A novel approach for mining closed clickstream patterns

A Novel Approach for Mining Closed Clickstream Patterns

Bao Huynh[a], Loan T. T. Nguyen[b,c], Huy M. Huynh[d], Adrianna Kozierkiewicz[e], Unil Yunf, Zuzana K. Oplatková[d], and Bay Vo[a]

[a]*Faculty of Information Technology, Ho Chi Minh City University of Technology (HUTECH), Ho Chi Minh City, Vietnam;*

[b]*School of Computer Science and Engineering, International University, Ho Chi Minh City, Vietnam;*

[c]*Vietnam National University, Ho Chi Minh City, Vietnam;*

[d]*Faculty of Applied Informatics, Tomas Bata University in Zlm, Zlm, Czech Republic;*

[e]*Faculty of Computer Science and Management, Wroclaw University of Science and Technology, Wroclaw, Poland; department of Computer Engineering, Sejong University, Seoul, Republic of Korea*

CONTACT: Loan T. T. Nguyen nttloan@hcmiu.edu.vn School of Computer Science and Engineering, International University, Ho Chi Minh City, Vietnam.

KEYWORDS: C-List, clickstream pattern mining, closed pattern, SPPC-tree

ABSTRACT

Closed sequential pattern (CSP) mining is an optimization technique in sequential pattern mining because they produce more compact representations. Additionally, the runtime and memory usage required for mining CSPs is much lower than the sequential pattern mining. This task has fascinated numerous researchers. In this study, we propose a novel approach for closed clickstream pattern mining using C-List (CCPC) data structure. Closed clickstream pattern mining is a more specific task of CSP mining that has been lacking in research investment; nevertheless, it has promising applications in various fields. CCPC consists of two key steps: It initially builds the SPPC-tree and the C-List for each frequent 1-pattern and then determines all frequently closed clickstream 1-patterns; next, it constructs the C-List for each frequent k-pattern and mines the remaining frequently closed k-patterns. The proposed method is optimized by modifying the SPPC-tree structure and a new property is added into each node element in both the SPPC-tree and C-Lists to quickly prune nonclosed clickstream. Experimental results conducted on several datasets show that the proposed method is better than the previous techniques and improves the runtime and memory usage in most cases, especially when using low minimum support thresholds on the huge databases.

Introduction

The explosion of big data technologies has led to a large volume of user's data that is archived. Analyzing the user's data helps the manager or the system to predict the user's interests and future actions. For this purpose, sequential pattern mining is one of the essential methods. It has a broad field in data science and applications in many sectors, such as shop analysis (Valle, Ruz, and Morriás **2018**; Moodley et al. **2019**; Astrova, Koschel, and Lee **2020**), web-based analysis (Danilowicz et al. **2000**; Nguyen **2000**, **2002**; Nguyen and Sobecki **2003**; Hagen and Stein **2018**; Prakash and Jaya **2020**), prediction (Fabra, Alvarez, and Ezpeleta **2020**; Soui et al. **2020**; Yan et al. **2021**), and bioinformatics analysis (Asite and Aleksejeva **2019**; Aberra et al. **2020**; Shihab, Dawood, and Kashmar **2020**). Its main goal is to identify recurrent patterns in any kind of data: images, transactions, sequences of figures. It can detect specific medical issues, customer habits, or characteristic user actions.

Sequential pattern mining can generate an exponential huge of patterns so it takes the high computational cost on the database that contains long sequences or on a huge database. Discarding useless or weak patterns by compressing the mined patterns is necessary in order to reduce the runtime overhead and increase speed. Maximal frequent pattern mining, frequent closed pattern mining, and top-k frequent pattern mining (Han et al. **2007**) are the nonredundant pattern mining methods that are often used. We can achieve a better compression runtime with the top-k frequent pattern methods and the maximal frequent pattern. However, there are chances that information may not be intact for both kinds of methods. Therefore, closed sequential pattern (CSP) mining is a better solution to mine nonredundant patterns. If a pattern does not have a superpattern with the same support, the pattern is considered as a closed pattern.

Although many algorithms have been proposed recently, such as BIDE (BI-Directional Extension) (Wang and Han **2004**), CloSpan (Closed Sequential pattern mining) (Yan, Han, and Afshar **2003**), ClaSP (Closed Sequential Patterns algorithm) (Gomariz et al. **2003**), CM-ClaSP (Cooccurrence MAP Closed Sequential Patterns algorithm) (Fournier-Viger et al. **2014**), CloFAST (Closed FAST sequence mining algorithm based on sparse id-lists) (Fumarola et al. **2016**), CloFS-DBV (Closed Frequent Sequences use Dynamic Bit Vectors) (Tran, Le, and Vo **2015**), pDBV-FCSP (parallel Dynamic Bit Vectors Frequent Closed Sequential Patterns) (Huynh, Vo, and Snasel **2017**), FCloSM (Frequent Closed Sequence Mining) (Le et al. **2017**), TKCS (Top-K Closed Sequences) (Pham et al. 2020), and NetNCSP (Nettree for Nonoverlapping Closed Sequential Pattern) (Wu et al. 2020), but their perform is not good when execution on huge databases.

In this study, we propose a novel approach for closed clickstream pattern mining using C-List (CCPC), a special variety of CSP mining. CCPC is a hybrid method relying on C-List, an extension of the B-List data structure (Bui et al. **2018**). The experimental results on many databases have exhibited that CCPC was more effective than the recent state-of-the-art algorithms, such as FCloSM (Le et al. **2017**), CloFAST (Fumarola et al. **2016**), CM-ClaSP (Fournier-Viger et al. **2014**), and CloSpan (Yan, Han, and Afshar **2003**), concerning runtime, particularly on a huge database with very small minimum support thresholds. Our contribution is briefly as follows:

- Propose a C-List data structure for efficient mining closed clickstream patterns.
- Propose the CCPC algorithm for mining closed clickstream patterns on various databases.
- Early perform closure checking mechanism to discarding redundant or nonclosed patterns based on C-List.
- Evaluate the proposed algorithm through many real-life test databases.

The study is structured as follows. In the section "Related Works," we describe the foundational concepts and the problem. The section "Preliminaries and Problem Definition" elaborates on some

related problems. The C-List structure and an effective CCPC algorithm for mining clickstream patterns are developed in the section “Proposed Algorithms”. The proposed CCPC algorithm is compared to other related methods in the section “Experimental Evaluation”. The conclusions and future development trends in this area are presented in the last section.

Related Works

Closed sequential pattern mining is a promising solution in sequential pattern mining, a core task of data mining, with many different applications in wide sectors. Instead of mining a set of completed patterns, closed pattern mining identifies a set of compact patterns, from which the information can fully be extracted. Especially, with the huge databases or low minimum support thresholds, the CSP is an efficient method. Several proposed approaches for CSP have existed, such as CloSpan (Yan, Han, and Afshar **2003**), BIDE (Wang and Han **2004**), ClaSP (Gomariz et al. **2003**), CM-ClaSP (Fournier-Viger et al. **2014**), CloFAST (Fumarola et al. **2016**), CloFS-DBV (Tran, Le, and Vo **2015**), pDBV-FCSP (Huynh, Vo, and Smsel **2017**), FCloSM (Le et al. **2017**), TKCS (Pham et al. **2020**), and NetNCSP (Wu et al. **2020**), but their performance on huge databases is still lacking. CloSpan consumes high memory usage and large search space because it used candidate maintenance-and-test paradigm and the equivalence classes for the projected pattern. A strict depth-first search order is used in BIDE to produce the CSPs. Additionally, any historical frequent CSP is not kept to check for a new pattern’s closure. This approach uses a pseudoprojection technique to reduce storage space; however, it is inefficient because it has to iterate the database multiple times for each prefix. ClaSP uses a vertical database scheme and a heuristic to eliminate nonclosed patterns. However, the technique has to maintain a set of candidates to perform pattern closure checking and discard the nonclosed patterns. This process has a more memory footprint and suffers the problem of the candidate explosion. CM-ClaSP uses the data structure named CMAP to store the necessary information of pattern with one database scan and based on the CMAP structure to pruning nonclosed patterns. Although this method is cut down the search space and candidates, it still spends much time to assess several candidates that do not exist. Dynamic bit vector (DBV) structure and a vertical data scheme are exploited in CloFS-DBV to quickly determine the support value of candidates based on bits. However, this method also generates several redundant patterns in the mining process. CloFAST uses sparse id-lists and vertical id-lists for mining closed frequent sequences. First, it finds all closed frequent patterns 1-sequence. Then, new k-sequences are produced by directly working on the sequences, without mining additional frequent itemsets. pDBV-FCSP is proposed a different approach for mining closed patterns, and it also uses DBV structure to find all CSPs by applying the multicore architecture to improve the performance cost. FCloSM uses CMAP (Fournier-Viger et al. **2014**) (cooccurrence map) which stores coexisting information to early eliminate child branches that have no frequent patterns. The algorithm also introduced new pruning terms called extended premature removal (3E) and an early pruning technique called EPCLO and LPCLO to eliminate nonclosed pattern at the next two levels. NetNCSP is based on the Nettree structure for mining Nonoverlapping Closed Sequential Pattern. To compute the nonoverlapping support count of a pattern on the Nettree and envisage the occurrence and proximity of the patterns before the production of the candidate patterns, the algorithm adopts a backtracking strategy.

TKCS is suggested to mine top-k closed sequence patterns using a vertical bitmap and adopting some useful strategies. The TKCS algorithm produces candidates by choosing a sequential pattern that has the highest support value and customizing the sequential pattern that has the minimum support value in the list of top-k.

Besides, some of the methods related to the sequential pattern were proposed recently to reduce the storage space and performance cost; for example, ISP-IC (Inter-Sequence Pattern with Item Constraint mining) (Le et al. **2018**) uses the DBV structure for mining intersequence patterns with item constraints, MCM-SPADE (Multiple threads CM-SPADE) (Huynh et al. **2018**) is used CMAP data structure to store the necessary information on the item and applying multiple threads technique for SPM with a very large database, CM-WSPADE (Huynh, Nguyen, Vo, Nguyen, et al. **2020**) is an extended version of CM-SPADE (Fournier-Viger et al. **2014**) algorithm, for mining frequent weighted clickstream patterns based on WIBList data structure and WCMAP (Weighted Cooccurrence MAP) with a pruning heuristic that is cohesive with the average weight, CUP (Clickstream pattern mining Using Pseudo-IDList) (Huynh, Nguyen, Vo, Yun, et al. **2020**) uses pseudo-IDList data structure for clickstream pattern mining and reduces candidates by using DUB (Dynamic intersection Upper Bound constraint), a pruning heuristic.

However, the computational cost of these techniques is large because of the vast search space. Therefore, it remains a significant challenge, especially for huge sequence databases due to the high cardinality of the events and the long sequences.

Preliminaries and Problem Definition

Let E = $\{e_1, e_2, ..., e_m\}$ be a set of m unique events in the same category. A subset of L = $\{u_1, u_2, ..., u_k\}$ is called an itemset, where $U_i \in E, i \in [1, k]$). A clickstream is a list of ordered itemset and denoted by $C = < c_1, c_2,..., c_n >$, where $c_i \in E, i \in [1, n]$ is an event. The length of the clickstream is the number of events it contains. In the other words, a clickstream with length k has k events and is denoted by k-clickstream. For example, a clickstream C = <a, c, b, a, b > is a 5-clickstream with three distinct events {a, b, c}, where event a exists in the position 1 and 4, and similarly, event b appears at the position 3 and 5 in clickstream C.

A clickstream $C_\alpha =< \alpha_1, \alpha_2,..., \alpha_u >$ is a subclickstream of another clickstream $C_\beta =< \beta_1, \beta_2, —, \beta_v >$, denoted by C_α C C_β, if there exist integers $1 \leq i_1 < i_2 < ...<i_u \leq v$ that $\alpha_k = \beta_{ik}, \forall k \in [1,u]$. C_β is also called a superclickstream of C_α.

A user clickstream is defined as the sequence of actions taken by the user through a Web site, and it consists of a series of ordered events triggered by user interactions.

A clickstream database, denoted by CDB, is a set of clickstream sequences. Each clickstream sequence is paired with a unique identification cid, if a clickstream sequence is a subclickstream of at least one or more user clickstreams in CDB, it is a clickstream pattern.

ε(P), which denotes the support count of a clickstream P, is the number of user clickstreams in CDB that containing clickstream of P, that is, $\varepsilon(P) = |\{C_i \in CDB \text{ j } P \text{ C } C_i\}|$.

Table 1 is an illustration of a clickstream database, the user clickstream (b, e, f, c, f, b) has cid = 200, and (a, c, b) is a clickstream pattern.

A clickstream pattern P is frequent if and only if $\varepsilon(P) \geq \delta$, where δ is the minimum support value defined by users. If P is frequent and does not have any superpattern with the same support, P is a closed clickstream pattern, that is, there does not exist Q such that $P \subset Q$ and ε(P) = ε(Q). The task of the closed clickstream pattern mining problem is uncovering all closed clickstream patterns in CDB.

Table 1. A clickstream sequence database.

Clickstream id	User clickstream
100	*a, c, b, a, b*
200	*b, e, f, c, f, b*
300	*a, c, d, f, a, a*
400	*a, c, b, d*

For example, with the clickstream database in **Table 1** and threshold δ = 0.3%, the number of clickstream patterns is 17 while the number of closed clickstream patterns is only 10. When we decrease the threshold δ = 0.1%, the number of clickstream sequential patterns explodes to 116 patterns, but the number of closed clickstream patterns is only 27. So, mining closed clickstream patterns is a good model because it saving much computation cost.

Proposed Algorithms

The original B-List (Bui et al. **2018**) does not support a way to optimize performance for closed clickstream pattern mining. It can use a normal way to determine closed clickstream patterns by checking if any pattern has a superpattern or a subpattern in the frequent pattern set. If it is either a superpattern or a subpattern, then the pattern is not closed. However, the process would be slow.

In this section, we depict the idea and the flow of our proposed CCPC method. CCPC is a mixed method for mining closed clickstream sequential patterns. The CCPC algorithm includes the following main steps:

- Identify all frequent clickstream 1-patterns and build SPPC-tree.
- Construct a C-List for each frequent clickstream 1-pattern based on SPPC-tree.
- Discover the rest of the frequent closed clickstream k-patterns (k > 1).

The specifications of the approach are presented in the section “C-List ”.

SPPC-Tree

SPPC-tree is developed from PPC-tree (Deng 2016). Each node in the tree consists of the following fields:

- Count is the number of sequences sharing a common path from the root to the current node.
- First-child is a list of the first children (i.e., the direct nodes 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 item-name. The list includes nodes that may exist in different branches of the tree.
- Precode is a preorder number assigned by the preorder traversal of the tree.
- Postcode is a postorder number assigned by the postorder traversal of the tree.

SPPC-tree is a compressed database that is made from the horizontal database; however, the sequence id information is lost during the conversion. Thus, one important optimization "SID count optimization" cannot be applied directly to CCPC because CCPC does not keep track of clickstream sequence cid during its process. To make it work, we need to add an element into the node elements in SPPC-tree and C-Lists. The new node structure has an additional element:

- Sum-id: It is mainly used for "SID count optimization".

Each node in SPPC-tree or C-List can be represented in a form of (precode, postcode, count, sum-id). The SPPC-tree can be built according to Algorithm 1 as follows.

Algorithm 1. Building SPPC-tree.

Input: User clickstream database *CDB* and minimum support threshold δ
Output: An SPPC-tree and a set of frequent 1-pattern F_1

1. Identify a set of frequent 1-pattern F_1 and infrequent 1-pattern F_1' in *CDB*
2. Let *CDB*' be the database that has been removed all infrequent 1-patterns in F_1'
3. Create an SPPC-tree T with a root node N_R
4. $N_R \leftarrow$ null
5. For each clickstream *seq* in *CDB*'
6. $N_t \leftarrow N_R$
7. For each event *e* in *seq* do
8. if N_t has any direct child C such as C.event-name = e.event-name
9. C.count $\leftarrow$ C.count + 1
10. C.sum-id $\leftarrow$ C.sum-id + seq.cid
11. else
12. Create a new node C with default value
13. C.count $\leftarrow$ 1
14. C.sum-id $\leftarrow$ *seq*.cid
15. Add C to N_t children list
16. $N_t \leftarrow C$
17. Traverse the SPPC-tree with preorder and postorder traversals to create the preorder and the postorder values for each node.
18. **return** SPPC-tree T

The algorithm first scans the clickstream database, finds all the clickstream patterns with a size of 1, and stores them into a new database *CBD'* (lines 1 and 2). Next, the SPPC-tree is built by the following steps. An empty node is created as the root node (line 3) at the beginning. Each event in a user clickstream is assigned with a new node and appended to the tree in the same order as they are in the user clickstream. The first event of the user clickstream is appended to the root, the second is appended to the first node, and so on. If a node has existed in the tree, it updates its node information by increasing the *count* and *sum-id* values (lines 8-10). Otherwise, a new child node appends to the tree (lines 12-15). After that, we traverse the tree using a depth-first search technique with preorder and postorder to generate the precode and the postcode values for each node.

For example, with the clickstream database CDB in **Table 1** and δ = 0.5%, the absolute value of δ is 2, and all the frequent 1-patterns are first discovered by scanning CDB. The final set is C_1 = {a, b, c, d, f and newly clickstream database as in **Table 2** was created after all infrequent items are removed.

Afterward, the steps of constructed SPPC-tree from **Table 2** are done as follows.

Step 1. Add user clickstream 100 = (a, c, b, a, b) into the SPPC-tree, and we have the following branch.

root	a⟨-, -, 1, 100⟩	c⟨-, -, 1, 100⟩	b⟨-, -, 1, 100⟩	a⟨-, -, 1, 100⟩	b⟨-, -, 1, 100⟩

Table 2. The clickstream database after removing all infrequent 1-patterns.

CID	User clickstream
100	*a, c, b, a, b*
200	*b, f, c, f, b*
300	*a, c, d, f, a, a*
400	*a, c, b, d*

Step 2. Add user clickstream 200 = (b, f, c, f, b) into the SPPC-tree, because the user clickstream 200 does not start with the same start event as the user clickstream 100. Thus, we create a new branch and add each event in this user clickstream into the tree (like what we did to the user clickstream 100).

root	b⟨-, -, 1, 200⟩ a⟨-, -, 1, 100⟩	f⟨-, -, 1, 200⟩ c⟨-, -, 1, 100⟩	c⟨-, -, 1, 200⟩ b⟨-, -, 1, 100⟩	f⟨-, -, 1, 200⟩ a⟨-, -, 1, 100⟩	b⟨-, -, 1, 200⟩ b⟨-, -, 1, 100⟩

Step 3. Add user clickstream 300 = (a, c, d, f, a, a) into the SPPC-tree. The sequence (a, c) has existed in the tree when added the user clickstream 100, so the count and sum-id of each same node are increased. The process for the rest of the events in the user clickstream is the same as adding new events to the tree.

Root	b⟨-, -, 1, 200⟩ a⟨-, -, 2, 400⟩	f⟨-, -, 1, 200⟩ c⟨-, -, 2, 400⟩	c⟨-, -, 1, 200⟩ d⟨-, -, 1, 300⟩ b⟨-, -, 1, 100⟩	f⟨-, -, 1, 200⟩ f⟨-, -, 1, 300⟩ a⟨-, -, 1, 100⟩	b⟨-, -, 1, 200⟩ a⟨-, -, 1, 300⟩ b⟨-, -, 1, 100⟩	a⟨-, -, 1, 300⟩

Step 4. Add user clickstream 400 = (a, c, b, d) into the SPPC-tree, and this process is the same as previous steps.

Root	b⟨-, -, 1, 200⟩ a⟨-, -, 3, 800⟩	f⟨-, -, 1, 200⟩ c⟨-, -, 3, 800⟩	c⟨-, -, 1, 200⟩ d⟨-, -, 1, 300⟩ b⟨-, -, 2, 500⟩	f⟨-, -, 1, 200⟩ f⟨-, -, 1, 300⟩ d⟨-, -, 1, 400⟩ a⟨-, -, 1, 100⟩	b⟨-, -, 1, 200⟩ a⟨-, -, 1, 300⟩ b⟨-, -, 1, 100⟩	a⟨-, -, 1, 300⟩

Finally, the SPPC-tree is built as shown in **Figure 1**

Step 5. Adding precode to the SPPC-tree. Precode ranks are assigned by preorder traversal as shown in Figure 2.

Step 6. Adding postcode into the SPPC-tree. Postcode ranks are assigned by postorder traversal as shown in Figure 3.

C-*List*

Each node in a C-List has the same structure as SPPC-tree in a form of (precode, postcode, count, sum-id). The prefix-path that is formed by walking from the root to the node in SPPC-tree is considered a shared prefix of user clickstreams starting from first event to the current event of the node. The count in each node is the number of user clickstreams that share the prefix. The sum-id is the sum of all clickstream sequence ids of user clickstreams that share the same **prefix-path**. A pattern also has an added element **sum-id** that is attached. Thus, we create a structure called a pattern with two elements: event-name and sum-id.

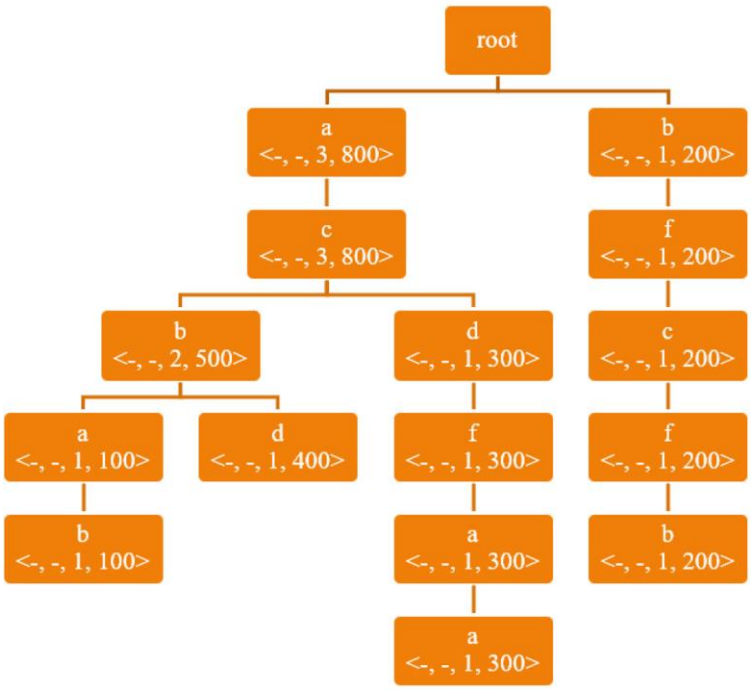

Figure 1. Building the initial SPPC-tree from **Table 2.**

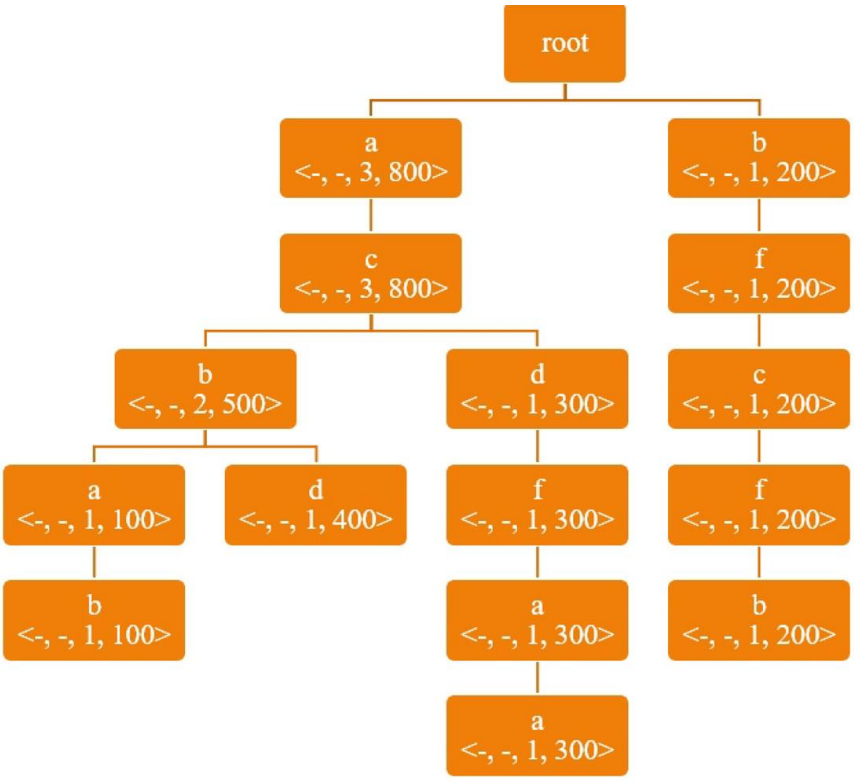

Figure 2. The SPPC-tree after adding precodes.

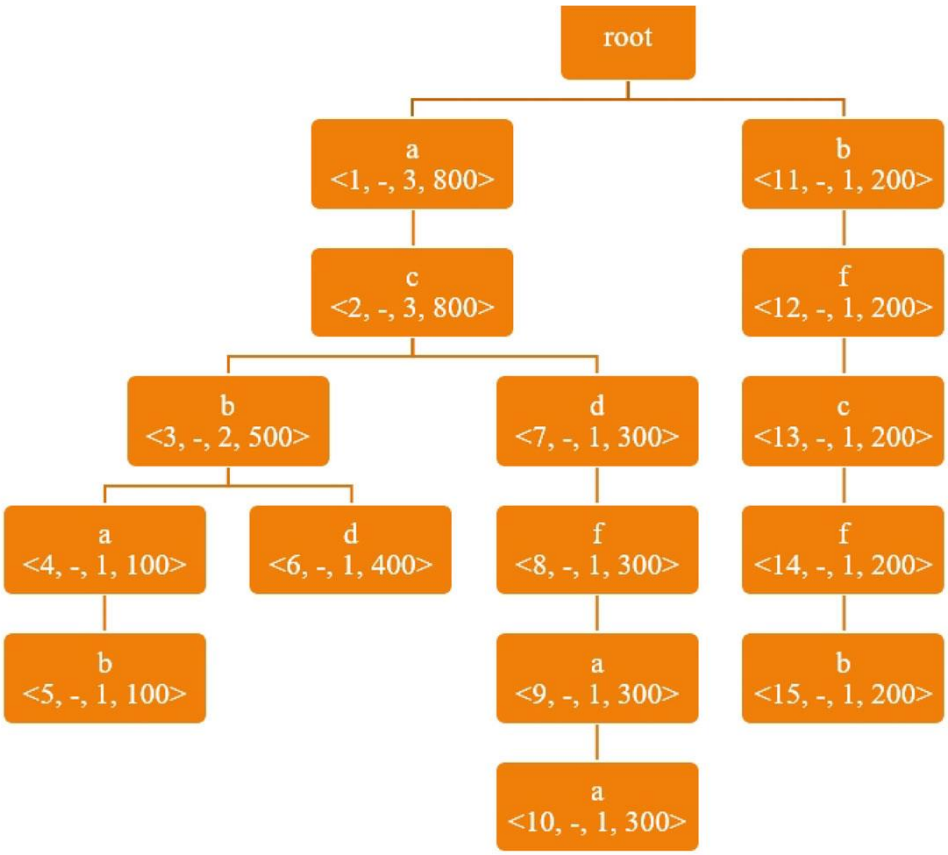

Figure 3. The SPPC-tree after adding postcodes.

Algorithm 2 is shown the process of constructing the C-List for clickstream 1-pattern from SPPC-tree.

Algorithm 2. Constructing C-Lists for all frequent 1-patterns from SPPC-tree.

Input: An SPPC-tree T and the set of frequent 1-pattern F_1
Output: All C-Lists for frequent 1-pattern in F_1

1. For each frequent 1-pattern p in F_1 do
2. Create an empty C-List *clist*
3. For each node N_t in T by traversing in preorder (root-left-right) do
4. if p.event-name = N_t.event-name then
5. if *clist* is empty then
6. $N_{highest} \leftarrow N_t$
7. p.sum-id $\leftarrow N_t$.sum-id
8. else
9. if NOT($N_{highest}$.precode $<$ N_t.precode AND $N_{highest}$.postcode $>$ N_t.postcode) then
10. $N_{highest} \leftarrow N_t$
11. p.sum-id $\leftarrow p$.sum-id + N_t.sum-id
12. Add N_t to *clist*
13. **return** SPPC-tree T

Initially, Algorithm 2 will create an empty C-List for each clickstream 1-pattern in F_1 (lines 1-2), and then it traverses the SPPC-tree in preorder for each node (line 3). If a clickstream pattern existed in SPPC-tree and the C-List is still empty, then add the node to C-List as the highest node (lines 4-7). In contrast, the algorithm has to check the precode and postcode of the current node compared to the highest node in C-List; if it satisfies, then assign this is the highest node to C-List and update the sum-id value (lines 9-12).

We only add the sum-id of the first common father node. Because the common father node contains duplicate information regarding count and sum-id. As mentioned above, the prefix-path from root to the common father is always a subsequence of any prefix-path from the root to any other node that includes the common father node. For example, let X = (root, a, c), Y = (root, a, c, b, a), and Z = (root, a, c, d, f, a) be prefix-paths from the example SPPC-tree, then the first (a) node in X is the common father node and the second (a) node in Y and Z is the descendant node. The common father (a) has a count value of 3 while the other two (a) only have a count value of 1. It means there are three user clickstreams (100, 300, and 400) that have the same prefix-path R1 from root to the first (a), while only one user clickstream has the prefix-paths R_2 (user clickstream 400) and R3 (user clickstream 300) from root to the second (a) in the other two nodes. This also implies that:

- One of three user clickstreams that share R1 is the same as the clickstream that has R2 or R3.
- We cannot accumulate count or sum-id of all three (a) nodes, but only the first (a) node (which is the common father node).

Let X be a pattern. The support count of X = sum of count of all last node in setX where set_X = {Y | ∀ prefix-path Y in SPPC-tree: (X is a subsequence of Y) AND (last item of X = last item of Y)}. Similarly, The sum-id of X = sum of all sum-id of set_X.

Let clist1 and clist2 be the C-Lists of two k-frequent sequences P_1 =< i1, i2,..., ik_1, x > and P2 =< i_1, i_2,..., i_{k-1},y > , P_1 and P_2 share the same (k-1) prefix, and the C-List of (k + 1)-sequence P_3 =< i_1,..., i_{k-1}, x,y > is created by the procedure in Algorithm 3. Otherwise, **Algorithm 3** only works between two frequent clickstream k-patterns that share (k _ 1) prefix. A special case is that frequent 1-patterns are considered sharing an empty prefix. **Algorithm 3** shows the process of constructing C-Lists for frequent clickstream k-patterns.

Algorithm 3. Constructing C-Lists for frequent clickstream k-patterns.

Input: Two patterns P_1 and P_2 that share the same $(k\text{-}1)$-prefix $(k \geq 2)$ and their C-Lists $clist_1$ and $clist_2$
Output: C-List $clist_3$ for the candidate pattern P_3

1. $index_1 \leftarrow 1$; $index_2 \leftarrow 1$
2. $clist_3 \leftarrow \emptyset$
3. While $index_1 \leq$ length of $clist_1$ AND $index_2 \leq$ length of $clist_2$ do
4. $N_1 \leftarrow$ the node at $index_1$ in $clist_1$
5. $N_2 \leftarrow$ the node at $index_1$ in $clist_2$
6. if N_1.precode $<$ N_2.precode AND N_1.postcode $>$ N_2.postcode then
7. if *clist*3 is empty then
8. P_3.sum-id $\leftarrow$ N_2.sum-id
9. $N_{highest} \leftarrow N_2$
10. else if NOT($N_{highest}$.precode $<$ N_2.precode AND $N_{highest}$.postcode $>$ N_2.postcode) then
11. P_3.sum-id $\leftarrow$ P_3.sum-id + N_2.sum-id
12. $N_{highest} \leftarrow N_2$
13. Add N_2 to $clist_3$
14. $index_1 \leftarrow index_1 + 1$
15. $index_2 \leftarrow index_2 + 1$
16. else if N_1.precode $\geq$ N_2.precode then
17. $index_2 \leftarrow index_2 + 1$
18. else if N_1.precode $<$ N_2.precode then
19. $index_1 \leftarrow index_1 + 1$
20. **return** $clist_3$

The mining process of closed clickstream patterns in Algorithm 4 can be described briefly as follows. First, the algorithm finds all closed clickstream 1-patterns from SPPC-tree T_1 and C-List C_1 (line 3). Next, it performs an expansion for k-pattern by the recursive procedure (line 4). With two patterns Pu and P_v in C_k that have the same (k-1) prefix, a new pattern P_v and the C-List of that pattern are created (lines 7-11), and then the algorithm calculates the support counts of the new pattern (line 9). If the new pattern is satisfying the threshold, it is put into C_k and T_k (line 12) and then we apply the closure property check to determine whether the pattern is closed (line 14). The closed pattern is added to the result and the process continues for (k + 1-pattern (line 16). **Algorithm 4** illustrates the processing of CCPC by pseudocodes.

Algorithm 4. Mining closed clickstream patterns.

Input: User clickstream database *CDB*, the minimum support threshold δ
Output: A complete set of all closed clickstream sequential patterns *CP* that exists in the database

1. *Build* $T_1 \leftarrow$ *SPPC-tree (Call Alg.1)*
2. *Initial* $C_1 \leftarrow$ *C-List (Call Alg. 2)*
3. *Call mining_Closed* (C_1, T_1)
4. $CP \leftarrow \emptyset$
5. ***Procedure**: mining_Closed* (C_k, T_k)
6. *Initial* $C_{k+1} \leftarrow \emptyset$, $T_{k+1} \leftarrow \emptyset$
7. *for each pattern* P_u *in* T_k *do*
8. *for each pattern* P_v *in* T_k do
9. $clist_u \leftarrow$ *the C-List of* P_u in C_k
10. $clist_u \leftarrow$ *the C-List of* P_u in C_k
11. *Create* P_z *and* $clist_z$ *from* P_u, P_v, $clist_u$, and $clist_v$, *(Call Alg.3)*
12. *if* $\varepsilon(P_z) \geq \delta$ *then*
13. $C_{k+1} \leftarrow C_{k+1} \cup clist_z$; $T_{k+1} \leftarrow Pz \cup T_{k+1}$
14. *if*(*isClosed*(P_z)) then
15. $CP \leftarrow CP \cup P_z$
16. *end if*
17. *end if*
18. *end for*
19. *Call mining_Closed*(C_{k+1}, T_{k+1})
20. *end for*
21. ***end procedure***

Experimental Evaluation

This section evaluates the effectiveness of the CCPC algorithm, and it is compared with CloFAST, CM-ClaSP, CloSpan, and FCloSM. The experiments were performed on a personal computer equipped with Intel Core i5-7200U CPU 2.5-GHz, 8 GB of RAM, Windows 10 Pro 64 bit and used Java language programming with JDK 13.0.2. The information of databases is shown in Table 3.

The experimental results exhibit that CCPC is better than other algorithms in most cases. Especially, we can see the improvement when comparing the CCPC algorithm with FCloSM1, an efficient algorithm for mining frequent CSP recently. The CCPC algorithm can be executed with a very small minsup threshold while the other methods are of high computation cost or could not be executed. All the chart visuals in this section are using the logarithmic scale base 10.

Runtime

Figures 4-9 show the runtimes of the CCPC algorithm and CloFAST, CM-ClaSP, CloSpan, and FCloSM algorithms for mining CSPs. The experimental results confirmed that the runtime of the CCPC algorithm is much better than other algorithms on all the databases and with various δ.

Table 3. Databases used in experiments[a].

Database	#sequences	#items	Comment
BIBLE	36,369	13,905	This dataset is a conversion of the Bible into a sequence database
FIFA	20,450	2,990	Clickstream data from the Web site of FIFA World Cup 98
Kosarak	990,002	41,270	Clickstream data of a Hungarian online news portal (source: http://fimi.uantwerpen.be/data/. . . . .)
MSNBC	989,818	17	Clickstream data from the MSNBC Web site
Chainstore	1,112,949	46,086	Customer transactions from a major grocery store chain in California, USA
KDDCup99	1,000,000	135	Clickstream data from the KDD-CUP 1999 https://archive.ics.uci.edu/ml/datasets/KDD+Cup+1999+Data. . . . .)

[a]**http://www.philippe-fournier-viger.com/spmf/index.php?link=datasets.php**

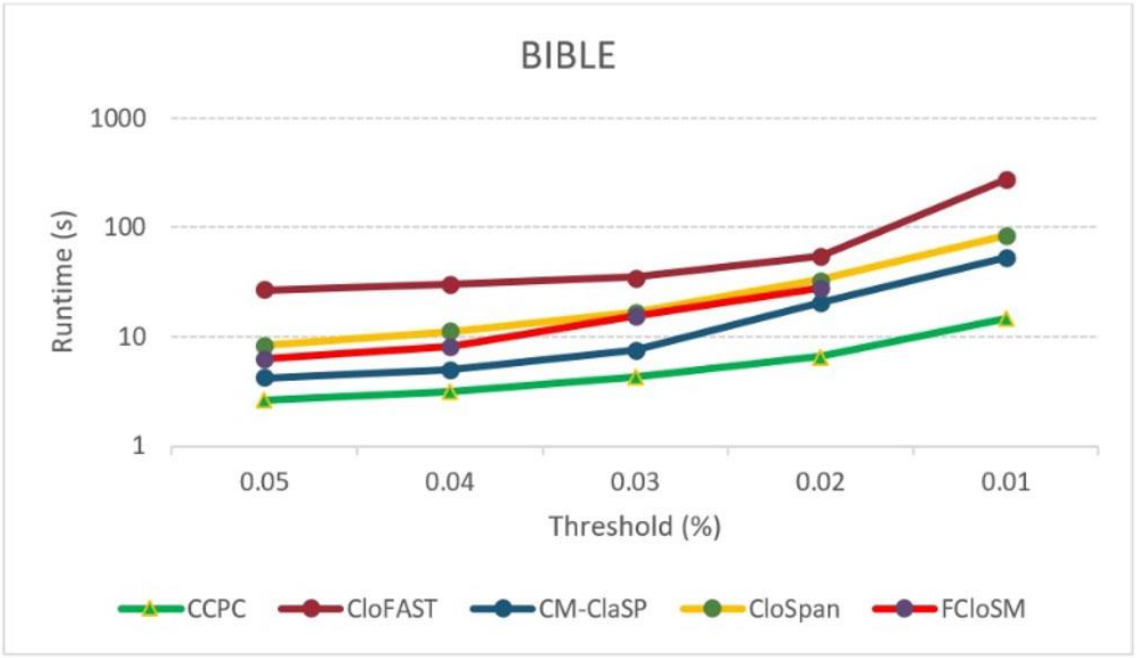

Figure 4. Runtime for exploiting FSP in the BIBLE database.

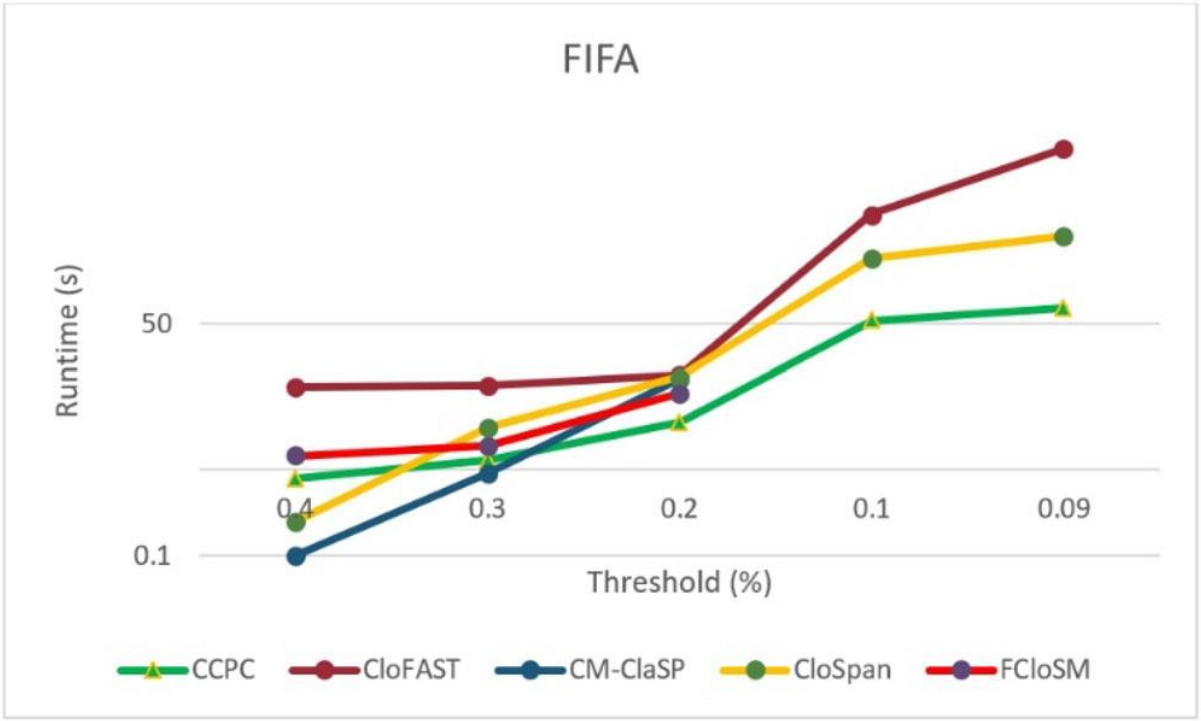

Figure 5. Runtime for exploiting FSP in the FIFA database.

Especially, when we set up a small δ and there is a large sequence database with many items, the CCPC algorithm is still worked fine while some of the algorithms cannot be executed.

In more detail, with the BIBLE database as shown in Figure 4 and d = 0.05%, the runtime of the CCPC algorithm is faster than the CloFAST, CM-ClaSP, CloSpan, and FCloSM algorithms having 10.3, 1.62, 3.2, and 2.4 runtimes, respectively.

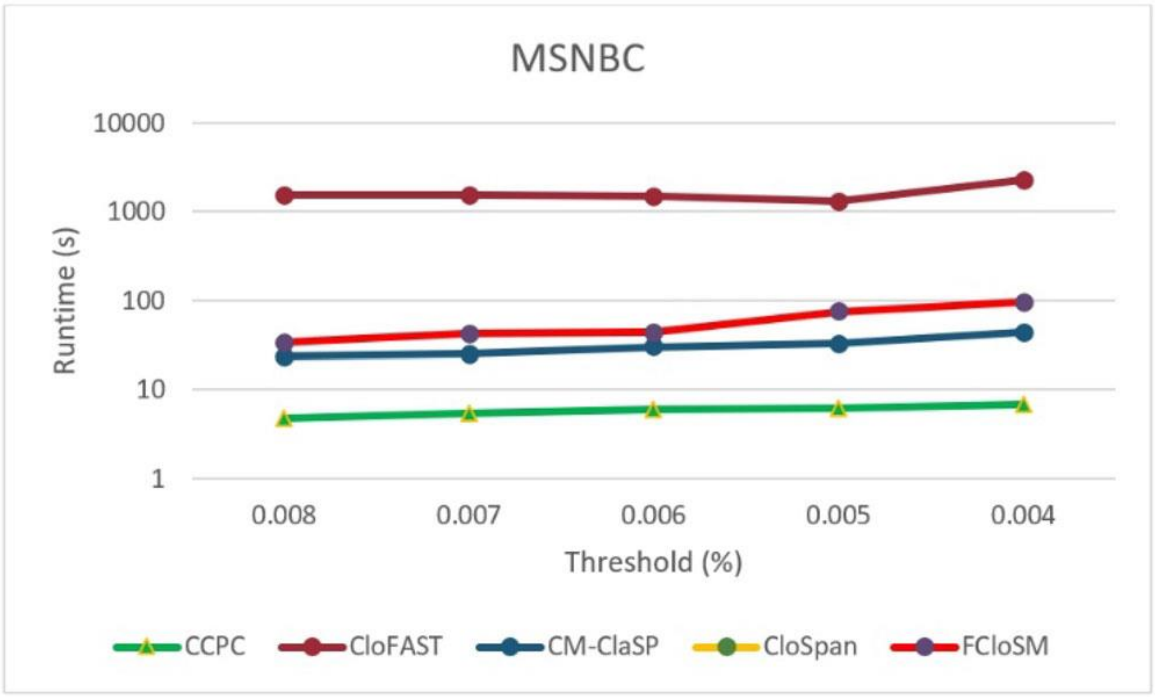

Figure 6. Runtime for exploiting FSP in the Kosarak database.

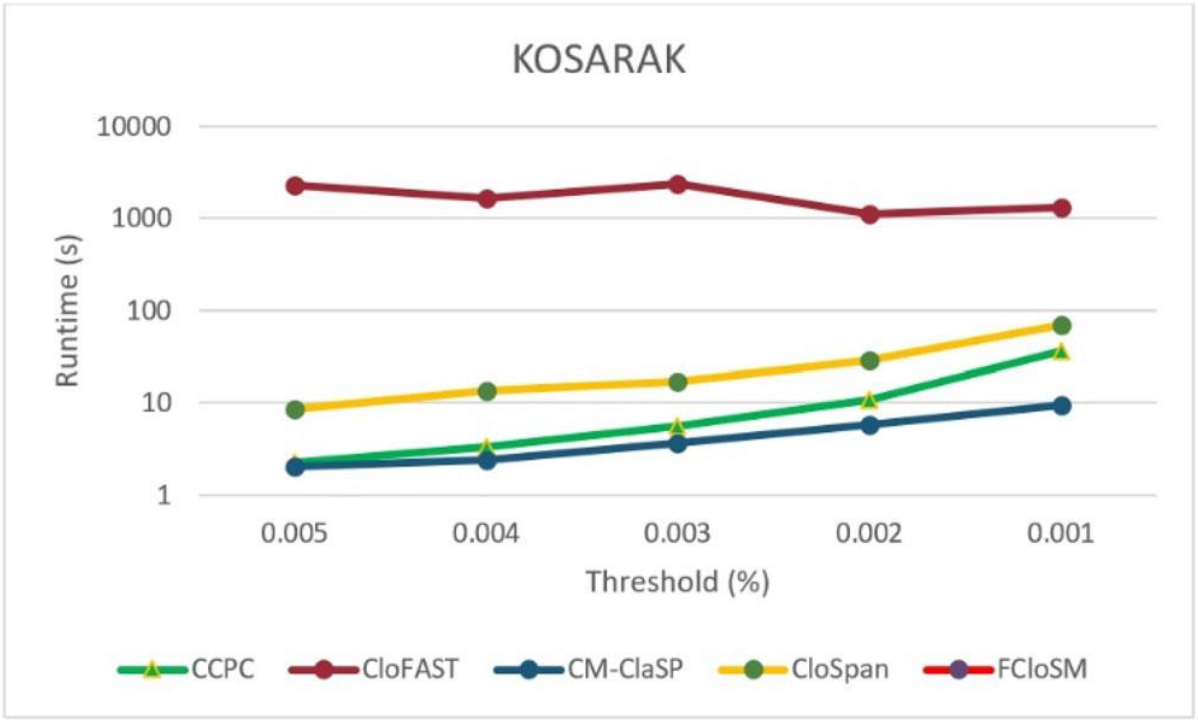

Figure 7. Runtime for exploiting FSP in the MSNBC database.

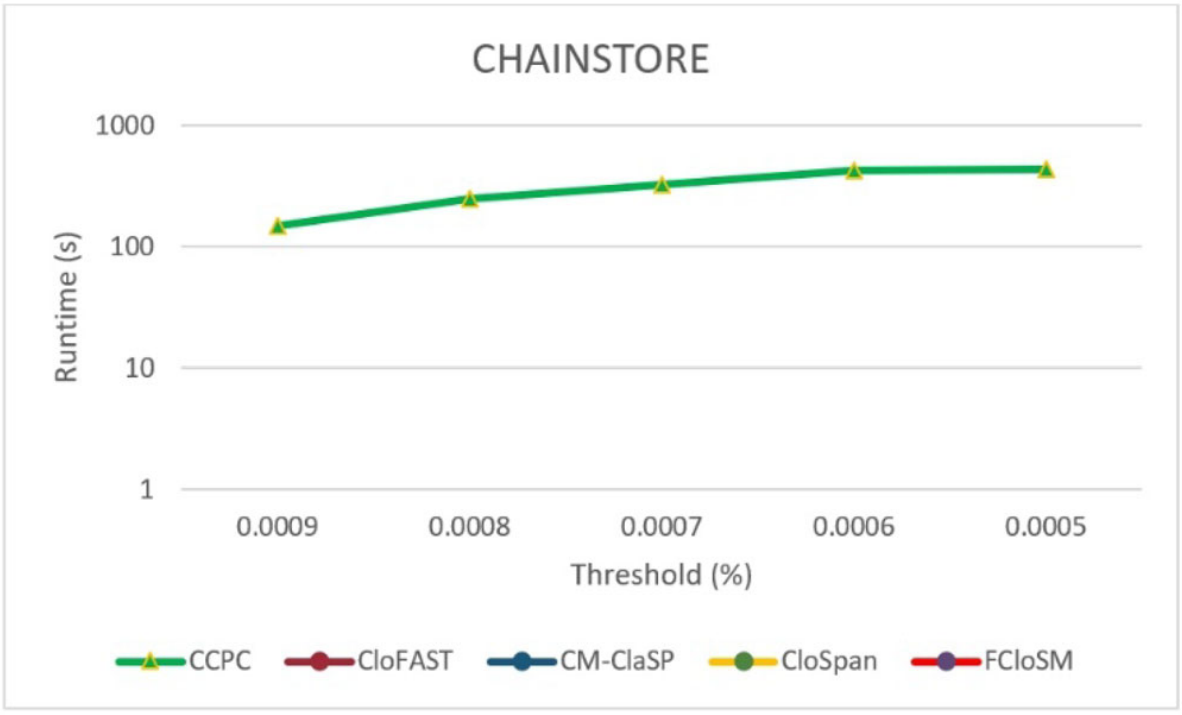

Figure 8. Runtime for exploiting FSP in the Chainstore database.

When δ is decreased to 0.01%, the gaps between runtime of the CCPC algorithm and the CloFAST, CM-ClaSP, and CloSpan algorithms increase to 18.5, 3.6, and 5.7 times, respectively.

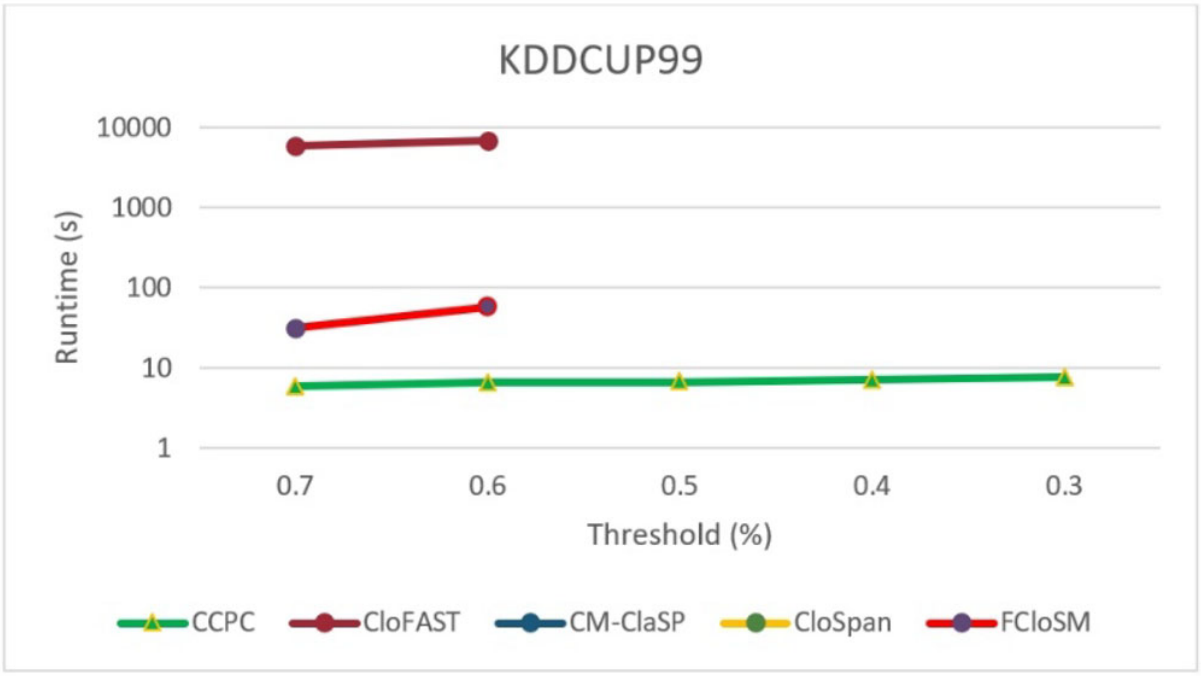

Figure 9. Runtime for exploiting FSP in the KDDCup99 database.

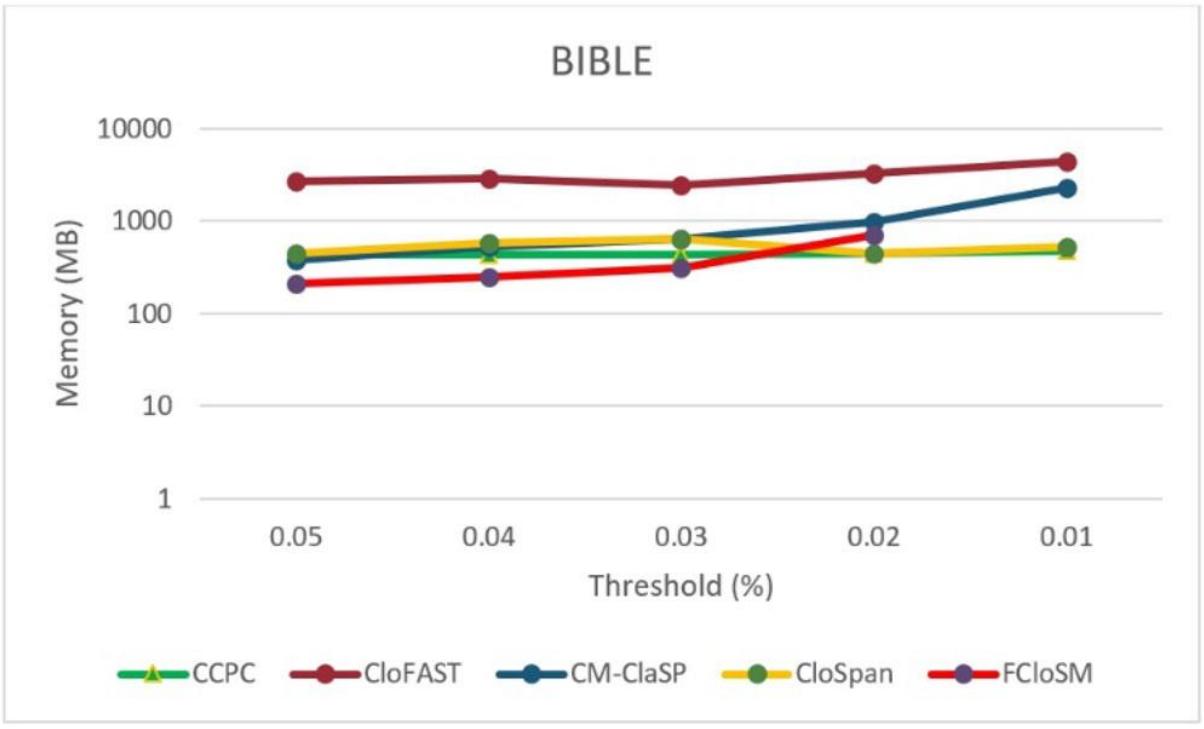

Figure 10. Memory usage for exploiting FSP in the BIBLE database.

Meanwhile, the FCloSM algorithm cannot be executed due to insufficient memory. Therefore, the marker point does not exist in the chart at that point.

With the FIFA database as shown in **Figure 5**, the CCPC algorithm is also faster than other algorithms. Although the CCPC algorithm is not faster than all others at the early δ values, CCPC is better than all other algorithms when $\delta < 0.2\%$. Especially, the CM-ClaSP and FCloSM algorithms cannot be executed at the $\delta = 0.1\%$ on this database.

Figure 6 shows the runtime of the CCPC algorithm is many times faster than CloFAST and CloSpan on all thresholds on the Kosarak database. Especially, the FCloSM cannot be executed with any δ value and this is a unique case that the CCPC algorithm is not faster than the CM-ClaSP algorithm.

Similar to the MSNBC database, **Figure 7** shows that the runtime of the CCPC algorithm is always faster than the other algorithms. The CloSpan algorithm cannot be executed at any δ value on this database.

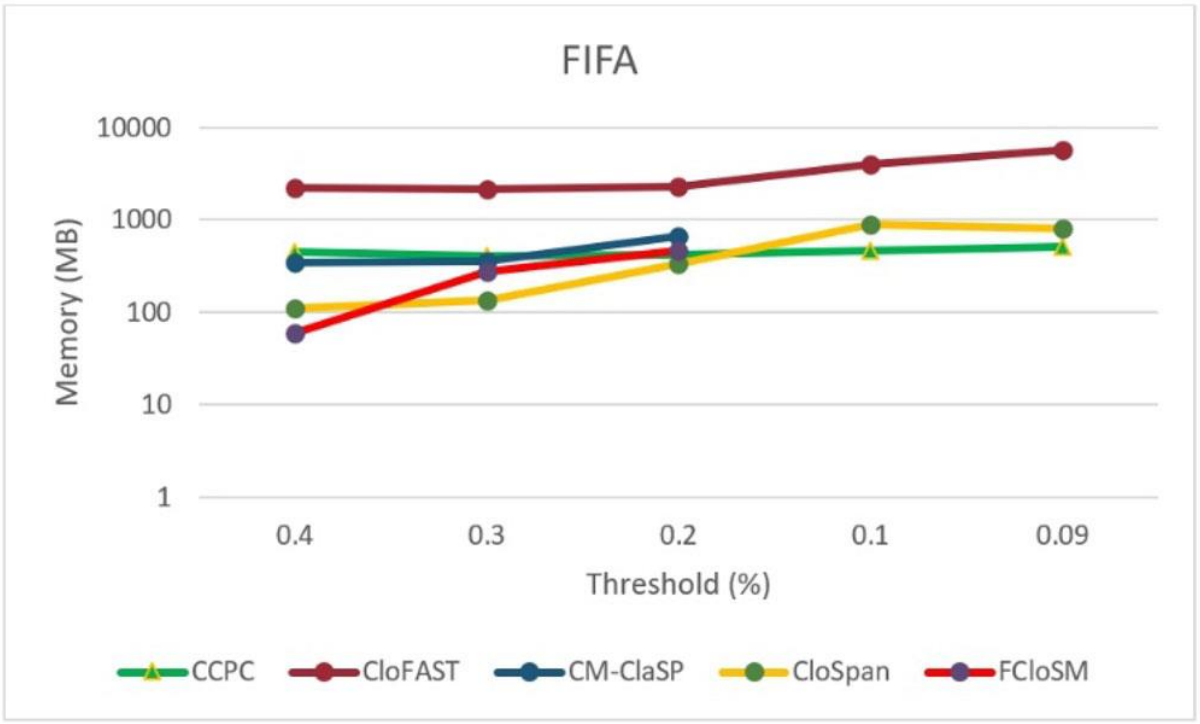

Figure 11. Memory usage for exploiting FSP in the FIFA database.

Same as previous experiments, Figures 8-9 show that the runtime of the CCPC algorithm is the best, and it gradually increases while the other algorithms cannot be executed on the Chainstore database.

The CloFAST and FCloSM algorithms are only executed with two d values 0.7% and 0.6% on the KDDCup99 database. Meanwhile, the CCPC can run with a smaller value of $\delta < 0.6\%$.

Memory Usage

Similar to runtime, the memory usage of CCPC is not lower than some algorithm at the early stage; however, with a low value of δ, the memory usage of CCPC is much better. **Figures 10-15** show the detailed memory usage of these algorithms.

Figure 10 shows that on the BIBLE database at δ = 0.05%, the memory usage of FCloSM is smallest; however, the CCPC algorithm is better than the CloFAST, CM-ClaSP, and CloSpan algorithms. When the minsup threshold is decreased (δ = 0.03%), the memory of FCloSM begins to increase. At δ = 0.02%, the memory usage of FCloSM significantly increases while CCPC slowly increases and becomes the lowest memory consumption method. The FCloSM algorithm cannot be executed at δ = 0.01%.

Similarly, **Figure 11** shows that the memory usage of CCPC is better than the other algorithms when δ is at low values (δ = 0.1%) and it continues to decrease afterward.

With the Kosarak database in **Figure 12**, the memory usage of the CCPC algorithm is not better than the CloSpan algorithm in most of δ values; however, it is still better than CloFAST and CM-ClaSP algorithms. The FCloSM algorithm is omitted because it cannot run on this database.

Figures 13-15 show that the memory usage of the CCPC algorithm is the best when compared with the others in all δ values. The CloSpan cannot run on the MSNBC database that is shown in **Figure 13**.

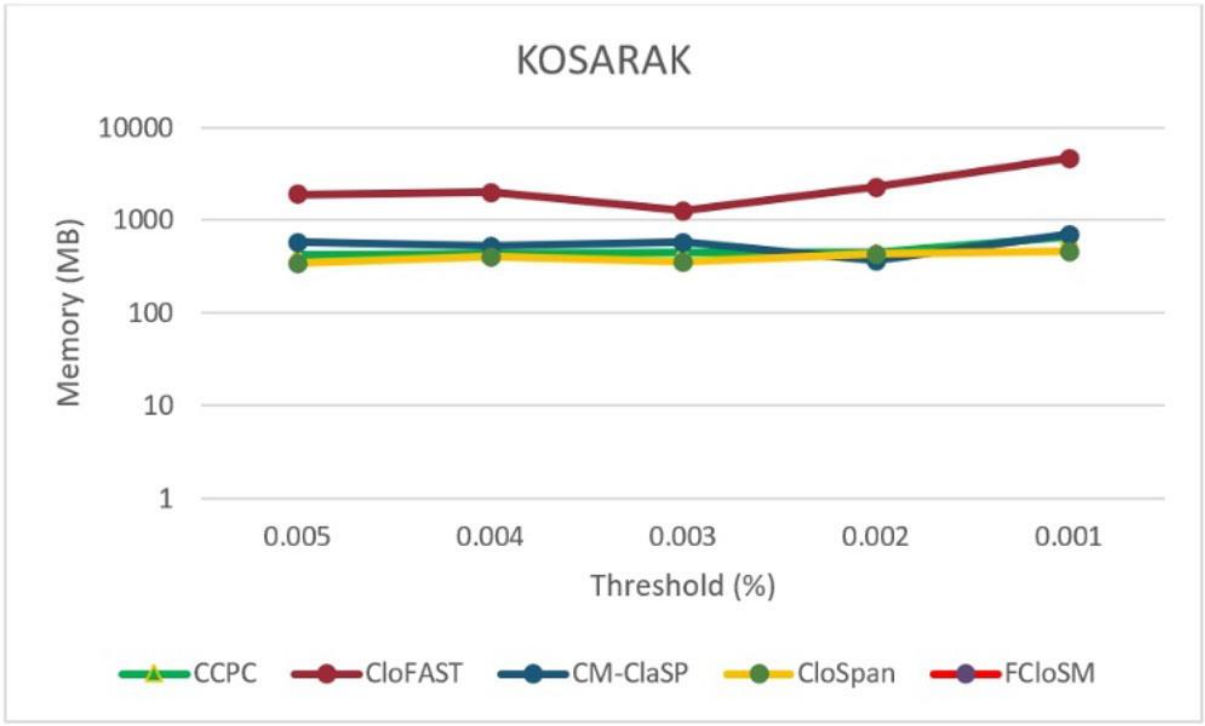

Figure 12. Memory usage for exploiting FSP in the Kosarak database.

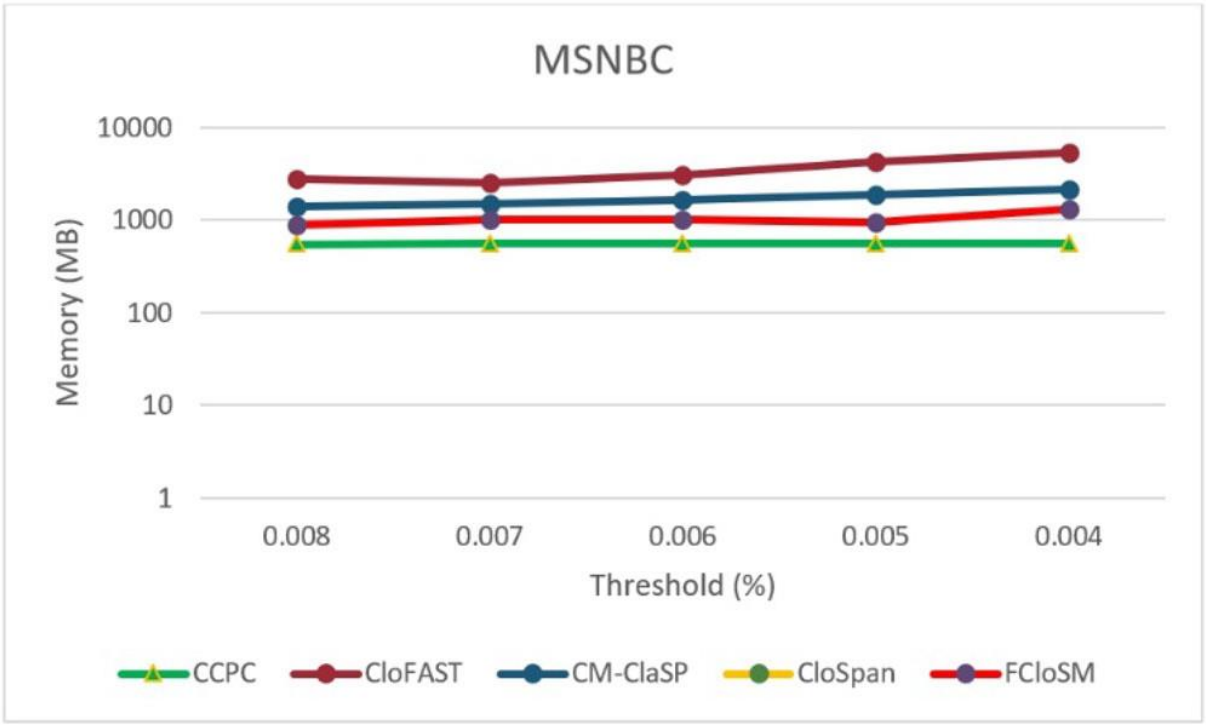

Figure 13. Memory usage for exploiting FSP in the MSNBC database.

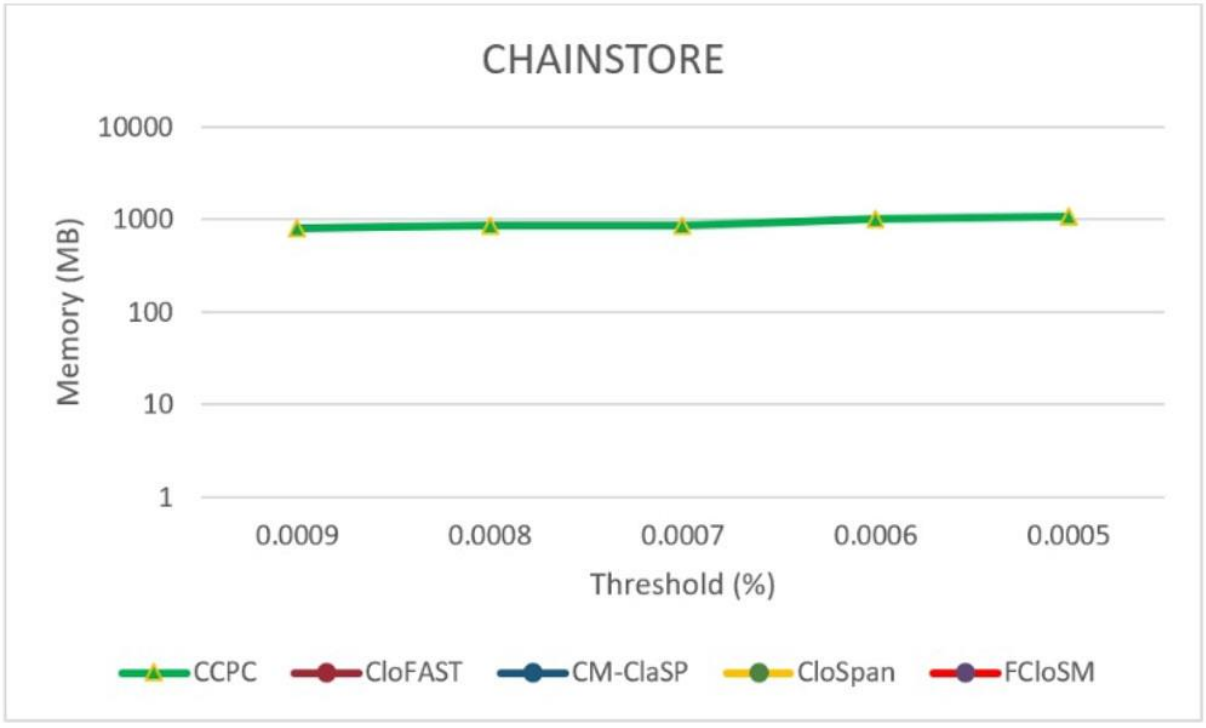

Figure 14. Memory usage for exploiting FSP in the Chainstore database.

With the Chainstore database in **Figure 14**, only the CCPC can be executed.

Figure 15 shows that the memory usage of the CCPC algorithm does not increase or increases very little while the CloFAST and FCloSM algorithms are only executed at two δ values 0.7% and 0.6%.

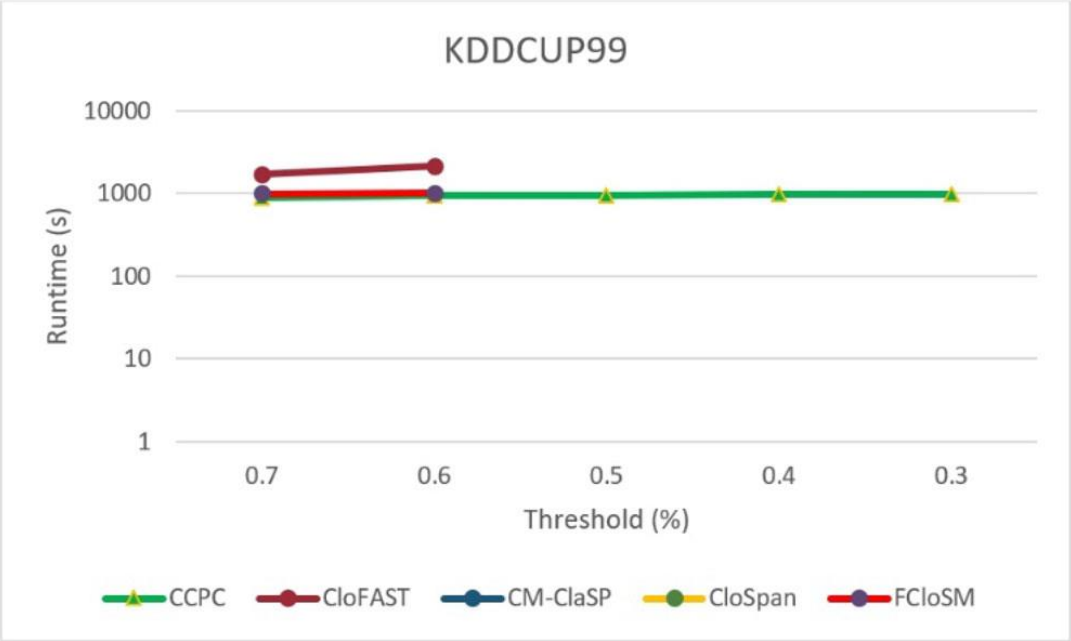

Figure 15. Memory usage for exploiting FSP in the KDDCup99 database.

Overall, the proposed algorithm has better performance than the others in terms of time and memory usage, especially when the minsup threshold δ is small on the huge databases.

References

Aberra, N., A. Sebastian, A. Maloy, C. Rees, M. Bartron, and I. Albert. 2020. Bioinformatics recipes: Creating, executing and distributing reproducible data analysis workflows. BMC Bioinformatics 21 (1):292.

Asite, M. G., and L. Aleksejeva. 2019. Classification methodology for bioinformatics data analysis. Automatic Control and Computer Sciences 53 (1):28-38.

Astrova, I., A. Koschel, and S. L. Lee. 2020. Using market basket analysis to find semantic duplicates in ontology. ICCSA 4:197-211.

Bui, B. V., B. Vo, H. M. Huynh, T. A. Nguyen-Hoang, and B. Huynh. 2018. An efficient method for mining clickstream patterns. In IJCSR 2018, LNAI 11103, 572-83.

Danilowicz, C., and N. T. Nguyen, 2000. Consensus-based methods for restoring consistency of replicated data. In: Klopotek. (eds), Advances in Soft Computing, Proceedings of 9th International Conference on Intelligent Information Systems'2000, Physica, 325-36.

Deng, Z. H. 2016. DiffNodesets: An efficient structure for fast mining frequent itemsets. Applied Soft Computing 41:214-23. **doi: 10.1016/j.asoc.2016.01.010.**

Fabra, J., P. Alvarez, and J. Ezpeleta. 2020. Log-based session profiling and online behavioral prediction in e-commerce websites. IEEE Access 8:171834-50. **doi: 10.1109/ ACCESS.2020.3024649**.

Fournier-Viger, P., A. Gomariz, M. Campos, and R. Thomas. 2014. Fast vertical mining of sequential patterns using co-occurrence information. PAKDD 1:40-52.

Fumarola, F., P. F. Lanotte, M. Ceci, and D. Malerba. 2016. CloFAST: closed sequential pattern mining using sparse and vertical id-lists. Knowledge and Information Systems 48 (2): 429-63. **doi: 10.1007/s10115-015-0884-x**.

Gomariz, A., M. Campos, R. Marin, and B. Goethals. 2003. ClaSP: An efficient algorithm for mining frequent closed sequences. PAKDD 2013, 7818:50-61.

Hagen, M., and B. Stein. 2018. Weblog analysis. New York, NY: Springer.

Han, J., H. Cheng, D. Xin, and X. Yan. 2007. Frequent pattern mining: current status and future directions. Data Mining and Knowledge Discovery 15 (1):55-86. **doi: 10.1007/ s10618-006-0059-1**.

Huynh, B., C. Trinh, H. M. Huynh, T. T. Van, B. Vo, and V. Snasel. 2018. An efficient approach for mining sequential patterns using multiple threads on very large databases. Engineering Applications of Artificial Intelligence. 74:242-51. **doi: 10.1016/j.engappai.2018.06.009**.

Huynh, B., B. Vo, and V. Snasel. 2017. An efficient parallel method for mining frequent closed sequential patterns. IEEE Access 5:17392-402. **doi: 10.1109/ACCESS.2017.2739749**.

Huynh, H. M., L. T. Nguyen, B. Vo, A. Nguyen, and V. S. Tseng. 2020. Efficient methods for mining weighted clickstream patterns. Expert Systems with Applications 142:112993. **doi: 10.1016/j.eswa.2019.112993.**

Huynh, H., L. Nguyen, B. Vo, U. Yun, Z. Oplatkovaa, and T. Hong. 2020. Efficient algorithms for mining clickstream patterns using pseudo-IDLists. Future Generation Computer Systems 107:18-30. **doi: 10.1016/j.future.2020.01.034**.

Le, B., H. Duong, T. Truong, and P. Fournier-Viger. 2017. FCloSM, FGenSM: Two efficient algorithms for mining frequent closed and generator sequences using the local pruning strategy. Knowledge and Information Systems 53 (1):71-107. **doi: 10.1007/s10115-017-1032-6**.

Le, T., A. Nguyen, B. Huynh, B. Vo, and W. Pedrycz. 2018. Mining constrained intersequence patterns: a novel approach to cope with item constraints. Applied Intelligence 48 (5):1327-43. **doi: 10.1007/s10489-017-1123-9**.

Maleszka, M., and N. T. Nguyen. 2011. A method for complex hierarchical data integration. Cybernetics and Systems 42 (5):358-78. **doi: 10.1080/01969722.2011.595341**.

Mianowska, B., and N. T. Nguyen. 2013. Tuning user profiles based on analyzing dynamic preference in document retrieval systems. Multimedia Tools and Applications 65 (1): 93-118. **doi: 10.1007/s11042-012-1145-6**.

Moodley, R., F. Chiclana, F. Caraffini, and J. Carter. 2019. Application of uninorms to market basket analysis. International Journal of Intelligent Systems 34 (1):39-49. **doi: 10.1002/ int.22039**.

Nguyen, N. T. 2000. Using consensus methods for solving conflicts of data in distributed systems. SOFSEM 2000, LNCS 1963:411-9.

Nguyen, N. T. 2002. Consensus systems for conflict solving in distributed systems. Information Sciences 147 (1-4):91-122. **doi: 10.1016/S0020-0255(02)00260-8**.

Nguyen, N. T., and J. Sobecki. 2003. Using consensus methods to construct adaptive interfaces in multimodal web-based systems. Universal Access in the Information Society 2 (4): 342-58. **doi: 10.1007/s10209-003-0050-1**.

Pham, T. T., T. Do, A. Nguyen, B. Vo, and T. P. Hong. 2020. An efficient method for mining top-K closed sequential patterns. IEEE Access 8:118156-63. **doi: 10.1109/ACCESS. 2020.3004528**.

Prakash, P. G.-O., and A. Jaya. 2020. WS-BD-based two-level match: Interesting Sequential Patterns and Bayesian Fuzzy Clustering for Predicting the Web Pages from Weblogs. The Computer Journal 63 (2):322-36. **doi: 10.1093/comjnl/bxz132**.

Shihab, A. I., F. A. Dawood, and A. H. Kashmar. 2020. Data analysis and classification of autism spectrum disorder using principal component analysis. Advances in Bioinformatics 2020:3407907. **doi: 10.1155/2020/3407907**.

Soui, M., S. Smiti, M. W. Mkaouer, and R. Ejbali. 2020. Bankruptcy prediction using stacked auto-encoders. Applied Artificial Intelligence 34 (1):80-100. doi: **10.1080/ 08839514.2019.1691849**.

Tran, T., B. Le, and B. Vo. 2015. Combination of dynamic bit vectors and transaction information for mining frequent closed sequences efficiently. Engineering Applications of Artificial Intelligence 38:183-9. **doi: 10.1016/j.engappai.2014.10.021**.

Valle, M. A., G. A. Ruz, and R. Morras. 2018. Market basket analysis: Complementing association rules with minimum spanning trees. Expert Systems with Applications 97:146-62. **doi: 10.1016/j.eswa.2017.12.028**.

Wang, J., and J. Han. 2004. BIDE: Efficient mining of frequent closed sequences. International Conference on Data Engineering, Boston, MA, 79-90.

Wu, Y., C. Zhu, Y. Li, L. Guo, and X. Wu. 2020. NetNCSP: Nonoverlapping closed sequential pattern mining. Knowledge Based Systems 196:105812. **doi: 10.1016/j.knosys.2020. 105812**.

Yan, R., Y. Li, D. Li, W. Wu, and Y. Wang. 2021. SSDBA: the stretch shrink distance based algorithm for link prediction in social networks. Frontiers of Computer Science 15 (1): 151301.

Yan, X., J. Han, and R. Afshar. 2003. CloSpan: Mining closed sequential patterns in large datasets. SDM