Journal of Scientific & Industrial Research
Vol. 84, May 2025, pp. 531-543
DOI: 10.56042/jsir.v84i5.5439

Adaptive Hierarchical Clustering and Batch-free Top-K Sequential Pattern Mining for Data Streams

K Poongodi[1]*, Dhananjay Kumar[2] & K Meenakshi[2]

[1]Department of Information Technology, Sri Eshwar College of Engineering, Coimbatore 641 202, Tamil Nadu, India
[2]Department of Information Technology, MIT Campus, Anna University, Chennai 600 044, Tamil Nadu, India

Received 08 September 2023; revised 22 May 2024; accepted 08 April 2025

The Sequential Pattern Mining (SPM) is a challenging task in data streams due to huge memory and computational costs to meet accuracy in mined results. Sequential patterns mined from target stream in traditional batch-based processing results in pattern loss when the batches are processed independently, where the pattern frequency is determined local to the batch. However, if a pattern is frequent in the stream and its items appear in various batches, then this pattern never becomes frequent and hence requires pruning. To address this issue, the sequences are clustered by similarity using Adaptive Hierarchical Clustering (AHC) and Batch-Free Top-K Sequential Pattern Mining (BFTKSPM) algorithms proposed to mine approximate sequential patterns over data streams. The BFTKSPM algorithm targets data stream in a continuous and batch-free manner. The top-*k* sequential patterns are extracted from data streams and are maintained in an inverted tree structure. The experimental results of the proposed algorithm are carried out on benchmark datasets for data streams and it outperforms the existing batch-based methods in terms of execution time, memory, precision, recall, and F1-score.

Keywords: Inverted tree, Maximum pattern length, Minimum support, Prediction, Subsequences

Introduction

In sequential pattern mining, the frequency of sequential patterns occurring in a streaming data has significant impact due to its various applications in market basket analysis[1], intrusion detection[2], webpage click-stream analysis[3], etc. Several sequential pattern mining algorithms are developed where many are stream-oriented, as they need multiple scans over the dataset, and time-aware since they fail to consider the time limitation of sequential patterns.[4–6] The existing approaches are unsuitable to handle the streaming data, as they require multiple pass over the data, and cannot be applied on streams as it is difficult to store enormous amount of data before processing where the length of the stream is unknown.

The sequential patterns are discovered from sequence databases[7] based on the minimum support threshold[8] criteria set by the user. The sequential pattern mining discovers the frequent subsequences from a group of sequences and identifies relationship that exists between the patterns present in a sequence database. In a large search space[9], discovering all frequent sequences in huge databases is a difficult task. SPM analyses data considering the sequential ordering of events or elements.

The sequential pattern mining from data streams[10] discovers the sequential patterns where multiple scanning over data is not required as the data is generated continuously. The basic streaming approach divides the entire target stream into pieces of stream called chunks and mining of patterns from stream chunks generates errors when the mining is carried out non-continuously, since the patterns containing items are lost from two continuous batches. If a pattern is frequent in the data stream and its items are spread among different batches, such patterns are pruned and are not considered frequent. To overcome this drawback of traditional batch-based processing, a Batch-Free Top-*K* Sequential Pattern Mining algorithm, BFTKSPM is proposed for mining sequential patterns from data streams. The proposed algorithm discovers top-*k* sequential patterns from the tree and predicts future events based on the stored patterns where the method is applied on benchmark streaming datasets to prove its effectiveness.

The algorithm mines the data stream in a batch-free manner continuously. The patterns are not lost by maintaining the track of items in the stream for every *L* items. Adaptive hierarchical clustering groups the

*Author for Correspondence
E-mail: poongodikk@gmail.com

sequences by similarity, that provides high availability of data to the top-*k* sequential pattern mining method. This speed up the mining process and the patterns are stored in an inverted tree structure to reduce the computational overhead in updating the tree. The top-*k* patterns are retrieved from the tree by traversing the path from node to the root, thereby reducing the execution time of the algorithm. The pruning strategy adopted to remove unpromising items reduces the memory usage.

Related Work

A prediction framework[11] is presented to mine sequential patterns from data streams. A score is assigned to each pattern and the patterns discovered are used to determine the future events. The limitations of this predictor are the usage of several user-chosen parameters and hence, the importance to patterns is not assigned in generating accurate results. A Batch-Free Sequential Pattern Miner (BFSPMiner) algorithm[12] discovers frequent patterns from streaming data and the predictor predicts next item in the stream.

A Sequential Pattern mining in Evolving Data Streams[13] is designed to mine sequential patterns from data streams. Batch window and tilted-time window models are used to mine sequential patterns in evolving data streams. An online algorithm[14] for mining sequential patterns in data streams is formulated, where an automaton-based data structure is used to store the discovered frequent sequential patterns. The patterns are generated based on the user-defined threshold.

Mendes *et al.*[15] introduced bounded error and memory bound methods in stream sequence miner to mine sequential patterns from data streams. These methods divide the stream into various batches and each batch is processed only once. Koper and Nguyen[16] proposed sequential pattern mining algorithms for streaming data using bounded error algorithm. A tree based data structure is used to maintain the input sequences from data streams. When the values of the significance support threshold and the batch support threshold is close to the minimum support threshold, the number of patterns increases which causes a decrease in the accuracy of SS-BE, SS-BE2 and SS-LC2 algorithms.

A greedy algorithm[17] is introduced to identify frequent sequential patterns from data streams. A technique is built using the sequence tree and divides the stream into windows or patches where the tree is updated using the previous patches. An algorithm to mine rare sequential patterns from data streams is developed using a sliding window.[18] Infrequent patterns like rare sequential patterns discovers knowledge from the databases.

An algorithm to generate high utility sequential patterns from data streams is formulated.[19] A tree structure is adopted using memory adaptive mechanisms for storing the sequential patterns which has high utility discovered from fast changing streaming data. Zihayat *et al.*[20] proposed data structures to maintain the significant information of high utility sequential patterns in both online and offline modes. Using the sequence-suffix utility measure, the search space is pruned to carry out the mining process. High average utility pattern mining approach[21] is presented for extracting patterns from data streams using a utility measure. This framework uses the damped window model to generate useful patterns in the stream environment with a pruning technique to perform the mining process.

An erasable pattern mining algorithm over data streams[22] is designed based on damped window model for data streams. Since the data stream is growing in size over time, the presented technique caused memory overload. A streaming algorithm is formulated for automatic sequential pattern discovery evolving over time from multi-dimensional event streams.[23] In this approach, the time-series streams are fragmented into segments and similar segments are grouped together. A streaming algorithm is presented by Patnaik *et al.*[24] to discover frequent episodes from dynamic event streams. Wang *et al.*[25] investigated top-*k* closed co-occurrence pattern mining algorithm over multiple streams using Laplace and exponential mechanisms to analyse the problems for top-*k* sequential pattern mining.

Preliminaries

The ordered collection of patterns or items is defined as sequence and a pattern is called sequential pattern if its support threshold is larger than or equal to minimum support set by user. Consider two sequences $S_1 = \{X_1, X_2, \dots, X_n\}$ and $S_2 = \{Y_1, Y_2, \dots, Y_m\}$. A sequence S_1 is said to be contained in another sequence S_2 if there exist integers $1 \leq j_1 < j_2 < \cdots < j_n \leq m$ such that $X_1 \subseteq Y_{j_1}, X_2 \subseteq Y_{j_2}, \dots, X_n \subseteq Y_{j_n}$ denoted by $S_1 \subseteq S_2$.

A data stream[26] represents the flow of data, where the sequences of transactions arrive in a continuous manner denoted by $DS = \{ds_1, ds_2, \dots, ds_n, \dots\}$ where ds_n is the n^{th} arrived transaction at time t_n. Each transaction consists of patterns. In the data streaming context, a sequence of patterns $p =$

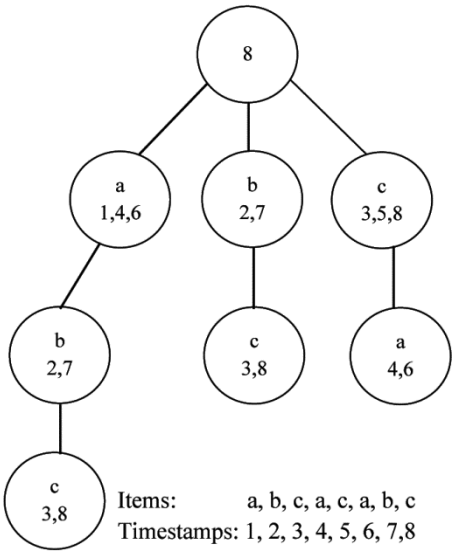

Fig. 1 — T_0 tree

$\{p_1, p_2, \ldots, p_n\}$ is available in the stream DS as a subsequence such that there exist integers $1 \leq i_1 < i_2 < \cdots < i_n \leq n$ where $p_1 \subseteq ds_{i_1}, p_2 \subseteq ds_{i_2}, \ldots, p_n \subseteq ds_{i_n}$.

For a data stream DS and a user-defined support threshold sup, such that $0 \leq sup \leq 1$, a subsequence q is called frequent pattern if

$$\frac{count_q}{n} \geq sup \quad \ldots (1)$$

where, $count_q$ is the number of times the pattern q has appeared in DS and n, the total number of patterns available in DS. The subsequence is termed as sequential pattern if the frequent patterns appear one after another in a sequential order.

A tree data structure is required to depict the patterns and T_0 is a tree structure[15], to maintain the details of all the sequential patterns which are considered as promising items and the root node of T_0 contains the total count of items appearing in the dataset. The other nodes in T_0 contain information about the patterns including number of times the pattern has occurred and name of item. The pattern present in corresponding node is shown by adopting the path from the root to the specific node considered. Example of T_0 tree is represented in Fig. 1.

Methodology

The Batch-Free Top-*K* Sequential Pattern Mining (BFTKSPM) algorithm is proposed to mine approximate sequential patterns in the data streams. The block diagram of the proposed algorithm is depicted in Fig. 2. Pre-processing is performed to remove the unwanted labels from the dataset and the algorithm uses the timestamp associated with the items for the generation of sequential patterns. The incoming items are pre-processed and clustered using adaptive hierarchical clustering based on similarity. On receiving the latest items, the algorithm generates the new patterns and these candidate patterns are stored in the inverted T_0 tree. Eventually, proposed algorithm generates top-*k* sequential patterns by pruning the unpromising patterns from the tree. From the generated patterns, the algorithm discovers the rules and the predictor predicts the next upcoming item from those rules which have high confidence values.

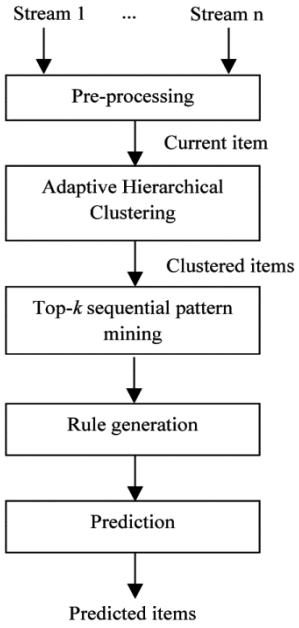

Fig. 2 — Block diagram of BFTKSPM system framework

A complete process of BFTKSPM algorithm with pseudo code is given in Algorithm 1.

Algorithm 1: Batch-Free Top-*K* Sequential Pattern Mining (BFTKSPM) algorithm

Input: Item list

Output: Top-*k* sequential patterns and predictions
Step 1: for each incoming item do
Step 2: Pre-process the item in the item list.
2.1: Remove the oldest item from the list.
2.2: Add the new pre-processed item to the list.
Step 3: Cluster the item using adaptive hierarchical clustering.
Step 4: Generate the top-*k* patterns from the clustered items.

Step 5: Print the top-*k* patterns.
Step 6: Determine the rules from the top-*k* patterns.
Step 7: Generate the predictions using the rules and the prediction parameter.
Step 8: Output the predictions.
Step 9: End for

Adaptive Hierarchical Clustering

The common patterns of major groups $p_1, p_2, \ldots, p_n$ are first mined in each local data stream. Then, the sequential patterns that are approximate to other sequences are summarized and effectively represent the sequences in a particular group. Let $DS_x = ds_1, ds_2, \ldots$ be the local data stream sequences and $dis(ds_i, ds_j)$ be the distance between the two sequences ds_i and ds_j in the range 0 to 1. DS_x can be divided into similarity groups $G_{x_1}, G_{x_2}, \ldots, G_{x_n}$ such that $\sum_{i \neq j} dis\left(ds_{ia}, ds_{jb}\right)$ is maximized while $\sum_{i=j} dis\left(ds_{ia}, ds_{jb}\right)$ is minimized where $ds_{ia} \in G_{x_i}$ and $ds_{jb} \in G_{x_j}$. This distance measure $dis(ds_i, ds_j)$ is used in BFTKSPM algorithm, where the similarity of groups is identified through clustering to group the patterns from local data stream sequences.

The Homogeneous Set (HS) can provide a model which is uniform to identify the similarity in the local patterns. For a given set of local patterns O, the homogeneous set has a range δ where the similarity between patterns p_i and p_j in the HS is not less than δ such that $p_i \in HS \wedge p_j \in HS \wedge sim(p_i, p_j) \geq \delta$ where $sim(p_i, p_j) = 1 - dis(p_i, p_j)$. Using function $sim(ds_i, ds_j) = 1 - dis(ds_i, ds_j)$, the similarity between the local data stream sequences can be identified.

Adaptive Hierarchical Clustering of *k* patterns contains *k* distinct clustering and is generally depicted as a dendrogram.[27] Let C_k denote the *k*-clusters and one of these clusters is again split into two to form (*k*+1) clustering denoted by C_{k+1} and so on. A binary tree is used to represent the overall structure in which the leaves are patterns and the internal nodes have exactly two children such that there are 2*k*-1 overall node. Each internal node is assigned with a unique split number between 1 and *k*-1 which satisfies the property that the split number of a parent is less than its children. Generally, a *k*-clustering is formed by removing the internal nodes numbered $1,2, \ldots, k-1$ such that each cluster consists of the resulting connected components. Let μ_i denote the mean of leaves (patterns) and AHC is represented by a binary tree with the patterns at the leaves where the internal nodes are ordered. Let $DS_1, DS_2, \ldots, DS_k$ be the clusters of C_k where the *k*-means cost function is given by

$$Cost(C_k) = \sum_{j=1}^{k} \sum_{x \in DS_j} \left|\left| x - \mu(DS_j) \right|\right|^2 \quad \ldots (2)$$

where, $\mu(DS_j)$ is the mean of the set DS. The adaptive hierarchical clustering presented in Algorithm 2 can be evaluated by combining the costs of all k intermediate clustering costs in a linear manner. The overall cost of the AHC is considered as

$$Overall\_cost_{AHC}(C_k) = \sum_{k=1}^{n} F_{aw_k}.Cost(C_k) \quad \ldots (3)$$

where, F_{aw_k} is generated via the fuzzy function which are non-negative weights. The default choice is to make all F_{aw_k} using a triangular function and it is defined by a lower limit c_1, an upper limit c_2, and a value c_3, where $c_1 < c_2 < c_3$. The weights are application specific where $aw_1 > aw_2 > aw_3 > \cdots > aw_k$.

$$F_{aw_k} = \mu_A(aw_k) = \begin{cases} 0, aw \leq c_1 \\ \frac{aw-a}{m-a}, a < aw \leq c_2 \\ \frac{b-aw}{b-m}, c_2 < aw < c_3 \\ 0, aw \geq c_3 \end{cases} \quad \ldots (4)$$

Algorithm 2: Adaptive hierarchical clustering

Input: Data stream
Output: Similar cluster groups
Step 1: Divide the data stream into local data streams where $DS_x = ds_1, ds_2, \ldots$
Step 2: Calculate the distance measure to identify the patterns between local data stream sequences using $dis(ds_i, ds_j)$.
Step 3: Approximate sequential patterns are discovered from the local data stream.
Step 4: A homogeneous set is formed to identify the similarity using the function $sim(ds_i, ds_j) = 1 - dis(ds_i, ds_j)$.
Step 5: Identify *k*-distinct clustering represented as C_k.
Step 6: Calculate the cost of each cluster using Eqs 2 and 3.
Step 7. Return patterns of similar cluster groups.

Generation of Top-*k* Sequential Patterns

The top-*k* sequential patterns are discovered from data streams and k is a user-defined threshold

parameter. Parameter k is a positive integer specified by user and can identify desired number of patterns to be found from streaming data. When minimum support threshold is low, several patterns are generated and when it is high, only few patterns are generated as output.

The BFTKSPM algorithm uses the maximum pattern length parameter set by user to limit the growth of patterns and the support of each pattern changes for every new incoming item and the time consumed to discover the patterns are constant. It does not depend on data stream and linearly scales with respect to incoming data. The proposed algorithm discovers the sequential patterns that are less than the maximum pattern length and these patterns are maintained in the tree structure. Patterns maintained in tree provide information regarding the patterns generated.

The proposed algorithm creates the patterns continuously depending on the flow of the incoming data stream. The items are analysed and the BFTKSPM creates the patterns where each new item is added as the last element. Suppose the item a is newly arriving item, the algorithm initiates this item where pattern length is one and in next iteration new item is added as a predecessor to this item a. This process is carried out in an iterative manner. For example, last item in the stream is c and then b as depicted in Fig. 1. Further, when the new item a arrives, it is processed and pattern a is created. Later, b, a is created, followed by the pattern c, b, a and so on and are added to the adapted T_0 tree. Algorithm 3 represents the generation of top-k sequential patterns by BFTKSPM algorithm.

Algorithm 3: Top-k sequential pattern generation

Input: S: Sequence database, I: Item list, curr-pat: current pattern, k: number of patterns

Output: Top-k sequential patterns with the new item
Step 1: Read the sequence database S and calculate minimum support for every item.
Step 2: Calculate length of item list and store it in C such that C=I.length().
Step 3: Create an array Top$k$Array to store the patterns.
Step 4: patterns.ClusterItem(I);
Step 5: while (curr-pat.length() ≤C) do
Step 6: curr-pat = I.get(temp – curr-pat.length()) + curr-pat;
Step 7: patterns.Add(curr-pat);
Step 8: Update_Tree(curr-pat);
Step 9: if (item_count % Pruning Count = = 0) then
Step 10: PruneTree()
Step 11: Store the patterns in Top$k$Array.
Step 12: return the top-k patterns.
Step 13: end if

Inverted T_0 Tree to Store Top-k Sequential Patterns

The nodes of T_0 tree represent the sequential pattern and the pattern can be found by following a predetermined path to the corresponding node. The occurrence time of the pattern is also stored in each node which is the timestamp. In the T_0 tree, the patterns are added as a prefix of its children and the BFTKSPM does not use the T_0 tree structure to store the patterns. Suppose, if the node n_1 is a child of another node n_2, then the node n_1 representing the pattern can be considered as prefix of the pattern specified by node n_2. However, patterns discovered by the proposed algorithm are added as postfixes where BFTKSPM creates pattern with the new item and append to the existing items. Hence, the proposed algorithm uses the inverted T_0 tree to store patterns instead of T_0 tree to reduce the huge computational overhead when the tree is updated with patterns. For every generated pattern, it searches a different path to insert new patterns into T_0 tree.

T_0 tree and inverted T_0 tree are represented in Fig. 3 for sequence a, b. In this inverted T_0 tree, each node represents the pattern with its corresponding path from the node considered to root following bottom up approach. The algorithm updates the tree with a single path instead of multiple paths and retrieval of patterns from the tree consumes less time. By using inverted T_0 tree, the algorithm identifies a single path to retrieve the patterns instead of determining multiple paths and updates the nodes. Occurrence of pattern a, b, f is represented as the rightmost path in Fig. 3. Inverted T_0 tree structure to update the patterns is shown in Fig. 3.

The BFTKSPM algorithm updates the tree with the newly generated patterns. In the process of T_0 tree updation in left side of Fig. 3, to update new item f, the algorithm uses three different paths, whereas in the right side of the inverted T_0 tree, algorithm uses a single path.

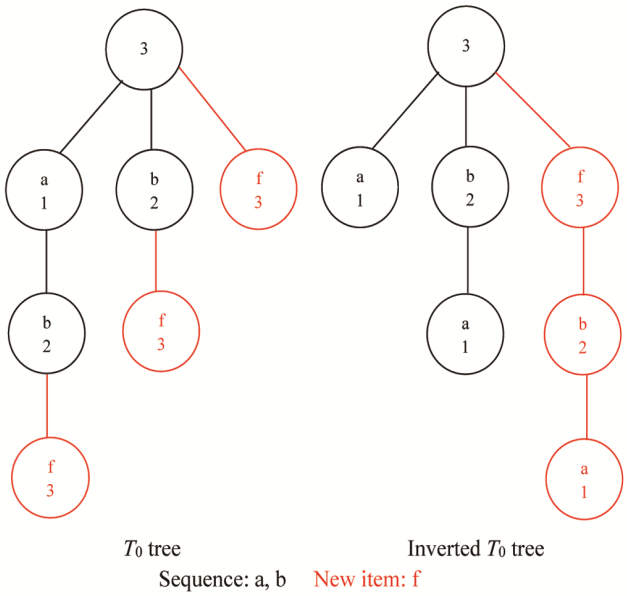

Fig. 3 — Inverted T_0 tree

Discarding Unpromising Patterns

The data stream usually contains infinite number of items and hence, the patterns stored in inverted T_0 tree are large. Pruning strategy[15] is used in BFTKSPM algorithm to discard the unpromising patterns eventually from the tree. This pruning strategy contains various parameters: *i) TS* represents the timestamp of the batch *ii) count* indicates the sequence count *iii) B* is number of batches passed *iv) batch_count* is total count of batches *v)* α and ϵ are two user-defined support thresholds to identify the promising patterns such that $0 \leq \alpha \leq \epsilon \leq supportthreshold$, *vi)* L is the length of the batch, and *vii)* δ is the pruning period. For these given parameters, based on the following condition, $count + (B - batch\_count) * (\lceil \alpha L \rceil - 1) \leq \epsilon BL$, which exists for a node considered, the sub tree rooted at this node can be pruned from the tree. The proposed algorithm tracks the number of items that are passed in the stream for every L items and the pruning process is repeated for every $L\delta$ items.

Rule Generation

The proposed algorithm generates top-*k* sequential patterns and these sequential patterns are split into left and right halves, which are antecedent and consequent of the rule. To generate the candidate rules, one item in the antecedent will be moved to the right, then two items will be moved, etc. and this process is iterated until a single item is available in the antecedent from which no further rules are generated. The confidence of these rules are then determined. A valid rule should satisfy minimum support and minimum confidence criteria. The algorithm prunes the search space by merging two valid rules that share the same prefix in the antecedent and the consequent of the rule and by determining subset of rule that does not have high confidence. The valid rules determined after pruning are sent as input to the prediction process.

Prediction of New Items

The valid rules generated are used for prediction which contains the frequent sequential patterns and its support. The support values are sent as input to predictor for generating new predictions. Initially, predictor searches frequent pattern list to identify rules that matches context followed by a search based on the most recent items that are available in the stream. The confidence of an association rule $A \rightarrow B$ is calculated using the Eq. 5.

$$conf(A \rightarrow B) = \frac{support(A,B)}{support(A)} \quad \ldots (5)$$

On finding the confidence of all rules that match with the context, items are predicted based on confidence measure and the next item to be predicted is the rule with the largest confidence value. Prediction of new items in data stream is presented in Algorithm 4.

Algorithm 4: Prediction of new items from the rules

Input: Item, top-*k* patterns, item_List,
Output: List of predicted items
Step 1: Discover the relevant rules from top-*k* sequential patterns
Step 2: Compute the confidence using Eq. 5 and filter the relevant rules.
Step 3: Compute precision using the Eq. 6.
Step 4: Compute recall using Eq. 7.
Step 5: Determine F1-score using Eq. 8.
Step 6: Return the predicted items from the item_List.

The parameter *m* is used for predicting *m* different items at the subsequent timestamp *t*. The parameter *span* is defined as time span to find the time taken to hold item for prediction. These two parameters are user-defined and suppose, one item has to be predicted in the next timestamp, the parameter setting corresponds to $m = 1$ and $span = 1$. If the parameter value is set as $m = 2$, which implies that there are two rules with confidence that are used for prediction such that the next item would be from any of these two. When $span = 3$ is set, which means that the prediction

is valid not only at time step *t*, *t + 1*, and *t + 2* but it is valid till time step *t + 3*. At time step *t*, if an item was predicted and if this item is again predicted during the *span* time steps, then this item will not be appended to the pending predictions list. This implies that each item prediction is done exactly once during the *span* time steps and further, this item will be predicted again later, ensuring number of predicted items is not higher than number of items available in the stream.

Results and Discussion

The performance of BFTKSPM algorithm is determined on four benchmark datasets REDD, Kosarak, MSNBC, and FIFA. It is compared against SS-BE[15], SS-LC[16], and BFSPMiner[12] approaches for its efficiency. Efficiency of BFTKSPM algorithm is determined using the metrics precision, recall, F1-score, memory usage, and execution time.

Dataset Information

The REDD[28] dataset contains information of power usage on electrical devices collected from several homes and high frequency voltage/current data of two of these homes is considered. Kosarak[29] is a large click-stream dataset collected from a Hungarian News Portal. Subset choices on a Hungarian news portal visited by a user in a given browsing session are deduplicated in this universal choice dataset.

MSNBC[29] dataset is a click-stream dataset which contains details about the pages visited by users of the site msnbc.com on September 28, 1999 for the whole day. A user's page visits are represented by each sequence during that 24-hour period in the dataset. FIFA[29] dataset collected from the FIFA World Cup 98 website is also a click-stream dataset. Player's data from FIFA 15 to FIFA 20 for the career mode is available in this dataset.

The reason for selecting these datasets is they contain features that are relevant to the problem with reasonable number of sequences. These are standard benchmark datasets and are widely used in sequential pattern mining problems.[11,12,19] They are taken from SPMF (Sequential Pattern Mining Framework) which are highly reliable. Details of datasets are provided in Table 1.

Table 1 — Dataset Information

Dataset	Number of sequences
REDD	45,924
Kosarak	396,000
MSNBC	846,000
FIFA	1,476,000

Experimental Setup

The proposed BFTKSPM algorithm is implemented in Java and the experiments were performed on a Windows 10 OS, Intel(R) Core(TM) i3-4130 CPU @ 3.40 GHz with x64-based processor and 4.00 GB RAM. The value of support threshold is set as 1%. The user defined thresholds ϵ and α to discover the promising patterns are set to 0.00999 and 0.00995, respectively. The value of pruning period δ is assigned to 5 with the initial batch size of 300.

Performance Metrics

Efficiency of BFTKSPM algorithm is evaluated by the metrics precision, recall and F1-score.

Precision: Precision is ratio of correct predictions to number of items predicted.

$$Precision = \frac{|CorrectPredictions|}{|Predictions|} \quad \ldots (6)$$

Recall: Recall is ratio of correct predictions to number of items processed.

$$Recall = \frac{|CorrectPredictions|}{|items|} \quad \ldots (7)$$

F1-score: F1-score determines accuracy of the predictions carried out where both precision and recall measures are considered.

$$F1\_score = 2 * \frac{Precision * Recall}{Precision + Recall} \quad \ldots (8)$$

The performance results comparison for three metrics for the REDD dataset is depicted in Fig. 4(a). When the *span* parameter value is increased, the values of all the three evaluation metrics are simultaneously increased, while the predicted items count are same and stays valid for a long period of time. If *span* parameter value increases, all the performance measures increases that allows for accurate prediction. It is obvious from Fig. 4(a), the proposed algorithm achieved a precision of 84% for the *span* value 6. This increase in performance measures are due to pruning strategy adopted in the algorithm. The performance results comparison for three different metrics for different *m* parameter value when *span = 3* is displayed in Fig. 4(b). Increase in *m* parameter results in increased recall and generates more number of valid rules that allow for improved prediction. The increase in recall indicates that too many predictions of items were carried out at subsequent time steps and they remain valid for a

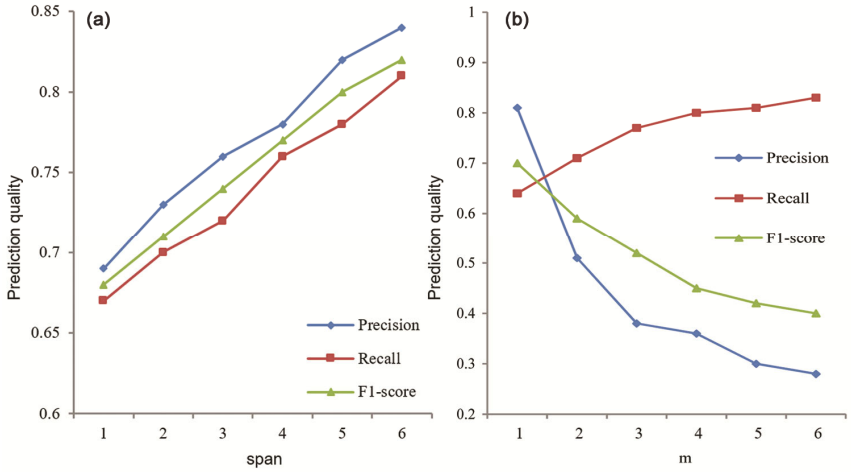

Fig. 4 — Performance measure comparison for REDD dataset: (a) Prediction quality vs. *span* (*m* = 1 for REDD dataset), & (b) Prediction quality vs. *m* (*span* = 3 for REDD dataset)

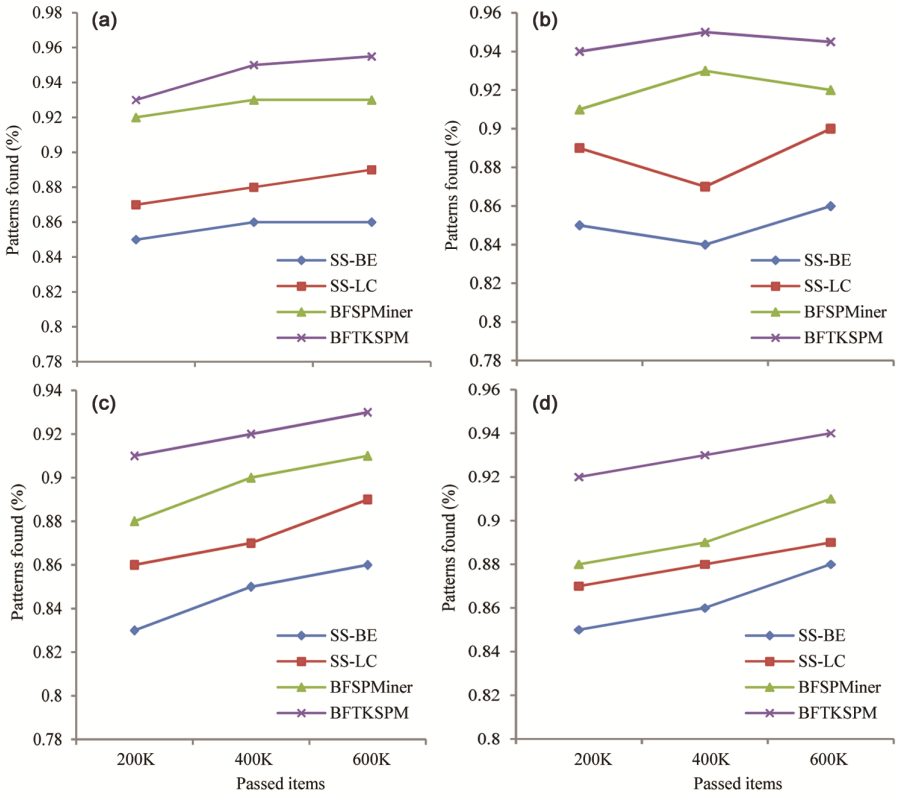

Fig. 5 — Relative number of patterns found by the algorithms for different datasets: (a) Kosarak, (b) MSNBC, (c) FIFA, & (d) REDD

longer period of time. The proposed algorithm computes recall value of 83% when the *m* value is 6.

Percentage of Patterns Found by Algorithms for Different Datasets

The percentages of patterns determined by algorithms for various datasets are shown in Figs. 5(a–d) respectively. Increase in percentage of patterns found by BFTKSPM algorithm when compared with the other methods for different datasets over 200K, 400K and 600K passed items is represented in the Table 2. The algorithms considered for comparison SS-BE, SS-LC and BFSPMiner could identify the percentage of patterns between 82% and 93%, while the proposed algorithm BFTKSPM discovered the percentage of patterns between 91% and 95.5%. The pruning technique adopted by the BFTKSPM algorithm

contributes to the increase in the percentage of patterns significantly by discarding the unpromising patterns and retains only the important patterns.

The SS-BE, SS-LC and BFSPMiner algorithms could find the percentage of patterns of 86%, 89%, and 93% respectively whereas BFTKSPM algorithm found the number of patterns of 95.5% for 600K passed items with respect to Kosarak dataset as shown in Fig. 5(a). BFTKSPM algorithm finds all the sequential patterns for the given maximum pattern length value of 10.

The percentage of patterns found by all the algorithms over passed items for different datasets are shown in the Figs 5(b–d). Considering the MSNBC dataset in Fig. 5(b), the SS-BE, SS-LC and BFSPMiner algorithms found the percentage of sequential patterns of about 86%, 90% and 92%, respectively, whereas the BFTKSPM algorithm could find the percentage of patterns of about 94.5% with respect to 600K passed items.

The SS-BE, SS-LC and BFSPMiner algorithms could find percentage of sequential patterns between 86% and 91%, while the proposed algorithm found the sequential patterns of about 93% for 600K passed items with respect to the FIFA dataset in Fig. 5(c). Considering REDD dataset in Fig. 5(d), the percentage of sequential patterns determined by the SS-BE, SS-LC and BFSPMiner algorithms are between 88% and 91%, whereas the BFTKSPM algorithm discovered 94% of sequential patterns for 600K passed items. This is due to pruning technique adopted in proposed algorithm and for all the datasets, the significant patterns are generated.

Accuracy Comparison

The percentage of increase in performance measures of the proposed algorithm BFTKSPM with respect to SS-BE, SS-LC and BFSPMiner for the REDD dataset for *span* value 6 is presented in Table 3. The increase in *span* value reflects the prediction quality, where the predicted items are retained for a longer time.

Accuracy comparison results of the evaluation methods with respect to *span* parameter for REDD dataset is presented in Fig. 6. The evaluation results show the prediction quality and reflect the changes in precision, recall and F1-score values by varying *span*.

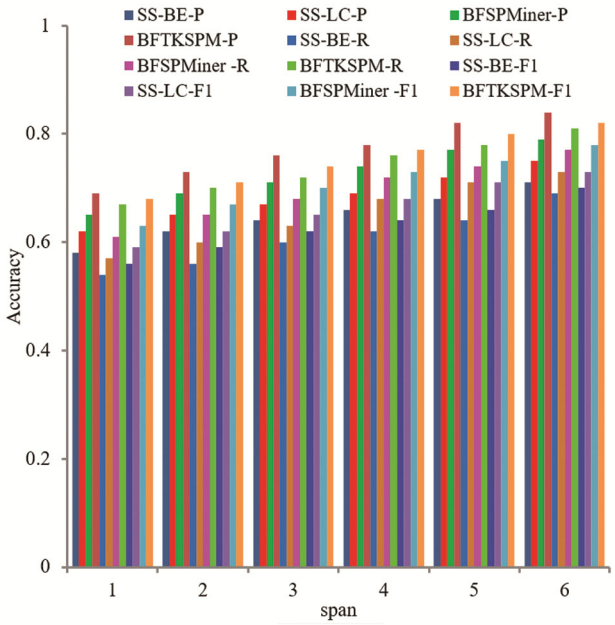

Fig. 6 — Accuracy comparison of the algorithms for the REDD dataset ($m = 3$)

Table 2 — Increase in percentage of patterns identified by BFTKSPM Algorithm

Number of passed items	Increase (%) of patterns found			Datasets
	SS-BE	SS-LC	BFSPMiner	
200K	8.6	6.4	1.0	Kosarak
	9.5	5.3	3.1	MSNBC
	8.7	5.4	3.2	FIFA
	9.8	6.5	2.1	REDD
400K	9.4	7.3	2.1	Kosarak
	11.5	8.4	2.1	MSNBC
	7.6	5.4	2.1	FIFA
	9.6	7.5	2.1	REDD
600K	9.9	6.8	2.6	Kosarak
	8.9	4.7	2.6	MSNBC
	7.5	4.3	2.1	FIFA
	9.5	6.3	2.1	REDD

Table 4 — Percentage of increase in performance measures of BFTKSPM algorithm for $m = 6$

Dataset	Performance measures	Increase (%) of performance measures		
		SS-BE	SS-LC	BFSPMiner
MSNBC	Precision	22..6	17.8	10.7
	Recall	18.4	13.1	2.6
	F1-score	20.6	16.2	7.5

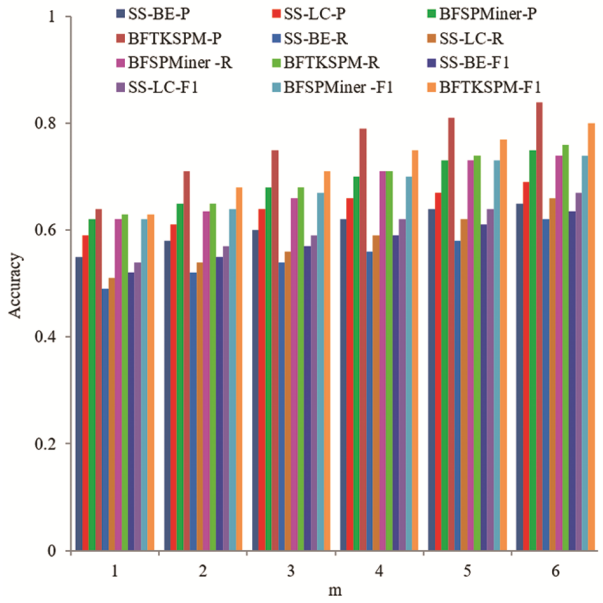

Fig. 7 — Accuracy comparison of the algorithms on the MSNBC dataset (*span* = 3)

BFTKSPM algorithm achieves precision, recall and F1-score values of 84%, 81% and 82% for *span* = 6. Hence, increase in *span* allows recall, precision and F1-score values to rise, which indicates that accurate predictions is performed by the algorithm.

The percentage of increase in performance measures of BFTKSPM algorithm compared with other algorithms for MSNBC dataset for *m* value 6 is represented in Table 4. From table, it is evident that, increase in *m* value increases the number of rules generated and hence more number of items are predicted by the predictor thereby increasing both prediction quality and performance measures.

The comparison of accuracy parameter of the evaluation methods for the prediction parameter *m* on the MSNBC dataset is presented in Fig. 7. Prediction quality of BFTKSPM algorithm with SS-BE, SS-LC and BFSPMiner algorithms for varying *m* is shown in the figure. The BFTKSPM algorithm achieved 20.6%, 16.2% and 7.5% higher F1-score when compared with SS-BE, SS-LC and BFSPMiner algorithms, respectively, for *m*=6 and *span*=3. The BFTKSPM algorithm has higher precision, recall and F1-score values when compared with existing algorithms.

Execution Time Comparison vs. Support Threshold (%) for Different Datasets

The execution time is compared with algorithms for varying support threshold for different datasets are shown in Figs. 8(a–d). For higher support threshold, faster processing of items occurs and less number of patterns is generated. Hence, fewer frequent patterns are used as rules for the predictor which results in less execution time. Percentage of decrease in execution time of BFTKSPM when compared with other techniques for different datasets for 50% of support threshold is represented in Table 5. From table, it is clear that BFTKSPM outperforms the other methods by consuming less execution time for 50% of support threshold with respect to different datasets. Execution time consumed by BFTKSPM algorithm in generating valid patterns is less because of the high availability of data by adaptive hierarchical clustering for generating the patterns and pruning strategy. BFTKSPM technique has attained lesser execution time on all the four datasets with respect to the existing approaches.

Running time comparison of algorithms on different datasets is represented in Table 6. BFTKSPM algorithm takes lesser execution time over other algorithms is indicated in the table. This is due to high availability of the data from the clusters for mining of patterns and the pruning strategy implemented in the algorithm to discover the significant patterns.

The percentage of decrease in running time of BFTKSPM for different datasets is represented in Table 7.

Memory Usage of Algorithms for Different Datasets

The memory consumed by the algorithms for various datasets is shown in Table 8. The proposed algorithm BFTKSPM consumed less memory space for different datasets is due to inverted T_0 tree used to store the patterns. The unpromising items are removed from the tree due to the adoption of the pruning technique.

The percentage of decrease in memory usage of BFTKSPM in comparison with other methods for different datasets is represented in Table 9.

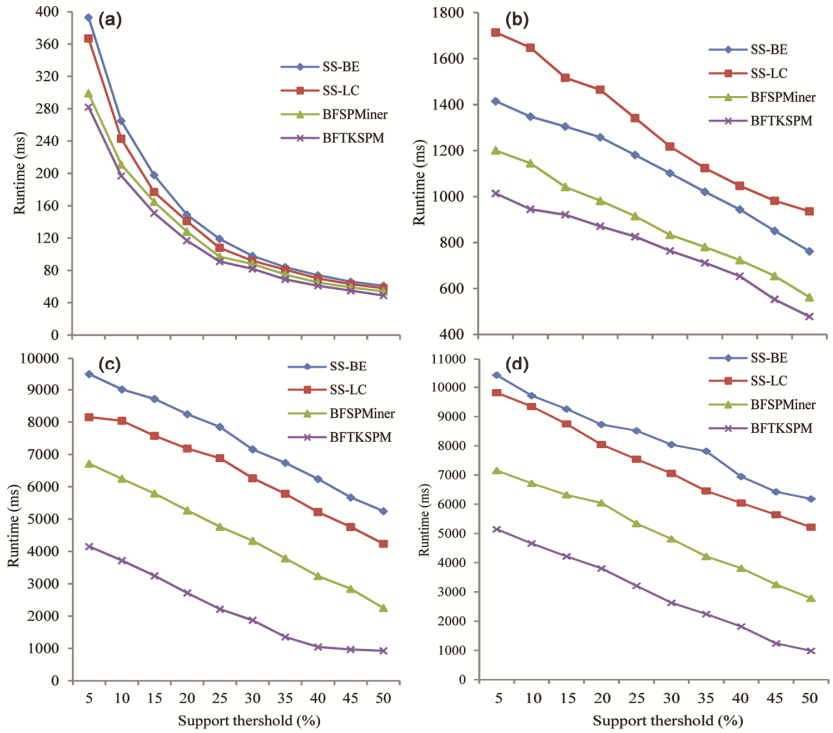

Fig. 8 — Runtime vs. support threshold (%) for algorithms on different datasets: (a) REDD, (b) Kosarak, (c) MSNBC, & (d) FIFA

Table 5 — Percentage of decrease in execution time of BFTKSPM algorithm for 50% of support threshold

Dataset	Decrease (%) of execution time (ms)		
	SS-BE	SS-LC	BFSPMiner
Kosarak	48.9	37.2	14.9
MSNBC	82.3	78.1	58.8
REDD	19.6	15.5	9.2
FIFA	84	81	64.5

Table 6 — Running time comparison of algorithms for different datasets

Dataset	Running time (ms)			
	SS-BE	SS-LC	BFSPMiner	BFTKSPM
Kosarak	1,654	1,925	27,859	1,258
MSNBC	15,625	14,582	4,123	2,849
REDD	515	689	6,915	457
FIFA	15,242	36,514	31,957	7,183

Table 7 — Percentage of decrease in running time of BFTKSPM algorithm for different datasets

Dataset	Decrease (%) of running time (ms)		
	SS-BE	SS-LC	BFSPMiner
Kosarak	23.9	34.6	95
MSNBC	81.7	80.4	30.8
REDD	11.2	33.6	93.3
FIFA	52.8	80.3	77.5

Table 8 — Memory usage comparison of algorithms for different datasets

Dataset	Memory usage (MB)			
	SS-BE	SS-LC	BFSPMiner	BFTKSPM
Kosarak	481	398	306	265
MSNBC	890	760	635	581
REDD	82	47	36	30
FIFA	1,872	1,256	1,078	987

Table 9 — Percentage of decrease in memory usage of BFTKSPM algorithm for different datasets

Dataset	Decrease (%) of memory usage (MB)		
	SS-BE	SS-LC	BFSPMiner
Kosarak	44.9	33.4	13.3
MSNBC	34.7	23.5	8.5
REDD	63.4	36.1	16.6
FIFA	47.2	21.4	8.4

Conclusions

A Batch-Free Top-*K* Sequential Pattern Mining algorithm was proposed to mine approximate sequential patterns from data streams. Sequential patterns that are similar to several sequences in group are clustered using Adaptive Hierarchical Clustering, which effectively denotes the sequences that belong to a specific group. The BFTKSPM algorithm mines the top-*k* sequential patterns that are smaller than the maximum pattern length from each cluster. The patterns discovered are stored in the inverted T_0 tree and each node in tree represents pattern, which is found by adopting a path from corresponding node to root following a bottom up approach. In inverted T_0 tree, the leaves specify the patterns with an ordering of the internal nodes and from this tree, the top-*k* sequential patterns are retrieved. From top-*k* patterns, the rules are generated and the upcoming items are predicted by the predictor for those rules which have high confidence. The proposed algorithm BFTKSPM was compared with the existing approaches for its efficiency and evaluated by the metrics precision, recall, F1-score, execution time, and memory. In mining sequential patterns from data streams, batch-free technique can be extended by introducing an adaptive threshold for maximum pattern length, which increases the prediction quality of datasets containing huge number of significant patterns with undefined pattern length. The prediction method can be further improved by using some statistical techniques and the batch-free stream clustering can be investigated for temporal pattern mining.

Acknowledgement

K. Poongodi received Anna Centenary Research Fellowship (ACRF 2016-2018) from Centre for Research, Anna University, Chennai to carry out this research work.

References

1 Gan W, Lin J C W, Fournier-Viger P, Chao H C & Yu P S, A survey of parallel sequential pattern mining, *ACM Trans Knowl Disc Data*, **13(3)** (2019) 1–34, https://doi.org/10.1145/3314107.

2 Wuu L C, Hung C H & Chen S F, Building intrusion pattern miner for snort network intrusion detection system, *J Syst Softw*, **80(10)** (2007) 1699−1715, https://doi.org/10.1016/j.jss.2006.12.546.

3 Huynh H M, Nguyen L T, Vo B, Yun U, Oplatková Z K & Hong T P, Efficient algorithms for mining clickstream patterns using pseudo-IDLists, *Fut Gen Comp Syst*, **107** (2020) 18–30, https://doi.org/10.1016/j.future.2020.01.034.

4 Ayres J, Flannick J, Gehrke J & Yiu T, Sequential pattern mining using a bitmap representation, in *ACM SIGKDD 8th Int Conf Knowl Disc Data Mining* (Association of Computing Machinery, New York) 2002, 429–435, https://doi.org/10.1145/775047.775109.

5 Aseervatham S, Osmani A & Viennet E, bitSPADE: A lattice-based sequential pattern mining algorithm using bitmap representation, in *IEEE 6thInt Conf Data Mining* (IEEE) 2006, 792–797, doi: 10.1109/ICDM.2006.28.

6 Pei J, Han J, Mortazavi-Asl B, Wang J, Pinto J H, Chen Q & Umeshwar D, Mining sequential patterns by pattern-growth: The prefixspan approach, *IEEE Trans Knowl Data Eng*, **16(11)** (2004) 1424–1440, doi: 10.1109/TKDE.2004.77.

7 Mabroukeh N R & Ezeife C I, A taxonomy of sequential pattern mining algorithms, *ACM Comp Surveys*, **43(1)** (2010) 1–41, https://doi.org/10.1145/1824795.1824798.

8 Agrawal R & Srikant R, Mining sequential patterns, in *IEEE 11th Int Conf Data Eng* (IEEE) 1995, 3–14, doi: 10.1109/ICDE.1995.380415.

9 Zaki M J, SPADE: An efficient algorithm for mining frequent sequences, *Machine Learn*, **42** (2001) 31–60, https://doi.org/10.1023/A:1007652502315.

10 Xu C, Chen Y & Bie R, Sequential pattern mining in data streams using the weighted sliding window model, in *IEEE 15th Int Conf Parallel and Dis Syst* (IEEE) 2009, 886–890, doi: 10.1109/ICPADS.2009.64.

11 Zhou C, Cule B & Goethals B, A pattern based predictor for event streams, *Expert Syst Appl*, **42(23)** (2015) 9294–9306, https://doi.org/10.1016/j.eswa.2015.08.021.

12 Hassani M, Töws D, Cuzzocrea A & Seidl T, BFSPMiner: An effective and efficient batch-free algorithm for mining sequential patterns over data streams, *Int J Data Sci and Analytics*, **8(3)** (2019) 223–239, https://doi.org/10.1007/s41060-017-0084-8.

13 Soliman A F, Ebrahim G A & Mohammed H K, SPEDS: A framework for mining sequential patterns in evolving data streams, in *IEEE Pacific Rim Conf Commun, Comp and Signal Processing* (IEEE) 2011, 464–469, doi: 10.1109/PACRIM.2011.6032938.

14 Vinceslas L, Symphor J E, Mancheron A & Poncelet P, SPAMS: A novel incremental approach for sequential pattern mining in data streams, *Adv Knowl Disc Manag*, **292** (2010) 201–216, https://doi.org/10.1007/978-3-642-00580-012.

15 Mendes L F, Ding B & Han J, Stream sequential pattern mining with precise error bounds, in *IEEE 8thInt Conf*

Data Mining (IEEE) 2008, 941–946, doi: 10.1109/ICDM.2008.154.

16 Koper A & Nguyen H S, Sequential pattern mining from stream data, in *7th Int Conf Adv Data Mining and Appl* (Springer, Berlin, Heidelberg) 2011, 278–291, https://doi.org/10.1007/978-3-642-25856-5_21.

17 Hijawi H & Saheb M, Sequence pattern mining in data streams, *Comput Inf Sci*, **8(3)** (2015) 64–70.

18 Ouyang W, Mining rare sequential patterns in data streams with a sliding window, in *IEEE 3rd Int Conf Syst and Informatics* (IEEE) 2016, 1023–1027, doi: 10.1109/ICSAI.2016.7811101.

19 Zihayat M, Chen Y & An A, Memory-adaptive high utility sequential pattern mining over data streams, *Mach Learn*, **106** (2017) 799–836, https://doi.org/10.1007/s10994-016-5617-1.

20 Zihayat M, Wu C W, An A, Tseng V S & Lin C, Efficiently mining high utility sequential patterns in static and streaming data, *Intell Data Anal*, **21(S1)** (2017) S103–S135, doi: 10.3233/IDA-170874.

21 Kim H, Yun U, Baek Y, Kim H, Nam H, Lin J C W & Fournier-Viger P, Damped sliding based utility oriented pattern mining over stream data, *Knowl Based Syst*, **213** (2021) 106653, https://doi.org/10.1016/j.knosys.2020.106653.

22 Baek Y, Yun U, Kim H, Nam H, Lee G, Yoon E, Vo B & Lin J C W, Erasable pattern mining based on tree structures with damped window over data streams, *Eng Appl Art Intell*, **94** (2020) 103735, https://doi.org/10.1016/j.engappai.2020.103735.

23 Kawabata K, Matsubara Y & Sakurai Y, Automatic sequential pattern mining in data streams, in *ACM 28th Int Conf Inf Knowl Manag* (Association of Computing Machinery, New York) 2019, 1733–1742, https://doi.org/10.1145/3357384.3358002.

24 Patnaik D, Laxman S, Chandramouli B & Ramakrishnan N, Efficient episode mining of dynamic event streams, in *IEEE 12th Int Conf Data Mining* (IEEE) 2012, 605–614, doi: 10.1109/ICDM.2012.84.

25 Wang J, Fang S, Liu C, Qin J, Li X & Shi Z, Top-k closed co-occurrence patterns mining with differential privacy over multiple streams, *Future Gen Comp Syst*, **111** (2020) 339–351, https://doi.org/10.1016/j.future.2020.04.049.

26 Laur P A, Nock R, Symphor J E & Poncelet P, Mining evolving data streams for frequent patterns, *Pat Recognition*, **40(2)** (2007) 492–503, https://doi.org/10.1016/j.patcog.2006.03.006.

27 Murtagh F & Contreras P, Algorithms for hierarchical clustering: an overview, *WIREs Data Mining Knowl Disc*, **2(1)** (2012) 86–97, https://doi.org/10.1002/widm.53.

28 Kolter J Z & Johnson M J, REDD: A public data set for energy disaggregation research, in *SIGKDD Workshop on Data Mining Appl in Sustainability* **(**San Diego, CA) 2011, 59–62.

29 Kosarak, MSNBC, FIFA datasets available at http://www.philippe-fournier-viger.com/spmf/index.php?link=datasets.php, (7 September 2023).