Event Set.**
The event set is called compatible event set if the events in this set satisfy the next three conditions:

1) They have the same label;
2) They donot belong to the same event sequence;
3) The insertion of their expected range of positions in BigSeq is not empty.

Example 4.1: Given the event sequence database in Table I. Table II reports the initial value of BigSeq (S_1 [The first row]), the remaining sequences (S_2, S_3, S_4, S_5, S_6, S_7, S_8 [The first column]), the events of each remaining sequence S' that belong to $LCS(S', BigSeq)$ [The second column], and $ERP(e_r)$ for any remaining event, e_r ($e_r \notin LCS(S', BigSeq)$) [The third column].

Note that the underlined events in the first and second columns belong to $LCS(S', BigSeq)$ and the parameter $\delta \geq 1$. To distinguish among the remaining events (e_r in the third column of Table II) that have the same label, we assign superscripts for these events as follows. A^{km} means the m-th remaining event in the sequence k.

Now we will determine the compatible sets of remaining events. The remaining event can be belonged to more than one compatible event set. In this case, we add this remaining event to only one compatible event set. From the definition of compatible event set, If two or more different remaining events belong to the same event sequence then we must add them to different sets of compatible events. For example, since the two different remaining events, A^{21} and A^{22} belong to the same event sequence (the second event sequence, S_2), they must be added to two different sets of comaptiable events. Based on the definition of compatible event set and $ERP(e_r)$ in Table II, we have three **co**mpatible sets of **r**emaining **e**vents, $core = \{core_1, core_2, core_3\}$, where $core_1 = \{A^{21}, A^{31}, A^{41}, A^{71}\}$, $core_2 = \{A^{22}, A^{32}, A^{61}, A^{82}\}$, and $core_3 = \{C^{81}\}$.

Next we will discuss the computations of these three compatible sets of remaining events in details and how we conserve all characteristics of the original event sequence in the final BigSeq with respect to the event sequence database and $ERP(e_r)$ in Tables I and II, respectively.

The first compatible set of remaining events is $core_1 = \{A^{21}, A^{31}, A^{41}, A^{71}\}$. Note that all events in $core_1$ have the same label, A (Condition 1 in Definition 4.1) and they do not belong to the same sequence but they belong to sequences S_2, S_3, S_4, and S_7 respectively (Condition 2 in Definition 4.1). We have $ERP(A^{21}) = [p, p + \delta[$, where $p > i_2$. Recall, because $i_3 > i_2$, we can set $p = i_3$. Now $ERP(A^{21}) = [i_3, i_3 + \delta[$. Recall, $\delta \geq 1$ then we can set δ = 2. In other words, $ERP(A^{21}) = [i_3, i_3 + 2[$. The other expected range of positions are $ERP(A^{31}) =]i_3, i_4[$, $ERP(A^{41}) =]i_3, i_5[$, and $ERP(A^{71}) =]i_3, i_4[$. As the result, we have $ERP(A^{21}) \cap ERP(A^{31}) \cap ERP(A^{41}) \cap ERP(A^{71}) \neq \phi$ (Condition 3 in Definition 4.1). Thus, we insert in BigSeq at position $i_3 + 1$ only one event with label A (e_{rep1}) as representative for $core_1$. Note that we select the position $i_3 + 1 \neq i_4$, since $i_3 + 1 \in ERP(A^{21}), ERP(A^{31}), ERP(A^{41}), and\ ERP(A^{71})$.

The second compatible set of remaining events is $core_2 = \{A^{22}, A^{32}, A^{61}, A^{82}\}$. Note that $core_2$ satisfy coditions 1 and 2 in Definition 4.1 since all events in $core_2$ have the same label, A. the events in $core_2$ do not belong to the same sequence but they belong to sequences S_2, S_3, S_6, and S_8 respectively (Condition 2 in Definition 4.1). We have $ERP(A^{22}) = [p + \delta, \infty[= [i_3 + 2, \infty[$. The other expected range of positions are $ERP(A^{32}) =]i_4, i_5[$, $ERP(A^{61}) =]i_4, i_5[$, and $ERP(A^{82}) = [i_4, \infty[$. As the result, we have $ERP(A^{22}) \cap ERP(A^{32}) \cap ERP(A^{61}) \cap ERP(A^{82}) \neq \phi$ (Condition 3 in Definition 4.1). We insert in BigSeq at position $i_4 + 1$ only one event with label A (e_{rep2}) as representative for $core_2$. Note that we select the position $i_4 + 1 \neq i_5$, since $i_4 + 1 \in ERP(A^{22}), ERP(A^{32}), ERP(A^{61}), and\ ERP(A^{82})$.

Finally, the third compatible set of remaining events is $core_3 = \{C^{81}\}$ with $ERP(C^{81}) =]i_1, i_2[$. Since $core_3$ has only one event then it is compatible set that satisfy the three conditions in Definition 4.1. Thus, we insert the event C in BigSeq as representative for $core_3$ [e_{rep3}] at position between i_1 and i_2. In other words, we can insert C in BigSeq at position $i_1 + 1 \neq i_2$ such that $i_1 + 1 \in]i_1, i_2[$.

TABLE II. EXPECTED RANGE OF POSITIONS FOR EACH REMAINING EVENT e_r, $ERP(e_r)$

Initial Value of BigSeq = $S_1 = ABCBC$ **(the first sequence in** $\mathcal{D}$**) and its indices are** $i_1 i_2 i_3 i_4 i_5$		
S'**: Remaining Seq**	$e \in LCS(S', BigSeq)$ **with pos. in BigSeq**	$ERP(e_r)$
S_2: $\underline{AB}AA$	$\underline{A}$ and $\underline{B}$ with pos. i_1 and i_2	$ERP(A^{21}) = [p, p+\delta[$, $p > i_2$ - $ERP(A^{22}) = [p+\delta, \infty[$
S_3: $\underline{C}A\underline{B}A\underline{C}$	$\underline{C}$, $\underline{B}$, and $\underline{C}$ with pos. i_3, i_4, and i_5	$ERP(A^{31}) =]i_3, i_4[$ and $ERP(A^{32}) =]i_4, i_5[$
S_4: $\underline{C}A\underline{C}$	$\underline{C}$ and $\underline{C}$ with pos. i_3 and i_5	$ERP(A^{41}) =]i_3, i_5[$
S_5: $\underline{ABCB}$	$\underline{A}$, $\underline{B}$, $\underline{C}$, and $\underline{B}$ with pos. i_1, i_2, i_3, and i_4	NULL
S_6: $\underline{CB}A\underline{C}$	$\underline{C}$, $\underline{B}$, and $\underline{C}$ with pos. i_3, i_4, and i_5	$ERP(A^{61}) =]i_4, i_5[$
S_7: $\underline{BCAB}$	$\underline{B}$, $\underline{C}$, and $\underline{B}$ with pos. i_2, i_3, and i_4	$ERP(A^{71}) =]i_3, i_4[$
S_8: $\underline{A}C\underline{BB}A$	$\underline{A}$, $\underline{B}$, and $\underline{B}$ with pos. i_1, i_2, and i_4	$ERP(C^{81}) =]i_1, i_2[$ and $ERP(A^{82}) =]i_4, \infty[$

TABLE III. BIGSEQ CONSTRUCTION

Temp. Pos.	i_1	i_2	i_3	i_4	i_5
BigSeq Events	A	B	C	B	C

(a) Initial BigSeq

Temp. Pos.	i_1	i_1+1	i_2	i_3	i_3+1	i_4	i_4+1	i_5
Actual Pos.	1	**2**	3	4	**5**	6	**7**	8
BigSeq Events	A	**C**	B	C	**A**	B	**A**	C

(b) Insertion of the Three Representatives of the Three Compatiable Sets in BigSeq

The initial BigSeq with its indices and the final BigSeq with its indices are reported at Table III(a) and Table III(b), respectively. The final BigSeq is $ACBCABAC$.Note that we insert the three representative C, A, and A in BigSeq at position $i_1 + 1$ (between i_1 and i_2), $i_3 + 1$ (between i_3 and i_4), and $i_4 + 1$ (between i_4 and i_5). Here the size of the final BigSeq is 8 after inserting the three representatives. Therefore, the actual indices of the events in the final BigSeq will be from 1 to 8 (i.e. 1, 2, 3, 4, 5, 6, 7, and 8). See the next definition for the size of the final BigSeq.

Definition 4.2: **The Final BigSeq Size.**

The Final BigSeq Size is $|final\_BigSeq|$ = $|initial\_BigSeq|$ + $|core|$, where $|core|$ is the count of compatible sets of remaining events, where $initial\_BigSeq$ is the initial value of BigSeq.

For example, with respect to the event sequence database and the data in Tables I and II respectively, we have the following $|final\_BigSeq|$ = $|initial\_BigSeq|$ + $|core|$ = $|S_1|$ + $|core|$ = 5 + 3 = 8.

From definition 4.2, to reduce the final BigSeq Size, we should reduce the count of compatible sets of the remaining events as much as possible.

To conserve all characteristics of the original event sequence in the final BigSeq, we should associate with each event e in BigSeq a list of sequence ids that contains the event e, namely $e.id\_list$ as follows. First, since we select the first sequence as the initial value for BigSeq, we will

TABLE IV. STEPS OF BIGSEQ CONSTRUCTION WITH SEQUENCE ID LIST

Pos.	i_1	i_2	i_3	i_4	i_5
BigSeq Events	A	B	C	B	C
Seq. Id List	1	1	1	1	1

(a) Initial BigSeq with Id List of the First Sequence

Pos.	i_1	i_2	i_3	i_4	i_5
BigSeq Events	A	B	C	B	C
Seq. Id List	1	1	1	1	1
	2	2	3	3	3
	5	5	4	5	4
	8	7	5	6	6
		8	6	7	
			7	8	

(b) Addition of Id List for each Event $e \in GCD(S', BigSeq)$

Temp. Pos.	i_1	$\mathbf{i_1+1}$	i_2	i_3	$\mathbf{i_3+1}$	i_4	$\mathbf{i_4+1}$	i_5
Actual Pos.	1	**2**	3	4	**5**	6	**7**	8
BigSeq Events	A	**C**	B	C	**A**	B	**A**	C
Seq. Id List	1	8	1	1	2	1	2	1
	2		2	3	3	3	3	3
	5		5	4	4	5	6	4
	8		7	5	7	6	8	6
			8	6		7		
				7		8		

(c) Insertion of the Three Representatives for the Three Compatible Sets in BigSeq with their Id List

add 1 (the id of the first sequence) to $e.id\_list$ for each event $e \in initial\_BigSeq = S_1$ [see Table IV (a)]. Second, suppose the case that the event $e \in LCS(S', BigSeq)$, where S' is a remaining sequence (i.e. $e \in S'$ and $e \in BigSeq$). In this case, we add the id of S' to $e.id\_list$ for each event $e \in BigSeq$ [see Table IV (b)]. Finally, we have three representative events for the three compatible sets of remaining events. As we mentioned before, we inserted the three representatives, A, A, and C in BigSeq at positions i_3+1, $i_4 + 1$, and $i_1 + 1$ respectively to generate the final BigSeq. For each representative event, e_{rep}, for the compatible set of remaining events, $core_k$, we add to $e_{rep}.id\_list$ the id of the event sequence that contains the remaining event e_r, for every $e_r \in core_k$ with k = 1, 2, and, 3 [see Table IV (c)].

Next algorithm outlines the BigSeq construction with sequence Id List.

TABLE V. BIGSEQ WITH BIT-VECTORS

Pos.	1	2	3	4	5	6	7	8
$e \in BigSeq$	A	C	B	C	A	B	A	C
$B(e) = \{B_1\}$	{10010011}	{10000000}	{11010011}	{01111101}	{01001110}	{11110101}	{10100110}	{00101101}

Algorithm: BigSeq Construction with Sequence Id List

Input: Event sequence database, $\mathcal{D}$
Output: $BigSeq$ with $e.id\_list$ for each event $e \in BigSeq$.

1. Select an event sequence $S \in \mathcal{D}$ as $BigSeq$
   // Initial value of BigSeq = S
2. Add the id of S to $e.id\_list$ for each $e \in BigSeq$
3. $\mathcal{D} = \mathcal{D} - S$
4. $ERP = \{\}$
   //the set of expected range of positions
5. **for** each event sequence $S' \in \mathcal{D}$ **do**
6. $lcs = LCS(S', BigSeq)$
7. **for** each event $e \in lcs$ **do** // $e \in S'$ and $e \in BigSeq$
8. Add the id of the sequence S' that contains the event e to $e.id\_list$ in $BigSeq$
9. **end for**
10. **for** each event $e_r \in S'$ and $e_r \notin lcs$ **do**
    //e_r is remaining event
11. Compute $ERP(e_r)$
12. $ERP = ERP \cup ERP(e_r)$
13. **end for**
14. **end for**
15. Find the compatible sets of remaining events, $core$, based on Definition 4.1 and ERP
16. **for** each compatible set $core_k \in core$
17. Insert the representative event, e_{rep}, for $core_k$ into $BigSeq$ at position $p \in ERP(e')\ \forall\ e' \in core_k$
18. Add to $e_{rep}.id\_list$ the id of the event sequence that contains the remaining event $e_r\ \forall\ e_r \in core_k$
19. **end for**
20. **return** $BigSeq$ // The final BigSeq with $e.id\_list$ for each event $e \in BigSeq$

V. COMPRESSING EVENTS SEQUENCES DATABASE USING $BigSeq$ METHOD

The objective of this paper is to compress the event sequence database in efficient way such that we conserve all characteristics of the original database. In other words, we will compress the event sequence database into compact BigSeq with sequence id_lists. But when the size of event sequence database is large, the id_list size of each event in the corresponding BigSeq will be large. To confine the larger size of these id_lists, we can represent $e.id\_list$ of each event $e \in BigSeq$ as a set of bit-vectors, $B(e) = \{B_1, B_2, ..., B_m\}$, where each B_i is 8 length bit-vector (i.e. each B_i occupy 1 byte in memory) and suppose that the maximum $id \in e.id\_list$ is n then $m = |B(e)| = n/8$. Each position in each B_i corresponds to event sequence $S_{id} \in \mathcal{D}$ where $id \in [8 \times (i-1) + 1, 8 \times i]$. The bit at position j in B_i represents the presence or absence of the event $e \in BigSeq$ in the event sequence $S_{8\times(i-1)+j}$. See next example.

Example 5.1: The first event in BigSeq (in Table V) is $e_1 = A$ with the set of bit-vectors $B(e_1) = \{B_1\} = \{10010011\}$. Note that $B(e_1)$ contains only one bit-vector, B_1, since the maximum id in $e_1.id\_list$ = 8 (i.e. $n = 8$), thus $m = n/8 = 1 = |B(e)|$. The bits in B_1 represent the presence or absence of the event e_1 in the event sequences that have $id \in [1, 8]$. The bit at position 1 in B_1 is one, this means that $e_1 \in S_1 = S_{8\times(1-1)+1}$, etc. Given the final BigSeq with id_lists in Table IV(c), its corresponding BigSeq with bit-vectors is reported in Table V.

A. Compression Benefit

Suppose each event e occupy 1 byte in memory, then the size (in terms of bytes) of the original event sequence database, $\mathcal{D}$ (in Table I) is 34 bytes ($\mathcal{D}$ contains 34 events). Recall each B_i occupy also 1 byte in memory, therefore the size (in terms of bytes) of the BigSeq with bit-vectors (in Table V) is $|BigSeq|$ + $|B(e)|$ = 8 + 8 = 16 bytes. We can use the compression ratio to measure how well the data is compressed. The compression ratio calculated by dividing the data size before compression with the size after compression. In the above example, the compression ratio is $34/16$ = 2.125. In other words, the space saving (%) is (1 - (compressed size / uncompressed size)) $\times$ 100 = $1 - (16/34) \times 100$ = 52.9%. Here, we have an optimization that based on the observation that there are many consecutive zeros in each row of the bit-vectors. This is clearly grossly inefficient. Therefore, we can compress these consecutive zeros in efficient way as follows. Given array of bits (0 and 1), Bit_Arr, and paramter n. The output is the same as the input except for consecutive zeros. Note that, may be there are many sets of consecutive zeros in Bit_Arr. For each set of consecutive zeros, CZ, we do the following. If $|CZ| \leq n + 2$, we do nothing. Otherwise, we compress CZ into compressed CZ with size $n+2$ bits. The first and the second bits in the compressed CZ are 0 (indicator for compressing CZ) and 1 (indicator for doing the compression) respectively. The other n bits in the compressed CZ indicate how many times of zeros were repeated consecutively.

In the experimental results section, we will show the better compression ratio of BigSeq method against the state-of-art algorithm, GoKrimp, on many real datasets.

VI. EXPERIMENTAL EVALUATION

This section reportes the results of experiments on many real dataset. We compare the performance of Our proposed method, namely BigSeq with GoKrimp algorithm [8] [9]. Here, we exclude the two algorithms SeqKrimp and SQS from this experiment since GoKrimp algorithm outperforms them by one to two orders of magnitude. BigSeq is implemented in standard C++ with STL library support and compiled with GNU GCC. Experiments were run on laptop with Intel i3 2.4 GHz and 8G memory running Linux.

A. Datasets

Experimental evaluation are performed on a group of real datasets as follows. We used five real datasets namely, msnbc [10], Gene [16], TCAS [17] [4], Activity [1], and JBoss [5]. The corresponding information of these real datasets is summarized in Table VI, where $|\mathcal{D}|$ represents the number of sequences, $|E|$ is the number of the events, min_L, max_L and avg_L denote the minimum length, maximum length and average length of the sequences respectively.

TABLE VI. SUMMARY STATISTICS OF THE REAL DATASETS USED IN THE EXPERIMENTS

Dataset	$\|\mathcal{D}\|$	$\|E\|$	min_L	max_L	avg_L
msnbc	31790	18	9	100	13.33
Gene	2942	5	41	216	86.53
TCAS	1578	75	8	70	36
Activity	35	10	12	43	21.14
JBoss	28	64	51	125	91

B. Effect of Optimization

In this experiment, we show the effect of optimization of compressing the consecutive zeros, namely Opt, with respect to the compression ratio. Table VII reports the compression ratio of BigSeq with and without Opt on the three datasets (Gene, TCAS, and JBoss). From this table, BigSeq with Opt has the better compression ratio in all datasets. Note that the larger compression ratio is the better compression we have.

TABLE VII. EFFECT OF OPTIMIZATION WITH RESPECT TO COMPRESSION RATIO

Dataset	BigSeq with Opt	BigSeq without Opt
Gene	2.428	1.348
TCAS	3.384	1.585
JBoss	3.303	2.700

C. Performance of BigSeq against GoKrimp

From the previous experiment, BigSeq with Opt has the best performance with respect to compression ratio. Therefore, in this experiment, we will use BigSeq with Opt and we will call it as BigSeq for abbreviation.

The proposed method, BigSeq is evaluated according to the following criteria:

- *Compression Ratio:* To measure how well the dataset is compressed using BigSeq.
- *Total Response Time:* To measure the efficiency of BigSeq.
- *The Number of Patterns:* The number of detected patterns that used for compression.

1) Compression Ratio: Table VIII reports the compression ratio of the two algorithms on the five datasets. Recall, the larger compression ratio is the better compression we have. The BigSeq algorithm shows a better compression ratio in all datasets. For example, in Gene dataset, the compression ratio of BigSeq is 2.428 while the compression ratio of GoKrimp is 1.251.

TABLE VIII. COMPRESSION RATIO OF THE TWO ALGORITHMS (BIGSEQ AND GOKRIMP)

Dataset	BigSeq	GoKrimp
msnbc	1.648	1.123
Gene	2.428	1.251
TCAS	3.384	2.951
Activity	1.520	1.077
JBoss	3.303	1.541

2) Total Response Time (Sec): Table IX reports total response time (Sec) of the two algorithms on the five datasets. The BigSeq algorithm has the best execution time on all datasets. On msnbc dataset, Gene dataset, TCAS dataset, Activity dataset, and JBoss dataset, BigSeq outperforms GoKrimp by more than two orders of magnitude, more than one order of magnitude, more than three orders of magnitude, approximately three factors, and more than one order of magnitude respectively.

TABLE IX. TOTAL RESPONSE TIME (SEC) OF THE TWO ALGORITHMS (BIGSEQ AND GOKRIMP)

Dataset	BigSeq	GoKrimp
msnbc	1.32	313
Gene	1.1	45.2
TCAS	0.135	199
Activity	0.066	0.239
JBoss	0.072	2

3) Number of Patterns: Table X reports the number of patterns that used for compression by the two algorithms on the five datasets. Note that BigSeq used only one pattern for all datasets. This pattern is the compact BigSeq itself.

TABLE X. THE NUMBER OF PATTERNS OF THE TWO ALGORITHMS (BIGSEQ AND GOKRIMP)

Dataset	BigSeq	GoKrimp
msnbc	1	27
Gene	1	6
TCAS	1	33
Activity	1	2
JBoss	1	5

VII. CONCLUSION

In this paper, we focus on summarizing the event sequence dataset. Existing methods summarize the event sequence dataset by mining a significant patterns that compress the dataset well. In contrast, the novel proposed method, BigSeq summarizes the event sequence dataset by merging the event sequences into compact big sequence. The construction of the compact big sequence is done via the longest common subsequence and the novel definition of the compatible event set. Our compact big sequence converses the all characteristics of the original event sequences. Experimental results show that BigSeq method can achieve better performance than the state-of-the-art methods such as GoKrimp in terms of compression ratio, total response time, and number of detected patterns. As future work, we plan to adapt the BigSeq method for mining the frequent, closed, and maximal patterns in the event sequence dataset.

ACKNOWLEDGMENT

I wish to express my deep gratitude to my mentor Prof Dr. Karam Gouda. I am very grateful to my parents, my wife, my brother, and my sisters for their continuous moral support and encouragement.

REFERENCES

[1] A. Asuncion and D. Newman. *UCI machine learning repository* University of California, Irvine, School of Information and Computer Sciences, 2007. [Online]. Available: http://archive.ics.uci.edu/ml

[2] B. Le, H. Duong, T. Truong, and P. Fournier-Viger. *FCloSM, FGenSM: Two New Algorithms for Efficiently Mining Frequent Closed and Generator Sequences using Local Pruning Strategy*. Knowledge and Information Systems (KAIS), Springer, 53(1):71-107, 2017.

[3] B. Vo, S. Pham , T. Le, Z-H Deng. *A novel approach for mining maximal frequent patterns*. Expert Syst Appl 73:178-186, 2017

[4] C. Liu, X. Yan, L. Fei, J. Han, and S. P. Midkiff. *SOBER: statistical model-based bug localization*. In ACM SIGSOFT ESEC-FSE, 2005.

[5] D. Lo, S.-C. Khoo, and Chao Liu. *Efficient Mining of Iterative Patterns for Software Specification Discovery*. In, KDD, 2007.

[6] F. Bariatti, P. Cellier, and S. Ferré. *GraphMDL: graph pattern selection based on minimum description length*. In: International symposium on intelligent data analysis (IDA). Springer, 54-66, 2020.

[7] F. Fumarola, P. F. Lanotte, M. Ceci, and D. Malerba. *CloFAST: closed sequential pattern mining using sparseand vertical id-lists*. Knowl Inf Syst, 48:429-463, 2016.

[8] H. T. Lam, F. Mörchen, D. Fradkin, and T. Calders, *Mining compressing sequential patterns*, In SDM, 2012.

[9] H. T. Lam, F. Mörchen, D. Fradkin, and T. Calders. *Mining compressing sequential patterns. Statistical Analysis and Data Mining*, 7(1):34-52, 2014.

[10] http://www.msnbc.com

[11] J. Pei, J. Han, B. Mortazavi-Asl, J. Wang, H. Pinto, Q. Chen, U. Dayal, M.-C. Hsu. *Mining sequential patterns by pattern-growth: The prefixspan approach*. IEEE Trans. Knowl. Data Eng. 16(11) 1424-1440, 2004.

[12] J. Rissanen, *Modeling by shortest data description*, Automatica, 14(1):465-471, 1978.

[13] J. Vreeken, M. van Leeuwen, and A. Siebes. *KRIMP: Mining itemsets that compress.* Data Min. Knowl. Disc., 23(1):169-214, 2011.

[14] J. Wang and J. Han. *BIDE: efficient mining of frequent closed sequences* In ICDE, 2004.

[15] K. Gouda, M. Hassaan, and M. J. Zaki. *Prism: An effective approach for frequent sequence mining via prime-block encoding*. Journal of Computer and System Sciences 76:88-102, 2010.

[16] L. Wei, M. Liao, Y. Gao, R. Ji, Z. He, and Q. Zou. *Improved and promising identification of human micrornas by incorporating a high-quality negative set* IEEE/ACM Transactions on Computational Biology and Bioinformatics, 11(1) 192-201, 2014.

[17] M. Hutchins, H. Foster, T. Goradia, and T. Ostrand. *Experiments of the effectiveness of dataflow- and controlflow-based test adequacy criteria*. In ICSE, 1994.

[18] N. Tatti and J. Vreeken. *The long and the short of it: Summarizing event sequences with serial episodes.* In KDD, 462-470. ACM, 2012.

[19] P. Fournier-Viger, A. Gomariz, M. Campos, and R. Thomas. *Fast vertical mining of sequential patterns using co-occurrence information*. In Pacific-Asia Conference on Knowledge Discovery and Data Mining. Springer, 40-52, 2014.

[20] P. Fournier-Viger, C.-W. Wu, A. Gomariz, and V. S. Tseng. *VMSP: Efficient Vertical Mining of Maximal Sequential Patterns*. Proc. 27th Canadian Conference on Artificial Intelligence (AI), Springer, LNAI, pp. 83-94, 2014.

[21] P. Grünwald, *The Minimum Description Length Principle*, MIT Press, 2007.

[22] T. Makhalova, S. O. Kuznetsov, and A. Napoli *Mint: MDL-based approach for Mining INTeresting Numerical Pattern Sets*. Data Mining and Knowledge Discovery 36:108-145, 2022.

[23] Y. Li, S. Zhang, L. Guo, J. Liu, Y. Wu, X. Wu. *NetNMSP: Nonoverlapping maximal sequential pattern mining*. Applied Intelligence, 52:9861-9884, 2022.

[24] Y. Wua, C. Zhu, Y. Li, L. Guo, X. Wue. *NetNCSP: Nonoverlapping closed sequential pattern mining*. Knowledge-Based Systems 196, 2020.