Applied Intelligence (2025) 55:221
https://doi.org/10.1007/s10489-024-06137-y

A tree-based framework to mine top-K closed sequential patterns

Redwan Ahmed Rizvee[1] · **Chowdhury Farhan Ahmed**[1] · **Carson K. Leung**[2]

Accepted: 30 November 2024

Abstract
Top-K closed sequential pattern (CSP) mining addresses the challenge of reducing the number of mined patterns and the dependency on the support threshold parameter. This study tackles top-K CSP mining from three angles: top-K generic CSPs, group CSPs, and redundancy-aware CSPs. We propose the novel SP-Tree-based *KCloTreeMiner* to mine these variations and introduce the *PaMHep* data structure for efficient candidate pattern maintenance. Two pruning strategies—namely, *pattern absorption* and SP-Tree-based temporary node projection—are also presented to reduce search space. This study offers a thorough theoretical analysis and establishes bounds for the top-K framework, covering everything from solution design to completeness and optimization. Evaluations on six real-life datasets show up to a 23% average runtime improvement for *KCloTreeMiner* over the benchmark algorithm TKCS. We also propose two greedy algorithms Max_{WC} and Max_{WOC} for pattern summarization and introduce *Subset Distance* for measuring distances between sequential patterns, improving K-medoid clustering results over average silhouette-width for the reported clusters.

Keywords Closed sequential patterns · Top-K mining · Tree-based mining · Pruning technique · Clustering · Pattern summarization

1 Introduction

The sequential pattern mining (SPM) problem seeks to identify sequential relationships within a dataset [25, 29, 47, 48]. First addressed by Srikant and Agrawal [38], it has since inspired numerous approaches from various perspectives [12]. The generic SPM problem focuses on discovering patterns that appear a specified number of times, defined by a minimum support threshold (min_sup). Due to its relevance in many real-life applications, SPM has been extended to various contexts, including attribute-based databases [27], negative SPM [47], and mining with specified gaps [25].

In the generic SPM problem, the number of mined patterns can vary greatly with the min_sup threshold, often resulting in uninteresting patterns. Estimating min_sup without knowing database characteristics is challenging. To address the issue of an overwhelming number of patterns, closed sequential pattern (CSP) mining was introduced [14]. A pattern is closed if no super-pattern has the same support. However, CSP mining still relies on the empirically set min_sup parameter.

To address the dependency on the min_sup parameter, the top-K sequential mining problem was introduced [10]. This approach focuses on mining K patterns with the highest support [10]. Additionally, the top-K closed SPM problem was developed to further reduce the number of patterns by mining the top-K closed patterns [41]. Due to its efficiency, variations like top-K rules mining [12], top-K high utility mining [20], and top-K contrastive pattern mining [49] were also explored.

A real-life example illustrates the importance of top-K CSP over generic SPM and CSP mining. In the seasonal fruit market, fruit availability varies by season. Using generic SPM to find frequently bought fruit collections requires setting a threshold, which is challenging due to seasonal variations and domain knowledge dependencies, e.g., availability of fruits based on seasons, fruit growth geographical patterns, etc. The top-K approach, however, does not depend on

✉ Chowdhury Farhan Ahmed
farhan@du.ac.bd

Redwan Ahmed Rizvee
rizvee@cse.du.ac.bd

Carson K. Leung
carson.leung@umanitoba.ca

[1] University of Dhaka, Dhaka 1000, Bangladesh

[2] University of Manitoba, Winnipeg, MB R3T 2N2, Canada

Published online: 26 December 2024

the distribution of entities within the database, simplifying the process of identifying sequential relationships. Applying top-K CSP further compacts and diversifies the mined fruit sets compared to top-K generic sequential patterns [34].

To the best of our knowledge, TKCS [34] is the current state-of-the-art algorithm for the top-K CSP problem, improving on TKS [10]. However, TKCS has issues that we address in our work. For instance, TKCS uses a list-based structure to store candidates and recursively extends patterns until their support decreases, then marks them as closed. We identified key points for improvement and developed our proposals accordingly:

1. *Necessary number of mined patterns:* TKCS [34] lacks a formal theoretical basis for top-K mining. To ensure completeness, more than K patterns may be needed. We establish a pattern-frequency-based condition to ensure the K-reported patterns are all CSPs with no higher support patterns. We also provide candidate ordering strategies.
2. *Customized heap-based data structure:* TKCS uses inefficient list-based storage for top-K mining. We introduce a heap-based structure—namely, Pattern Max Heap (*PaMHeap*)—with customized attributes for efficient candidate management.
3. *Bitmap-based projection storage:* TKCS's bitmap-based storage is biased by database size. We use SP-Tree [36] for faster, flexible projection storage, and introduce SP-Tree*, a memory-optimized version that uses *node IDs* to save projection information efficiently.

In addition to our main contributions, we also introduce pruning strategies to reduce pattern search space. Pruning is crucial for efficient mining algorithms to avoid redundant pattern detection and unnecessary searches. We adopt the breadth-first-based support counting strategy (BF-S) from Ref. [36], which approximates maximum support for patterns and guides projection requirements in SP-Tree for top-K mining. Our new pruning strategy—namely, *pattern absorption*—identifies redundant sequence extensions by comparing support values of smaller patterns with their super patterns and eliminates redundant searches accordingly.

Combining the aforementioned strategies and rationales, we introduce the novel mining algorithm *KCloTreeMiner* to extract top-K CSPs or generic CSPs. We refer to this version as "generic" because we also introduce two other variations of top-K CSP mining in the current article: the *top-k group CSP* mining problem and the *top-k redundancy-aware CSP* mining problem.

The top-K group CSP problem aims to mine groups of CSPs with the highest K unique supports, ensuring all remaining CSPs have lower support. While it can be addressed using generic top-K CSP mining, determining the exact value of K often requires experimentation, which is costly in terms of memory and time. Applications seeking patterns within the highest K unique supports may struggle to adjust K for generic solutions. For example, consider a clickstream database in which new data appear at regular intervals. A challenge is to mine all the CSPs within the top-3 unique support values (K=3) for each new set of data at each interval. In such a scenario, the concept of extracting the top-K group CSPs would be highly beneficial for identifying these patterns.

The top-K redundancy aware CSP problem aims to report CSPs while ensuring two constraints: remaining unmined CSPs either have lower support or are subsets of reported ones. This approach captures more variation compared to generic CSP mining for the same K value by avoiding repeating subsequences. The problem allows for a larger K window to include more patterns, resulting in increased variational subsequences. However, it may not exhibit a monotonically increasing property over the number of mined patterns due to continuous generation and removal of patterns. We provide a theoretical framework to analyze the mining nature in this redundancy-aware version.

We address this variation of top-K redundancy-aware CSPs to achieve broader pattern coverage, reducing repetition compared to the generic version. For example, while the generic approach might report multiple patterns like $< (a)(b) >: 10$, $< (a)(c) : 9$, $< (b)(c) : 8$, and $< (a)(b)(c) : 7 >$, the redundancy-aware approach only reports $< (a)(b)(c) : 7 >$. Our algorithm KCloTreeMiner handles these variations with minimal modifications. Unlike maximal pattern mining, which focuses on larger sequences to capture frequent patterns, our approach considers both pattern support and maximality in enclosing smaller patterns.

Top-K CSP mining is a well-established problem in sequential pattern mining [34]. Our study suggests that the top-k framework can extend to other variations relevant to different scenarios and applications [10, 41]. We present two such variations with examples. We also highlight significant observations related to performance metrics like memory, runtime, and completeness from problem formulation to solution design. The current article provides detailed discussions with strong mathematical formulations and bounds to guide researchers in this domain.

We extend the top-K group CSP mining problem to include finding representative or summarizing patterns for each group of CSP [5]. We focus on three strategies:

1. A maximal pattern-based approach, in which summarized patterns do not need to be explicitly present in the database (Max_{WOC}).
2. A maximal pattern-based approach, in which each summarized pattern must be explicitly present in the database (Max_{WC}).

3. A clustering-based approach, which uses cluster centroids as summarized patterns.

For the first two strategies, we propose two greedy algorithms ensuring sub-optimality and discuss the complexity of the optimal solution using dynamic programming. For the third strategy, we introduce a new distance metric called Subset Formation Distance, along with existing widely used measures [40]. We employ k-medoid clustering to cluster patterns and find centroids as summarized patterns. These strategies are generic and applicable to a wide range of representative pattern-finding problems [6].

In the light of the aforementioned discussion, the *key novel contributions* of this study can be summarized as follows:

1. **Top-K framework:** We extensively explore the top-K Closed Sequential Pattern (CSP) mining framework, discussing three related variations:
   (a) Top-K generic CSP: Mining the top-K CSPs while ensuring no remaining patterns have higher support.
   (b) Top-K group CSP: Mining groups of CSPs with the highest k unique supports.
   (c) Top-K redundancy aware CSP: Mining top-K closed sequential patterns while considering redundancy among mined CSPs' sub-sequences occurrences.
2. **Mining algorithm and theoretical background:** We introduce the SP-Tree based mining algorithm—namely, *KCloTreeMiner*—capable of mining top-K CSPs for each addressed variation. We provide theoretical insights guiding the top-K mining framework and implementation optimizations. Additionally, we propose PaMHep as a heap-based efficient candidate pattern maintenance structure.
3. **Adopted and new pruning strategies:** We adopt the BF-S pruning strategy, which approximates the maximum possible support for patterns before complete SP-Tree projection. We also introduce a new pruning strategy—namely, *pattern absorption*—to further reduce search space. Additionally, we incorporate other relevant existing pruning strategies for sequential mining into KCloTreeMiner.
4. **Pattern summarization:** We propose three generic strategies:
   (a) Max_{WOC}: Maximal pattern based approach where summarized representative patterns do not need to be in the database.
   (b) Max_{WC}: Maximal pattern Based approach where all summarized representative patterns must be present in the database.
   (c) $CROID$: Centroid-based approach to find cluster centroids as pattern summarizers.
   We introduce two greedy algorithms for calculating Max_{WOC} and Max_{WC}. Additionally, we incorporate a new distance measure, *Subset Distance*, to improve performance by focusing on summarization and integrating it with centroid-based techniques.
5. **Evaluation:** We present a detailed, analytical and experimental discussion to evaluate our proposals by applying them to various real-life datasets.

The reminder of the current article is organized as follows. Section 2 discusses the relevant pieces of literature that match the problems presented in the current article. Alongside, we also formally present the variations associated with top-K closed sequential pattern mining with definitions and examples. Section 3 contains all of our proposals and their supporting reasoning. In Section 4, we evaluate our proposals based on various metrics and key points to highlight their efficiencies and novelties. Finally, we finish this study by providing our concluding remarks in Section 5.

2 Background study

In this section, we review relevant literature that lies close to our addressed variations of top-K closed sequential pattern mining. We further present relevant preliminaries, terminologies, and problem definitions.

2.1 Literature review

The domain of data mining focuses on extracting interesting information from data [3, 44]. It has various subdomains addressing multiple aspects, such as pattern mining [24, 30], clustering [8], association rule mining [13], outlier detection [45], and top-K mining [16]. This literature specifically examines sequential pattern mining (SPM), a pattern mining subdomain that aims to discover sequential relationships in a database. The SPM problem has several variations, including generic SPM [35, 36], weight-based SPM [23], uncertainty-based SPM [37], and parallel-environment SPM [22]. The current article focuses on the **top-K closed SPM** problem, a significant variation within the SPM domain.

The generic SPM problem aims to discover frequent sequential patterns from sequence databases. Various strategies have been developed to address this issue. GSP [38] applies candidate generation and testing but suffers from inefficiency due to generating many unnecessary candidates. PrefixSpan [33] uses pattern-growth and database projection, but the projection process can be costly. SPAM [4] employs a binary vector representation and bit operations, although storing bit information is resource-intensive. SPADE [52] converts the database to a vertical format but still relies on GSP's candidate generation method. SP-Tree [36] stores the

database in a tree structure, allowing for efficient projection and pruning despite requiring additional memory for the structural design. Other approaches, such as parallel mining algorithms, have also been proposed to improve efficiency [15].

A major issue in pattern mining is the explosion of candidate patterns when the minimum support threshold is low, especially as database size increases. To address this, much research has focused on mining closed frequent patterns rather than traditional frequent patterns [28, 31]. This concern extends to the generic SPM problem, prompting research into mining closed sequential patterns (CSPs) [14, 19, 39, 43]. Mining CSPs poses additional challenges compared to closed frequent pattern mining due to the need to maintain the order of entities in the dataset.

CloSpan [50] was one of the first algorithms for mining closed sequential patterns (CSPs) using candidate generation and testing. CloSpan generates a candidate set of closed sequential patterns, enumerates the search space, and performs post-pruning, but suffers from generating many unnecessary candidates similar to GSP. BIDE [43] introduced a bi-directional extension and back-scan strategy to find CSPs without maintaining candidates, but requires multiple database scans, resulting in an increased runtime. FCSM-PD [19] mines CSPs using positional information and a pattern growth approach, but incurs high memory usage and projection costs.

Inspired by SPADE's vertical representation [52], CloFAST used a vertical id-list-based data structure to calculate CSPs, but required significant memory for maintaining separate database snapshots during pattern extension [14]. FCloSM addressed this by eliminating and pruning non-CSPs early without testing supersequence or subsequence relationships among patterns in the prefix tree. Additionally, parallel computation techniques have been applied to mine CSPs [21].

Closed frequent pattern mining reduces the number of mined patterns but still requires a user-specified support threshold, which can be challenging to set without domain knowledge [41]. To address this, the top-K CSP mining problem was introduced [41]. In this approach, users specify a variable K, and the algorithms mine the CSPs with the highest K supports, avoiding the need for an empirical support threshold. This makes setting the threshold more accessible and interpretable than generic SPM or CSP problems. TSP proposed techniques for top-K sequential and CSP mining, outperforming CloSpan but suffering from costly recursive database projections [41]. TKS improved upon TSP by using a precedence map structure to reduce candidates and adopted SPAM's data representation, significantly enhancing runtime and memory efficiency [10].

TKCS mines top-K closed sequential patterns (CSPs) from the sequential database, ensuring that no remaining CSP has more support than the mined ones [34]. It employs a vertical bitmap-based data representation and uses two buffer lists, L and R. Between them, L holds patterns identified as CSPs or those to be replaced by their CSPs, while R contains candidate sets processed in descending order. The minimum support threshold at any moment is the lowest support observed in the L buffer, which controls candidate generation. According to the presented results, TKCS is more efficient than TKS.

TKCS has several concerns. For instance, TKCS lacks an efficient candidate storage structure, reducing performance during redundant pattern searches. Supports attached to patterns can vary widely, necessitating a presentation that allows efficient queries for the highest and lowest supports. Additionally, TKCS does not provide a theoretical framework for top-K mining and uses SPAM's bit vector-based projection strategy, which requires significant memory when transaction width or database breadth increases.

Other solutions exist but were not designed for generic top-K CSP mining. For instance, Wang et al. [42] mined top-K closed co-occurrence patterns in multiple streams using an exponential mechanism for candidate patterns, while Le et al. [27] mined top-K itemsets from uncertain databases. Additionally, the latter focused on top-K frequent itemset mining in uncertain databases, and proposes an algorithm for top-K high-utility sequential patterns, prioritizing high utility without preserving closedness.

This study addresses the generic top-K CSP mining problem along with two variations: the top-K group CSP problem aims to find CSPs with the highest K unique supports, while top-K redundancy-aware CSP aims to find top-K CSPs while minimizing redundancy in mined subsequences. We establish a theoretical framework for top-K mining, covering solution design, bounding, and optimizations. Additionally, we propose old and new pruning strategies and an efficient candidate maintenance heap structure.

Pattern summarization is a crucial problem and has always gained significant attention in the research community. Pan et al. [32] focused on designing an error-robust multi-view subspace clustering based on tensors. They used Cauchy pseudo norms, adaptive weights to approximate the true tensor rank by nonconvex tensor nuclear norm, and hypergraph embedding to preserve local structures. Guo et al. [17] proposed another multi-view clustering problem to process high-dimensional data. They also proposed a tensor-based adaptive consensus graph learning model. They solve the issue of neglecting high-order correlation while emphasizing the common representation issue of multiple tensor-based views. In this regard, they use consensus graph learning to assign weights to each similarity graph to form a unified graph matrix. Guo et al. [18] also worked on a multi-view clustering problem and addressed similar concerns. Their solution relied on learning consensus matrix from differ-

ent representations with adaptive weights. Yang et al. [51] worked on a multi-view unsupervised feature selection problem to reduce the dimensionality of heterogeneous data. Their approach included the learning of hypergraph Laplacian matrices for each view with adaptive weights to preserve the manifold structure.

Summarizing problems on sequential data also possesses a significant amount of complexity due to the underlying subsequence relationship among the entities [26, 40]. For instance, Kumar et al. [26] proposed two distance metrics to measure the degree of similarity between two patterns and applied that to conduct rough-set partitioning and fuzzy clustering over a set of patterns. In this study, we modify the distance measure, including a new metric to capture subset formation distance property to embed in the cluster centroids weighting the larger patterns more as the centroid candidates. We also propose two maximal pattern-based strategies and their sub-optimal greedy solutions to summarize a group of patterns. Clustering patterns and discovering the co-relations can be combined to improve the performance of various applications [1, 2, 46]. Ali et al. [2] proposed a deep dynamic attention-mechanism integrated neural network to find the spatio-temporal correlations within urban traffic crowd flows. As proposed by Wiroonsri [46], these correlation measures can guide the generation of various cluster-validating indexes leading to a better-quality cluster.

2.2 Preliminaries and problem definition

The important terminologies related to our problem are stated below. We also present three variations related to top-K closed sequential pattern mining (CSPM) problem afterward:

- Let $A = \{i_1, i_2, i_3, ..., i_m\}$ be an **itemset**, i.e., a set of items. An item is the smallest entity or unit observed in any problem. A is the set of all unique items observed in the problem.
- A **sequence** $s = \{e_1, e_2, e_3, ..., e_p\}$ is a list of ordered itemsets or events where each event $e_j \subseteq A$.
- A **database** $D = \{s_1, s_2, s_3, ..., s_n\}$ consists of multiple sequences.
- All the ordered sub-sequences of a sequence can be considered as a pattern P. A **pattern** $P = \{e'_1, e'_2, e'_3, \ldots, e'_p\}$ is contained in another pattern $Q = \{e''_1, e''_2, e''_3, \ldots, e''_q\}$ iff each event $e'_i \subseteq e''_j$ where $e'_i \in P$, $e''_j \in Q$ and each e'_i is observed in Q maintaining their respective order in P.
- The number of sequences $s \in D$ that contains P denotes the **support** S_P of P.

Definition 1 (Frequent sequential pattern mining problem (FSP)) The problem of mining Frequent Sequential Patterns (FSP) tries to discover the pattern(s) P that has $S_p \geq \delta$ where δ denotes a user-defined minimum support threshold. In this study, δ will represent the minimum support constraint.

Definition 2 (Closed sequential pattern mining problem (CSP)) If a pattern P contains another pattern Q and $S_P = S_Q$, then it is said that P encloses Q or P is a closed pattern, and Q is a non-closed pattern. The problem of finding CSPs focuses on discovering all the closed patterns P from D where for each P, $S_P \geq \delta$.

The problem of FSP and CSP is that both need a minimum support threshold parameter, δ, to mine the sequences. Only the patterns that have support at least δ are reported as the mined final set of patterns.

Definition 3 (Top-K generic closed sequential pattern mining (Top-K generic CSP)) The problem of mining top-K CSP focuses on discovering K closed sequential patterns with the highest supports where the remaining unreported CSPs do not have support more than the mined ones. Here K is a parameter set by the user.

For the discussion of the current article, we will use the database shown in Table 2. A group of symbols has been used in this literature to conduct the discussion. Table 1 contains the summarized list of symbols widely used during the discussion. There are mainly two types of symbols that are shown. Generic denotes generic symbols, and Tree indicates the symbols associated with the SP-Tree*.

For $K = 3$, for the database shown in Table 2, as top-K generic CSP, we might have a set as such $\{< (a, b)(c) >: 5, < (b, c) >: 5, < (a, b)(b, c) >: 4\}$. There can be another set of reported CSPs as well, for example, $\{< (a, b)(c) >: 5, < (b, c) >: 5, < (a, b)(d) >: 4\}$ or $\{< (a, b)(c) >: 5, < (b, c) >: 5, < (c)(c) >: 4\}$, etc. The remaining unmined CSPs do not have more support than the mined ones. Items enclosed by the same first bracket denote that they are in the same event.

Definition 4 (Top-K group closed pattern mining (Top-K group CSP)) The problem of mining top-K Group CSP focuses on discovering all the closed sequential patterns with the highest K unique supports observed in the database. Here K is given as input to the problem.

As before, for $K = 3$ for the database shown in Table 2, as top-K group CSP, the patterns we obtain are shown in Table 3. These patterns fall in the set of having the highest K unique supports.

Definition 5 (Top-K redundancy aware closed pattern mining (Top-K redundancy aware CSP)) The problem of

Table 1 Summarized list of the most used symbols and short terms

Type	Symbol	Description
Generic and Tree	A and I	I denotes an item and A denotes itemset
Generic	s and D	s denotes a sequence and D denotes the database
Generic	P	Pattern
Generic	S_P	Support of pattern P
Generic	N_P	Projection Nodes of pattern P
Generic	δ	Minimum support threshold constraint for P
Generic	$\|...\|$	Bar symbol denotes the size of a list or a string
Generic	(...)	Dot operator is used to access the attributes.
Generic	[...]	This operator is used to access array like elements. 0 based indexing is followed.
Generic	SE	SE denotes the short form of Sequence Extension
Generic	IE	IE denotes the short form of Itemset Extension
Tree	ev	Event number represented by each node of SP-Tree*
Tree	C	Count attribute of each node of SP-Tree*
Tree	nl	Next link attribute of each SP-Tree* node N comprised of $N.nl_S[\gamma]$ and $N.nl_E[\gamma]$ denoting the start and end pointer for an item γ
Tree	$N^s(\gamma)$	Set of nodes in the underlying subtree of SP-Tree* Node N reachable using next links for item γ
Tree	id	Denotes node id for each SP-Tree* node
Tree	PB_{INF}	Denotes parent info attribute
PaMHep	Can	A buffer of candidate patterns in a PaMHep node
PaMHep	Clo	A buffer of closed patterns in a PaMHep node

mining top-K CSP focuses on discovering K closed sequential patterns where the remaining unreported CSPs either do not have more support than the mined ones or it is a subset of a reported CSP.

For $K = 3$ as top-K redundancy aware CSP we might have $\{< (a,b)(b,c) >: 4, < (a,b)(d) >: 4, < (c)(c) >: 4\}$ or $\{< (a,b)(b,c) >: 4, < (a,b)(d) >: 4, < (c)(d) >: 4\}$, etc. The unmined sequences will belong to two sets—either they will have lesser support than the mined ones or they will be a subset of the mined patterns.

Recall the primary motivation behind addressing the top-K framework over the minimum support threshold parameter is that it is difficult to understand the distribution of patterns of a dataset prior without any domain-specific knowledge. Setting δ is difficult compared to setting K. Because δ denotes the actual minimum support expected for each pattern, K is focused on the number of patterns or unique supports.

Table 2 Example sequence database

SID	Sequences
1	<(ab)(e)(bc)(d)(ef) >
2	<(df)(abc)(c)(cd) >
3	<(cd)(ab)(d)(bc) >
4	<(ab)(cd)(d)(bc)(d) >
5	<(c)(ab)(c)(bc)(e) >

3 Proposed method

This section will discuss our proposed strategies to solve the top-K CSP mining problem over the basic tree-based structure SP-Tree [36].

Table 3 Top-3 group closed sequential patterns

Support	Pattern	Support	Pattern
5	<(ab)(c) >	3	<(ab)(c)(d) >
5	<(bc) >	3	<(bc)(d) >
4	<(ab)(bc) >	3	<(cd) >
4	<(ab)(d) >	3	<(c)(bc) >
4	<(c)(c) >	3	<(d)(bc) >
4	<(c)(d) >	3	<(d)(b)(d) >
3	<(ab)(c)(c) >		

3.1 Modified sequential pattern tree (SP-Tree*)

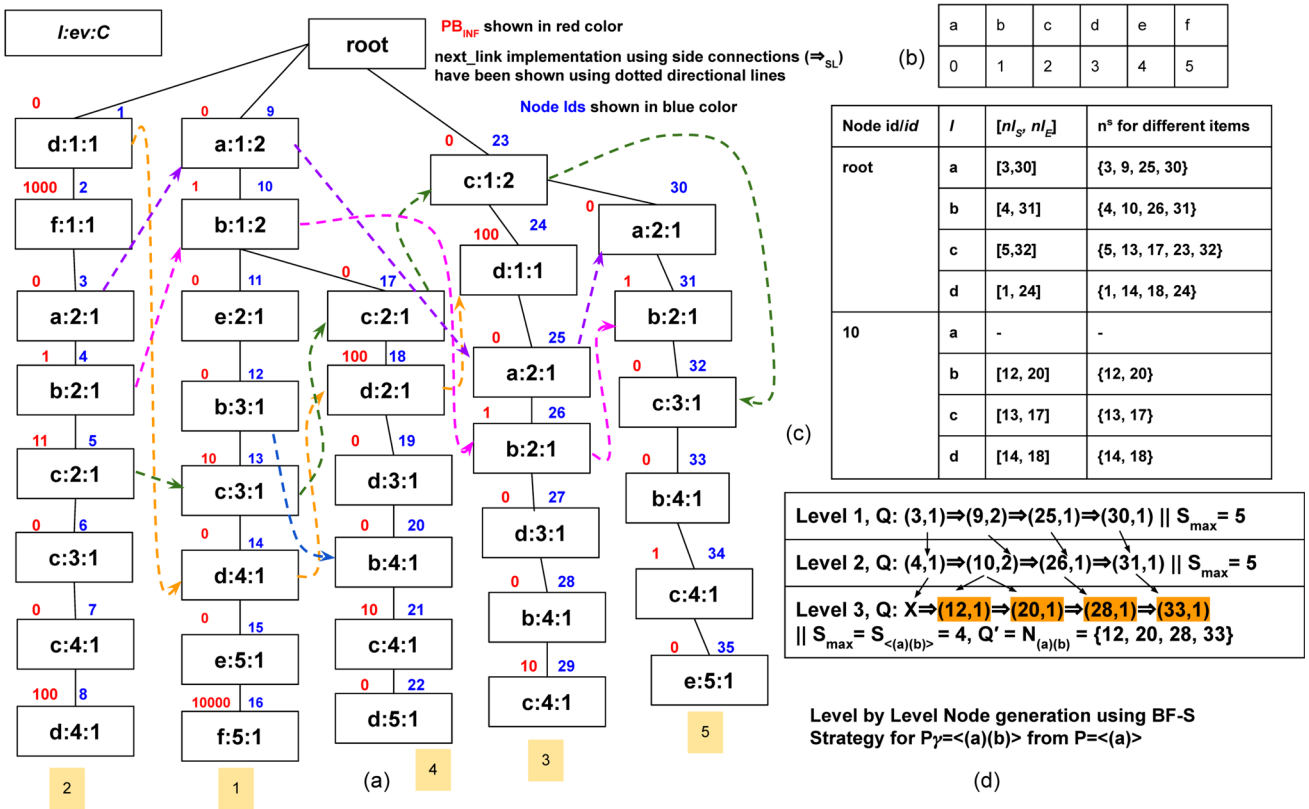
In this study, we modify some features of our previously proposed tree-based structure SP-Tree [35, 36]. First, we will provide a brief discussion of the existing attributes. Then, we will discuss the modified part compared to the previous one. To point out expressly, we devise a new strategy to improve the calculation of the prior proposed Next Link attribute.

We introduce the attributes directly fetched from the SP-Tree to SP-Tree* as follows:

1. Item I: A node N will represent an **item** I of some sequence. For the overlapping nature, the nodes or items will be shared.
2. Event number ev: A node N will contain the specific **event number** ev associated with the concerned item found in the database.
3. **Count** C: It indicates the number of times a node N is overlapped during sequence insertion.
4. Child node(s) ch: A non-leaf node N of the tree will have **child node(s)** ch for unique $\{I, ev\}$ combination.
5. Parent information PB_{INF}: A node N will contain the **parent information** PB_{INF} of the items found in its ancestor nodes having the same ev as it in the bitset form.
6. Next links nl: For a node N, using **next links** nl for an item γ, we can reach a set of nodes in the underlying subtree of N, $N^s(\gamma)$ where $\forall n \in N^s(\gamma), n.I = \gamma$. Although Rizvee et al. [36] introduced the idea of next links to perform faster traversal in the tree, their implementation was not stated. Hence, in the current study, we propose *side connection-based* implementation to reuse information and reduce memory.

Definition 6 (Side connection-based next links) A node N using side connection ($N \rightarrow_{SL}$) points to another node M where $N.I$=$M.I$. N also holds two node pointers, $nl_S[\gamma]$ and $nl_E[\gamma]$ for an item γ. Traversing from $nl_S[\gamma]$ to $nl_E[\gamma]$ using the side connections, complete $N^s(\gamma)$ can be extracted.

In Fig. 1(a), the corresponding SP-Tree of the sample database shown in Table 2 has been presented. Each database sequence can be found by traversing the paths from the SP-tree's root. For each node N its corresponding I, ev, C, and



a	b	c	d	e	f
0	1	2	3	4	5

Node id/*id*	*I*	$[nl_S, nl_E]$	n^s for different items
root	a	[3,30]	{3, 9, 25, 30}
	b	[4, 31]	{4, 10, 26, 31}
	c	[5,32]	{5, 13, 17, 23, 32}
	d	[1, 24]	{1, 14, 18, 24}
10	a	-	-
	b	[12, 20]	{12, 20}
	c	[13, 17]	{13, 17}
	d	[14, 18]	{14, 18}

Fig. 1 (a) SP-Tree* for the database shown in Table 2. (b) Item table to calculate PB_{INF} attribute. (c) Next Links' implementation using side connections. (d) Example of Breadth-First based Support Counting Strategy for $< (a)(b) >$ from $< (a) >$. For each entry, the node id and corresponding count attribute is shown

PB_{INF} values are shown. The nodes are also numbered in blue, denoting the node ids or id attribute. To store the value of PB_{INF}, the table shown in Fig. 1(b) has been used; for each item, the corresponding bits are set. For example, for node 5, the value is 11 (0^{th} and 1^{st} bit are set) because it has items a and b in nodes 3 and 4 having the same ev as it.

Implementation of next links using side connections ($\rightarrow_{SL}$) has also been shown for some nodes in Fig. 1(c). For example, if we consider the root node, for item a, it stores $nl_S[a]$ and $nl_E[a]$ as 3 and 30, respectively. Using side connections, we can extract complete set of nl for root for item a, $3 \rightarrow_{SL} 9 \rightarrow_{SL} 25 \rightarrow_{SL} 30$. To calculate the nl attribute using side connections, we apply a single in-order traversal of the tree in a depth-first manner to create an ordering among the nodes. This node ordering is used during mining while applying pruning strategies.

3.2 Pattern generation from SP-Tree*

The current article follows a similar strategy to generate patterns from the SP-Tree* [36]. For each pattern P, its projection nodes N_P are the first occurrence of nodes where each concatenation ends in different disjoint subtrees and support S_P=$\sum_{n \in N_P} C(n)$. In Fig. 1, for $P =< (ab) >$, $N_P = \{4, 10, 26, 31\}$ and S_P=$1 + 2 + 1 + 1 = 5$.

Let a pattern P ends at a node A and another node B lying in the underlying subtree of A having $I = \gamma$. Pattern extension (PE) conditions are given as follows:

- Sequence extension (SE: $P \rightarrow P\{\gamma\}$): B can extend A as SE for γ iff $A.ev < B.ev$. An occurrence of P=$< (a) >$ can be found in A=3 in Fig. 1(a). B=6 can extend A as SE for γ=C, as $A.ev < B.ev$. B is an occurrence for $< (a)(c) >$.
- Itemset extension (IE: $P \rightarrow \{P\gamma\}$): B can extend A as IE for γ iff $A.ev = B.ev$, and γ is lexicographically larger compared to all the items of the last event of P and $B.PB_{INF}$ will contain all those items as a bitset. For example, pattern $P =< (b) >$ has two occurrences in node A=10. Node B=13 having γ=c cannot extend A as IE($A.ev \neq B.ev$) but can extend A=12 because now $A.ev$=$B.ev$=3 and $B.PB_{INF}$ contains item b in set bit(s) and $\gamma = c > b$.

Pattern generation will be performed using the next links through the $BF - S$ strategy [35]. We omit the completeness of the pruning strategy as it has already been discussed in the earlier literature [35]. Intuitively, this strategy approximates a pattern's maximum possible support in each step without projecting completely. So, based on this temporary projected condition, we can make various decision factors related to top-K mining. We present the idea of BF-S with temporary SP-Tree projected nodes in Definition 7. An illustrative example has been shown in Fig. 1(d) to present how SP-Tree nodes are explored level by level and possible support is updated.

Definition 7 (BF-S with temporary SP-Tree projected nodes) To extend a pattern P having S_P to $P\gamma$ from N_P, nodes are processed using a queue in first-in-first-out (FIFO) order. Let Q=N_P, $S^{RS}_{P\gamma}$=S_P. A node $n \in Q$ is removed. If n satisfies PE, it is stored in Q' that represents $N_{P\gamma}$, else complete $n^s(\gamma)$ is added in Q and $S^{RS}_{P\gamma}$=$S^{RS}_{P\gamma} - C(n) + \sum C(n^s(\gamma))$ is updated for further processing. Here $\sum C(n^s(\gamma)$ denotes the summation of the count attribute of all the underlying subtree nodes reachable using the next links for γ of n. Additionally, at any step, $N^T_{P\gamma} = Q \cup Q'$ denotes the set of temporary projected nodes for $P\gamma$.

As per Definition 7, for a support threshold δ, if at any moment $S^{RS}_{P\gamma}$ is less than δ, we omit projecting further and halt with $N^T_{P\gamma}$. During top-K mining, we can prioritize the ordering of the extended patterns based on this strategy. For any node $n \in N^T_{P\gamma}$, either it is an actual projected node for $P\gamma$ or it may need further projection in the underlying subtree.

For example, for pattern $P =< (ab) >$, its corresponding N^T_P can be {4, 9, 26, 31}, where 4, 26 and 31 are actual projection nodes for $< (ab) >$ in different subtrees. However, node 9 needs further searching in the underlying subtree to get the actual projection nodes. When the projected nodes are discovered, they are always processed in the order of their appearance. We replace a node with its underlying subtree, but the in-between node ordering is never changed. Lefter nodes will always be lefter (earlier positions), and righter nodes will always be righter (later positions) in the temporary projection node list. We use a linked list to implement this idea to maintain node ordering.

3.3 Pruning strategies

This section summarizes the adopted pruning strategies and then states the newly proposed strategy *Pattern Absorption*. We also discuss its formal proof and other related theoretical concepts. Let $sList$ and $iList$ for a pattern P denote which items may extend P as SE and IE respectively and have $S_P \geq \delta$.

1. Co-occurrence map (CMAP) [9]: If $S_{(\beta)(\alpha)} < \delta$ or $S_{(\gamma)(\alpha)} < \delta$ or both then always $S_{(\alpha)(\beta\gamma)(\alpha)} < \delta$. Similarly, if $S_{(\beta)(\gamma)} < \delta$ then $S_{(\alpha\beta\gamma)} < \delta$. Observing the two-length occurrences, we can apply this pruning.
   Implementation strategy: We update CMAP Table during mining on the fly. For example, we are trying to calculate the support value of $(\alpha)(\gamma)$ (a two-length pattern extension). Through applying the BF-S strategy, we

either calculate the complete projection of the pattern or early prune the searching and extract the temporary projection nodes. In the first case, we get the actual possible support of $(\alpha)(\gamma)$ and in the second case, we get an upper bound regarding the maximum possible support $S^{RS}_{\alpha\gamma}$. This value guides the later phase of the mining. For example, before calculating the projection nodes of $< (\alpha)(\beta)(\gamma) >$, we can check if $(\alpha)(\gamma)$ satisfies our desired support threshold or not.

2. Recursive sList & iList pruning [4] : If $S_{(\alpha)(\beta)} \geq \delta$ and $S_{(\alpha)(\gamma)} < \delta$, then $S_{(\alpha)(\beta\gamma)} < \delta$ (iList prune) and $S_{(\alpha)(\beta)(\gamma)} < \delta$ (sList prune). Observing the extended patterns' sLists and iLists we can compact the sLists and iLists during next level extensions.

Theorem 1 (**Pattern absorption** ($Q \subseteq_A P$)) *Let a pattern P enclose another pattern Q, denoted as $Q \subseteq_E P$. If $\forall x \in N_P$, a node $y \in N_Q$ exists where x is in the subtree of y having same ev, then P absorbs Q, and Q does not need SE. Then, it can be regarded as P absorbs Q, denoted as $Q \subseteq_A P$.*

Proof We know that P is a closed pattern of Q. So, $S_P = S_Q$. Also, for each $y \in N_Q$, a $x \in N_P$ is found where they lie in the same event, and x is in the subtree of y. So, there are two possible cases:

- Case 1: Nodes are same ($y = x$) or different nodes but no branch separation ($c(y) = c(x)$).
- Case 2: Multiple branch creation after occurrence of Q but P can be found in each branch.

In each case, to perform SE using the next links for γ, we must reach such nodes where we exceed the current event number. Both for P and Q to reach such nodes, there can be at most two-level next link traversals (First in the same event nodes, second to the nodes exceeding the current event number). If P needs two level traversals, then Q will also need two level next link traversals (For example, $Q = < (a) > \rightarrow \ Q = < (a)(d) >$ and $P = < (ac) > \rightarrow \ P = < (ac)(d) >$). However, if P needs one level traversal, then Q will also need one level traversal if the last item of both of the pattern matches (For example, $Q = < (c) > \rightarrow \ Q = < (c)(d) >$ and $P = < (ac) > \rightarrow \ P = < (ac)(d) >$). However, Q might need two level traversals if the last item is lexicographically smaller (For example, $Q = < (a) > \rightarrow \ Q = < (a)(b) >$ and $P = < (ac) > \rightarrow \ P = < (ac)(b) >$). □

Intuitively, Theorem 1 states that, if two different patterns have the same set of projected nodes, then for SE they will move similarly in SP-Tree*. For example, in Fig. 1(a), $Q \subseteq_E P$ where P=$< (a)(b) >$, Q=$< (b) >$ and $S_P = S_Q =$ 5, N_P={4, 10, 26, 31} and N_Q={4, 10, 26, 31}. $\forall x \in N_P$ a valid $y \in N_Q$ can be found. So, $< (b) > \subseteq_A < (ab) >$ resulting in no SE required for $< (b) >$.

3.4 Pattern Max Heap (*PaMHep*)

This section presents our proposed candidate maintenance heap-based structure Pattern Max Heap (*PaMHep*). We discuss the node attributes, corresponding properties of the structure and relevant formal arguments.

For each unique support, there will be a dedicated node in PaMHep, and the nodes will be created on the fly during mining iterations. PaMHep nodes will be ordered based on the support values. It is a max heap-based structure, so the node with the current highest support will be the root of the binary tree. Before addressing the node attributes of PaMHep, we talk about *Pattern Block* that comprises the lowest unit entity in our framework.

Definition 8 (**Pattern Block (Pb)**) A pattern Block is a linked list node pointing to the next and previous pattern blocks. Each pattern block has the following attributes:

1. Pattern $Pb.P$: Each pattern block Pb represents a **pattern** P.
2. Projection nodes $Pb.N_P$: For the P, each pattern block stores its **projection SP-Tree pointer nodes** (or **projection nodes** for short) N_P where the P has ended in different disjoint subtrees as their first occurrences. N_P can be stored in a raw format, or we can use bit-based representation by setting the corresponding bits of the positions representing node ids. For example, for $P = < (ab) >$, its N_P can be either a raw list {4, 10, 26, 31} or a number $(1 << 4) + (1 << 10) + (1 << 26) + (1 << 31) = 16 + 1024 + 67108864 + 2147483648 = (2214593552)_{10}$.
3. Projection status $Pb.N^S_P$: To state the condition of the projected node, we maintain a corresponding bit character string to capture **projection status** N^S_P. So, if the size of N_P ($|N_P| = 4$) is 4, then the size of N^S_P will be 4 as well. If the entry is 1 for each position, the corresponding projected node in N_P is an actual projected node in some subtree. However, if the value is set to 0, the corresponding node in N_P needs further searching in its underlying subtree. For example, for $P = < (ab) >$ if we have $N_P = \{4, 10, 26, 31\}$ then $N^S_P =$ "1111". However, if $N_P = \{3, 9, 26, 31\}$ then $N^S_P =$ "0011".
4. $Pb.S_{EX}$ and $Pb.I_{EX}$: For each pattern P, there will be two lists of items that denote which items or symbols may extend P as sequence extension SE(S_{EX}) and itemset extension IE(I_{EX}), respectively.
5. $Pb.F_{CL}$ and $Pb.F_{SE}$ flags: For a pattern, $F_{CL} = 1$ indicates that the pattern can be closed, while $F_{CL} = 0$ means

it can never be a closed pattern. Initially, the value is set to 1. If a pattern P is encompassed by another super pattern with similar support, F_{CL} is set to 0. Likewise, $F_{SE} = 1$ signifies that this pattern requires SE to be performed, and $F_{SE} = 0$ indicates that SE is not needed. The pattern absorption pruning strategy changes F_{SE} from 1 to 0.

6. $Pb.Prev$ and $Pb.Next$: Pattern blocks are used as linked list nodes in a chain of nodes in our solution. So, there are two links denoting the next($Next$) and previous($Prev$) pattern blocks. These links maintain ordering among pattern blocks in linked list implementation.

Each pattern-related entry stored in the PaMHep node is done using pattern blocks. The node attributes of PaMHep are stated in Definition 9.

Definition 9 (**Pattern Max Heap Node** ($PaMHep_N$) Each $PaMHep_N$ holds the following attributes:

1. Support $PaMHep_N^S.S$: Each node represents a unique **support** S. All the patterns, aka pattern blocks stored in this node, are relevant to this support. The relevant PaMHep node bearing support S is denoted by $PaMHep_N^S$.
2. Candidate($PaMHep_N^S.Can$) pattern blocks: Each candidate pattern or pattern block of relevant support are stored against this key in the node. Pattern blocks are separated based on their lengths. This means that $Can[x]$ holds a linked list of pattern blocks where patterns are of length x.
3. Closed($PaMHep_N^S.Clo$) pattern blocks: This attribute behaves exactly similar to the last attribute Can except it holds the found closed patterns. As before, $Clo[x]$ holds a linked list of pattern blocks representing the found closed patterns of length x of support S.

The main properties of $PaMHep$ consisting of $PaMHep$ nodes are stated below.

Properties 1. (Insertion) Patterns or pattern blocks are always inserted at the end of the current linked list of the $PaMHep$ node's Can or Clo's buffer where the length of the pattern matches. The big-O complexity to insert a pattern will be $\mathcal{O}(1)$ and to insert a $PaMHep$ node will be $\mathcal{O}(log\overline{S})$ where $\overline{S}$ denotes the maximum possible support in the database.

Properties 2. (Pop or Deletion) The general behavior to delete a pattern from $PaMHep$ will be to delete the pattern or pattern block from the $PaMHep$ node at the root (node bearing the current highest support). Then during deletion, pattern blocks are removed in FIFO order in ascending order over the lengths of the patterns. It means that the patterns of smaller sizes are first popped or deleted, whereas the earlier inserted patterns will be removed earlier. In general, the big-O complexity of deleting a pattern will be $\mathcal{O}(1)$ and the big-O complexity to delete a $PaMHep$ node will be $\mathcal{O}(log\overline{S})$. However, due to linked list-based implementation, the deletion operation of any pattern block from any arbitrary position will take $\mathcal{O}(1)$ complexity. Let the function that executes this task be *DeleteFromPaMHep*. We will be using this function in our pseudocode.

Let us consider an example to illustrate this concept. In Fig. 2, we display the status of PaMHep nodes across different iterations during the mining process. When processing a candidate pattern, we always examine the root node (which contains the patterns with the highest current support value). In iteration 1, we start with the PaMHep node labeled *A* with

itr	PaM Hep$_N$	Can	Clo	Count
1	A:5	1: { ‹(a)›, ‹(b)›, ‹(c)›}	-	K_{GE}:0 K_{GR}:0 K_{RA}:0
	B:4	1: {‹(d)›}	-	
	C:2	1: {‹(e)›,‹(f)› }	-	
5	A:5	2: {‹(ab)›, ‹(bc)›}	GE/GR/RA: 2:{‹(a)(c)›}	K_{GE}:1 K_{GR}:0 K_{RA}:1
	B:4	1: {‹(d)›}		
		2: {‹(a)(b)›, ‹(a)(d)›, ‹(a)(e)›, ‹(a)(f)›, ‹(ac)›, ‹(ad)›, ‹(ae)›, ‹(af)›, ‹(bd)›, ‹(be)›, ‹(bf)›, ‹(c)(a)›, ‹(c)(b)›, ‹(c)(c)›,‹(c)(d)›, ‹(c)(e)›, ‹(c)(f)›, ‹(cd)›, ‹(ce)›, ‹(cf)›}		
		3: {‹(a)(c)(a)›, ‹(a)(c)(b)›, ‹(a)(c)(c)›, ‹(a)(c)(d)›, ‹(a)(c)(e)›, ‹(a)(c)(f)›, ‹(a)(cd)›, ‹(a)(ce)›, ‹(a)(cf)›}		
	D:3	2: {‹(a)(a)›}		
	C:2	1: {‹(e)›,‹(f)›}		

Itr	PaM Hep$_N$	Can	Clo	Count
15	A:5	-	GE/GR/RA: 2: {‹(bc)›} 3:{‹(ab)(c)›}	K_{GE}:3 K_{GR}:1 K_{RA}:3
	B:4	2: {‹(ad)›, ‹(ae)›, ‹(af)›, ‹(bd)›, ‹(be)›, ‹(bf)› ‹(c)(a)›, ‹(c)(b)›, ‹(c)(c)›,‹(c)(d)›, ‹(c)(e)›, ‹(c)(f)›, ‹(cd)›, ‹(ce)›,‹(cf)› }	GE/GR/RA: 2: {‹(a)(d)›}	
		3: { ‹(a)(c)(a)›, ‹(a)(c)(b)›. ‹(a)(c)(c)›, ‹(a)(c)(d)›, ‹(a)(c)(e)›, ‹(a)(c)(f)›, ‹(a)(cd)›, ‹(a)(ce)›, ‹(a)(cf)›, ‹(ab)(b)›, ‹(ab)(d)›, ‹(ab)(e)›, ‹(ab)(f)›, ‹(abc)›, ‹(abd)›, ‹(abe)›, ‹(abf)›, ‹(bc)(b)›, ‹(bc)(c)›, ‹(bc)(d)›, ‹(bc)(e)›, ‹(bc)(f)›, ‹(bcd)›, ‹(bce)›, ‹(bcf)›, ‹(a)(bc)›}		
		4: {‹(ab)(c)(a)›, ‹(ab)(c)(b)›,‹(ab)(c)(c)›, ‹(ab)(c)(d)›, ‹(ab)(c)(e)›, ‹(ab)(c)(f)›, ‹(ab)(cd)›, ‹(ab)(ce)›, ‹(ab)(cf)›}		
	D:3	2: {‹(a)(a)›, ‹(d)(a)›, ‹(d)(b)›,‹(d)(c)›, ‹(d)(d)›, ‹(d)(e)›, ‹(d)(f)›, ‹(de)›, ‹(df)›, ‹(a)(e)›, ‹(a)(f)›, ‹(ac)›}		
		3: {‹(ab)(a)›, ‹(bc)(a)›,‹(a)(b)(a)›, ‹(a)(b)(b)›, ‹(a)(b)(c)›, ‹(a)(b)(d)›, ‹(a)(b)(e)›, ‹(a)(b)(f)›, ‹(a)(bd)›, ‹(a)(be)›, ‹(a)(bf)›,‹(a)(d)(a)›, ‹(a)(d)(b)›, ‹(a)(d)(c)›, ‹(a)(d)(d)›, ‹(a)(d)(e)›, ‹(a)(d)(f)›, ‹(a)(de)›,‹(a)(df)›}		
	C:2	1: {‹(e)›,‹(f)›}		

Fig. 2 Some of the mining iterations of KCloTreeMiner during Top-3 CSP mining

a support of 5. When dealing with patterns of varying lengths, we prioritize those with shorter lengths. Thus, in iteration 15, when we consider node B, we first process patterns of length 2 before moving on to patterns of lengths 3 and 4.

In Fig. 2, while explaining our mining algorithm KCloTreeMiner, we present the status of PaMHep for different nodes with their corresponding attributes' values, e.g., support, closed and candidate buffer. This figure, mainly presents how patterns are organized in different blocks for different nodes in PaMHep to bring efficiency in searching.

3.5 KCloTreeMiner

In this section, we first introduce the generic principles related to our proposed mining algorithm KCloTreeMiner. Then, we present the logical reasoning with supporting arguments behind the crucial steps of our solution sketch. Afterward, we present the mining algorithm KCloTreeMiner and other important points related to the three addressed variations. KCloTreeMiner can address all three variations. The discussion is segmented into separate sub-sections for clarification.

3.5.1 Formal reasoning and behavioral characteristics relevant to top-K mining framework

The generic principles or properties of our mining approach are summarized as follows:

Properties 1. (Pattern growth approach) We follow the pattern growth approach. We apply a one-length pattern forward or suffix extension during mining iterations in each step. For example, $< (c)(d) > \rightarrow < (c)(d)(e) >$ or $< (c)(de) >$, etc.

Properties 2. (Setting support threshold δ): During the extension of a pattern P having support S for another item β we set a pseudo support threshold $\delta \leftarrow S$. While applying the BF-S strategy to find the projected nodes, whenever we see that the maximum possible support of the extended pattern has fallen below δ, we immediately stop further projection and save the temporary projected nodes along with other necessary information of $P\beta$ as a pattern block in the corresponding PaMHep node's candidate buffer. It follows the insertion property of PaMHep, stated in the earlier section.

Properties 3. (Candidate choosing criteria) The pattern with the maximum support and the least length is always extended first. In the case of a tie, we use the FIFO strategy. As stated in the earlier section, the insertion and deletion property of the PaMHep node ensures this by default.

Properties 4. (Closed pattern detection criteria) While applying the pattern growth property over a pattern P with support S, we check whether any of the extended patterns have gained the support S. If there exists at least one such pattern, P will never be a CSP. Again, if there is no other pattern, then P can be a CSP if no different super pattern P' has already ruled out P by enclosing it. It can also occur that, during following mining iterations, some super pattern may arrive and nullify P's CSP property.

Properties 5. (Minimum support fixation for $PaMHep$ and redundant items removal) While tracking the initial one-length patterns, if we observe that we have more than K unique supports, then we apply two types of strategies as pruning mechanisms in top-K generic and group CSP mining problems. First, we remove all items with support lesser than top-K supports. These items do not contribute to any pattern extensions. Second, we fix a bounding that can be the minimum support possible in any PaMHep node.

Next, we discuss some formal reasoning that guide the steps of our proposed KCloTreeMiner algorithm.

Theorem 2 (**Closed coverage**) *For a found closed pattern P of length m and support S, the algorithm checks in $PaMHep_N^S.Clo[x], \forall x < m$ to see if any prior determined closed pattern is enclosed by P. If it does, those patterns are removed from $PaMHep_N^S.Clo$.*

Proof Suppose we have found pattern P of length m having support S enclosed by another super pattern Q of length n having support S ($m < n$). So, by definition, P cannot be a CSP and must be removed from tracking. Based on Property 3, P will be processed earlier, and Q will be processed later. Also, based on Property 4, P can be detected as CSP during earlier mining iterations. P can be of any length between 1 to $(n-1)$. Thus, to remove P, we need to search in the buffer of closed patterns in all smaller lengths of $PaMHep_N^S$. Another point to note is that we cannot order between two different patterns of the same length and same support to identify which pattern will give a larger length CSP prior without completing the extensions, let alone between the patterns of different lengths. □

Theorem 3 (**Candidate search space behavior control**) *For a found closed pattern P of support S of length m, the algorithm searches in $PaMHep.Can[m-1]$ to find its $(m-1)$ length sub-patterns of similar support. Each such pattern will never be a closed pattern ($F_{CL} \leftarrow 0$), and if they are absorbed by P, then they do not require SE ($F_{SE} \leftarrow 0$).*

Proof We have already stated that we follow property three during mining iterations. P's $(m-1)$ length sub patterns will have support greater or equal to S. So, if they have larger support than S, they will be automatically processed earlier. However, if they have support equal to S, then they will be in the $(m-1)$ length candidate buffer in $PaMHep_N^S$. These patterns will never be CSPs by definition. So, each F_{CL} flag is set to 0. Among them, if some sub-patterns are absorbed by P (enclosed and projection nodes exhibit similar extension

characteristics) as stated in Theorem 1, they do not require SE as well. That is why for them F_{SE} flag will be set to 0.

Theorems 2 and 3 altogether ensure that no pattern except closed is stored in the buffers (candidate and closed) of PaMHep for a particular support S. Theorem 3, also applies pattern absorption pruning strategy by setting corresponding flags, F_{SE} to zero. Due to the efficient management of PaMHep, only closed and candidate buffers are used by Theorems 2 and 3, respectively.

Theorem 4 (**Finding all the CSPs to ensure correctness**) *The algorithm mines all the CSPs of a particular support S to ensure correctness in finding the valid set of CSPs.*

To illustrate Theorem 4, let us consider three patterns P_1, P_2 and P_3, each with support S and none enclosing the others. Additionally, consider a super pattern P^* with the same support S that encloses at least one of P_1, P_2 and P_3. According to Theorem 2, it is impossible to determine an order among the patterns that will construct P^* first. Thus, it cannot be predetermined which pattern P' will eventually extend to P^*. Therefore, to ensure completeness in identifying the full set of closed patterns (CSPs) for a given support S, it is necessary to find all patterns with support S to encompass the sub-patterns within the larger patterns.

Theorem 5 (**Unbounded property of** $PaMHep$) *PaMHep cannot be bounded with the K nodes representing the highest K unique supports at any moment.*

Proof The reasoning is that all the patterns are not fully projected during mining. During extension, if a pattern P with the current maximum possible support S falls under the desired support threshold δ, its projection is stopped, and temporarily projected nodes are again stored in PaMHep in a node representing lesser support than S. This reduced support may or may not be P's actual support.

Suppose we have more than K nodes, and we decide to prune the nodes bearing support less than higher K supports. In that case, we might prune some valid candidates because the patterns now expected to have better support might get lesser during further projection and become suitable patterns for those nodes.

As we progress with BF-S and temporal projection nodes, the current support S for a pattern P cannot be identified as its final support. So, PaMHep keeps nodes representing the candidates and closed patterns of different supports not necessarily bounded by K unique or highest support values. For example, we can consider this scenario, from the max support 5 during extension, we get patterns with three lesser supports 4, 3 and 2. All the extended patterns will not be projected entirely. The patterns expected to have support 4 or 3 may get reduced to 2 or lower at some point. If in the top-3, we ignore patterns with support two completely; we might remove some very valid expected CSPs as per definition.

3.5.2 Mining algorithm: KCloTreeMiner

Now, accumulating all the ideas stated up to now; we state the proposed mining approach KCloTreeMiner in Algorithm 3. The primary mining principle of each of the three variations is very similar. That is why KCloTreeMiner can address each of them. In Algorithms 1 and 2, we have written some utility and pattern extension functions widely used by KCloTreeMiner.

In Algorithm 1, in lines 1-5, we show a procedure named BFSProjection to find the projected nodes of a pattern P using the strategy stated in Theorem 7. Gradual progression is done using a queue, and a linked list is used to maintain the order among the projected nodes. Anytime support reduces from the expected threshold δ; the searching stops with the temporary or final set of projected nodes N_P with a bounding over the maximum possible support S^P. In lines 5-11, we write a procedure to create a new pattern block and store its attribute values based on Definition 8. In lines 12-17, we write a procedure to find which candidates are enclosed and afterward absorbed by the passed pattern block Pb based on Theorem 3. Similarly, in lines 18-25, we write a procedure to find which prior considered closed patterns within a specific support S can be enclosed by the pattern block Pb based on Theorem 2. We also update the mined pattern counter variables in line 25, denoting that the number of mined CSPs in the generic and redundancy-aware framework has reduced by 1.

In Algorithm 2, we write a procedure that denotes how we extend a pattern for a list of items based on the extension type, Ex. First, we apply CMAP pruning based on the discussion presented in Section 3.3. Suppose the possible support falls under δ. In that case, we prohibit searching in the SP-Tree, create a new pattern block with the reduced support and temporary projected nodes for the extended pattern, and insert it into PaMHep (lines 9-12). In the other case, we apply projection using the procedure BFSProjection to search for the projected nodes in SP-Tree (line 13). If the projected pattern $P\gamma$ has support equal to δ, we have found a CSP that encloses P, so P can never be a CSP, we update F_{CL} flag of P (line 15). In line 16, we create a new pattern block Pb' for $P\gamma$ and insert it into PaMHep. If all the projected nodes are completely projected meaning all the bits in $N^S_{P\gamma}$ are set, we apply closed coverage and candidate coverage (lines 18-19). Finally, we return the list of newly created pattern blocks with their corresponding extended symbols (line 20).

In Algorithm 3, we first extract the 1-length items from the SP-Tree (line 5). Then, the items are sorted, and we track the items with the highest K unique supports; we remove the items with lesser support and set a bounding over PaMHep regarding the minimum support (line 6). In the case of generic and group mining problems, PaMHep will never insert any node with support less than this bound. For simplicity, we have omitted frequent checking of this strategy. We set the

Algorithm 1 A set of utility functions.

1: **procedure** BFSPROJECTION(Pattern P, Projection Nodes N_P, Projection Status N_P^S, Support Threshold δ)
2: Let γ denotes the last item of the pattern to be extended. Ex denotes the extension type as SE or IE
3: Apply projection using BF-S Stated in Theorem 7. For the SP-Tree nodes $n \in N_P$ where projection is not complete, we search in its underlying subtree keeping the order among the nodes consistent. The corresponding value in N_P^S denotes the node's projection status as final or temporary.
4: Returns updated support S_P, Projection Nodes N_P and Projection status N_P^S.
5: **procedure** CREATENEWPB(Pattern P, Projection Nodes N_P, Projection Status N_P^S, SE items S_{EX}, IE items I_{EX}, F_{CL}, F_{SE})
6: If not given, default values are, $F_{CL} \leftarrow 1$, $F_{SE} \leftarrow 1$
7: Initialize Memory for new Pattern block Pb'
8: Set projection information $Pb'.P \leftarrow P$, $Pb'.N_P \leftarrow N_P$, $Pb'.N_P^S \leftarrow N_P^S$
9: Set extended items' information $Pb'.S_{EX} \leftarrow S_{EX}$, $Pb'.I_{EX} \leftarrow I_{EX}$
10: Set flags $Pb'.F_{CL} \leftarrow F_{CL}$, $Pb'.F_{SE} \leftarrow F_{SE}$
11: Return Pb'
12: **procedure** CANDIDATECOVERAGE(Pattern block Pb, Support S)
13: Let $PMapHep_N^S$ denotes the node representing support S in PaMHep
14: Let m denotes the length of pattern $Pb.P$
15: **for** each $ent \in PMapHep_N^S.Can[m-1]$ **do**
16: **if** $ent.P$ is enclosed by $Pb.P$ **then** Set $ent.F_{CL} \leftarrow 0$
17: **if** $ent.N_P$ is absorbed by $Pb.N_P$ **then** Set $ent.F_{SE} \leftarrow 0$.
18: **procedure** CLOSEDCOVERAGE(Pattern block Pb, Support S)
19: Let $PMapHep_N^S$ denotes the node representing support S in PaMHep
20: Let m denotes the length of pattern $Pb.P$
21: **for** each $len \in [1, m-1]$ **do**
22: **for** each $ent \in PMapHep_N^S.Clo[len]$ **do**
23: **if** $ent.P$ is enclosed by $Pb.P$ **then**
24: Delete ent from $PMapHep_N^S.Clo[len]$.
25: Set $K_{GE} \leftarrow K_{GE} - 1$, $K_{RA} \leftarrow K_{RA} - 1$

Algorithm 2 Pattern extension.

1: **Input: Pattern block** Pb**, Support Threshold** δ**, Extension type** Ex
2: **Output:** Newly created pattern blocks, new_pbs
3: **procedure** EXTENDPATTERN
4: If Ex denotes "SE", then Set $list \leftarrow Pb.S_{EX}$. Else Set $list \leftarrow Pb.I_{EX}$.
5: Set $new\_pbs = \{\}$
6: **for** each $\gamma \in list$ **do** ▷ $P\gamma$ as generic extension
7: Apply CMAP Pruning for $P\gamma$ to get maximum possible support $S_{P\gamma}$.
8: Let, $P \leftarrow Pb.P$, $m \leftarrow |Pb.P|$
9: **if** $S_{P\gamma} < \delta$ **then**
10: Set $N_{P\gamma}^S \leftarrow$ A bit string consisting of zeroes for all $n \in N_P$
11: Set $Pb' \leftarrow$CreateNewPb($P\gamma$, $Pb.N_P$, $N_{P\gamma}^S$, $Pb.S_{EX}$, $Pb.I_{EX}$, 1, 1). Considering the projection nodes of P as temporary projection nodes for $P\gamma$.
12: Insert Pb' into PaMHep. Set $new\_pbs \leftarrow new\_pbs \cup \{(Pb', \gamma, ex)\}$. Go to line 6.
13: Set $S_{P\gamma}, N_{P\gamma}, N_{P\gamma}^S \leftarrow$ BFSProjection($P\gamma$, $Pb.N_P$, $Pb.N_P^S$, δ)
14: If $|P\gamma|$ equals 2, update CMAP Table for the corresponding combination.
15: **if** $S_{P\gamma}$ equals δ **then** Set $Pb.F_{CL} \leftarrow 0$.
16: Set $Pb' \leftarrow$CreateNewPb($P\gamma$, $N_{P\gamma}$, $N_{P\gamma}^S$, {}, {}, 1, 1). Insert Pb' in PaMHep.
17: Set $new\_pbs \leftarrow new\_pbs \cup \{(Pb', \gamma)\}$
18: **if** $P\gamma$ is fully projected **then**
19: Apply CandidateCoverage(Pb', $S_{P\gamma}$) and ClosedCoverage(Pb', $S_{P\gamma}$)
20: Return new_pbs

mined pattern counter variables in line 7. In line 8, we insert all the extracted single-length items as pattern blocks in the PaMHep in the corresponding nodes that bear their supports. Initially, all the items are kept in lists as sequences and itemset extended symbols (sList, iList), keeping the idea of lexicographical order in mind. The mining continues till the breaking condition is not satisfied (lines 9-35). We pop out the pattern block from the root of PaMHep (line 10). If there is no pattern block, it indicates that we have completed finding all the CSPs of particular support, we check breaking conditions, and if not satisfied, we start with the next highest supported PaMHep node as root (line 13). We set the support threshold δ in line 14, which denotes that we want to extend only those patterns with this support. If the pattern block Pb is not completely extended, we extend it. If it fails to satisfy δ, we do not extend it further but rather store it as a new pattern block in PaMHep in the corresponding node (lines 15-19). However, if Pb satisfies δ, we start extending it (lines 21-22) based on the F_{SE} flag stated in Theorem 3. If we see that no other pattern of similar support has enclosed Pb, we identify it as a CSP and update the corresponding variables (lines 23-25). In lines 26-30, we prepare the lists of the newly created pattern blocks that will try to extend them as SE and IE based on the discussion presented in Section 3.3. For redundancy-aware mining, we also prune the previously considered CSPs of higher supports which are subsets of Pb (lines 33-35).

3.5.3 Breaking conditions and the number of mined patterns in each variation

The algorithm evaluates the breaking condition after processing the CSPs of a particular support in each variation (Algorithm 3, line 11). In the following points, we analyze the breaking conditions of the three addressed variations.

- Top-K generic CSP: After processing the CSPS of a particular support, the algorithm evaluates if it has already found at least K CSPs. If it has, then the mining stops.
- Top-K group CSP: The algorithm processes the CSPs of exactly the highest K unique supports. The algorithm stops afterward.
- Top-K redundancy aware CSP: The breaking condition of top-K redundancy aware CSP is very tricky because,

Algorithm 3 KCloTreeMiner: mining algorithm.

1: **Input:** SP-Tree Root $SPTree$, K, mining type $type$. The value of $type$ can be "generic", "group," or "redundancy-aware".
2: **Output:** Based on mining type mined top-K CSPS, $KCSP = \{\}$
3: **Global:** All pattern blocks, PaMHep, mined pattern counter variables.
4: **procedure** KCLOTREEMINER
5: Extract all 1-length items I, corresponding support S_I and projection nodes N_I in $SPTree$ using next links and store in L. $L \leftarrow L \cup \{(I, S_I, N_I)\}$
6: Sort L based on support. If $type$ is "generic" or "group" and L has more than K unique supports remove redundant items and set a minimum support bound for $PaMHep$. (Property 5, Section 3.5).
7: Set mined pattern counter variables to track the number of mined CSPs for "generic", "group," and "redundancy aware" as $K_{GE}, K_{GR}, K_{RA} \leftarrow 0, 0, 0$ respectively.
8: For each $en \in L$, create pattern blocks with the information stated in Definition 8 and insert in the corresponding node in $PaMHep$ based on their support.
9: **while** Breaking condition is not satisfied **do**
10: $Pb \leftarrow$DeleteFromPaMHep().
11: **if** Pb is empty **then**
12: If there was at least one pattern from the last highest support, Set $K_{GR} \leftarrow K_{GR} + 1$.
13: Remove PaMHep root. Check Breaking condition. If Fails, go to line 10.
14: $\delta \leftarrow$Support value of PaMHep root.
15: **if** $Pb.N_P$ is not completely projected **then**
16: $S_P, N_P, N_P^S \leftarrow$ BFSProjection$(Pb.P, Pb.N_P, Pb.N_P^S, \delta)$
17: **if** S_P not equals δ **then**
18: Set $Pb' \leftarrow$ CreateNewPb$(Pb.P, N_P, N_P^S, Pb.S_{EX}, Pb.I_{EX}, Pb.F_{CL}, Pb.F_{SE})$.
19: Insert Pb' in PaMHep in the concerned node $PaMHep_N^{S_P}$. Go to line 10.
20: Set $new\_pbs_{SE}, new\_pbs_{IE} \leftarrow Pb.S_{EX}, \{\}$
21: **if** $Pb.F_{SE}$ equals 1 **then** $new_pbs_{SE} \leftarrow$ExtendPattern(Pb, δ, SE)
22: $new_pbs_{IE} \leftarrow$ExtendPattern(Pb, δ, IE)
23: **if** $Pb.F_{CL}$ equals 1 **then**
24: Let $m \leftarrow |Pb.P|$. Insert Pb in $PaMHep_N^{\delta}.Clo[m]$.
25: $K_{GE} \leftarrow K_{GE} + 1, K_{RA} \leftarrow K_{RA} + 1$.
26: $T_{SE}, T_{IE} \leftarrow \{\}, \{\}$
27: **for** all $\{Pb', \gamma\} \in new\_pbs_{SE}$ **do** $T_{SE} \leftarrow T_{SE} \cup \{\gamma\}$
28: **for** all $\{Pb', \gamma\} \in new\_pbs_{IE}$ **do** $T_{IE} \leftarrow T_{IE} \cup \{\gamma\}$
29: **for** all $\{Pb', \gamma\} \in new\_pbs_{SE}$ **do**
30: $Pb'.S_{EX} \leftarrow \{\forall \beta \in T_{SE}\}$, $Pb'.I_{EX} \leftarrow \{\forall \beta \in T_{SE}$ where $\beta >_{order} \gamma\}$
31: **for** all $\{Pb', \gamma\} \in new\_pbs_{IE}$ **do**
32: $Pb'.S_{EX} \leftarrow \{\forall \beta \in T_{SE}\}$, $Pb'.I_{EX} \leftarrow \{\forall \beta \in T_{IE}$ where $\beta >_{order} \gamma\}$
33: **if** $type$ is "redundancy aware" and $Pb.F_{CL}$ equals 1 **then**
34: **for** all $sup \in$PaMHep **do**
35: **if** $PaMHep_N^{sup}.S > \delta$ **then** ClosedCoverage(Pb, sup)

in this variation, the total number of mined CSPs after processing the CSPs of particular support does not follow monotonically increasing property. Let us consider, X_T and X_{T+1} denoting the total number of mined patterns after T^{th} and $(T+1)^{th}$iteration. Also, let us consider x_{T+1} and R_{T+1} denoting the number of new CSPs that enclose some prior considered CSPs of higher support and the number of new CSPs that are not considered to enclose some other CSPs of higher supports after $(T+1)^{th}$ iteration respectively. Formally,

$$X_{T+1} = X_T - x_{T+1} + R_{T+1} \tag{1}$$

because, to enclose a pattern, we need at least one super pattern

$$x_{T+1} \leq X_T \tag{2}$$

to increase the total number of mined patterns in successive iterations:

$$X_{T+1} > X_T \tag{3}$$

$$X_T - x_{T+1} + R_{T+1} > X_T \tag{4}$$

$$R_{T+1} > x_{T+1} \tag{5}$$

So, to maintain the monotonic property over the total number of mined patterns, we need more patterns that do not enclose prior considered CSPs (i.e., higher value in R_T) and lesser patterns that enclose prior considered CSPs (i.e., lesser value in x_{T+1}). However, the issue here is that, with reduced support value, we get both new variational patterns (R_{T+1}) consisting of combinations of different items and super patterns that enclose many prior considered CSPs. So, the expected relationship between x_{T+1} and R_T is not always maintained, resulting in non-monotonic property. The algorithm in this variation continues till we get to a point where we have at least K CSPs. The worst-case scenario of the algorithm will be a set of transactions that enclose all the patterns observed in the database.

Now, we summarize the number of patterns that can be found or reported in each variation of the addressed top-K mining problems.

- Top-K generic CSP: KCloTreeMiner might mine more than K patterns during mining iterations. It might happen because we always completely calculate all the CSPs of a particular support to ensure correctness. To report as the final set of patterns, we can omit reporting some CSPs bearing the least support being processed in KCloTreeMiner.
- Top-K group CSP: We mine exactly K groups of patterns bearing the highest K unique supports using KCloTreeMiner. Often the total number of mined patterns will be more than K.
- Top-K redundancy aware CSP: If the number of mined patterns is more than K, we need to carefully omit some CSPs not used to enclose some smaller length CSPs of

higher support. Because if we omit a CSP that is used to enclose a CSP of higher support we shall violate the constraint stated in Definition 5.

In Fig. 2, we present a snapshot of different iterations (Itr) during mining. Each iteration shows which PaMHep nodes, $PaMHep_N$ (alphabet level) of different supports exist now, along with the corresponding Can and Clo buffer of different lengths. As stated prior, Can and Clo denote the candidate and closed patterns found with the corresponding support shown in the second column after the PaMHep node label and column. For simplicity, we are not showing all the information about the pattern blocks. We also show the mined pattern counter variables along with. An iteration denotes extending a single pattern for different items from its list of items as SE and IE lists in the pattern blocks. We also show the value of the counter variables observed at the beginning of the iteration in the last column denoted as Count. Fig. 2 demonstrates how patterns are maintained in PaMHep for storage or further processing.

Most of the candidate patterns (stored in column Can) are temporarily projected. Their support is continuously updated with their projected nodes in the following iterations. The main difference among the variations is the set of mined CSPs and the difference in the value of the counter-variables. We can see how K_{GR} varies compared to K_{GE} and K_{RA}. Though the set of CSPs for the generic and redundancy aware seems similar here, with gradual progression, many prior CSPs of higher support (stored in column Clo) will be enclosed by other larger-length CSPs of smaller supports, making the finding more challenging. The final set of top-3 mined CSPs for top-K generic and top-k redundancy aware is shown in Definitions 3 and 5. The final set of top-3 group CSP is shown in Table 3. Figure 2 gets generated during Top-3 mining by KCloTreeMiner.

3.6 Pattern summarization

We stated in Section 3.5.3 that, in the top-K group CSP problem, the number of reported patterns can be more than K. In this regard, we discuss some techniques to summarize the patterns that might be useful for some applications in this section. As highlighted in Section 1, we show three strategies in this study.

3.6.1 Max_{WOC}: maximal patterns without strict occurrence constraint

In this strategy, we summarize each group of CSPs by a single maximal pattern P_k where P_k does not have the constraint to be strictly present in the database. Here, our objective is to minimize the length of P_k for each group of CSPs.

The problem of finding the optimal maximal pattern to summarize each group bears exponential complexity depending on all possible merging of the sub-sequences or the patterns. That is why we have focused on designing a greedy solution to find a maximal summarized pattern that is suboptimal. We present in Algorithm 4. The algorithm first sorts the pattern based on length in descending order. Then, in each step, the greedy choice chooses a pattern that can be merged with the ongoing solution with the least change. The big-O complexity of the solution will be $\mathcal{O}(n^2 \times m^2)$, where n denotes the number of patterns which is summarized and m denotes the length of the sequence found by placing all the patterns side by side. The procedure MergePattern stated in Algorithm 4 uses a dynamic programming approach to find the strategy to optimally merge pattern Q within P by memorizing the overlapping sub-problems. In Fig. 3, we show the summarized patterns found by applying Algorithm GreedyMAXWOC over the top-3 groups of CSPs.

Algorithm 4 Greedy suboptimal algorithm to calculate Max_{WOC} for a group of CSPs.

1: **procedure** MERGEPATTERN(Pattern P, Pattern Q, P's event P_i, Q's event Q_j)
2: **if** All the events of Q is inserted in P **then** Return 0
3: **if** All the events of P is checked but some events in Q remains uninserted **then**
4: Return the total number of characters in the remaining events of Q
5: Set $t \leftarrow$the minimum number of character edition to make event $Q_j \subseteq P_i$
6: Set $v_1 \leftarrow t+$MergePattern(P, Q, P_{i+1}, Q_{j+1}), $v_2 \leftarrow$ MergePattern(P, Q, P_{i+1}, Q_j)
7: Return minimum(v_1, v_2)
8: **procedure** GREEDYMAXWOC(A Group of CSPs G_k)
9: Sort all the patterns $P_k \in G_k$ in Descending order over their lengths
10: Set $A_k \leftarrow$the first element in the sorted G_k
11: **while** $|G_k| > 1$ **do**
12: Find $F_k \in G_k$ that can be merged with A_k with minimum character edition using the MergePattern procedure
13: Merge A_k with F_k based on the findings of the MergePattern procedure to merge with the minimum character edition. Update A_k ▷ Local Greedy Choice
14: Remove F_k from G_k
15: Return summarized pattern A_k

3.6.2 Max_{WC}: maximal patterns with strict occurrence constraint

In this strategy, we summarize each group of CSPs by a group G_k of maximal patterns where each pattern $P_k \in G_K$ occurs in the database. Here, our first objective is to reduce the size of G_k as small as possible, and the second objective is to reduce the size of P_k. In this regard, we also propose another suboptimal greedy strategy stated in Algorithm 5. Like before,

Fig. 3 Example of pattern summarization based on Max_{WOC} and Max_{WC} over top-3 group CSPs

S	G_k	MAX_{WOC}	MAX_{WC}
5	<(ab) (c)>, <(bc)>	<(ab) (bc)>	<(ab) (bc)>
4	<(ab) (bc)>, <(ab) (d)>, <(c) (c)>, <(c) (d)>	<(abc) (bcd)>	<(ab) (cd) (d) (bc)>
3	<(ab) (c) (c)>,<(ab) (c) (d)>, <(d) (b) (d)>, <(bc) (d)>, <(d) (bc)>, <(c) (bc)>, <(cd)>	<(d) (ab) (bc) (bcd)>	<(ab) (cd) (d) (bc) (d)>

the optimal solution requires ordering all possible pattern combinations and finding the coverages by the transactions which bear exponential complexity. In the algorithm, as the greedy strategy, we choose the transaction that covers the maximum number of remaining uncovered patterns in each step. We use a max heap-based structure to find such transactions. After finding a summarizing pattern t, we remove the redundant items that are not necessary to cover the intended patterns to reduce t's length. The size of A_k can be more than 1. In Fig. 3, we have shown a demo to summarize Top-3 group CSPs with this strategy in the second column.

Algorithm 5 Greedy suboptimal algorithm to calculate MAX_{WC} for a group of CSPs.

1: **procedure** GREEDYMAXWC(A Group of CSPS G_k)
2: Set $TrWithPa \leftarrow \{\}$ to store the mapping between transactions and patterns
3: $A_k \leftarrow \{\}$
4: **for** each $P_k \in G_k$ **do**
5: Find a set of transactions T_k that covers it.
6: For each $t \in T_k$, update $TrWithPa[t] \leftarrow TrWithPa[t] \cup \{P_k\}$.
7: **while** $TrWithPa$ is not empty **do**
8: Find t that has the maximum number of elements in $TrWithPa[t]$
9: Remove redundant items from t that is not necessary to cover the patterns $P_k \in TrWithPa[t]$
10: Set $A_k \leftarrow A_k \cup \{t\}$
11: **for** each $P_k \in TrWithPa[t]$ **do**
12: Remove P_k from all $m \in TrWithPa$ where $P_k \in TrWithPa[m]$. If $TrWithPa[m]$ gets empty, remove $TrWithPa[m]$.
13: Return the set of maximal patterns A_k

3.6.3 *CROID*: centroid based summarization approach

In this strategy, we focus on a distance measure for the similarity value among the patterns. Based on that, we group the mined CSPs into a number of clusters and consider the cluster centroids to be representative or summarized patterns. The distance measure to find the dissimilarities between two patterns A and B is given in (9). Here α, β, and γ denote weights over different metrics which sums up to 1. A higher value represents more dissimilarity between A and B. In the following paragraphs, we describe each metric TD, $LCSD$, and SD. All the metrics in (9) are non-negative, normalized between [0, 1], and symmetric:

$$M_1 = TD(A, B) \tag{6}$$
$$M_2 = LCSD(A, B) \tag{7}$$
$$M_3 = SD(A, B) \tag{8}$$
$$Distance(A, B) = \alpha M_1 + \beta M_2 + \gamma M_3 \tag{9}$$
$$\alpha + \beta + \gamma = 1 \tag{10}$$
$$0 \leq Distance(A, B) \leq 1 \tag{11}$$

Definition 10 (Transaction-Wise Distance, TD) Given two patterns A and B, the transaction-wise distance is calculated based on (12) [40]. Here T_A and T_B denote the transactions where A and B have occurred, respectively. $|T_A \cup T_B|$ and $|T_A \cap T_B|$ denote the union and intersection of the lists respectively. The higher value denotes more dissimilarity:

$$TD(A, B) = 1 - \frac{|T_A \cap T_B|}{|T_A \cup T_B|} \tag{12}$$

where $0 \leq TD(A, B) \leq 1$.

Now, a challenge comes to calculating (12) based on SP-Tree because the transactions are not explicitly stored in the tree-based structure. In this regard, we highlight an important property of SP-Tree in the form of Definition 11. We have also stated in earlier sections that by traversing from a SP-Tree node to the root, we get a transaction fragment in reverse order.

Definition 11 (Transactions Within Nodes) For any node N in SP-Tree, if it can be observed that $C(N) > \sum_{child \in N.ch} C(child)$, it means that there are $C(N) - \sum_{child \in N.ch} C(child)$ transaction(s) of the actual database ending at node N. By default, all the leaf nodes of SP-Tree support this statement.

Let L_A denote the set of SP-Tree nodes that points to the transactions in which A can be found based on Definition 11. Now, we can re-write (12) as follows:

$$TD(A, B) = 1 - \frac{\sum_{n \in \{L_A \cap L_B\}} (C(n) - \sum_{ch \in N.ch} C(ch))}{\sum_{n \in \{L_A \cup L_B\}} (C(n) - \sum_{ch \in N.ch} C(ch))} \tag{13}$$

Let $A = <(c)>$ and $B = <(ac)>$. Then, $N_{<(c)>} = \{5, 13, 17, 23\}$, $N_{<(ac)>} = \{5\}$. Combining the

leaf nodes where transactions end of the corresponding branches $L_{<(c)>} = \{8, 16, 22, 29, 35\}$, $L_{<(ac)>} = \{8\}$. So, $TD(<(c)>, <(ac)>) = 1 - \frac{1}{1+1+1+1+1} = 1 - \frac{1}{5} = \frac{4}{5} = 0.8$.

$LCSD$ metric focuses on the length of the longest common subsequence observed between two patterns A and $B(LLCS(A, B))$ normalized by the size of the largest pattern($max(|A|, |B|)$). A common subsequence means a fragment of an ordered character set found between two sequences where the characters do not need to be consecutive but to maintain respective order among them. The equation used to measure $LCSD(A, B)$ is shown in (14):

$$LCSD(A, B) = 1 - \frac{LLCS(A, B)}{max(|A|, |B|)} \tag{14}$$

For example, if $A = <(a)(b)(c)>$ and $B = <(b)>$, then $LCSD(A, B) = 1 - \frac{1}{3} = 0.67$.

Subset Distance(SD) between two patterns A and B is formulated with the help of Procedure MergeDistance in Algorithm 4 (lines 1-7). This study proposes this additional metric, primarily focusing on a pattern's summarizing attribute. MergeDistance procedure given two patterns P and Q tries to merge Q within P with minimum character edition. Based on this procedure, $SD(A, B)$ can be written via (15):

$$SD(A, B) = \min\{sd'(B \subset A), sd'(A \subset B)\} \tag{15}$$

where

$$sd'(B \subset A) = \frac{\text{MergeDistance}(A, B)}{|B|} \tag{16}$$

$$sd'(A \subset B) = \frac{\text{MergeDistance}(B, A)}{|A|} \tag{17}$$

Here, $\frac{\text{MergeDistance}(A,B)}{|B|}$ states a ratio that the number of new characters needed to be inserted within A to make $B \subset A$. A higher ratio means many new characters are required to make $B \subset A$. SD simulates both choices to make $(B \subset A)$ and $(A \subset B)$ and picks the one with the least value or dissimilarity. $LCSD$ focuses on the matched characters whereas SD focuses on the mismatched characters. As we plan to summarize a pattern group, the large patterns enclosing many other small patterns should get some additional priority as per our intuition. For example, $LCSD(<(a)(bc)(d)>, <(c)>) = 1 - \frac{1}{4} = \frac{3}{4} = 0.75$ though $<(c)>$ can be completely summarized by $<(a)(bc)(d)>$. Using $SD(<(a)(bc)(d)>, <(c)>)$ we get $\min\{\frac{0}{1}, \frac{3}{4}\} = 0$. Another example can be used to understand how the metric works. For example, Let $B = <(ab)(c)>$, then if $A_1 = <(b)(c)(de)>$, then $SD(A_1, B) = \frac{1}{3}$, but if $A_2 = <(a)(de)(b)>$, then$SD(A_2, B) = \frac{2}{3}$ denoting that A_1 is more close to B than A_2. In SD, the denominator terms are also not generic like maximum or minimum but subset pattern specific, making the value crisper.

Now, applying the distance measure stated in (9), we use clustering techniques to cluster the patterns and find each cluster's centroid. We plan on finding crisp centroids, that is why we choose the *K-medoid* algorithm in our study that tries to keep the clusters as separated as possible over rough sets or fuzzy clusters [7, 26, 40]. In Fig. 4, we present the distance matrix and three centroids found after applying the *K-medoid* algorithm over all the patterns mined as Top-3 group CSPs. The clustering approach can be modified based on an application's specificity. As weights, we have set $\alpha = 0.25$, $\beta = 0.25$, and $\gamma = 0.5$ weighting more on the pattern enclosure. The respective number denoting patterns are also shown as the superscript alongside the patterns in the

#/#	1	2	3	4	5	6	7	8	9	10	11	12	13
1	0	0.42	0.11	0.3	0.47	0.55	0.16	0.16	0.6	0.68	0.6	0.6	0.52
2	0.42	0	0.17	0.47	0.55	0.55	0.54	0.54	0.18	0.52	0.18	0.18	0.6
3	0.17	0.17	0	0.39	0.6	0.72	0.4	0.4	0.53	0.67	0.44	0.35	0.65
4	0.3	0.55	0.39	0	0.85	0.42	0.44	0.12	0.4	0.4	0.56	0.73	0.48
5	0.47	0.42	0.54	0.85	0	0.47	0.19	0.59	0.57	0.81	0.48	0.15	0.44
6	0.55	0.42	0.66	0.42	0.47	0	0.59	0.19	0.15	0.48	0.48	0.57	0.44
7	0.16	0.54	0.34	0.44	0.19	0.59	0	0.31	0.65	0.71	0.65	0.42	0.56
8	0.16	0.54	0.34	0.12	0.59	0.19	0.31	0	0.29	0.48	0.65	0.72	0.56
9	0.6	0.18	0.53	0.31	0.65	0.23	0.65	0.29	0	0.38	0.38	0.45	0.54
10	0.6	0.52	0.67	0.31	0.81	0.48	0.65	0.42	0.38	0	0.25	0.62	0.42
11	0.6	0.18	0.44	0.56	0.56	0.48	0.65	0.65	0.38	0.25	0	0.38	0.42
12	0.6	0.18	0.35	0.65	0.23	0.57	0.48	0.72	0.45	0.62	0.38	0	0.54
13	0.52	0.47	0.59	0.56	0.44	0.44	0.56	0.56	0.54	0.5	0.42	0.54	0

(a)

Cluster 1: Representative ‹(c)(c)›

pattern	distance	TD	LCSD	SD
‹(c)(c)›^{5}	0	0	0	0
‹(ab)(c)(c)›^{7}	0.19	0.25	0.5	0
‹(cd)›^{13}	0.44	0.25	0.5	0.5

Cluster 2: Representative ‹(bc)›

pattern	distance	TD	LCSD	SD
‹(bc)›^{2}	0	0	0	0
‹(ab)(bc)›^{3}	0.17	0.2	0.5	0
‹(bc)(d)›^{9}	0.18	0.4	0.33	0
‹(d)(bc)›^{11}	0.18	0.4	0.33	0
‹(c)(bc)›^{12}	0.18	0.4	0.33	0

(b)

Cluster 2: Representative 3 ‹(ab)(c)(d)›

pattern	distance	TD	LCSD	SD
‹(ab)(c)(d)›^{8}	0	0	0	0
‹(ab)(d)›^{4}	0.12	0.25	0.25	0
‹(ab)(c)›^{1}	0.16	0.4	0.25	0
‹(c)(d)›^{6}	0.19	0.25	0.5	0
‹(d)(b)(d)›^{10}	0.48	0.6	0.5	0.4

Fig. 4 (a) Distance matrix (b) Clusters generated by applying K-Medoid algorithm, maximum number of iterations=200 and patience=70. Superscripts alongside the patterns denote their serial numbers used in the distance matrix

Table 4 Dataset description

Name	Sequence Count	Item Count	Avg. Seq Length	Type of Data
Leviathan	5834	9025	33.81	Book
Bible	36369	13905	21.64	Book
Sign	800	267	51.99	Sign Language
FIFA	20450	2990	34.74	Web Click Stream
BMSWeBView1	59601	497	2.42	Web Click Stream
MSNBC	989818	17	13.23	Web Click Stream

figure. Patience denotes the threshold to check the number of successive iterations to observe improvement in total loss.

4 Evaluation

In this section, we evaluate the performance of KCloTreeMiner and other proposals of the current article based on different metrics. We present the novelties of our proposals from different viewing angles. For the comparison, we considered three relevant and recent algorithms, TKCS[34], TKS[10] and TKUS[53]. To apply the TKUS, we considered all the items equally weighted and have support 1 in each event. All the algorithms were implemented in Python language. The experiments were performed in a 64-bit machine having an Intel Core-i5 8265U processor (1.60-1.80 GHz), 16 GB ram, and Windows 10 OS. We evaluated six real-life sequential datasets collected from SPMF[11]. Table 4 summarizes the datasets. Both sparse and dense datasets $\left(\frac{Avg.SeqLength}{ItemCount} \geq 19\%\right)$ [36] have been used for the comparison. Datasets were discretized to make multi-item events based on random selection and merging. Our implementation can be found in our GitHub.[1]

All the datasets are widely acknowledged in sequential pattern mining literature and representatives of various scenarios related to sequential knowledge representation and extraction. For example, if we consider the MSNBC dataset, we can lay out an intuitive explanation. MSNBC dataset contains user click stream data denoting their browsing history within the MSNBC website [11]. Through generic sequential mining, we find the sequences that are frequently observed in the browsing history of the users. The additional layer of top-K mining focuses on extracting only the frequent top-K ones over all the frequent ones by reducing the dependency on the exact distribution of the support values of the items.

[1] Implementation will be available https://github.com/rizveeredwan/top-k-closed-tree-miner

4.1 Runtime evaluation of KCloTreeMiner in finding Top-K generic CSPs

To create a fair comparison to mine the top-K CSPS, we had to increase the value of K in TKS and TKUS. Because, for the same value of K, TKS and TKUS might not extract the top-K CSPs but a different set. We experimentally set the desired value of K for all the other algorithms in such a manner so that all of them, in the end, have at least the top-K generic CSPs. For TKCS, we do not need to adjust K in such a regard as it automatically discovers top-K generic CSPs.

Runtime Performance of KCloTreeMiner and other compared algorithms for finding top-K generic CSPs are shown in Fig. 5. Observed from the figure, KCloTreeminer takes the least amount of time. TKS mines top-K SPs, whereas TKCS mines top-K CSPs. Thus, TKCS mines fewer patters, resulting in improved runtime. TKUS applies some pruning strategies to tackle the utilities of the items, but as all the items are equally weighted here, those strategies actually added more cost. TKCS, TKS and TKUS all use a pattern grow extension strategy; the prior two apply the idea of bit-based data structure, and the last one uses vector id-list. The improvement shown by KCloTreeMiner is due to the following reasons,

1. Using a heap-based structure PaMHep over a list-based structure to maintain the candidates is much more efficient. Because we can easily find out the nodes or patterns with higher support values.
2. TKCS has to perform complete projection to calculate the support of a pattern, whereas using the BF-S strategy, we can on-the-fly identify if this pattern satisfies our expected support threshold. Based on this, we control the behavior of pattern expansion and progress with temporary projection nodes.
3. We use additional pruning strategies like pattern absorption and CMAP pruning. The prior strategy detects extension redundancy, and the following one gives an upper bound regarding the maximum possible support before extending the pattern in the tree. Pattern projec-

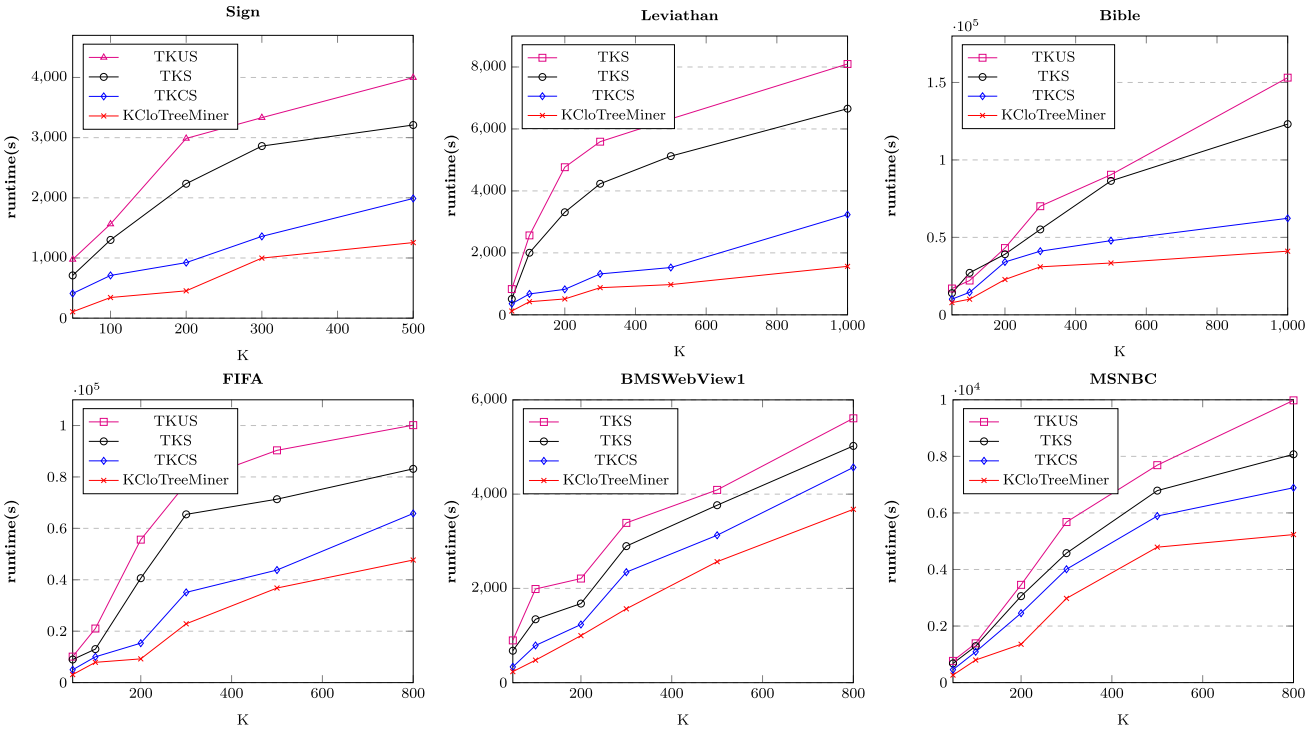


Fig. 5 Runtime usage of six datasets in finding top-K generic CSPs

tion is the most costly operation in mining, where we reduce several such projections.

4. The overlapping node property of SP-Tree* compacts the database, and the next link attribute helps to move faster in the tree resulting in improved runtime.

For the smaller values of K, the performance of each algorithm is quite close because the number of desired patterns is few. However, with increasing K, the difference is quite visible. From the numeric analysis, it can be understood better; for example, in the Sign dataset in Fig. 5, for $K = 50$, KCloTreeMiner performs 1.5 times faster than TKCS whereas, for $K = 500$, the factor is around 1.9 times. We have observed similar improvement factors for different Ks in all the other datasets. If we want to compare the runtime performance of top-k group CSP and top-K redundancy aware CSP with TKS, TKCS and TKUS, we need to adjust their parameter K's to a much higher level to discover the desired set of CSPs. The above experiment shows that, with higher K's, the runtime performance improvement becomes more visible.

4.2 Memory usage of KCloTreeMiner in finding Top-K generic CSPs

Memory usage of KCloTreeMiner with other algorithms have been shown in Fig. 6 for two datasets, sign (dense) and Leviathan (sparse). The observed behavior was found to be quite similar in all the datasets. KCloTreeMiner uses SP-Tree* structure to store the sequential database along with PaMHep to maintain the candidates and mined closed patterns. It uses SP-Tree* nodes as the projection pointers. It also

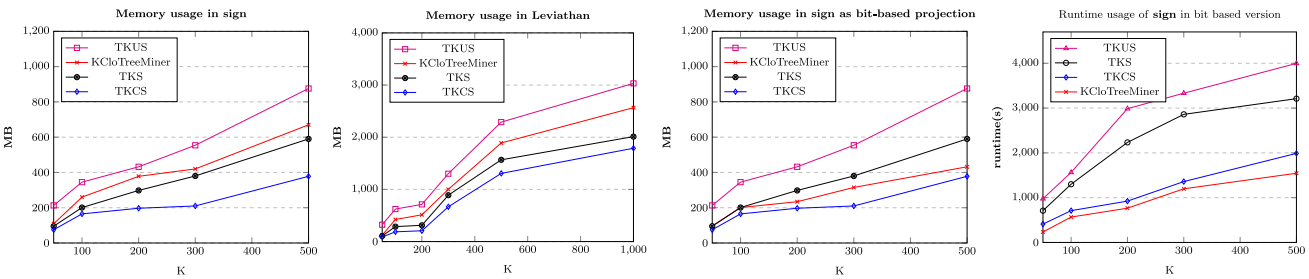


Fig. 6 Memory usage of KCloTreeMiner and optimization using bit-based projection

uses a table named CMAP inspired from Rizvee et al.'s work [36] equivalent to PMAP [10] to prune search space. TKCS stores the candidates in a raw list-based format; for each candidate pattern, it stores the information in a bit vector-based format. TKS also maintains a vector-bit-based format similar to TKCS to hold the candidates for expansion. TKUS uses a vector id-list with utility metadata to store the projection information and to maintain the candidates using a list-based structure.

From our results, we have observed that the memory usage of each algorithm is quite close. In contrast, KCloTreeMiner takes slightly more memory compared to the TKS and TKCS due to its structural representation of the database in a tree-based format and maintaining some supporting structures to execute pruning and maintaining the candidates. However, it comparatively takes lesser memory than TKUS. Overall KCloTreeMiner's memory usage is not significantly higher than the peer algorithms. As the earlier sections state, we can also use a bit-based node projection scheme to compress the information. This strategy will reduce memory to a reasonable amount but add extra runtime costs while unpacking the corresponding bitsets for each pattern. We also show its impact in Fig. 6. In the third figure, we show in the Sign dataset how using bit based node projection scheme; we can reduce memory(around 35.5% for $K = 500$) and get close to TKCS while our runtime also increases to a factor(around 19% for $K = 500$), shown in the fourth figure. To unpack the projection nodes or find the set bits in a value, the combination of bitwise XOR and AND operations in the detection of LSB is significantly faster than traditional remainder checking. For example, for a random value of 10^5 bits where 100 bits are set, the first method takes 0.002 seconds, and the second takes 4.33 seconds.

4.3 Setting of parameter *K* for finding Top-K group CSP in other algorithms

As stated in the earlier sections, to find a similar set of patterns for a particular K that we mine using top-K group CSPs, we need to fix a comparatively higher value of K in other algorithms. In Section 4.1, we have already shown that in higher K's KCloTreeMiner gives observably improved runtime performance. In this section, we highlight that in order to find top-K group CSPs, we need to increase the value of K in other algorithms. In Fig. 7, we show the results for the MSNBC dataset. In the x-axis, we vary the value of K that we use in KCloTreeMiner. In the corresponding y-axis, we present the value of K that we set in other algorithms to get the desired set of patterns. The figure represents the increasing factor for each algorithm for a particular K; with higher values of K, the difference becomes more prominent. The change in TKUS is abrupt because, in the high-utility framework, a pattern's total utility is influenced by both support and length. So, a small pattern with higher support and a larger pattern with smaller support both might fall into top-K leading to either a larger result set or a smaller result set of patterns.

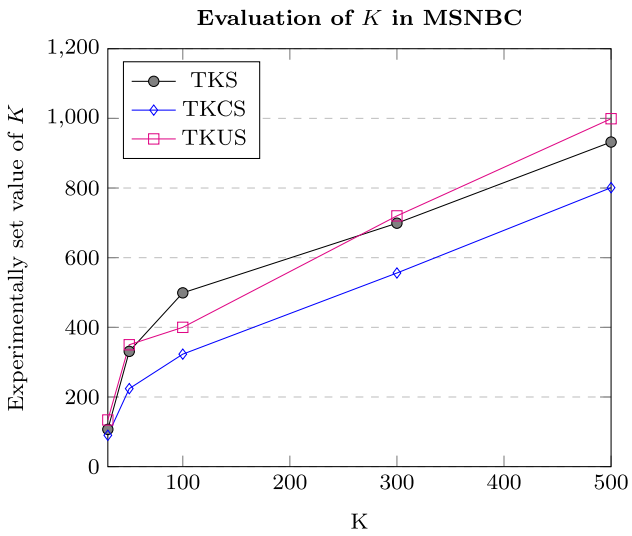


Fig. 7 Increasing factor of K in other algorithms during top-K Group CSPs

4.4 Number of mined pattern curve in Top-K redundancy aware CSP

In Section 3.5.3, we have shown that meeting the breaking condition during top-K redundancy-aware CSP mining is challenging. In this section, we observe this behavior in two datasets. We observe how the number of mined patterns varies for a particular K after processing the CSPs of a specific support. We present the results in Fig. 8. In the x-axis, we provide the supports observed during the top-K highest supports processing. In the y-axis, we keep the total number of mined patterns after processing the CSPs of the corresponding support given in the x-axis. We can see some downward lines represent the breaking of the monotonicity property. For example, In the FIFA dataset, while processing the supports within 600-700, the total number of mined patterns reduces a bit. Both curves were found during top-300 CSPs processing.

4.5 Capturing variation through Top-K redundancy aware CSP

The main motivation behind addressing top-K redundancy-aware CSP is to capture patterns that contain a wide range of relationships among the database entities within a small K value. This should help to capture the versatile behavioral relationships among the elements of a database. For example,

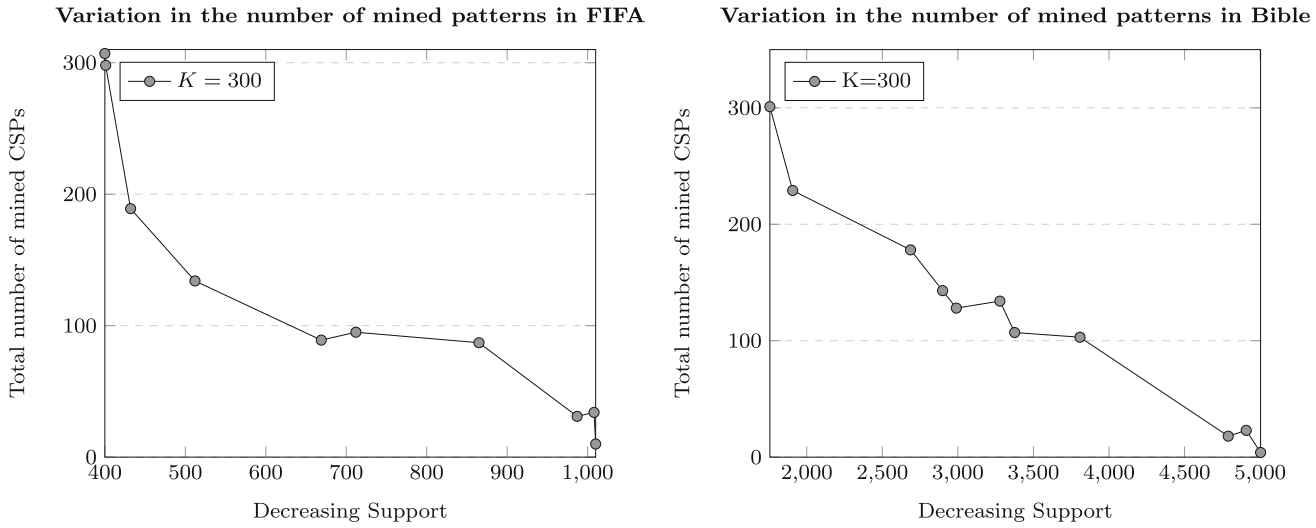


Fig. 8 Continuous number of mined patterns in top-K redundancy aware CSP

reporting both $< (a) >$ and $< (a)(b) >$ as Top-2 CSP capture lesser variation compared to reporting $< (a)(b) >$ and $< (d)(e) >$. In this section, we present data that shows for a particular K the number of unique unigrams, bigrams and trigrams observed in the reported CSPs by TKCS with the output of KCloTreeMiner during top-K redundancy aware mining. In Table 5, we present the results. For each dataset for a particular K, we show a tuple denoting the number of unigrams, bigrams and trigrams, respectively, for both TKCS and KCloTreeMiner. As a head-to-head comparison, we cannote that in the redundancy-aware version, KCloTreeMiner always captured more variation, giving higher values for unique unigrams, bigrams and trigrams.

4.6 Tree construction cost

This section observes the runtime cost associated with tree construction. In Table 6, we present the summarized information to state the percentage of runtime that is needed to construct the SP-Tree* along with completing the calculation of the attributes compared to the total runtime for a particular K for each of the datasets. It can be easily seen that the concerned cost is very insignificant compared to the total mining time. This cost's dependency lies in the database's size and its nature as dense or sparse—these factors control the branching and length factor of the constructed SP-Tree*. In general, as stated in our previous work [36], the construction of the SP-Tree structure is relatively insignificant compared to the mining time. This additional cost helps traverse the database faster, using the next links during pattern generation for sparse and dense databases.

Table 5 Observed unique combinations in unigram, bigram and trigram

Dataset	K	TKCS	KCloTreeMiner
Leviathan	30	(13, 43, 134)	(23, 74, 203)
Bible	50	(29, 134, 246)	(37, 234, 345)
Sign	30	(21, 65, 123)	(37, 97, 224)
FIFA	50	(12, 101, 301)	(19, 191, 498)
BMSWeb-View1	50	(7, 176, 201)	(12, 225, 298)
MSNBC	75	(32, 256, 398)	(42, 343, 430)

4.7 Compactness of SP-Tree* over SP-Tree

We have already stated that SP-Tree [36] should have mentioned the implementation strategy of the introduced attribute next link, which is used to traverse the tree-based structure faster. So, as per definition, the raw list-based format was used to store that information in each SP-Tree node. In this study, we use a novel and more compact strategy to save the attribute. In Fig. 9, we have shown two ways to store the next link attribute. The prior one comes per the definition of SP-Tree, and the following one comes from the definition stated in SP-Tree* using side connection. Here for nodes E and G, we show how they store the next link nodes for a particular symbol α. Curved lines denote that these nodes should be tracked as the next linked nodes, which lie in the underlying subtree. The straight line represents a direct parent-child node relationship, and dotted arrowed lines indicate side connections. Let the item attribute value of node E is not α. We can observe here that, in the second diagram in Fig. 9, to store the information for α as the next links, we need to keep lesser values for both nodes G and E compared to the first figure, (A, B, C, D) and (A, B, C, D, F) vs. (A, D) and (A, F) respectively.

Table 6 Tree construction runtime cost

Name	K	Tree construction T	Total mining M	Percentage (T/M)*100%
Leviathan	1000	2	1565	0.13
Bible	1000	35	41092	0.09
Sign	500	1.1	1256	0.09
FIFA	800	17	47689	0.04
BMSWeb-View1	800	49	3678	1.33
MSNBC	800	103	5234	1.97

For each dataset, we have also conducted experiments to see how the memory of the tree-based structure varies with this strategy for SP-Tree* over SP-Tree. In Fig. 10, we show the memory reduction of SP-Tree* in this regard. It can be easily understood that this strategy's performance depends on the dense or sparse database characteristics: the more node sharing, the more improvement in the concerned memory reduction.

4.8 Analysis to understand the effectivity of pruning strategies

In this study, we mainly apply three major pruning strategies, BF-S, CMAP, and pattern absorption. Pattern absorption strategy directly prunes the projection of a set of patterns. BF-S reduces the number of complete projections of the patterns in SP-Tree*. CMAP delays the starting of projections of the patterns or dictates that a group of patterns does not need projections based on a possible support threshold for the extended patterns. All of them have been used during mining using KCloTreeMiner, and Fig. 5 shows the corresponding runtime improvement. It is difficult to individually point out the novelties of each pruning strategy during mining, but we will provide some data to support our argument.

In Fig. 11, we show how the pattern absorption strategy improves the performance. We present the number of patterns that are projected at some point in SP-Tree* without and with the presence of this strategy for a particular K in each dataset. We may observe a sound reduction in the number here; the difference denotes the number of unprojected patterns.

To understand the effect of BF-S and CMAP, we present Fig. 12. It shows that, among the total patterns that were initiated to be projected, how many patterns were not wholly projected due to BF-S and CMAP. The y-axis denotes the number of patterns. BF-S stops the projection of the patterns to various degrees when it understands that these patterns will never satisfy the currently expected support threshold. They are kept in the buffer with temporary projection nodes for future extensions. An example can be used to understand their logical activity. Let us consider this scenario. We have performed complete projection for $< (a)(b) >$ detecting its actual support as 10, whereas we have performed temporary projection using BF-S and saw that support of $< (a)(c) >$ can be at most 8. Then, during extension of $< (a)(b)(c) >$ using CMAP, we can get an idea that support of $< (a)(b)(c) >$ will be ≤ 8 before applying any projection.

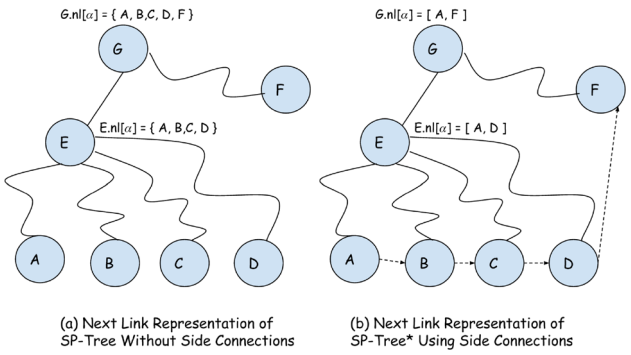


Fig. 9 Compactness of side connection based next link implementation of SP-Tree*

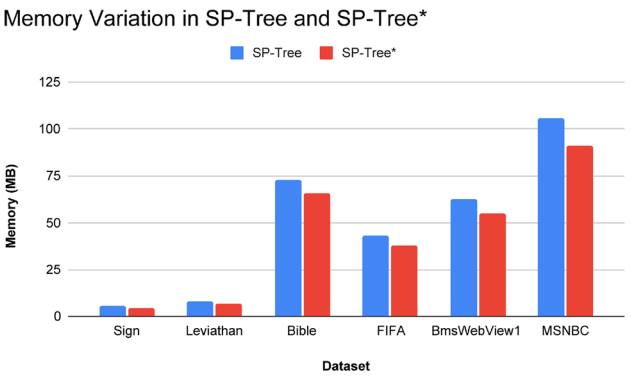


Fig. 10 Improvement in calculating next links using side connections of SP-Tree* over SP-Tree

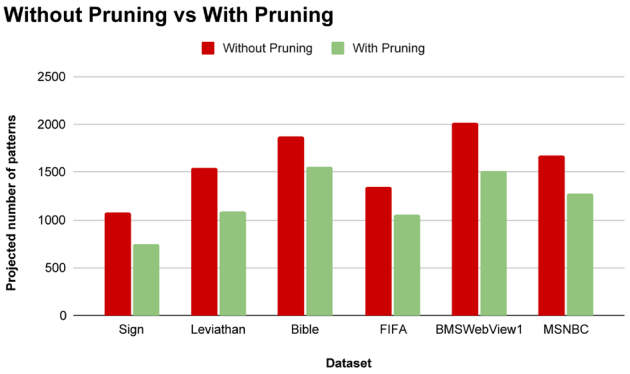


Fig. 11 Number of patterns projected in various degrees in the SP-Tree* with and without pattern absorption pruning strategy. As K, 300, 600, 500, 500, 600 and 400 were chosen respectively for the datasets in top-K generic CSP

4.9 Analysis of pattern summarization

This section analyzes our sequential pattern summarization approach from various angles.

4.9.1 Number of summarized patterns

As stated in Section 3.6, we can apply three strategies to summarize a group of patterns. To use MAX_{WOC}, we pick CSPs of particular support and then generate a single maximal pattern that contains all the CSPs of that support group as subsequences. So, for K group CSPs, we shall have K summarized patterns. We apply the K-medoids algorithm to get the summarized patterns using *CROID*. The k-medoids algorithm needs a parameter that denotes the number of clusters. In our simulation, we first merge all the mined CSPs into a single group. Then, fix the input K as the number of clusters and apply the algorithm to find the centroids. So, finally, there will be K summarized patterns. However, a crucial challenge comes with finding Max_{WC} for each group of CSPs as there can be one or more summarized patterns for each group. In Table 7, we present data that shows to summarize each dataset for a particular K, the total number of mined CSPs(C), the total number of summarized Max_{WC}(S), average number of Max_{WC}(Avg $|Max_{WC}|$) and average number of CSPs(Avg $|CSP|$) found for each group. This denotes that the total number of Max_{WC} is significantly smaller compared to the total mined CSPs. For each group, this number is also insignificant compared to the total mined CSPs of that group.

4.9.2 Runtime and memory usage of Max_{WOC}, Max_{WC} and *CROID*

In this section, we present an analysis to understand the summarization strategies' memory usage and runtime requirement. In Figs. 13 and 14, we present two sets of data that represent the required runtime and memory usage, respectively. To apply the K-medoid algorithm, the max number of iterations was picked as 200, with the patience value set to 70 to track the number of consecutive iterations with no improvement. The runtime usage of CROID depends on the number of iterations and the distance computation. Max_{WOC}, at each step, chooses the pattern that can be merged with the ongoing solution with the least cost or distance. Max_{WC} in each step selects the transaction that can enclose the maximum number of patterns. From our analysis, we found that the runtime usage of Max_{WC} and Max_{WOC} is quite similar, while Max_{WC} comparatively took less time. The underlying reasoning can be in each step of Max_{WOC}, we multiple times calculate the distance metric that uses the optimal merging and lcs calculation based on dynamic programming strategy. Also, Max_{WC} is cost-prone when the length of the sequences becomes larger along with a group of CSPs requiring various transactions to be summarized. From Fig. 13, we can see that Max_{WOC} and Max_{WC} took an almost similar amount of time while *CROID* took consid-

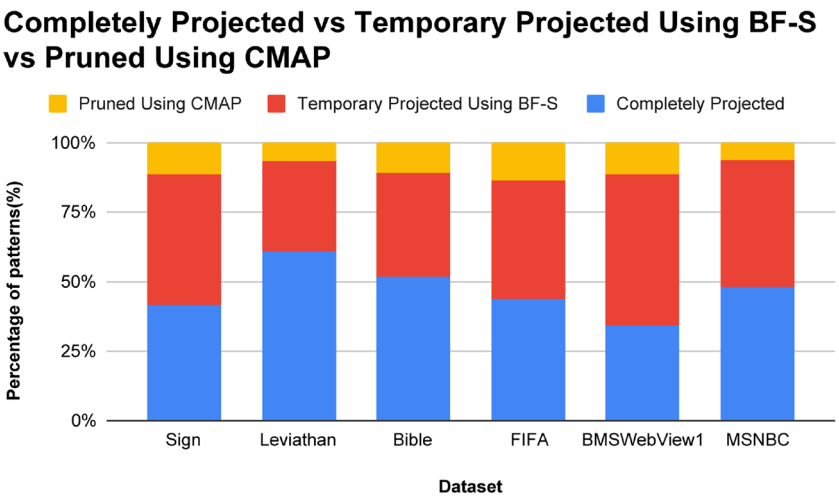


Fig. 12 Percentange of completely projected patterns, temporary projected patterns, and pruned by CMAP. As K, 300, 600, 500, 500, 600, and 400 were chosen respectively for the datasets in top-K generic CSP

Table 7 Number of found Max_{WC} for different datasets in top-K group CSP

Database	K	C	S	Avg |MaxWC|	Avg |CSP |
Sign	10	345	18	1.8	34.5
Leviathan	25	567	43	1.72	22.68
Bible	15	897	51	3.4	59.8
FIFA	15	435	36	2.4	29
BMSWebView1	10	201	27	2.7	20.1
MSNBC	25	598	61	2.44	23.92

erably longer due to distance calculations and having several iterations. In a single iteration, the runtime cost of *CROID* should also be quite similar.

As per the memory consumption analysis, we found that each strategy has no specific memory bottleneck. However, with the use of dynamic programming, sequences of longer lengths may impose memory constraints. Furthermore, in practical scenarios, such length is rare. Based on our observation, we found that memory usage is almost similar for each strategy, presented in Fig. 14.

4.9.3 Analysis on silhouette width

In this section, we present an analysis of silhouette width [40] to understand the cluster quality achieved by K-medoid. For each pattern, the silhouette coefficient value $[-1, +1]$ denotes its closeness with the corresponding cluster centroid where the pattern has been kept. Here $+1$ represents that the pattern is very compact or close to the cluster centroid, whereas -1 denotes the opposite. In (18), we present the mathematical formula used to calculate the silhouette coefficient value for a pattern P. Here a denotes the intra-cluster average distance, and b indicates the minimum average inter-cluster distance for P. The silhouette width of a cluster is the average of the silhouette coefficient values of all the patterns belonging to that cluster. A higher positive value as the silhouette width will denote a compacter cluster.

$$silhouette(P) = \frac{b - a}{max(b, a)} \tag{18}$$

We present data for the sign dataset in Fig. 15. All the other datasets exhibit similar kind of characteristics. To experiment, we fixed $\alpha = 0.25$, $\beta = 0.25$ and $\gamma = 0.5$ as weights in our used distance metric while applying the K-medoid algorithm. Also, for the comparison, we tested with $\alpha = 0.5$, $\beta = 0.5$ and $\gamma = 0$ setup as used in Ref. [40]. Figure 15 shows the average silhouette co-efficient for each cluster and the average silhouette width of all clusters. For ease of visualization, we set the number of clusters as 5 while we mined top-10 group CSPs. We also present the average subsequence length(l) covered by the corresponding centroid written in percentage for each cluster. Our analysis states that the average silhouette width observed via our distance metric in our setup is comparatively higher(0.38 vs 0.11). Moreover, the average subsequence length covered by the centroids is also relatively higher(0.46 vs 0.36). This represents that through our weighted distance metric, most of the patterns belonged to the correct cluster with a good positive silhouette coefficient. Also, the discovered centroids tried to cover the patterns of the same cluster as much as possible.

4.10 Ablation study over recent algorithms

TKCS, TKUS and TKS are the most recent algorithms that closely align with our research problem and are relevant to top-K, CSP and sequential pattern mining challenges. For this reason, we formulated our discussion and analysis surrounding these algorithms to understand the core improvement factors in metrics such as runtime, memory usage, etc. In this section, we present an ablation study comparing the very

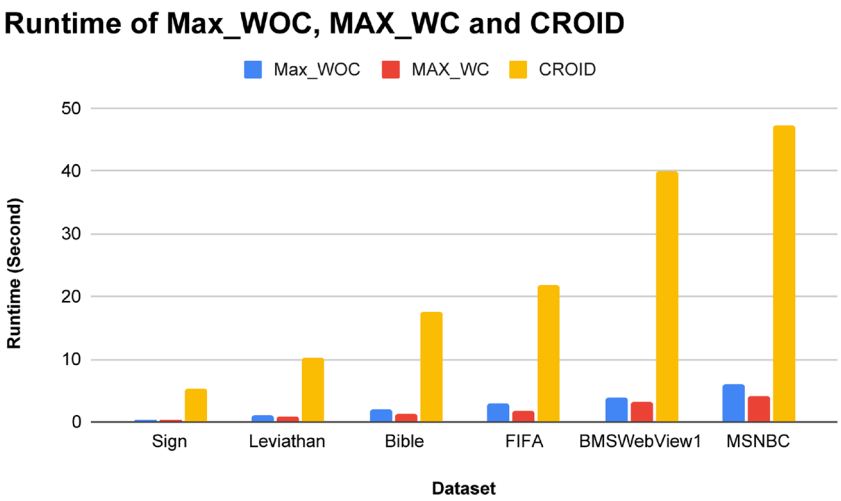


Fig. 13 Runtime Usage of Max_{WOC}, Max_{WC} and $CROID$ in seconds for six datasets for $K = 10, 25, 15, 15, 10$ and 25 respectively

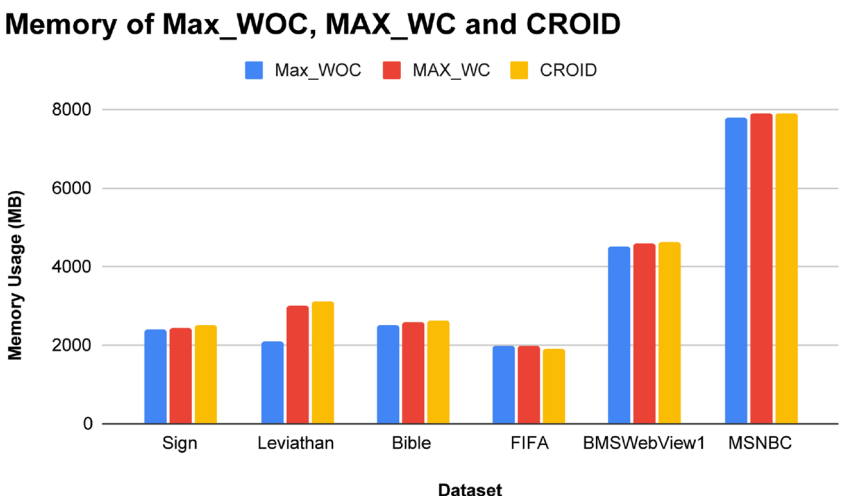


Fig. 14 Memory usage of Max_{WOC}, Max_{WC} and $CROID$ in MB for six datasets for $K = 10, 25, 15, 15, 10$ and 25 respectively

recent algorithms with respect to our proposal, KCloTreeMiner.

Among other very recent algorithms for sequential pattern mining, Refs. [57] and [58] both work with the problem of Targetted Sequential Pattern Mining (TaSPM) alongside an additional attribute entitled contiguity. TaSPM problem is significantly different from ours in terms of the definition—considering a targetted variable as a query to mine sequential patterns over a threshold. As we do not have any prior sequence-based variable to consider or to prune the search space, we do not compare KCloTreeMiner with Refs. [57, 58].

Another group of very recently published literature contains Refs. [59, 60]. In particular, UUCPM[59] combines the problem of high utility sequential mining and uncertainty observed within a database. To have a fair comparison with it, we need to omit the uncertainty and utility factor from consideration, similar to our evaluation with TKUS. As per our study, their pruning strategies are based on constructing two bounds over the values of utility and uncertainty. If these two factors are omitted, other features of their proposal do not comply with any improvement concerning top-K mining. In a similar analogy, we can also omit the comparison with Ref. [60] because the latter extends the problem of high-utility mining problem to high utility-based sequential rule mining. They also construct a bound based on the utility factor to prune search space. Extending the prior arguments, we can also denote that, removing the utility factor will deter the effect of the bound resulting in no additional improvement for top-K mining.

Another very recent notable study addresses the problem of top-K frequent itemset mining in very large databases [61]. The main reasoning behind not choosing this work lay in not considering the order among the items which is the main distinction between normal itemset mining and sequential pattern mining. So, comparing with this particular work

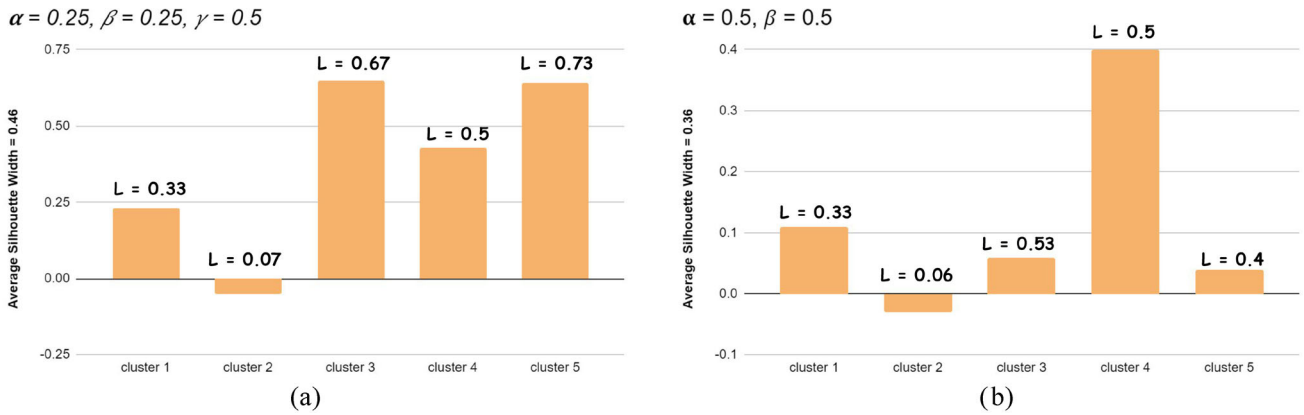


Fig. 15 Silhouette width and average pattern coverage for each cluster(l) in sign for $K = 10$ with 5 clusters where (a) $\alpha = 0.25, \beta = 0.25, \gamma = 0.5$ and (b) $\alpha = 0.5, \beta = 0.5, \gamma = 0$

will require undermining a lot of challenges that are closely related to sequential mining and will not represent the effectiveness of our proposal.

4.11 Discussion

Our experiments showed that SP-Tree* based KCloTreeMiner provides a quality improvement in runtime during top-K generic CSP mining. From our analysis, we achieved around 23% improvement in average runtime for the experiments conducted over the datasets shown in Fig. 5 compared to TKCS, the close benchmark algorithm upon which we improve. To calculate the average we consider the runtime values obtained from the highest K tested for each dataset for both TKCS and KCloTreeMiner and perform averaging. As top-K group and redundancy aware CSPs are a variation over this generic framework, they also carry similar runtime enhancements. We showed that to extract a similar set of CSPs mined by top-K group and redundancy aware, other algorithms need to lower their K experimentally, which is also challenging.

Through memory usage analysis, we showed that KCloTreeMiner does not increase memory usage by a significant amount. Moreover, we also showed its bitset node projection-based version that can reduce the memory to a reasonable extent with some additional runtime costs. We also presented the compactness of the modified SP-Tree* over the prior proposed SP-Tree along with the discussion on tree construction cost, which is relatively insignificant compared to the total mining time. Moreover, we also tried to show how BF-S and CMAP delay the complete projection of the patterns, whereas Pattern Absorption completely omits projection for some patterns. We also discussed the memory usage and runtime costs observed while applying summarization strategies, indicating that their performance was similar and close. In this context, in Section 4.9.1, we present how the present summarizing algorithms Max_{WC} and Max_{WOC} can find the representative pattern(s) for a particular group: Max_{WC} always sticks to one representative pattern whereas Max_{WOC} might generate one or more. To demonstrate the quality of the proposed measure Subset Distance we highlight the discussion of Section 4.9.3 based on Silhouette Width. As per the discussion, we state how by incorporating this measure along with existing metrics, the width gets reduced, meaning the clusters' elements were comparatively closer.

In terms of hyperparameters, our solution requires very few and the results are reproducible. In the top-K mining portion, K is the only hyperparameter. For a discretized sequential dataset, as stated in the introduction portion of Section 4, the mining result will only vary based on the values of K. In the clustering portion, where we try to find the representatives for a group of mined sequential patterns, the results can vary depending on the values of weight parameters α, β and γ used in the weighted distance measure of (9). Furthermore, for running different clustering algorithms, we may also require another parameter that denotes the number of clusters (e.g., K-medoid, K-Means, etc.) to be generated. Depending on this parameter, the results can also vary.

4.12 Future research directions

In summary, the current work (a) established a complete theoretical formulation associated with top-K sequential pattern mining, (b) proposed a novel algorithm to address three types of top-K sequential pattern mining problems, as well as a set of strategies and algorithms to find representatives among a set of sequential patterns. Multiple research directions can be followed based on this work.

To elaborate, this current work was designed for a single-machine environment where the database is first represented in a tree-based structure. A possible direction would be to explore the potential of the tree-based solution in a distributed or parallel computation environment, where the tree can be fragmented into multiple blocks or different parts of the mining process can be performed altogether. Another possible research direction would be to explore constructing more compact representatives of a group of sequential patterns. Currently, we use the K-Medoid algorithm for our proposed distance measure. Understanding the performance of other clustering algorithms or developing sequential pattern mining-focused specialized clustering algorithms can also be notable directions to extend the research. Furthermore, as we have established a very detailed framework related to top-K mining, the concepts presented in the current article can be a good guideline in exploring the top-K mining problem for other types of structured and unstructured datasets (e.g., graph, weighted sequential dataset, etc.) or other application-centric problems (e.g., targeted mining [54], fake news analysis [55], contrastive mining [56], etc.)

5 Conclusion

In this study, we addressed three variations of top-K CSP mining: top-K generic CSP, top-K group CSP, and top-K redundancy-aware CSP. We established a theoretical framework outlining general strategies and boundaries for the top-K mining problem. We introduced a novel pruning strategy named pattern absorption and utilized an existing BF-S strategy to reduce search space. Combining these strategies, we proposed the KCloTreeMiner algorithm, which mines CSPs using SP-Tree*. Additionally, we suggested three pattern summarization strategies—namely, Max_{WOC}, Max_{WC} and $CROID$—to incorporate a new subset distance metric for weighted summarization. We devised a side connection-based next-link attribute calculation to minimize memory

usage. Our experiments demonstrate the efficiency of our proposals in terms of runtime improvement and mining more contextful patterns. We also presented a set of possible research directions that would be a part of our follow-up extension of this work.

Acknowledgements This work is partially supported by (a) Natural Sciences and Engineering Research Council of Canada (NSERC) and (b) University of Manitoba. We would like to thank all the reviewers for their valuable time and suggestions to help improve the current article.

Author Contributions Redwan Ahmed Rizvee: Conceptualization, Methodology, Software, Validation, Formal Analysis, Investigation, Data Curation, Writing - Original Draft. Chowdhury Farhan Ahmed: Supervision, Investigation, Conceptualization, Resources, Writing - Review and Editing. Carson K. Leung: Supervision, Investigation, Conceptualization, Writing - Review and Editing, Funding Acquisition.

Data Availability All the used data are publicly available and accessible.

Declarations

Competing Interests The authors declare that they have no competing interests.

References

1. Ali A, Zhu Y, Zakarya M (2021) Exploiting dynamic spatio-temporal correlations for citywide traffic flow prediction using attention based neural networks. Inf Sci 577:852–870
2. Ali A, Zhu Y, Zakarya M (2022) Exploiting dynamic spatio-temporal graph convolutional neural networks for citywide traffic flows prediction. Neural Netw 145:233–247
3. Arefin MF, Ahmed CF, Rizvee RA, Leung CK, Cao L (2022) Mining contextual item similarity without concept hierarchy. In: Proceedings of the 16th International Conference on Ubiquitous Information Management and Communication (IMCOM 2022). IEEE, pp 229-236
4. Ayres J, Flannick J, Gehrke J, Yiu T (2002) Sequential pattern mining using a bitmap representation. In: Proceedings of the eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. pp 429–435
5. Chen Y, Liu Z, Li J, McAuley J, Xiong C (2022) Intent contrastive learning for sequential recommendation. In: Proceedings of the ACM Web Conference 2022. pp 2172–2182
6. Dhanaraj RK, Ramakrishnan V, Poongodi M, Krishnasamy L, Hamdi M, Kotecha K, Vijayakumar V (2021) Random forest bagging and x-means clustered antipattern detection from SQL query log for accessing secure mobile data. Wirel Commun Mob Comput 1–9:2021
7. Djenouri Y, Lin JC-W, Nørvåg K, Ramampiaro H, Yu PS (2021) Exploring decomposition for solving pattern mining problems. ACM Trans Manag Inform Syst 12(2):1–36
8. Ezugwu AE, Ikotun AM, Oyelade OO, Abualigah L, Agushaka JO, Eke CI, Akinyelu AA (2022) A comprehensive survey of clustering algorithms: State-of-the-art machine learning applications, taxonomy, challenges, and future research prospects. Eng Appl Artif Intell 110:104743
9. Fournier-Viger P, Gomariz A, Campos M, Thomas R (2014) Fast vertical mining of sequential patterns using co-occurrence information. In: Proceedings of 18th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD 2014), Part I. Springer, pp 40–52
10. Fournier-Viger P, Gomariz A, Gueniche T, Mwamikazi E, Thomas R (2013) TKS: efficient mining of top-k sequential patterns. In: Proceedings of the 9th International Conference on Advanced Data Mining and Applications (ADMA 2013), Part I. Springer, pp 109–120
11. Fournier-Viger P, Lin JC-W, Gomariz A, Gueniche T, Soltani A, Deng Z, Lam HT (2016) The SPMF open-source data mining library version 2. In: Proceedings of European Conference on Machine Learning and Knowledge Discovery in Databases (ECML-PKDD) 2016, Part III. Springer, pp 36–40
12. Fournier-Viger P, Lin JC-W, Kiran RU, Koh YS, Thomas R (2017) A survey of sequential pattern mining. Data Sci Pattern Recogn 1(1):54–77
13. Fu L, Wang X, Zhao H, Li M (2022) Interactions among safety risks in metro deep foundation pit projects: An association rule mining-based modeling framework. Reliabil Eng Syst Safety 221:108381
14. Fumarola F, Lanotte PF, Ceci M, Malerba D (2016) CloFAST: closed sequential pattern mining using sparse and vertical id-lists. Knowl Inf Syst 48:429–463
15. Gan W, Lin JC-W, Fournier-Viger P, Chao H-C, Yu PS (2019) A survey of parallel sequential pattern mining. ACM Trans Knowl Disc Data 13(3):1–34
16. Gomasta SS, Dhali A, Anwar MM, Sarker IH (2022) Query-oriented topical influential users detection for top-k trending topics in twitter. Appl Intell 52(12):13415–13434
17. Guo W, Che H, Leung M-F (2024) Tensor-based adaptive consensus graph learning for multi-view clustering. IEEE Trans Consum Electron 70(2):4767–4784
18. Guo W, Che H, Leung M-F, Yan Z (2024) Adaptive multi-view subspace learning based on distributed optimization. Internet Things 26:101203
19. Huang G-Y, Yang F, Hu C-Z, Ren J-D (2010) Fast discovery of frequent closed sequential patterns based on positional data. In: Proceedings of the 2010 International Conference on Machine Learning and Cybernetics (ICMLC), volume 1. IEEE, pp 444–449
20. Huang S, Gan W, Miao J, Han X, Fournier-Viger P (2023) Targeted mining of top-k high utility itemsets. Eng Appl Artif Intell 126:107047
21. Huynh B, Vo B, Snasel V (2017) An efficient parallel method for mining frequent closed sequential patterns. IEEE Access 5:17392–17402
22. Huynh U, Le B, Dinh D-T, Fujita H (2022) Multi-core parallel algorithms for hiding high-utility sequential patterns. Knowl-Based Syst 237:107793
23. Ishita SZ, Ahmed CF, Leung CK (2022) New approaches for mining regular high utility sequential patterns. Appl Intell 52(4):3781–3806
24. Islam MA, Rafi MR, Azad A-A, Ovi JA (2022) Weighted frequent sequential pattern mining. Appl Intell 52(1):254–281
25. Ke Y-H, Huang J-W, Lin W-C, Jaysawal BP (2020) Finding possible promoter binding sites in DNA sequences by sequential patterns mining with specific numbers of gaps. IEEE/ACM Trans Comput Biol Bioinf 18(6):2459–2470
26. Kumar P, Krishna PR, Bapi RS, De SK (2007) Rough clustering of sequential data. Data Knowl Eng 63(2):183–199
27. Le T, Vo B, Huynh V-N, Nguyen NT, Baik SW (2020) Mining top-k frequent patterns from uncertain databases. Appl Intell 50:1487–1497
28. Lin JC-W, Djenouri Y, Srivastava G, Fourier-Viger P (2022) Efficient evolutionary computation model of closed high-utility itemset mining. Appl Intell 52(9):10604–10616
29. Lin JC-W, Djenouri Y, Srivastava G, Li Y, Yu PS (2021) Scalable mining of high-utility sequential patterns with three-tier mapreduce model. ACM Trans Knowl Disc Data 16(3):1–26

30. Liu Z, Ma Y, Zheng H, Liu D, Liu J (2022) Human resource recommendation algorithm based on improved frequent itemset mining. Futur Gener Comput Syst 126:284–288
31. Pamalla V, Rage UK, Penugonda R, Palla L, Hayamizu Y, Goda K, Toyoda M, Zettsu K, Sourabh S (2023) A fundamental approach to discover closed periodic-frequent patterns in very large temporal databases. Appl Intell 53(22):27344–27373
32. Pan B, Li C, Che H (2024) Error-robust multi-view subspace clustering with nonconvex low-rank tensor approximation and hyper-Laplacian graph embedding. Eng Appl Artif Intell 133:108274
33. Pei J, Han J, Mortazavi-Asl B, Wang J, Pinto H, Chen Q, Dayal U, Hsu M-C (2004) Mining sequential patterns by pattern-growth: the PrefixSpan approach. IEEE Trans Knowl Data Eng 16(11):1424–1440
34. Pham T-T, Do T, Nguyen A, Vo B, Hong T-P (2020) An efficient method for mining top-k closed sequential patterns. IEEE Access 8:118156–118163
35. Rizvee RA, Ahmed CF, Arefin MF, Leung CK (2024) A new tree-based approach to mine sequential patterns. Expert Syst Appl 242:122754
36. Rizvee RA, Arefin MF, Ahmed CF (2020) Tree-miner: Mining sequential patterns from SP-Tree. In: Proceedings of the 24th Pacific-Asia Conference in Knowledge Discovery and Data Mining (PAKDD 2020), Part II. Springer, pp 44–56
37. Roy KK, Moon MHH, Rahman MM, Ahmed CF, Leung CK-S (2022) Mining weighted sequential patterns in incremental uncertain databases. Inf Sci 582:865–896
38. Srikant R, Agrawal R (1996) Mining sequential patterns: Generalizations and performance improvements. In: Proceedings of the Fifth International Conference on Extending Database Technology (EDBT 1996). Springer, pp 1–17
39. Thiet PT (2016) Applying the attributed prefix tree for mining closed sequential patterns. Viet J Sci Technol 54(3A):106–106
40. Tripathy B et al (2019) Fuzzy clustering of sequential data. Int J Intell Syst Appl 11(1):43
41. Tzvetkov P, Yan X, Han J (2005) TSP: Mining top-k closed sequential patterns. Knowl Inf Syst 7:438–457
42. Wang J, Fang S, Liu C, Qin J, Li X, Shi Z (2020) Top-k closed co-occurrence patterns mining with differential privacy over multiple streams. Futur Gener Comput Syst 111:339–351
43. Wang J, Han J (2004) BIDE: Efficient mining of frequent closed sequences. In: Proceedings of 20th International Conference on Data Engineering (ICDE 2004). IEEE, pp 79–90
44. Wang T, Duan L, Dong G, Bao Z (2020) Efficient mining of outlying sequence patterns for analyzing outlierness of sequence data. ACM Trans Knowl Disc Data 14(5):1–26
45. Wang Y, Chen C, Lai J, Fu L, Zhou Y, Zheng Z (2023) A self-representation method with local similarity preserving for fast multi-view outlier detection. ACM Trans Knowl Disc Data 17(1):2.1–2.20
46. Wiroonsri N (2024) Clustering performance analysis using a new correlation-based cluster validity index. Pattern Recogn 145:109910
47. Wu Y, Chen M, Li Y, Liu J, Li Z, Li J, Wu X (2023) ONP-Miner: One-off negative sequential pattern mining. ACM Trans Knowl Discov Data 17(3):1–24
48. Wu Y, Luo L, Li Y, Guo L, Fournier-Viger P, Zhu X, Wu X (2021) NTP-Miner: nonoverlapping three-way sequential pattern mining. ACM Trans Knowl Disc Data 16(3):1–21
49. Wu Y, Wang Y, Li Y, Zhu X, Wu X (2021) Top-k self-adaptive contrast sequential pattern mining. IEEE Trans Cybern 52(11):11819–11833
50. Yan X, Han J, Afshar R (2003) CloSpan: Mining closed sequential patterns in large datasets. In: Proceedings of the 2003 SIAM International Conference on Data Mining (SDM 2003), pp 166–177
51. Yang X, Che H, Leung M-F, Wen S (2024) Self-paced regularized adaptive multi-view unsupervised feature selection. Neural Netw 175:106295
52. Zaki MJ (2001) SPADE: An efficient algorithm for mining frequent sequences. Mach Learn 42(1):31–60
53. Zhang C, Du Z, Gan W, Philip SY (2021) TKUS: Mining top-k high utility sequential patterns. Inf Sci 570:342–359
54. Huang G, Gan W, TaSPM PSYu (2024) Targeted sequential pattern mining. ACM Trans Knowl Disc Data 18(5):1–18
55. Djenouri Y, Belhadi A, Srivastava G, Lin JC (2023) Advanced pattern-mining system for fake news analysis. IEEE Trans Comput Soc Syst 10(6):2949–2958
56. Sun C, Ren X, Dong X, Qiu P, Wu X, Zhao L, Guo Y, Gong Y, Zhang C (2024) Mining actionable repetitive positive and negative sequential patterns. Knowl-Based Syst 302:112398. Elsevier
57. Huang G, Gan W, Yu PS (2024) TaSPM: Targeted sequential pattern mining. ACM Trans Knowl Disc Data 18(5):114
58. Hu K, Gan W, Huang S, Peng H, Fournier-Viger P (2024) Targeted mining of contiguous sequential patterns. Inform Sci 653:119791. Elsevier
59. Chen Z, Gan W, Huang G, Zheng Y, Yu PS (2024) Towards utility-driven contiguous sequential patterns in uncertain multi-sequences. Knowl-Based Syst 284:111314. Elsevier
60. Zhang C, Lyu M, Gan W, Yu PS (2024) Totally-ordered sequential rules for utility maximization. ACM Trans Knowl Disc Data 18(4):80
61. Wan X, Han X (2024) Efficient top-k frequent itemset mining on massive data. Data Sci Eng 9:177–203. Springer

Redwan Ahmed Rizvee earned his B.Sc. and M.S. degrees in Computer Science from the Department of Computer Science and Engineering, University of Dhaka, Bangladesh, in 2019 and 2021, respectively. Currently, he is working as a lecturer in the same department. Before joining the University of Dhaka, he worked as a Research Scientist at TigerIT Bangladesh Limited from 2020 to 2022 and as a lecturer in the Department of Computer Science and Engineering at East West University, Bangladesh from 2022 to 2023. His research interests focus on Data Mining, High-Performance Computing, Programming Systems and Deep Learning.

Chowdhury Farhan Ahmed received the B.Sc. and M.Sc. degrees in Computer Science from the University of Dhaka, Bangladesh in 2000 and 2002 respectively, and the Ph.D. degree in Computer Engineering from the Kyung Hee University, South Korea in 2010. He was a postdoctoral research fellow in the ICube Laboratory, University of Strasbourg, France from 2013 to 2015. He worked as a visiting scholar in the Faculty of Engineering and Information Technology, University of Technology Sydney, Australia in March, 2019. Since 2004, he has been working as a faculty member (and as a Professor since 2016) in the Department of Computer Science and Engineering, University of Dhaka, Bangladesh. His research interests are in the areas of data mining, knowledge discovery, data science and machine learning.

Carson K. Leung received his B.Sc.(Hons.), M.Sc. and Ph.D. degrees, all in computer science, from the University of British Columbia, Vancouver, Canada. He is currently a Professor at the University of Manitoba, Canada. He has contributed more than 400 refereed publications on the topics of applied intelligence, artificial intelligence, big data, bioinformatics, data analytics, data mining, data science, health informatics, machine learning, social network analysis, and visual analytics. These include publications in international journals and conferences such as *ACM Transactions on Database Systems (TODS), ACM Transactions on Knowledge Discovery from Data (TKDD)*, IEEE ICDE, IEEE ICDM, and PAKDD. Moreover, he has served as the Editor-in-Chief for *Advances in Data Science and Adaptive Analysis (ADSAA)* and for *Analytics*, as well as an Associate Editor for international journals like Springer's *Social Network Analysis and Mining (SNAM)*. He has served on the Organizing Committees of the ACM CIKM, ACM KDD, ACM SIGMOD, DaWaK, IEEE DSAA, IEEE ICDM, and other conferences; he has also served as a PC member of numerous conferences including ACM KDD, ECML/PKDD and PAKDD. He is a Senior Member of both the ACM and the IEEE; he is also an IEEE Computer Society Distinguished Contributor.