Efficient Methods for Clickstream Pattern Mining on Incremental Databases

BAY VO\textsuperscript{1}, (Member, IEEE), HUY-CUONG NGUYEN\textsuperscript{1}, BAO HUYNH\textsuperscript{1}, AND TUONG LE\textsuperscript{2,3}

\textsuperscript{1}Faculty of Information Technology, HUTECH University, Ho Chi Minh City 700000, Vietnam
\textsuperscript{2}Informetrics Research Group, Ton Duc Thang University, Ho Chi Minh City 700000, Vietnam
\textsuperscript{3}Faculty of Information Technology, Ton Duc Thang University, Ho Chi Minh City 700000, Vietnam

Corresponding author: Bao Huynh (hq.bao@hutech.edu.vn)

This work was supported by the Vietnam National Foundation for Science and Technology Development (NAFOSTED) under Grant 02/2019/TN.

\textbf{ABSTRACT} Sequence pattern mining is a core task of data mining in various fields. Clickstream pattern mining, a variant of sequence pattern mining, is very useful in e-commerce, and is used to analyze, evaluate and predict online customer behaviors, attracting the interest of many researchers. Clickstream pattern mining has become important due to its wide range of applications. However, most previous approaches are not suitable to apply when clickstreams are inserted into a database, because this task is too time consuming.

The challenge of this problem is how to find a solution to minimize the runtimes and reduce the number of times the original databases are scanned to reduce the computational cost in the mining process on incremental databases. In this paper, we propose two effective methods for mining clickstream patterns from incremental databases, named inCMUB and Eff-inCMUB, based on the pre-large concept. inCMUB inserts new clickstreams from an inserted database into the existing tree and mines all frequent clickstream patterns, while Eff-inCMUB is a new approach, and builds a new tree from the inserted database to find pre-large 1-patterns and then it updates the pre-large clickstream patterns mined from the original database to extract frequent clickstream patterns. The experiments show that our proposed methods outperform the SMUB algorithm in terms of runtimes, memory usage and scalability on real-world clickstream databases.

\textbf{INDEX TERMS} Incremental database, clickstream pattern, sequence pattern mining, SPPC-tree.

\section*{I. INTRODUCTION}

Sequential pattern mining is an essential technique of data mining, and has been widely applied in various fields such as market analysis [1], stocks [2], weather prediction [3], medical treatment [4]–[6], mobile robots [7], and so on. Recently, many methods based on sequential pattern mining were developed, such as those used to analyze intrusion detection systems (IDS) [8], generate sitemaps [9], extract all recency-based sequential patterns (RF-Miner) [10], and carry out Vehicle Trajectory Prediction (VTP) [11].

A variant of sequential patterns, called clickstream patterns has also attracted a number of researchers in recent years. A clickstream is a list of ordered actions that the user takes when navigating a website. Clickstream patterns are usually applied in two popular fields, e-commerce and network traffic analysis. In e-commerce, clickstream pattern mining is used to analyze, evaluate and predict online customer behaviors to determine the effectiveness of the site as a marketing channel, and is especially useful in the digital era. In network traffic analysis, the server will track how many pages are served to the visitor, how long it takes each page to load, how much data is transmitted before the user moves on, etc.

Nevertheless, most existing clickstream pattern mining methods [12]–[16] only focus on static sequence databases, ignoring incremental databases. This is despite the fact that many databases are updated incrementally, such as customer online transaction databases in e-commerce, which grow because new transactions are appended into the existing databases daily when new customers or existing customers buy goods, the same as happens with stock price sequences which grow incrementally over time. There are two kinds of database updates in applications, which are inserting new clickstreams and/or appending new actions to the existing clickstreams. Previous approaches are not suitable for handling this situation because the result mined from the old
database is no longer valid on the updated database, and it is extremely inefficient to mine the updated databases. The challenge of this problem is how to find a solution to increase the runtime and reduce the number of times the original databases are rescanned to reduce the computational cost in mining clickstream patterns from incremental databases.

In this paper, we apply the pre-large concept and B-List structure generated from an SPPC-Tree to propose two effective methods for mining clickstream patterns from incremental databases. In this context, we only consider the case that inserts new clickstreams into the existing database. Our experiments on various real-time databases have shown that the proposed methods outperform the SMUB algorithm [13] in terms of runtimes and memory consumption, especially on large databases with low minimum support thresholds. Our contributions can be summarized as follows:

- We apply a compact data structure named SPPC-tree to compress the original database and reduce the cost of database scans in clickstream pattern mining.
- We reduce the rescanning of the original database based on a safety threshold. If the number of inserted clickstreams is not greater than a safety threshold, then the algorithm does not need to rescan the original database. This is the first study to apply the pre-large concept to a clickstream database. A pre-large clickstream acts like a buffer and is used to reduce the movement of clickstreams directly from large to small and vice versa during the incremental mining process.
- Two effective methods for mining clickstream patterns from incremental databases are proposed. The first method inserts the large 1-clickstream patterns from the inserted database into the existing tree, then mines all frequent clickstream patterns from the updated tree. The second is a novel method, it creates a new tree for pre-large and large 1-clickstream patterns from the inserted database and updates the existing list of pre-large/large clickstream patterns mined from the original database, then selects all frequent clickstream patterns. The proposed methods do not rescan the original database until the number of newly inserted clickstream exceeds a safety threshold.
- We evaluate the proposed algorithms on real-word and synthetic clickstream databases.

The remainder of this paper is organized as follows. Related works are briefly presented in section 2. The basic concepts are described in section 3. In section 4, we present the two proposed methods for clickstream pattern mining from incremental databases. The experimental results for showing the performance of the proposed methods are provided in section 5, and the conclusion is given in section 6.

II. RELATED WORKS

Sequential pattern mining (SPM) is an essential task of data mining, and has been widely applied in numerous applications, with many algorithms having been proposed. These algorithms can be classified into two groups:

- The horizontal database format includes AprioriAll [17], GSP [18], FreeSpan [19], and PrefixSpan [20]. AprioriAll is a popular algorithm that uses the “generate and test candidate” approach, GSP is an extension of AprioriAll, FreeSpan reduces the size of the databases slowly by using a projection process, and PrefixSpan is an improved version of FreeSpan that uses a prefix projection method.
- The other group contains those algorithms with a vertical database format, such as SPADE [21], SPAM [22], PRISM [23], CM-SPADE and CM-SPAM [24]. SPADE is one of the most efficient algorithms for SPM, and it identifies all frequent sequential patterns by BFS or DFS based on lattice decomposition. SPADE uses a depth-first search strategy to generate candidate patterns and a bitmap structure to store position information and encode the database. PRISM is based on primal block encoding to present the database. CM-SPADE and CM-SPAM are improved versions of SPAM and SPADE by integrating the CMAP structure, which stores co-occurrence information about items to prevent redundant candidates from being generated.

Many compact pattern mining approaches have been proposed instead of mining a full set of sequential patterns, such as closed sequential pattern mining [25]–[26], mining sequence pattern with constraints [27]–[28], and mining top-k frequent patterns [29]–[30].

Parallel computing approaches have also been studied [31] to speed up sequence pattern mining processing, with pDBV-SPM [32] using a dynamic bit vector structure to quickly calculate the support values, while MCM-SPADE [33] uses the CMAP (Co-occurrence MAP) structure for storing co-occurrence information, with both pDBV-SPM and MCM-SPADE being based on multi-core architecture. Recently two methods named DPC-Compact-SPADE and APCompact-SPADE [34] were proposed for mining frequent weighted clickstream patterns, based on hyper-threading technology. In addition, many proposed methods are based on computers with distributed memory [35]–[37].

Recently, researchers turned their attention to clickstream patterns, a special case of sequential patterns that are used in market behavior analysis and network traffic analysis. Many methods have been proposed for mining this kind of pattern, such as SMUB [13], CM-WSPADE [15], CUP [14], and CCPC [16]. SMUB mines the clickstream patterns based on the B-List structure, CM-WSPADE mines frequent weighted clickstream patterns based on the WIBList data structure and WCMAP (Weighted Co-Occurrence Map). The CUP mines clickstream pattern based on the pseudo-IDLList data structure and applies the DUB (Dynamic intersection Upper Bound) constraint to reduce candidates. The CCPC mines closed clickstream patterns based on the C-List structure.

In many applications, databases are always updated incrementally, such as customer online transaction databases in e-commerce that grow due to the new transactions being inserted into the existing databases. However, the previous
works are not suitable for mining on incremental databases because the result mined from the original database is no longer valid on the updated database.

Many algorithms for incremental pattern mining have thus been proposed, such as PreFUSP-TREE-INS [38], IncWTP [39], SSP-Tree [40]. PreFUSP-TREE-INS mines sequential patterns based on the PreFUSP-tree structure. It does not rescan the original database until the number of newly added sequences exceeds a safety threshold. The preFUSP-tree structure is a combination of the pre-large concept and the FUSP-tree structure. IncWTP is used to detect and delete the failed sequence data in a timely manner, and SSP-Tree mines rare association rules from incremental databases.

In addition, various pattern mining approaches for use with incremental databases have been studied, such as Pre-FUT [41], FCIL. [42]–[43], MU2P-Miner [44], IWEL [45], PRE-HAUIMI [46], APHAUP [47], RHUPS [48], LIMHAUUP [49], and ILDHUP [50]. Pre-FUT uses the Trie data structure and the concept of pre-large itemsets for mining incremental frequent itemsets. FCIL (Frequent Closed Itemset Lattice) builds an incremental frequent (closed) itemsets lattice. MU2P-Miner mines incremental maximal frequent patterns from univariate uncertain data. IWEL mines erasable patterns in an incremental database with weighted conditions. PRE-HAUIMI mines the set of high average-utility itemsets based on the average-utility-list structure and the pre-large concept to reduce the number of database scans. APHAUP uses two upper-bounds to reduce the size of candidates and decrease the number of candidates for HAUIM. RHUPS mines high utility patterns without generating candidate patterns by applying the sliding window model that calculates the influence of the insertion time of each batch in stream database. LIMHAUUP mines high average utility patterns from dynamic databases based on a compact data structure named HAU-Lists. ILDHUP mines HUI patterns from dynamic databases based on the HUPM approach and damped window model to decrease the importance of older data over time.

The challenges of this problem are how to find a solution to minimize the memory usage and reduce the number of rescans of the original database to reduce the computational cost in mining clickstream pattern process on incremental databases.

In this paper, we propose two effective methods for mining clickstreams from an incremental database based on the SPPC tree, a compact data structure to compress the databases and reduce the rescanning of the original database when the inserted database into the existing database based on pre-large concept. A pre-large clickstream acts like a buffer and is used to reduce the movement of clickstreams directly from large to small and vice versa during the incremental mining process.

III. BACKGROUND AND PROBLEM DEFINITION

Given $A = \{a_1, a_2, \ldots, a_n\}$ is a set of $n$ distinct actions with the same category. A subset $X = \{x_1, x_2, \ldots, x_m\}$, where $x_i \in A, i \in [1, m]$, is called an itemset.

Definition 1: A clickstream is a list of actions taken through a website by user interactions in order, denoted by $C = \{c_1, c_2, \ldots, c_m\}$, where $c_i \in A$ is an action ($i \in [1, m]$).

Definition 2: A set of clickstreams is called a clickstream database, denoted by $CDB$. Each clickstream in a clickstream database has a unique identification number, called a cid.

Definition 3: The length of the clickstream $C = \{c_1, c_2, \ldots, c_m\}$ is the number of actions it contains. A clickstream with length $k$ actions is called a $k$-clickstream. For example, clickstream $C = \{c, a, b, c\}$ is a 4-clickstream with three unique actions $\{a, b, c\}$, where action $c$ exists in positions 1, 4, action $a$ exists in position 2 and action $b$ appears at position 3.

Definition 4: A clickstream $\alpha = \{\alpha_1, \alpha_2, \ldots, \alpha_u\}$ is a sub-clickstream of another clickstream $\beta = \{\beta_1, \beta_2, \ldots, \beta_v\}$, denoted by $\alpha \subseteq \beta$, if there exist integers $1 \leq k_1 < k_2 < \ldots < k_u \leq v$ such that $\alpha_j = \beta_k, \forall j \in [1, u]$. In other words, $\beta$ is also called a super-clickstream of $\alpha$.

Definition 5: A clickstream, which is a sub-clickstream of at least one user clickstream in $CDB$, is called a clickstream pattern.

Definition 6: The support count of a clickstream $X$, denoted $\theta (X)$, is the number of user clickstreams in $CDB$ that contain $X$, i.e., $\theta (X) = |\{C_i \in CDB | X \subseteq C_i\}|$.

A clickstream $X$ is a frequent clickstream pattern if and only if its support is more than or equal to the given threshold value defined by users, $\theta (X) \geq \psi$, where $\psi$ is the minimum support threshold value defined by the user. The clickstream pattern mining task is to mine all frequent clickstream patterns in $CDB$.

Definition 7: An inserted database, denoted as $\Delta D$, is a set of new clickstreams added into an existing database. The new and original databases integrated together in an incremental database and denoted as $D^*$, where $D^* = CDB \cup \Delta D$.

TABLE 1 is an illustration of an original clickstream database. Given threshold $\psi = 2$, the user clickstream $\{c, a, b, c\}$ has $cid = 104$, and $\{c, b, c\}$ is a clickstream pattern due to $\theta (\{c, b, c\}) = 3 > \psi$.

| cid  | Clickstream |
|------|-------------|
| 101  | bcaabcaabc |
| 102  | dba         |
| 103  | aecbc       |
| 104  | cabc        |
| 105  | bffecf      |
| 106  | dabc        |

A. PROBLEM STATEMENT

Given a clickstream database $CDB$, a minimum support threshold $\psi$, and an inserted clickstream database $\Delta D$.

The problem of clickstream pattern mining on incremental databases is to mine all frequent clickstream patterns from $D^* = CDB \cup \Delta D$. 
To resolve this problem, we can modify existing algorithms for mining sequence patterns such as CM-SPADE, SMUB, CUP. However, the inserted database is usually very small compared to the original database. Thus, rescanning the incremental database takes much time and is not needed in most cases. The problem is how to mine or update the clickstream patterns from the inserted database without/reducing rescanning the original database to reduce the computational cost in the mining process.

Section 4 presents more details of the proposed methods for mining clickstream patterns from incremental databases.

IV. CLICKSTREAM PATTERN MINING ON INCREMENTAL DATABASES

In this section, we firstly present the pre-large concept to improve the mining process and SPPC-Tree structure. Based on the pre-large concept and B-List structure generated from the SPPC-Tree structure [13], two effective clickstream pattern mining methods for incremental databases are then proposed.

A. PRE-LARGE CONCEPT

The pre-large concept [51] uses two thresholds, a lower support threshold ($S_L$) and an upper support threshold ($S_U$). $S_U$ is the same as the minimum support used in static mining algorithms. Assume that $SC_U$ and $SC_L$ are the support count of $S_U$ and $S_L$, respectively. A clickstream pattern with the support count ($SC$) $\geq SC_U$ is a large clickstream pattern (or frequent clickstream pattern). A clickstream pattern with $SC < SC_L$ is a small clickstream pattern (or infrequent clickstream pattern). On the other hand, a clickstream pattern with $SC_L \leq SC < SC_U$ is considered to be a pre-large clickstream, and it may be a large clickstream in the future. A pre-large clickstream acts like a buffer and is used to reduce the movement of clickstreams directly from large to small and vice versa during the incremental mining process.

To reduce the need for rescanning the original database, a safety threshold $f$ based on the pre-large concept was applied. The safety threshold $f$ is defined as follows [45]:

$$f = \left\lfloor \frac{(S_U - S_L) \times |CDB|}{(1 - S_U)} \right\rfloor,$$

where

$$S_U, S_L \in (0, 1)$$

$$S_L < S_U,$$

where $S_U$ is the upper threshold, $S_L$ is the lower threshold, and $|CDB|$ is the number of clickstreams in the original sequence database $CDB$. The $f$ value will be rounded down to be an integer value.

Given a clickstream $X = \{x_1, x_2, \ldots, x_k\}$, there are three cases:

- $\theta(X) < SC_L$, $X$ is a small clickstream pattern
- $SC_U \leq \theta(X) < SC_L$, $X$ is a pre-large clickstream pattern
- $\theta(X) \geq SC_U$, $X$ is a large clickstream pattern

Property 1 [52]: A large clickstream pattern in $D^*$ must be large in either $CDB$ or $\Delta D$ (or both).

Property 2 [52]: A small clickstream pattern in both $CDB$ and $\Delta D$, it also small in $D^*$.

Lemma 1 [51]: If $|\Delta D| < f$, the algorithm does not need to rescan the original database.

A small clickstream in the original database that is a large clickstream in the inserted database cannot be a large clickstream for the incremental database.

Example, consider the inserted database in TABLE 2, assume $S_U = 50\%$, $S_L = 30\%$ we have

$$f = \left\lfloor \frac{(0.5 - 0.3) \times 6}{(1 - 0.5)} \right\rfloor = 2,$$

where $|CDB| = 6$. The algorithm does not need to rescan the original database in TABLE 1 to confirm the small clickstreams in the original database.

| cid | Clickstream |
|-----|-------------|
| 107 | f c b e e   |
| 108 | f e         |

B. SPPC-TREE

SPPC-tree [16] is a data structure, and each node in the tree consists of the following fields:

- **action-name** is the current action name.
- **count** is the number of actions sharing a common path from the root to the current node.
- **first-child** is a list of the first children that are expanded from the current node.
- **first-father** is the first previous node that is reached from the root node.
- **right-sibling** is the first next node with the same level as the current node.
- **label-sibling** is a list of nodes with the same action-name, and includes nodes that exist in different branches of the tree.
- **pre-code** is a pre-order number assigned by the pre-code traversal of the tree.
- **post-code** is a post-order number assigned by the post-code traversal of the tree.

The B-List structure is generated from SPPC-Tree and has the following form (pre-code, post-code, count). For example, with the clickstream database $CDB$ in TABLE 1, assume $S_U = 50\%$, $S_L = 30\%$, the SPPC-Tree is shown in FIGURE 1. In more detail, the node $a(2,4):1$ in FIGURE 1(b) includes **action-name** is a, **count** is 1, **first-child** is $a(3, 3):1$, **first-father** node is $c(1,6):2$, **right-sibling** node is $e(7,5):1$, **label-sibling** is $a(3,3):1$, **pre-code** is 2 and **post-code** is 4.

161308
Algorithm 1 Pre-CMUB (Building an SPPC-Tree With Lower Support \( S_L \))

**Input:** A clickstream database \( D \) and threshold \( S_L \)

**Output:** An SPPC-tree \( T \)

1. Scan \( D \) to find all 1-clickstreams with their support \( \geq S_L \times |D| \)
2. Eliminate all small 1-clickstream patterns (support \( < S_L \times |D| \)) in \( D \)
3. Build an SPPC-tree \( T \) with a root node \( R = \text{NULL} \)
4. For each clickstream \( cls \) in \( D \)
   5. \( r = R \)
   6. For each action \( a \) in \( cls \)
      7. If \( r \) has a child node \( c \) such that \( c\text{-action-name} = a\text{-action-name} \)
         8. \( c\text{-count} = c\text{-count} + 1 \)
      9. Else
         10. Create a new node \( c \) with \( c\text{-action-name} = a\text{-action-name} \)
            11. \( c\text{-count} = 1 \)
            12. Add node \( c \) as a first-child node of \( r \)
            13. \( r = c \)
   14. Traverse the tree \( T \) to create pre-code and post-code for each node after it is built.
   15. Return \( T \)

1) AN UPDATED TREE METHOD FOR INCREMENTAL CLICKSTREAM PATTERN MINING USING B-LIST

**a: inCMUB ALGORITHM**

A SPPC tree must be built in advance from the original database before new clickstreams are inserted. When new clickstreams are added, then based on the safety threshold, the algorithm will determine whether it needs to rescan the original database or not. However, small clickstreams from the original database will at most become pre-large and cannot become large, thus reducing the amount of rescanning necessary.

inCMUB does not need to rescan the original database if \( |\Delta D| \leq f \), it just scans the inserted database to find all 1-clickstream patterns that satisfy \( S_L \) and inserts them into the existing SPPC-tree built from the original database. Based on the updated tree, the algorithm mines all large \( k \)-clickstream patterns from \((k-1)\)-clickstream patterns and its B-List. This approach saves much time as it does not rescan the whole database to build the tree. Details of the inCMUB algorithm are shown in Algorithm 2.

Initially, the original database \( CDB \) is empty, so the first incremental database \( \Delta D \) is to be the original database, and the algorithm determines the safety threshold \( f \) (lines 1-2). If the number of clickstreams in the incremental database is greater than the safety threshold, the algorithm updates the safety threshold \( f \) and builds the SPPC-tree for an integrated database by executing the pre-CMUB procedure (lines 3-4).

In contrast, the algorithm scans the inserted database once, finds and updates the tree with 1-clickstream patterns that satisfy \( S_L \) by calling the Update-Tree procedure (or a pre-large pattern) (line 6).

Next, the algorithm creates the B-List for each large 1-clickstream pattern from the tree (line 7). Finally, the
full set of large k-clickstream patterns is mined by joining (k-1)-clickstream patterns (lines 8-9).

The same as the tree-building process, the Update-Tree procedure removes small 1-clickstream patterns from the inserted database (line 10). Next, it traverses the existing tree for each action in clickstreams in ΔD. If a node has existed, it updates its node information by increasing the count (lines 15-16). Otherwise, a new child node appends to the tree (lines 18-21). Finally, it updates the pre-code and post-code values for each node after it is built.

The last mining procedure finds all large k-clickstream patterns by joining the (k-1)-clickstream patterns that have the same prefix (lines 24-26). For each candidate pattern, c, if their support count of c is no smaller than SC, then the procedure adds c to CFP (lines 27-29), finally, the procedure calls itself recursively to conduct mining processes to extend the clickstream patterns (line 30).

**b: AN EXAMPLE**

Consider the original database CDB in TABLE 1, inserted database ΔD in TABLE 2, \( S_U = 50\% \) and \( S_L = 30\% \).

Initially, \( |CDB| = 6, SC_U = |S_U \times |CDB| | = [50\% \times 6] = 3, SC_L = |S_L \times |CDB| | = [30\% \times 6] = [1.8] = 2. \)

The tree T was built from \( F_1 = \{a, b, c, d, e\} \), the clickstream patterns satisfying \( SC_L = 2 \) are shown in FIGURE 1, and the B-List \( L_1 \) for each large 1-clickstream pattern is shown in TABLE 3.

| Large clickstream | B-List | count |
|-------------------|--------|-------|
| a                 |        | 5     |
| b                 |        | 6     |
| c                 |        | 4     |
| d, ac, eb, cc, abc, ebc | | 3     |

The large k-clickstream patterns are generated by joining large (k-1)-clickstream patterns. TABLE 4 shows the full set of frequent clickstream patterns.

| Large clickstream | count |
|-------------------|-------|
| a                 | 5     |
| b                 | 6     |
| c, ac, eb, cc, abc, ebc | 3     |

Consider inserted database \( \Delta D \) in TABLE 2, \( SC_L = 3, SC_U = 4 \) and the safety threshold \( f \) are re-calculated. The algorithm does not need to rescan the original database CDB because \( |\Delta D| = 2 \leq f \). \( \Delta F_1 = \{a, b, c, d, e\} \) is found from \( \Delta D \) and inCMUB updates each clickstream in \( \Delta D \) into the tree \( T \).

For \( cid = 107 \), the \( seq = \{c, b, c, e\} \) starting from \( c \) clickstream, node \( (18, 21) \) c:1 in \( T \) is updated \( (18, 21) \) c:2. Because the node \( (18, 21) \) c:2 does not have any child nodes, the next node \( b, c, e \) is appended under node \( c \), and a similar thing happens to \( cid = 108 \) for \( e \) clickstream. The tree \( T \) is updated as shown in FIGURE 2, which includes pre-code and post-code values.

Clickstream \( d \) is eliminated because the support count does not satisfy the upper threshold, so the list of large 1-clickstream patterns is shown in TABLE 5.

A recurrent clickstream pattern extends from (k-1)-clickstream patterns, and the full set of large k-clickstream patterns that are mined is shown in TABLE 6.

**2) EFFICIENT METHOD FOR UPDATED FREQUENT CLICKSTREAM PATTERNS IN INCREMENTAL DATABASES**

**a: EFF-INCMUB**

The process of finding and inserting all pre-large 1-clickstream patterns satisfying \( S_t \) from a database inserted into the existing tree is time consuming, especially when the
TABLE 5. The B-List of large 1-clickstream patterns in SPPC-Tree updated with the new database.

| Large clickstream | B-List | count |
|-------------------|--------|-------|
| a                 | (2,4):1→(3,3):1→(10,8):1→ | 5     |
| b                 | (11,1):1→(13,17):1→(19,20):1 | 7     |
|                  | (0,7):2→(4,2):1→(9,9):1→(12,10):1 |       |
|                  | →(16,14):1→(20,19):1→(22,23):1 |       |
|                  | (1,6):2→(5,1):1→(15,15):1→ | 5     |
| c                 | (17,13):1→(18,24):2→(21,18):1→ |       |
|                  | (23,22):1 |       |
| e                 | (7,5):1→(14,16):1→(24,21):1→ | 4     |
|                  | (25,25):1 |       |

TABLE 6. The set of large clickstream patterns that are mined.

| Large clickstream | Count |
|-------------------|-------|
| a, c, bc          | 5     |
| b                 | 7     |
| e, ab, cb, cc, abc| 4     |

size of the SPPC tree from the original database is large but the number of inserted clickstreams is small. Therefore, a new approach named Eff-inCMUB is proposed. Assume that we have pre-large and large clickstream patterns (PCP) mined from the original database. When the inserted database $\Delta D$ arrives, Eff-inCMUB will scan $\Delta D$ to build SPPC tree $T'$ and B-List $L_2$ of the 1-clickstream patterns that satisfy $S_L$. Finally, it traverses clickstreams in PCP and $L_2$ to update the support count for each clickstream in PCP. Frequent clickstream patterns (FCP) are easy to get by traveling the updated PCP to find patterns that satisfy $S_U$.

The description of the Eff-inCMUB algorithm is shown in Algorithm 3.

First, the original database $CDB$ is empty, so the first inserted database $\Delta D$ is the original database, and the algorithm initializes the safety threshold $f$ (lines 1-2). The algorithm builds the SPPC-tree $T$ and B-List $L_1$ for the integrated database (lines 4-5). Next, all pre-large/large clickstreams are generated in this phase (lines 6-7) in cases when the number of clickstreams in an incremental database is greater than the safety threshold.

In contrast, the Eff-inCMUB algorithm just scans the inserted database once and eliminates the small 1-clickstream patterns (line 9). It then builds the new tree $T'$ and creates the B-List $L_2$ pre-large/large 1-clickstream patterns of inserted database $\Delta D$ (lines 10-11). Next, the Update-List procedure is called to update the support count for each 1-clickstream pattern in PCP after updating (line 12). The full set of 1-clickstream pattern from tree $T'$ is extracted from the PCP that satisfies the upper threshold value (line 13).

The Update-List procedure traverses each node in B-List $L_1$ and calls the TraverseAndUpdate procedure to update its node information by $L_2$ (lines 14-15). If a node has existed both in $L_1$ and $L_2$, it updates its node information by summing the count of two nodes (lines 17-19). Next, it creates a new node and its B-List by joining each of the child nodes in $L_1$. If the count of the parent and child nodes is greater than the $S_L$ threshold, then update the $L_2$ with the new node (lines 20-24). Finally, the procedure calls itself recursively to conduct mining processes for each child node (lines 25-26).

The Eff-inCMUB algorithm is a new approach that can greatly outperform inCMUB with regard to runtime because
it does not need to traverse the tree that was built from the original database.

**b: AN EXAMPLE**

Similar to the example used with the inCMUB, we now consider the original database \( CDB \) in TABLE 1, new database \( 1D \) in TABLE 2, \( SU = 50\% \) and \( SL = 30\% \). Initially, \(|CDB| = 6, SC_{U} = \lceil SU \times |CDB| \rceil = \lceil 50\% \times 6 \rceil = 3 \) and \( SC_{L} = \lceil SL \times |CDB| \rceil = \lceil 30\% \times 6 \rceil = \lceil 1.8 \rceil = 2 \).

The \( CP_{1} = \{a, b, c, d, e\} \) and the SPPC-tree \( T \) is shown in FIGURE 1. The B-List \( L_{1} \) is generated from \( T \) for 1-clickstream patterns satisfying the \( SL \) threshold, as shown in TABLE 7.

### TABLE 7. The B-List of pre-large 1-clickstream patterns.

| pre-large clickstream | B-List \( L_{1} \) | count |
|-----------------------|------------------|-------|
| a                     | (2,4):1 → (3,3):1 → (10,8):1 → (11,11):1 → (13,17):1 → (19,20):1 | 5     |
| b                     | (0,7):2 → (4,2):1 → (9,9):1 → (12,10):1 → (16,14):1 → (20,19):1 | 6     |
| c                     | (1,6):2 → (5,1):1 → (15,15):1 | 4     |
| d                     | (6, 0, 1) → (8,12):2 | 3     |
| e                     | (7,5):1 → (14,16):1 | 2     |

The set of clickstream patterns that meet the \( SL \) threshold and is mined from \( L_{1} \) is shown in TABLE 8.

### TABLE 8. The set of clickstream patterns that meet the \( SL \) threshold.

| pre-large clickstream | count |
|-----------------------|-------|
| a                     | 5     |
| b                     | 6     |
| c, ab, bc             | 4     |
| d, abc, ac, cb, cbc, cc | 3 |
| e, ba, ca, da, db, cab, cabc, cac | 2 |

The algorithm scans \( \Delta D \) once (|\( \Delta D \)| \( \leq f \)) to find 1-clickstream patterns \( \Delta CP_{1} = \{a, b, c, d, e\} \) for which the support count is not less than the lower support. The new SPPC-tree \( T' \) is shown in FIGURE 3.

### FIGURE 3. The new \( T' \) from \( \Delta D \).

And the B-List \( L_{2} \) created from \( T' \) is shown in TABLE 9.

### TABLE 9. The B-List \( L_{2} \) is created from \( T' \).

| pre-large clickstream | B-List \( L_{2} \) | count |
|-----------------------|------------------|-------|
| b                     | (1,2):1 | 1     |
| c                     | (0,3):1 → (2,1):1 | 1     |
| e                     | (3,0):1 → (4,4):1 | 2     |

The update procedure is executed for \( L_{1} \) and \( L_{2} \). Node \( a \) and its child nodes are omitted because node \( a \) does not exist in \( L_{2} \). Next, node \( b \) exists in both \( L_{1} \) and \( L_{2} \), and the support count is updated to 7 in \( L_{1} \). There are two child nodes of \( b \), which are \( \{ba, bc\} \). The child node \( \{ba\} \) is omitted because node \( a \) does not appear in \( L_{2} \). The B-List of nodes \( \{bc\} \) is created, and this node is put into \( L_{2} \) because the support count (\( \{bc\}.count \) in \( L_{1} \) is 4 + \( \{bc\}.count \) in \( L_{2} \) is 1 > \( SL \times |D'| \)), satisfying the \( SL \) threshold, so it is put into \( L_{2} \) and the support count is updated to 5 in \( L_{1} \). A similar process happens for the remaining nodes \( c, d, e, \) and the B-List \( L_{2} \) after being updated is shown in TABLE 10.

### TABLE 10. The B-List \( L_{2} \) after being updated.

| pre-large clickstream | B-List \( L_{2} \) | count |
|-----------------------|------------------|-------|
| b, cb                 | (1,2):1 | 1     |
| bc, cc, cbc           | (2,1):1 | 1     |
| c                     | (0,3):1 → (2,1):1 | 1     |
| e                     | (3,0):1 → (4,4):1 | 2     |

The \( PCP \) unified list of \( L_{1} \) and \( L_{2} \) is shown in TABLE 11.

### TABLE 11. The \( PCP \) list merged by \( L_{2} \).

| pre-large clickstream | count |
|-----------------------|-------|
| a, c, bc              | 5     |
| b                     | 7     |
| e, ab, cb, cc, cbc    | 4     |
| d, abc, ac            | 3     |
| e, ba, ca, da, db, cab, cabc, cac | 2 |

The full set of large clickstream patterns extracted from the \( PCP \) is the same as that shown in TABLE 6.

### V. EXPERIMENTAL EVALUATION

This section evaluates the effects of the experimental methods, which are executed on a computer equipped with an Intel Core i5-5300U 2.3 GHz, 16GB main memory and Windows 10 operating system. The experiments are carried out on real-life datasets, with the details shown in TABLE 12.

The results show that the two proposed methods outperform the SMUB algorithm in both memory usage and running time, especially on huge clickstream datasets with a lower threshold.
TABLE 12. Databases used in the experiments.

| Database | #Clickstream | #Items    | Source                                      |
|----------|--------------|-----------|---------------------------------------------|
| Kosarak  | 990,002      | 41,270    | Clickstream data of a Hungarian online news portal |
| FIFA     | 20,450       | 2,990     | KDD Cup 1999                                |
| MSNBC   | 989,818      | 17        | Retail store customer transactions from NU-Mine Bench |
| BMS2     | 77,512       | 3,340     | Clickstream data from an e-commerce site that was used in KDD CUP 2000 |
| SUSY     | 5,000,000    | 18        | https://archive.ics.uci.edu/ml/datasets/SUSY# |

To depict the process for incremental databases, each experimental database is split into two parts, the first part is the original database, and the second is an incremental database for which the shares are 99.9% and 0.1%, respectively, for all databases, with the exception of the FIFA database, for which they are 98% and 2%. The second part continues to split the data into ten equal sub-parts, and each part is then inserted into the original database in the mining process.

A. THE RUNTIME

The results in this section confirmed that the runtime of the two proposed methods is always much better than the SMUB algorithm on all experimental databases. In particular, both of them can be executed on a large clickstream database with a very low threshold. A weakness of the proposed methods is that they are costly when a database is rescanned. However, most of the incremental databases are so small that this does not affect the overall cost. Moreover, with the smaller lower threshold the differences in the runtimes of Eff-inCMUB, inCMUB and SMUB are larger. FIGURE 4 to FIGURE 7 compare the execution times of the SMUB, inCMUB, and Eff-inCMUB algorithms.

In FIGURE 4, we compare the runtimes of SMUB, inCMUB, and Eff-inCMUB on the FIFA database with various settings. When the lower and upper threshold settings are $S_L = 0.08$ and $S_U = 0.1$, we found that the runtime of the inCMUB is faster than CMUB while that of Eff-inCMUB is the best, and the runtimes of these methods are stable when the new clickstream database is inserted. Eff-inCMUB is approximately 20 and 30 times faster than inCMUB and SMUB, respectively. When ten inserted databases are inserted both inCMUB and Eff-inCMUB only rescans the database once.

FIGURE 5 performs the same experiment for the BMS database. We found that the runtimes of inCMUB and Imp-inCMUB are much better than that of SMUB. Eff-inCMUB is the best, although it rescanned the database twice.

B. THE MEMORY USAGE

This sub-section reports the memory usage of the algorithms on the databases with various settings. FIGURE 8 shows the memory usages of Eff-inCMUB, inCMUB and SMUB for the FIFA database. With the setting is $S_L = 0.003$ and $S_U = 0.004$, we found that the runtime of the Eff-inCMUB does not increase, except for in the case of rescanning the database, while the runtime of inCMUB is also better than that of SMUB.

FIGURE 7 compares the runtimes of Eff-inCMUB, inCMUB and SMUB for the Kosarak database under various settings. When the setting is $S_L = 0.001$ and $S_U = 0.002$, and the runtime of Eff-inCMUB changes very little, while inCMUB and Eff-inCMUB are about 6 and 8 times faster than SMUB, respectively.

FIGURE 6 shows that the runtimes of SMUB, inCMUB and Eff-inCMUB for the Kosarak database with various settings. When the setting is $S_L = 0.003$ and $S_U = 0.004$, we found that the runtime of the Eff-inCMUB does not increase, except for in the case of rescanning the database, while the runtime of inCMUB is also better than that of SMUB.

FIGURE 9 reports the memory usages of Eff-inCMUB, inCMUB and SMUB with various settings for BMS2.
database. The memory usage of Eff-inCMUB is also better than that of inCMUB in the phase without rescanning the database, but it is not better when rescanning the database occurred, while the memory usages of inCMUB and CMUB are the same. Overall, the total memory usage of Eff-inCMUB is better than those of inCMUB and SMUB.

FIGURE 10 reports the memory usages of Eff-inCMUB, inCMUB and SMUB for the Kosarak database. With setting $S_L = 0.003$ and $S_U = 0.004$, the memory usage of Eff-inCMUB is the best, while those of inCMUB and SMUB are nearly the same.

FIGURE 11 shows the memory usages of SMUB, Eff-inCMUB and inCMUB for the MSNBC database. With setting $S_L = 0.001$ and $S_U = 0.002$, the memory usage of SMUB is better than those of Eff-inCMUB and inCMUB in some of the early stages. However, when the number of incremental databases is large, the memory usage of SMUB increases significantly, while the memory usages of Eff-inCMUB and inCMUB are stable. The MSNBC database has a large average clickstreams length (13.23), which leads to a large difference in the mining results in the two thresholds $S_L$ ($S_L = 0.001$, 16641 clickstreams) and $S_U$ ($S_U = 0.002$, 4251 clickstreams). Because the CMUB only executed a $S_U$ threshold so its memory usage in some first iterations is better than those of inCMUB and Eff-inCMUB, both of which run on the thresholds $S_L$ and $S_U$.

C. SCALABILITY ON LARGE DATABASE
To verify the performance and scalability on large data, we execute the proposed methods on the SUSY database that contains five million clickstreams.

1) THE RUNTIMES
The runtimes of both inCMUB and Eff-inCMUB are better than that of SMUB, and Eff-inCMUB is always the best method for mining clickstream patterns for the
and inCMUB are better than that of SMUB. $S_0 = 0.109$ and $S_U = 0.11$ in two settings. With this database, the memory usages of Eff-inCMUB and SMUB for the SUSY database are shown in FIGURE 13. Similarly, the memory usages of Eff-inCMUB, inCMUB and SMUB are nearly the same, while that of Eff-inCMUB is much smaller, at just half the memory usage of inCMUB. Similar results are found for setting $S_L = 0.109$ and $S_U = 0.11$, the memory usages of inCMUB and SMUB are nearly the same, while that of Eff-inCMUB is much smaller, at just half the memory usage of inCMUB. Similar results are found for setting $S_L = 0.109$ and $S_U = 0.11$, as the memory usages of Eff-inCMUB and inCMUB are better than that of SMUB. The difference in runtime between SMUB and the two proposed methods is significant. In particular, with a smaller lower threshold the runtime of the proposed methods is always better.

VI. CONCLUSION AND FUTURE WORK

This paper proposed two methods named inCMUB and Eff-inCMUB to mine the clickstream patterns on incremental datasets. Both inCMUB and Eff-inCMUB are very effective, especially in cases when the number of new clickstreams added is smaller than the safety threshold, as they do not need to rescan the original database. In addition, an outstanding feature of the proposed methods is that they are very efficient for large databases.

inCMUB mines large 1-clickstream patterns from the original clickstream database, then inserts the large 1-clickstream patterns from a new clickstream database into the existing tree for mining all large clickstream patterns. Eff-inCMUB first finds all clickstream patterns satisfying the lower threshold (known pre-large clickstream patterns). Next, it updates the set of pre-large clickstream patterns by using the pre-large 1-clickstream patterns from the inserted database, then extracts the full frequent clickstream patterns from the set of pre-large clickstream patterns that was updated. This method is more cost-effective than inCMUB as it does not traverse the tree.

In the future, we will improve the Eff-inCMUB to further reduce the running time in rescanning the database. We will also study strategies for mining clickstream patterns with constraints or those weighted on incremental databases. In addition, we will study how to use distributed architectures to improve the efficiency of the mining process.

REFERENCES

[1] F. Du, S. Guo, S. Malik, E. Koh, S. Kim, and Z. Liu, “Interactive event sequence prediction for marketing analysts,” in Proc. Extended Abstr. CHI Conf. Hum. Factors Comput. Syst., Apr. 2020, pp. 1–8, doi: 10.1145/3334480.3382971.

[2] Y. Shi, Y.-R. Tang, and W. Long, “Finding patterns of stock returns based on sequence alignment,” Proc. Comput. Sci., vol. 139, pp. 256–262, Dec. 2018, doi: 10.1016/j.procs.2018.10.265.

[3] C. Kunjumon, S. S. Nair, D. Rajan S., S. Suresh, and S. L. Preetha, “Survey on weather forecasting using data mining,” in Proc. Conf. Emerg. Devices Smart Syst. (ICEDSS), Mar. 2018, pp. 262–264, doi: 10.1109/ICEDSS.2018.8544326.

[4] C. Ou-Yang, C. P. Wulandari, R. A. R. Hariadi, H.-C. Wang, and C. Chen, “Applying sequential pattern mining to investigate cerebrovascular health outpatients’ re-visit patterns,” PeerJ, vol. 6, p. e5183, Jul. 2018, doi: 10.7717/peerj.5183.

[5] K. H. Hoang and T. B. Ho, “Learning and recommending treatments using electronic medical records,” Knowl.-Based Syst., vol. 181, Oct. 2019, Art. no. 104788, doi: 10.1016/j.knosys.2019.05.031.

[6] L. Langouche, A. Aralar, M. Sinha, S. M. Lawrence, S. I. Fraley, and T. P. Coleman, “Data-driven noise modeling of digital DNA melting analysis enables prediction of sequence discriminating power,” Bioinformatics, vol. 36, nos. 22–23, pp. 5337–5343, Dec. 2020, doi: 10.1093/bioinformatics/btaa1053.

[7] P. Duckworth, D. C. Hogg, and A. G. Cohn, “Unsupervised human activity analysis using artificial mobile robots,” Artif. Intell., vol. 270, pp. 67–92, Dec. 2019, doi: 10.1016/j.artint.2018.12.005.

[8] Y. Wang, Y. Liang, H. Sun, and Y. Ma, “Intrusion detection and performance simulation based on improved sequential pattern mining algorithm,” Cluster Comput., vol. 23, no. 3, pp. 1927–1936, Sep. 2020, doi: 10.1007/s10586-020-03129-8.

[9] M. Ceci and P. F. Lanotte, “Closed sequential pattern mining for sitemap generation,” World Wide Web, vol. 24, no. 1, pp. 175–203, Jan. 2021, doi: 10.1007/s11228-020-00839-2.
[50] H. Nam, U. Yun, E. Yoon, and J. C.-W. Lin, “Efficient approach of recent high utility stream pattern mining with indexed list structure and pruning strategy considering arrival times of transactions,” Inf. Sci., vol. 529, pp. 1–27, Aug. 2020, doi: 10.1016/j.ins.2020.03.030.

[51] T.-P. Hong, C.-Y. Wang, and S.-S. Tseng, “An incremental mining algorithm for maintaining sequential patterns using pre-large sequences,” Expert Syst. Appl., vol. 38, no. 6, pp. 7051–7058, Jun. 2011, doi: 10.1016/j.eswa.2010.12.008.

[52] H. Cheng, X. Yan, and J. Han, “IncSpan: Incremental mining of sequential patterns in large database,” in Proc. ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining, 2004, pp. 527–532, doi: 10.1145/1014052.1014114.

BAY VO (Member, IEEE) received the Ph.D. degree in computer science from the University of Science, Vietnam National University, Ho Chi Minh City, Vietnam, in 2011.

He is currently an Associate Professor and the Dean of the Faculty of Information Technology, HUTECH University, Ho Chi Minh City. His research interests include association rules, classification, mining in incremental database, distributed databases, and privacy preserving in data mining. He was a PC Member of several conferences, such as IJCAI, PAKDD, ICONIP, SMC, ICCCI, and ACIIDS. He is a Manager Editor of Vietnam Journal of Computer Science and an Associate Editor of ICIC Express Letters, Part B: Applications.

HUY-CUONG NGUYEN received the B.Sc. degree from the University of Science, Ho Chi Minh City, Vietnam, in 2008. He worked at Data Design Company, Vietnam, for 11 years with many roles as a developer, a team leader, and a project manager in software development. He is currently with the Faculty of Information Technology, HUTECH University, Ho Chi Minh City. Since 2019, he has been begin researching about sequence pattern, clickstream pattern, and graph in data mining.

BAO HUYNH received the M.Sc. degree in computer science from the Posts and Telecommunications Institute of Technology, Ho Chi Minh City, Vietnam, in 2011, and the Ph.D. degree in computer science from the VSB-Technical University of Ostrava, Czech Republic, in 2017. He is currently with the Faculty of Information Technology, HUTECH University, Ho Chi Minh City. Since 2017, he has been continuously researched and developed new algorithms that are helpful for real applications in various fields. He has over ten years of experience in designs, implements network infrastructure systems, and security network systems. His research interests include data mining, privacy-preserving, big data analytics, parallel computing, social networks, network infrastructure, and network security.

TUONG LE is currently a Researcher with the Informetrics Research Group, Ton Duc Thang University, Ho Chi Minh City, Vietnam. He is also with the Faculty of Information Technology, Ton Duc Thang University. He has published many articles in prestigious journals, such as Information Sciences, Expert Systems with Applications, IEEE Access, and Engineering Applications of Artificial Intelligence. His research interests include machine learning, imbalanced learning, deep learning, business intelligence, data analysis, data mining, and pattern mining. He served as a Reviewer for several journals, like IEEE TRANSACTIONS ON CYBERNETICS, IEEE ACCESS, Applied Soft Computing, Neural Computing and Applications, Applied Intelligence, PLOS ONE, and Engineering Applications of Artificial Intelligence.