Preprint of:
Lan, D., Sun, C., Dong, X., Qiu, P., Gong, Y., Liu, X., Fournier-Viger, P., Zhang, C. (2025). .TK-RNSP: Efficient Top-K Repetitive Negative Sequential Pattern mining. Information processing and management (IPM). 62(3): 104077.

TK-RNSP: Efficient Top-K Repetitive Negative Sequential Pattern Mining

Abstract

In the field of behavioral analysis, Repetitive Negative Sequential Patterns (RNSPs) provide crucial information for uncovering the significance of events. However, due to the uncertainty of datasets, users often find it challenging to set appropriate support thresholds, resulting in decreased mining efficiency. To address this issue, we propose an efficient Top-K Repetitive Negative Sequential Pattern mining algorithm (TK-RNSP). This algorithm leverages Top-K techniques to eliminate the difficulty of setting support thresholds, ensuring efficient completion of Top-K RNSP mining tasks even when the support is infinitesimally close to or equal to zero. TK-RNSP conducts pattern mining under self-adaptive gap conditions, generating candidate patterns through sequence extensions and itemset extensions. We demonstrate that within the framework defined in this paper, RNSP mining satisfies the anti-monotonicity property. Based on this observation, a pruning strategy based on the anti-monotonicity is integrated in TK-RNSP to quickly raise the minimum support threshold. Moreover, to further enhance the efficiency, TK-RNSP also relies on a novel technique using bitmaps for pattern information storage and support calculation. This method combines bitwise operations with depth-first search and backtracking strategies to achieve efficient pattern search and support calculation. Experimental results indicate that TK-RNSP exhibits excellent efficiency and stability in Top-K RNSP mining tasks, demonstrating its substantial potential as a data analysis tool.

Keywords: Sequential pattern mining, Negative sequential pattern, Top-K repetitive negative sequential pattern, Nonoverlapping, Self-adaptive gap

1 Introduction

Behavioral analysis is pervasive in people's daily lives [1], with one of its significant tasks being to understand and interpret the complexity and impact of non-occurring behaviors (NOBs) [2]. In recent years, increasing attention has been paid to Negative Sequential Pattern (NSP) mining, as it facilitates the understanding of NOBs [3]. NSPs are frequent sequences that contain both non-occurring and occurring behaviors (also called negative and positive behaviors in behavior and sequence analysis) [4]. A NOB is for instance, a vehicle not yielding to pedestrians at a crosswalk or a patient not taking a prescribed medication during treatment.

Mining and analyzing NSPs play an indispensable role in various real-world applications such as plan optimization, short video recommendation, biological genomic research, and criminal investigation [5–7]. For instance, in short video recommendation sequences, a positive sequential pattern (PSP) $s = \langle abcX \rangle$ may indicate that after watching some videos a, b, and c, a user is likely to purchase a product X. Conversely, a NSP $s = \langle a\neg bcY \rangle$ indicates that after watching video a and skipping video b, and then watching video c, a user is likely to purchase a product Y. As illustrated by this example, identifying NSPs allows uncovering important behavior patterns that cannot be revealed by PSPs alone.

Despite the growing interest in Negative Sequential Pattern (NSP) mining in recent years, there are still relatively few algorithms; notable among them are e-NSP [4], F-NSP [8], NegINSP [9], and sc-NSP [10]. However, these algorithms are limited as they exclusively focus on the presence of NSPs in sequences but overlook the essential feature of pattern repetitions. Recent research on NSP mining has highlighted that Repetitive Sequential Patterns (RSPs) are crucial for uncovering comprehensive and valuable patterns and rules within data, thereby enhancing decision-making [11]. For example, consider the sequence $s = \langle ab\neg c\, ab\neg c\, ab\neg c\, Y \rangle$, representing a suspect's behavior: repeated actions a (loitering) and b (observing), followed by the absence of action c (purchase), may be a strong indicator of criminal activity Y (theft). Thus, analyzing the repeated occurrences of the pattern $p = \langle ab\neg c \rangle$ within the sequence s is vital for addressing retail theft cases.

Although intensive efforts have been made to develop Repetitive PSP (RPSP) mining algorithms [12–24], Repetitive NSPs (RNSPs) cannot be discovered by these algorithms, and thus the information carried by these patterns is missed. The e-RNSP algorithm is the first to effectively address the RNSP mining problem [11]. ONP-Miner, by applying strict one-off pattern matching conditions, provides valuable pattern recognition support for the field of bioinformatics [25]. The SN-RNSP algorithm employs relatively loose constraints and utilizes bitmap structures, significantly improving mining accuracy and efficiency [26]. But all these methods are based on setting a minimum threshold (such as the minimum support ms). However, due to limited domain expertise, users often find it very difficult to set a suitable minimum threshold to obtain an expected number of patterns. A too small value of ms may generate thousands of patterns, while a too large value may yield no results. Adjusting ms is thus often done by trial and error and very time-consuming.

A similar problem of threshold adjustment is found in PSP mining and RPSP mining and was solved by proposing algorithms for top-k PSP and RPSP mining, where k is the expected number of PSPs or RPSPs to be output [27–31]. Top-k algorithms can identify the most valuable k patterns by performing dynamic improvement, starting from a small initial support threshold and increasing it through the search. Up to now, only the top-k NSP+ algorithm has provided a preliminary solution to address the threshold setting problem in NSP mining [32]. It effectively mines the first k most useful NSPs by introducing weighted support interest measures and pruning strategies. But NSP+ does not consider the recurring characteristics of NSPs and is hence unable to perform the task of top-k RNSP mining. This is because mining top-k

RNSPs is much more difficult than mining the top-k NSPs, particularly due to the following two intrinsic complexities:

1. **Pruning techniques limitations**. While Top-k RNSP mining would avoid the need for users to set a support threshold, ensuring the accuracy of mining results requires starting with a low minimum support threshold (close to or equal to zero) and gradually increasing it. Consequently, pruning techniques that can quickly raise the threshold to reduce the search space are essential. However, the existing frequency-oriented RNSP mining framework does not provide an anti-monotonicity property and thus cannot leverage the search space pruning strategies used for PSPs. More importantly, current RNSP pruning techniques are often inefficient and may result in the loss of truly top-k patterns, making them unsuitable for Top-k RNSP mining. Therefore, developing efficient and reliable pruning methods for RNSPs is a critical challenge in achieving Top-k RNSP mining.
2. **Heavy computational burden.** To ensure the complete results of Top-k RNSP mining, it is necessary to conduct mining in a large candidate space, which incurs great computational costs. Therefore, it is necessary to develop high efficiency methods for pattern search and support calculation. But most RNSP methods derive negative sequential candidates (NSCs) by converting PSPs that meet certain conditions, resulting in a two-stage mining process that cannot directly and rapidly perform pattern search and calculations for RNSPs. Although existing methods add negative constraints to limit the number of NSCs, a large number of invalid intermediate patterns are still generated. Additionally, some NSCs may not be generated during the conversion process, which clearly cannot meet the requirements for efficient and complete Top-k RNSP mining. Therefore, how to efficiently and quickly perform pattern search and support calculations for RNSPs in a very large candidate space is a key challenge.

To overcome the above challenges, we propose an efficient Top-K RNSP mining algorithm named TK-RNSP. The main contributions of this paper are:

First, we formally define the Top-k RNSP mining task and provide a new definition for RNSP occurrence. In this definition, RNSP mining satisfies an anti-monotonicity property, and a detailed theoretical proof is given. This allows us to approximately transform the Top-k RNSP problem into that of Top-k RPSP mining and perform efficient and accurate pruning to ensure the accuracy of the mining results.

We propose a bitmap-based Depth-First Backtracking Search (DFBS) strategy that enables efficient pattern generation, as well as the search and support calculation for both RPSPs and RNSPs. By leveraging the DFBS strategy, the algorithm incrementally explores candidate patterns and generates corresponding bitmap representations for support calculation, while the bitmap mechanism ensures efficient support calculation for both RPSPs and RNSPS.

In addition, we propose a one-stage efficient Top-k RNSP mining algorithm, called TK-RNSP. This algorithm mines patterns under relatively loose constraints such as self-adaptive gap constraints, and relies on the pattern generation method of NegPSPan to ensure a relatively complete candidate space; it adopts an anti-monotonicity-based pruning method to quickly narrow the search space, allowing the

threshold to be quickly raised and uses the DFBS strategy to search for RPSPs and RNSPs, calculating pattern support through bit operations, thereby improving the algorithm's computational efficiency.

Finally, through extensive experiments on eight datasets, the correctness of the pruning strategy and the effectiveness of the RNSP pattern search strategy are verified, and the excellent mining performance of TK-RNSP is demonstrated. The results show that TK-RNSP has good adaptability and efficiency in various data environments (including synthetic and real datasets, and datasets containing itemsets and single items). In particular, in the test of completing the Top-k RNSP task, the accuracy and efficiency of TK-RNSP significantly surpassed E-RNSP and SN-RNSP.

The structure of the paper is as follows: Section 2 reviews related work, Section 3 defines relevant concepts, Section 4 details the algorithm design and implementation, Section 5 validates the algorithm's performance through experiments, and Section 6 concludes the paper.

2 Related work

Current research has extensively explored PSPs, including RPSPs, high-utility PSPs, PSPs in data streams, Top-K PSPs, and interdisciplinary studies involving these pattern types. Concurrently, research on NSPs is progressing, primarily focusing on the efficiency of NSP mining. NSP research provides a novel perspective for PSP analysis, serving as a significant complement and further refining the analysis of sequential data. We provide an overview of the development of NSP mining, the exploration of RPSPs and RNSPs, and the state and progress of research on Top-K PSP and NSP mining.

2.1 Research on negative sequential pattern mining

In the domain of NSP mining, the landscape is rich with algorithms such as E-NSP [4], F-NSP [8], SC-NSP [10], NegI-NSP [9], and E-RNSP [11], which are designed to enhance mining efficiency and provide additional features. In particular, F-NSP [8] utilizes bitmap structures for quick NSP support calculations, while SC-NSP [10], NegI-NSP [9], and Zheng's genetic approach [33] ease F-NSP's constraints or innovate on pattern generation. Qiu et al.'s msNSPFI [34] can extract insights from infrequent sequences, and the MLMS-NSP [35] algorithm stands out for its ability to handle uniform support thresholds with its multilevel scheme. The HUNSPM algorithm [36] explored high-utility NSP mining but did not address the processing of repetitive occurrences.

Different researchers have used various constraints, as summarized in Table 1. For instance, Cao et al. [4] and Zheng et al. [37] adopted the first four constraints; Guyet et al. [38] used the second, fourth, and fifth constraints and proposed the NegPSpan algorithm; Wang et al. [39] in their VM-NSP algorithm relaxed the negative element constraint and adjusted other constraints. Xu et al. in E-msNSP [40] shifted to multiple minimum support constraints, Wu et al. in ONP-Miner [25] applied the second, fourth, and fifth constraints along with various gap constraints, Gong et al. in e-NSPFI [41] relaxed the format constraint, Qiu et al. in NegI-NSP [9] loosened the

Table 1: Comparison of Constraints Used in Different Algorithms

Algorithm	Constraints				
	Negative Element	Frequency	Format	Minimum Support	Gaps
Cao et al. [4]	Yes	Yes	Yes	Yes	No
Zheng et al. [37]	Yes	Yes	Yes	Yes	No
Guyet et al. [38]	No	Yes	No	Yes	Yes
Wang et al. [39]	Loose	Yes	Yes	Yes	No
Xu et al. [40]	Yes	Yes	Yes	Multiple	No
Wu et al. [25]	No	Yes	No	Yes	Yes
Gong et al. [41]	Yes	Yes	Loose	Yes	No
Qiu et al. [9]	Loose	Yes	Loose	Yes	No
Qiu et al. [32]	Yes	Yes	Yes	No	No
Our Algorithm	Yes	Yes	Yes	No	No

negative element constraint and format constraint, while Top-K NSP [32] removed the minimum support constraint.

In NSP mining, key factors for algorithm efficiency include generating and storing NSCs, pruning strategies, and support calculations. The diverse negative containment definitions and constraints lead to various methods. The NegGSP algorithm by Zheng et al. [37] and the PNSP algorithm by Hsueh et al. [42] extended the GSP [43] algorithm. Guyet et al.'s [38] NegPSpan, akin to PrefixSpan [44] is adapted for the maxgap and maxspan constraints. F-NSP [4] efficiently computes support by redefining negative containment as positive, avoiding database rescans. F-NSP [8] uses bitmap structures for faster support calculations, while SC-NSP [10] improves space efficiency with enhanced PrefixSpan and bitmap structures. VM-NSP [39] uses a vertical mining framework to efficiently find NSP sets, and its bitmap-based BM-NSP approach improves mining efficiency under relaxed constraints. These methods illustrate the diversity and complexity of NSP mining algorithms and suggest ways to enhance efficiency.

2.2 Research on repetitive positive and negative sequential pattern mining

In the domain of RPSP mining, the nonoverlapping condition serves as a balanced gap constraint, facilitating the extraction of a suitable number of patterns that comply with the anti-monotonicity property. Various algorithms have been developed to enhance efficiency and address the limitations of previous approaches. Notably, Wu et al. [12] introduced the NOSEP algorithm, which utilizes the Net-tree and NETGAP algorithms for effective pattern support calculations and pruning. Further advancements were made by Wu et al. [13] with NWP-Miner, Shi et al. [14] with NetNPG, and Wu et al. [15] with NetNCSP, incorporating strategies for nonoverlapping pattern matching and closed sequential pattern mining. Additionally, Li et al. [16] designed the MCoR-Miner algorithm for discovering nonoverlapping sequential rules, automating support and confidence threshold calculations with depth-first and backtracking strategies. Wu et al.'s [17] NTP-Miner targeted nonoverlapping tri-directional sequential patterns, introducing interest levels to minimize redundancy. To address the challenge of setting gap constraints without prior knowledge, Wang et al. [18] reduced

candidate patterns through pattern joining and an incomplete Net-tree structure.The NTP-Miner algorithm [20] leverages a single-root Net-tree alongside depth-first and backtracking strategies for efficient support calculation and employs a pattern joining strategy to curtail the generation of candidate patterns. In the field of time series data analysis, the COP-Miner [21] and RNP-Miner [22] algorithms introduced by Wu et al. have expanded the methodologies for analyzing time series data. Additionally, Wu et al.'s work [23, 24] on high utility factors has broadened the scope of sequential pattern mining, offering enhanced flexibility for users.

In the realm of NSP mining, although research is somewhat limited, algorithms such as the E-RNSP [11], ONP-Miner [25] and SN-RNSP [26] have provided new insights into this area. The E-RNSP algorithm optimizes the calculation of support by transforming repetitive negative containment checks into positive containment checks. The ONP-Miner algorithm, which mines patterns under one-off conditions, can unearth more interesting patterns. Additionally, the SN-RNSP algorithm leverages a bitmap structure and a bitmap-based mining method for rapid mining of RNSPs.

2.3 Research on Top-K positive and negative sequential pattern mining

Top-K algorithms, which mine the K patterns with the highest support/utility without setting a support threshold, have primarily been explored in PSP mining. For instance, Huang et al. [45] introduced the SARA and SARS algorithms for mining TIED patterns, characterized by time intervals and durations, while Lei et al. [30] developed a two-step sampling algorithm for mining approximate Top-K PSPs. Davashi et al. [46] proposed the ITUFP algorithm, an efficient method for interactive mining of Top-K frequent patterns in uncertain data. Wang et al. [47] investigated the problem of Top-K high-utility PSP mining and proposed the HUS-Span and TKHUS-Span algorithms. Zhang et al. [27] proposed the TKUS algorithm,which employs projection and local search mechanisms to reduce the search space. Kieu et al. [48] proposed a suite of algorithms for mining top-k co-occurrence items with sequential patterns, including the Naive Approach Mining (NAM), Vertical Approach Mining (VAM), and Vertical with Index Approach Mining (VIAM). For RPSPs, Wu et al. [28] introduced the SCP-Miner algorithm, which effectively mines the K patterns with the greatest contrast. Huang et al. [49] proposed the TMKU algorithm, which effectively integrates target patterns and Top-k pattern mining to achieve efficient mining of target high-utility itemsets. Nguyen et al. [50] introduced the ETARM algorithm, which incorporates two new pruning properties, effectively addressing the computational overhead issues in Top-k association rule mining. Qiu et al.'s Top-K NSP algorithm [32] represents one of the earliest and only works in NSP mining, utilizing optimization strategies such as weighted support and pruning. However, as our research focuses on RNSP mining, we did not perform direct experimental comparisons with their work, which is centered on NSPs. Additionally, the Top-K NSP algorithm is still in its early stages, leaving room for improvements in accuracy and efficiency.

Inspired by two previous studies, namely SN-RNSP [26] and Top-K NSP [32], this paper proposes a novel Top-K RNSP mining scheme. Although SN-RNSP [26] demonstrates certain efficiency advantages in mining RNSPs, it relies on a preset

fixed support threshold, which limits its flexibility in different data environments. On the other hand, Top-K NSP [32] selects Top-K PSP patterns and transforms them into Top-K NSP patterns, but its design is not capable of handling the Top-K RNSP mining task, and the algorithm is still in its early stages, requiring improvements in accuracy and efficiency.

In this paper, we address the limitations of fixed support threshold by incorporating Top-K techniques and propose a solution that does not require preset support values. Moreover, we innovatively employ a DFBS method for pattern search, and efficiently calculate support through bitmap operations. More importantly, the RNSP mining framework proposed in this paper satisfies the anti-monotonicity property, and we utilize this property for pruning, which significantly enhances the efficiency of the algorithm.

3 Problem statement

Let $I = \{i_1, i_2, \ldots, i_q\}$ be a set of items. An itemset is a subset of I. Without loss of generality, it is assumed that the items within an itemset are arranged in lexicographic order. A sequence is an ordered list of multiple itemsets, denoted as $s = \langle e_1 e_2 \ldots e_n \rangle$, where $e_k \subseteq I$ $(1 \leq k \leq n)$. Each e_k is also referred to as an element of the sequence. The number of elements in a sequence s is termed the size of the sequence, denoted as size(s). If size(s) $= n$, then the sequence s is called an n-Size sequence. The total number of items contained in all elements of sequence s is known as the length of the sequence, denoted as length(s). If length(s) $= r$, then sequence s is referred to as an r-length sequence. For example, the sequence $s = \langle (abc)b(cd) \rangle$ has a size of 3 and a length of 6.

Table 2: Sequence database

Sequence ID (sid)	Sequence
1	$\langle aabc \rangle$
2	$\langle (abc)(ab)c(bc) \rangle$
3	$\langle abc \rangle$

A sequence database DB is a collection of tuples $< sid, s >$, where s represents a sequence, and sid denotes the identifier of sequence s, expressed as $DB = \{s_1, ..., s_N\}$. In the example DB of Table 2, $DB = \{s_1, s_2, s_3\}$.

Definition 1 *(Subsequence and Supersequence) A sequence* $s = \langle e_1 e_2 \ldots e_m \rangle$ *is a subsequence of a sequence* $s^\alpha = \langle e_1^\alpha e_2^\alpha \ldots e_n^\alpha \rangle$ *and* s^α *is a supersequence of* s*, denoted by* $s \subseteq s^\alpha$*, if there exist integers* $1 \leq l_1 < l_2 < \ldots < l_m \leq n$ *such that* $e_1 \subseteq e_{l_1}^\alpha, e_2 \subseteq e_{l_2}^\alpha, \ldots, e_m \subseteq e_{l_m}^\alpha$.

Example 1 *In Table 2,* $p = \langle ab \rangle$ *is a subsequence of* $s_2 = \langle (abc)(ab)c(bc) \rangle$.

Definition 2 *(Occurrence and Pattern Instance) Let there be a sequence* $s = \langle e_1 e_2 \ldots e_m \rangle$ *that is a subsequence of a sequence* $s^\alpha = \langle e_1^\alpha e_2^\alpha \ldots e_n^\alpha \rangle$ *as per Definition 1. Such a sequence of integers* $\langle l_1, l_2, \ldots, l_m \rangle$ *is called an occurrence of* s *in* s^α*. Given a sequence database* $DB = \{s_1, s_2, \ldots, s_N\}$*, if an occurrence of* $p = \langle e_1, e_2, \ldots, e_m \rangle$

exists as $\langle l_1, l_2, \ldots, l_m \rangle$ *in* s_j *(*$s_j \in DB$*), the tuple* $(j, \langle l_1, l_2, \ldots, l_m \rangle)$ *is referred to as an instance of* p *in* DB*.*

Example 2 *In Table 2, the pattern* $p = \langle ab \rangle$ *has two occurrences in* s_1*:* $\langle 1, 3 \rangle$ *and* $\langle 2, 3 \rangle$*; three occurrences in* s_2*:* $\langle 1, 2 \rangle$*,* $\langle 1, 4 \rangle$*, and* $\langle 2, 4 \rangle$*; and one occurrence in* s_3*:* $\langle 1, 2 \rangle$*. Overall,* $p = \langle ab \rangle$ *has six instances:* $(1, \langle 1, 3 \rangle)$*,* $(1, \langle 2, 3 \rangle)$*,* $(2, \langle 1, 2 \rangle)$*,* $(2, \langle 1, 4 \rangle)$*,* $(2, \langle 2, 4 \rangle)$*, and* $(3, \langle 1, 2 \rangle)$*.*

Definition 3 *(Self-Adaptive Gap Constraints) A self-adaptive pattern* p *with size* m *can be expressed as* $p = \langle e_1 * e_2 * \cdots * e_m \rangle$*, where* $*$ *represents any number of intervening elements (including zero). The gap between two elements* e_i *and* e_{i+1} *refers to the number of elements appearing between them in the sequence. "Self-adaptive" means that the number of these intervening elements is unrestricted and can vary freely.*

Example 3 *In Table 2,* $\langle 1, 2 \rangle$ *represents an occurrence of the pattern* $p = \langle ab \rangle$ *in* s_2*. Simultaneously,* $\langle 1, 2 \rangle$ *also signifies an occurrence of* p *in* s_3*. To distinguish occurrences in different sequences, the concept of a pattern instance is introduced.*

Definition 4 *(Nonoverlapping Instance) Two instances of a pattern* $p = \langle e_1 e_2 \ldots e_m \rangle$ *in* DB*,* $(j^\alpha, \{l_1^\alpha, l_2^\alpha, \ldots, l_m^\alpha\})$ *and* $(j^\beta, \{l_1^\beta, l_2^\beta, \ldots, l_m^\beta\})$*, are nonoverlapping if* $j^\alpha \neq j^\beta$ *or* $\forall 1 \leq k \leq m, l_k^\alpha \neq l_k^\beta$*.*

Example 4 *In Example 2,* $(2, \langle 1, 2 \rangle)$ *and* $(2, \langle 2, 4 \rangle)$ *are nonoverlapping. However,* $(2, \langle 1, 2 \rangle)$ *and* $(2, \langle 1, 4 \rangle)$ *do not meet the nonoverlapping condition because "a" is used twice at the same position in* s_2*.*

Definition 5 *(Occurrence set and Instance set) The set of occurrences of pattern* p *within the sequence is referred to as the occurrence set, where each occurrence is nonoverlapping with any other. The set of instances of pattern* p *within the database* DB *is referred to as the instance set, where each instance is nonoverlapping with any other. Note that there may exist more than one Instance set of a pattern.*

Example 5 *For the instances in Example 2,* $S_1 = \{(1, \langle 1, 3 \rangle), (2, \langle 1, 2 \rangle), (2, \langle 2, 4 \rangle), (3, \langle 1, 2 \rangle)\}$*,* $S_2 = \{(1, \langle 1, 3 \rangle), (2, \langle 1, 4 \rangle), (3, \langle 1, 2 \rangle)\}$*,* $S_3 = \{(1, \langle 2, 3 \rangle), (2, \langle 1, 2 \rangle), (2, \langle 2, 4 \rangle), (3, \langle 1, 2 \rangle)\}$*, and* $S_4 = \{(1, \langle 2, 3 \rangle), (2, \langle 1, 4 \rangle), (3, \langle 1, 2 \rangle)\}$ *are four instance sets of* $p = \langle ab \rangle$ *satisfying that any two instances are nonoverlapping.*

Definition 6 *(Support Set and Support) The support of a pattern* p *in* DB *is defined as the maximal instance number of all possible instance sets in which any two instances are nonoverlapping. The support of* p *is denoted by* $sup(p)$*. An instance set containing* $sup(p)$ *instances is called a support set of* p*. Note that there may exist more than one support set of a pattern. For more detailed explanations on support and support set, please refer to [26].*

Example 6 *In Example 5,* S_1 *is a support set of* p *in* DB *because it possesses the maximal instance number. The support of* p *is 4, i.e.,* $sup(p)$*=4. In addition,* S_3 *is also a support set of* p *in* DB*.*

Corollary 1 *From the relationship between the support set and the instance set described in Definition 6, it is evident that the size of the support set is always greater than or equal to that of the instance set. For a single sequence, the size of the corresponding support set is equal to the size of the occurrence set with the highest number of occurrences for the sequence.*

Example 7 *In Example 2, the occurrence set of* $p = \langle a, b\rangle$ *in* s_2 *is* $\{\langle 1, 2\rangle, \langle 2, 4\rangle\}$*, and the support set in* s_2 *is* $\{(2, \langle 1, 2\rangle), (2, \langle 2, 4\rangle)\}$*. The support of* p *in* s_2 *is 2.*

Definition 7 *(Ascending Order of Instance) Given two instances* $(j^\alpha, \{l_1^\alpha, \ldots, l_m^\alpha\})$ *and* $(j^\beta, \{l_1^\beta, \ldots, l_m^\beta\})$ *of pattern* p *in* DB*, the instance* $(j^\alpha, \{l_1^\alpha, \ldots, l_m^\alpha\})$ *occurs before* $(j^\beta, \{l_1^\beta, \ldots, l_m^\beta\})$ *in the ascending order if:* $(1) j^\alpha < j^\beta$ *or* $(2)(j^\alpha = j^\beta \wedge l_k^\alpha < l_k^\beta$ *for* $1 \leq k \leq m)$*.*

Example 8 *In Table 2, the instances of* $p = \langle bc\rangle$ *are* $(1, \langle 3, 4\rangle)$*,* $(2, \langle 1, 3\rangle)$*,* $(2, \langle 1, 4\rangle)$*,* $(2, \langle 2, 3\rangle)$*,* $(2, \langle 2, 4\rangle)$*,* $(3, \langle 2, 3\rangle)$*. Among these,* $(2, \langle 1, 4\rangle)$ *and* $(2, \langle 2, 3\rangle)$ *do not satisfy the ascending order because they violate the condition* (2)*,* $1 < 2$ *but* $4 > 3$*. The instances* $(2, \langle 1, 4\rangle)$ *and* $(3, \langle 2, 3\rangle)$ *satisfy the ascending order as they occur in different sequences, fulfilling condition* (1)*.*

Definition 8 *(Leftmost Support Set) In a database* DB*, for a pattern* $p = \langle e_1 e_2 \ldots e_m\rangle$*, the leftmost support set* S *is the set of all support sets that are positioned farthest to the left. Under the condition of ascending order of instance, let* $S = \{(j^{(w)}, \langle l_1^{(w)}, \ldots, l_m^{(w)}\rangle) \mid 1 \leq w \leq sup(p)\}$*. For any other support set* $(j^{\alpha(w)}, \langle l_1^{\alpha(w)}, \ldots, l_m^{\alpha(w)}\rangle)$*, it must satisfy for all* $1 \leq w \leq sup(p)$ *and* $1 \leq k \leq m$ *that* $l_k^{(w)} \leq l_k^{\alpha(w)}$*.*

Example 9 *In Example 6, we know that the two support set of* $p = \langle ab\rangle$ *in* DB *are* S_1 *and* S_3*. The leftmost support set of* $p = \langle ab\rangle$ *is* $S_1 = \{(1, \langle 1, 3\rangle), (2, \langle 1, 2\rangle), (2, \langle 2, 4\rangle), (3, \langle 1, 2\rangle)\}$*. As can be seen from the Definition 8, the leftmost support set for a pattern is unique in* DB*.*

Definition 9 *(Non inclusion) An element* e *is not included in another element* e^α*, denoted by* $e \not\subseteq e^\alpha$*, if* $\forall i \in e, i \notin e^\alpha$*. Furthermore,* e *does not belong to a sequence* $s = \langle e_1 e_2 \ldots e_n\rangle$*, denoted by* $e \notin s$*, if* $\forall 1 \leq k \leq n, e \not\subseteq e_k$*.*

Example 10 *(df) does not include (cd), i.e., (cd)* $\not\subseteq$ *(ef), but (cd)* $\not\subseteq$ *(cf) is false because c occurs in (cf). Besides, (cd)* $\not\subseteq \langle b(e)f\rangle$*, but (cd)* $\not\subseteq \langle b(c)f\rangle$ *is false because (cd)* $\subseteq$ *(cf) is false.*

Definition 10 *(Positive Partner) The positive partner of a negative element* $\neg e$ *is* e*, denoted by* $p(\neg e) = e$*. Especially, the positive partner of a positive element* e *is itself, denoted by* $p(e) = e$*.*

The constraints are defined to ensure rigor and clarity:

Constraint 1 *Negative Element Constraint: The smallest component in any NSP is a single element, which must be uniformly positive or negative, e.g.,* $\langle a(a\neg b)ca\rangle$ *violates this, while* $\langle a\neg(ab)ca\rangle$ *complies.*

Constraint 2 *Size Constraint: A negative sequence pattern (NSP) cannot exceed the size of the data sequence that supports it. In other words, a data sequence cannot support an NSP that is bigger than the sequence itself.*

Constraint 3 *Format Constraint: Negative elements cannot consecutively appear within a sequence.*

Constraint 4 *Positional Constraint: A pattern 'p' must not start or end with a negative element, e.g.,* $< \neg ab >$ *and* $< b\neg a >$ *are not allowed.*

Definition 11 *(Maximum Positive Subsequence). The maximum positive subsequence of a negative sequence* ns *is an ordered list of all the positive elements in* ns*, denoted by* $MPS(ns)$*.*

Definition 12 *(Negative Occurrence) A sequence* $s = \langle e_1 e_2 \dots e_n \rangle$ *contains a negative sequence* $ns = \langle e_1^\alpha e_2^\alpha \dots e_m^\alpha \rangle$*, if there exists an occurrence* $\langle \dots, l_{k-1}, l_{k+1}, \dots \rangle$ *of* $MPS(ns)$*, for each negative element* e_k^α *in* ns *such that* $\exists \langle a_{l_{k-1}+1} \dots a_{l_{k+1}-1} \rangle$*,* $p(e_k^\alpha) \not\subseteq \langle a_{l_{k-1}+1} \dots a_{l_{k+1}-1} \rangle$*. Furthermore, such occurrence* $\langle \dots, l_{k-1}, l_{k+1}, \dots \rangle$ *is called an occurrence of* ns*.*

Example 11 *Take the DB in Table 2 as an example and consider a negative sequence* $ns = \langle a \neg ab \rangle$*. Instances of* $MPS(ns) = \langle ab \rangle$ *are* $(1, \langle 1, 3 \rangle)$*,* $(1, \langle 2, 3 \rangle)$*,* $(2, \langle 1, 2 \rangle)$*,* $(2, \langle 1, 4 \rangle)$*,* $(2, \langle 2, 4 \rangle)$*, and* $(3, \langle 1, 2 \rangle)$*. In* s_2*,* $\langle 1, 4 \rangle$ *has the occurrence of the positive element* b *in the corresponding intervals, so it is not an eligible occurrence.* $\langle 2, 3 \rangle$ *in* s_1*,* $\langle 1, 2 \rangle$ *in* s_2*, and* $\langle 1, 2 \rangle$ *in* s_3 *do not have any elements in the corresponding intervals, which is also not allowed. Moreover,* $\langle 1, 3 \rangle$ *in* s_1 *and* $\langle 2, 4 \rangle$ *in* s_2 *are two occurrences of* ns*, because the elements in the corresponding intervals* $\langle 1, 3 \rangle$ *and* $\langle 2, 4 \rangle$ *do not contain the positive element* b*. Furthermore, the support sets of* ns *are* $\{(1, \langle 1, 3 \rangle), (2, \langle 2, 4 \rangle)\}$*, and this set is also the leftmost support set.*

Definition 13 *(Top-K Repetitive Negative Sequential Pattern) The Top-K RNSP mining task aims to identify the Top-K sequential patterns with the highest frequency of repetition within a database, referred to as Top-K RNSPs. Among these patterns, the support of the K-th pattern having the lowest support, called Min-sup, represents the minimum threshold required to satisfy the frequency condition.*

The output of the task is K patterns. However, there are two special cases, which may occur. First, if K is set to a very large number and fewer than K patterns meet the frequency condition, the result will include all patterns that meet the condition, even if there are fewer than K patterns. Second, if there are more than K patterns with exactly the same support, then the Top-K patterns will include the K patterns that were inserted first into the min-heap. When the heap is full, the minimum support (Min-sup) threshold will be updated to the value of the support of these patterns. Subsequently, any patterns with support less than or equal to this Min-sup will not be included in the result set.

Example 12 *In the case of Table 2, when performing Top-K RNSP mining with* $K = 3$*, the resulting Top-K patterns are:* $\langle ab \neg ac \rangle : 1$*,* $\langle a \neg ac \rangle : 1$*, and* $\langle aa \neg ac \rangle : 3$*. However, during the pattern generation process, another pattern* $\langle a \neg cb \rangle : 1$ *is also generated. However, when filtering this pattern, the minimum support Min-sup* $= 1$*, and since this pattern does not meet the condition of having a support greater than the minimum support, it does not appear in the final results.*

4 TK-RNSP algorithm

TK-RNSP is an innovative method for mining Top-K RNSPs, operating under nonoverlapping and self-adaptive gap constraints. The various components of the TK-RNSP algorithm are discussed in the following sections. Section 4.1 describes the storage structure. Section 4.2 explains the pattern generation strategy. Section 4.3 gives the proof of anti-monotonicity. Section 4.4 introduces the pruning strategy. Then, Section 4.5 explains how pattern search and support calculation are conducted, while 4.6 presents the DFBS strategy used by the algorithm. Lastly, the overall framework of the TK-RNSP algorithm is presented in Section 4.7.

4.1 Storage structure

Table 3: Representation of single-item bitmaps in the database of Table 2

item	s_1	s_2	s_3
a	1100	1100	1000
b	0010	1101	0100
c	0001	1011	0001

To enhance computational efficiency, the TK-RNSP algorithm employs the same bitmap-based storage structure as the SN-RNSP algorithm [26]. In the context of the TK-RNSP algorithm, the bitmap storage structure is a binary array used for storing all occurrences of an item or pattern in the database. It indicates whether an item or pattern appears at a specific element in a specific sequence.

The content of the bitmap structure is as follows: within the bitmap, only the positions where the last element of a pattern appears are marked as "1", while all other positions remain "0". Each item is associated with an independent bitmap, the size of which is determined by the maximum sequence length (*MAX-SIZE*) and the total number of sequences in the database ($|DB|$), calculated as *MAX-SIZE* $\times|DB|$. In practical applications, if the bitmap extends beyond the original length of a sequence, the corresponding bitmap positions are filled with "0" to ensure consistency and completeness.

To effectively demonstrate the use of this bitmap storage structure, an example is given next. Example 13 explains how information from a database is transformed into the bitmap representation.

Example 13 *As shown in Table 3, which is a bitmap representation of Table 2, it is clear that MAX-SIZE=4 and* $|DB| = 3$*, thus each item's bitmap comprises 12 bits. For the construction of the bitmap for* $\langle c\rangle$*, the leftmost support set of* $\langle c\rangle$ *consists of* $\{(1, \langle 4\rangle), (2, \langle 1\rangle), (2, \langle 3\rangle), (2, \langle 4\rangle), (3, \langle 3\rangle)\}$*. Therefore, in the bitmap section for* s_1*, the fourth position is set to "1"; in* s_2*'s bitmap section, the first, third, and fourth positions are set to "1"; in* s_3*'s bitmap section, the third position is set to "1". Since* s_3 *only contains three elements, the fourth position in this area is padded with "0" to indicate that there is no element at that position.*

If a pattern p is found in a sequence s, we only record the position of the last element of the pattern to identify the leftmost support. Example 14 further elucidates the specific application of storing patterns.

Example 14 *Using the DB described in Table 2, which contains three sequences with the longest sequence being four elements in length, a 12-bit bitmap is allocated for the pattern* $p = \langle aab\rangle$*. The leftmost support set for this pattern includes* $\{(1, \langle 1, 2, 3\rangle), (2, \langle 1, 2, 4\rangle)\}$ *corresponding to the leftmost occurrences in sequences* s_1 *and* s_2*, respectively. In terms of the bitmap settings: (1) In the bitmap interval corresponding to* s_1 *(the first four bits), the pattern completely appears at the third position, therefore the third bit is set to "1". (2) In the bitmap interval for* s_2 *(the fifth to eighth bits), the pattern completely appears at the fourth position, thus the eighth bit is set to "1".*

Consequently, the bitmap representation for the pattern $p = \langle aab \rangle$ is 0010 0001 0000.

4.2 Pattern generation strategy

This section introduces the pattern generation strategy of TK-RNSP.

Compared to traditional methods, TK-RNSP does not require mining positive sequential patterns (PSPs) first and then converting them into negative sequential candidates (NSCs). In this study, we adopted the same pattern generation method as the NegPSpan algorithm. NegPSpan utilizes a prefix-based extension approach, combined with both positive and negative sequence extension techniques, to efficiently mine NSPs [38]. During the itemset extension process, elements within the itemset must be arranged in lexicographical order, meaning that a new element can only be added to the itemset if it is greater than the last element in the current itemset.

The specific operations are as follows:

Positive Extension (P-Extension):

- **Positive Sequence Extension (P-Sstep):** Let there be a patterm $P = \langle p_1, p_2, \ldots, p_n \rangle$, where each p_i is an itemset. A positive sequence extension adds a new itemset p_{n+1} to the end of the pattern, resulting in $P' = \langle p_1, p_2, \ldots, p_n, p_{n+1} \rangle$.
- **Positive Itemset Extension (P-Istep):** If the penultimate element of the candidate pattern is positive, an item x is added to the last itemset p_n, forming the pattern $P' = \langle p_1, p_2, \ldots, p_n \cup x \rangle$.

Negative Extension (N-Extension):

- **Negative Sequence Extension (N-Sstep):** For a pattern $P = \langle p_1, p_2, \ldots, p_n \rangle$, a negative sequence extension inserts a new negative itemset p'_n between p_{n-1} and p_n, resulting in $P' = \langle p_1, p_2, \ldots, p_{n-1}, p'_n, p_n \rangle$.
- **Negative Itemset Extension (N-Istep):** If the penultimate element of the pattern is negative, an item y is added to the second-to-last negative itemset p_{n-1}, forming the sequence $P' = \langle p_1, p_2, \ldots, p_{n-1} \cup y, p_n \rangle$.

As the length of patterns increases in RNSP mining, the computational costs also rise. Traditional RNSP mining mechanisms fail to maintain anti-monotonicity, which is fundamental to the pruning strategy. To address this issue, we adopt the concept of NSP partial order from the NegPSpan algorithm [38]. This definition not only provides the theoretical foundation for the anti-monotonicity of negative sequential patterns in this paper but also serves as the key premise for implementing the pruning strategy.

Definition 14 *(NSP partial order) Let* $p^\alpha = \langle a_1^\alpha \neg b_1^\alpha a_2^\alpha \neg b_2^\alpha \ldots a_{j^\alpha-1}^\alpha \neg b_{j^\alpha-1}^\alpha a_{j^\alpha}^\alpha \rangle$ *and* $p^\beta = \langle a_1^\beta \neg b_1^\beta a_2^\beta \neg b_2^\beta \ldots a_{j^\beta-1}^\beta \neg b_{j^\beta-1}^\beta a_{j^\beta}^\beta \rangle$ *be two NSPs such that* $a_k^\alpha \neq \emptyset$ *for all* $1 \leq k \leq j^\alpha$ *and* $a_k^\beta \neq \emptyset$ *for all* $1 \leq k \leq j^\beta$*. The partial order* $p^\alpha \lhd p^\beta$ *iff* $j^\alpha \leq j^\beta$ *and: (1)* $\forall k \in [1, j^\alpha - 1]$, $a_k^\alpha \subseteq a_k^\beta \wedge b_k^\alpha \subseteq b_k^\beta$*;(2)* $a_{j^\alpha}^\alpha \subseteq a_{j^\beta}^\beta$.

Example 15 *Consider the patterns* $P_1 = \langle a \neg c(ab) \rangle$, $P_2 = \langle a \neg (cd)(ab) \rangle$, *and* $P_3 = \langle a \neg (cd) a \rangle$*. Then* $P_1 \lhd P_2$ *and* $P_3 \lhd P_2$*, but* $P_1 \lhd P_3$ *is not true because a at the third position in* P_3 *does not contain* (ab) *in the third position in* P_1*.* P_1 *and* P_3 *are subsequences generated from* P_2*.*

Example 15 clearly demonstrates that the pattern generation strategy we employ ensures that for any pattern p, the subsequences p' generated from p satisfy the partial order condition $p' \triangleleft p$.

4.3 The antimonotonicity of RNSP

In this section, we will demonstrate that the RNSP mining process satisfies the anti-monotonicity property within the framework defined in this paper.

Theorem 2 (Anti-monotonicity) *If pattern p is infrequent, then all its supersequences are also infrequent. Conversely, if p is frequent, then all its subsequences are frequent as well.*

Guyet and Quiniou [38] introduced the definition of NSP partial order (Definition 14) and discussed the anti-monotonicity of NSP mining under weak occurrence and partial order. An NSP p is said to weakly-occur in a sequence s if there is at least one strict or soft embedding of p in s, and one of these embeddings satisfies the negative constraint (Definition 9). For more details, please refer to the original work by Guyet and Quiniou [38]. The generation rules for patterns indicate that the superpattern p^* of p is derived from p, and any superpattern p^* of p satisfies $p \triangleleft p^*$. Sun [26] also provided a method for RPSP mining under nonoverlapping self-adaptive gap conditions that satisfies anti-monotonicity, a foundational concept that significantly informs the demonstration of anti-monotonicity in RNSP mining within this paper. Based on these critical theoretical underpinnings, we will further elaborate on how RNSP mining within the framework of this study satisfies an anti-monotonicity property.

Proof 1 *Given an NSP $p^\alpha = \langle a_1^\alpha \neg b_1^\alpha a_2^\alpha \neg b_2^\alpha \dots a_{j^\alpha-1}^\alpha \neg b_{j^\alpha-1}^\alpha a_{j^\alpha}^\alpha \rangle$, its supersequence is $p^\beta = \langle a_1^\beta \neg b_1^\beta a_2^\beta \neg b_2^\beta \dots a_{j^\beta-1}^\beta \neg b_{j^\beta-1}^\beta a_{j^\beta}^\beta \rangle$. When $j^\alpha \leq j^\beta$, each element of p^α is contained in the corresponding element of p^β. Assume S^β is the support set of p^β in a sequence $s = \langle e_1 e_2 \dots e_n \rangle$, for every occurrence $(l_1, \dots, l_{j^\alpha}, \dots, l_{j^\beta}) \in S^\beta$, the following conditions are satisfied:*

1. $\forall k \in [1, j^\beta - 1], a_k^\beta \subseteq e_{l_k}$ *and* $\exists \langle e_{l_k+1} \dots e_{l_{k+1}-1} \rangle, b_k^\beta \not\subseteq \langle e_{l_k+1} \dots e_{l_{k+1}-1} \rangle$
2. $a_{j^\beta}^\beta \subseteq e_{l_k}$.

In terms of partial order, we can deduce that:

1. $\forall k \in [1, j^\alpha - 1], a_k^\alpha \subseteq a_k^\beta \subseteq e_{l_k}$ *and* $\exists \langle e_{l_k+1} \dots e_{l_{k+1}-1} \rangle, b_k^\alpha \subseteq b_k^\beta \not\subseteq \langle e_{l_k+1} \dots e_{l_{k+1}-1} \rangle$
2. $a_{j^\alpha}^\alpha \subseteq a_{j^\alpha}^\beta \subseteq e_{l_{j^\alpha}}$.

Hence, for each $(l_1, \dots, l_{j^\alpha}, \dots, l_{j^\beta}) \in S^\beta$ corresponding to an occurrence $(l_1, \dots, l_{j^\alpha})$, and where $(l_1, \dots, l_{j^\alpha})$ is an occurrence of p^α, all of them constitute an occurrence set S_{occ}^α of p^α and every occurrence satisfies the nonoverlapping condition. By inference, if p^β appears in a sequence, then p^α must also appear in that sequence because every element of p^α is a subset of p^β. Therefore, any sequence that supports p^β also supports p^α, indicating that the support of p^α is at least equal to or greater than that of p^β. And the size of the support set of p^β is $sup(p^\alpha) = |S^\alpha|$, because the support set has the maximal number of occurrences, and any two occurrences are nonoverlapping, $|S^\alpha| \geq |S_{occ}^\alpha|$. Let S^β be the support set of p^β, then it is known that $|S_{occ}^\alpha| = |S^\beta|$, i.e., the size of the occurrence set S_{occ}^α is equal to the size of the support set S^β. From

the relationship between the support set, the instance set and the occurrence set in a single sequence, it can be obtained that:

$$sup(p^{\alpha}) = |S^{\alpha}| \geq |S^{\alpha}_{occ}| = |S^{\beta}| = sup(p^{\beta})$$

In summary, $sup(p^{\alpha}) \geq sup(p^{\beta})$, the instances and occurrences of RNSPs satisfy the anti-monotonicity property. This still holds true in the database.

Example 16 *In Table 2, the support for pattern $p = \langle a\neg cc\rangle$ is $sup(p) = 3$, its subsequence $p' = \langle ac\rangle$ has a support of $sup(p') = 4$, and the supersequence $p^* = \langle a\neg ccb\rangle$ has a support of $sup(p^*) = 1$.*

4.4 Pruning strategy

Our TK-RNSP algorithm's design of the MAXG-MINUP strategy is inspired by the implementation of the pruning strategy in the TKHUS-SpanBFS [47] algorithm. The TKHUS-SpanBFS algorithm [47] uses a max heap to store unvisited nodes and expands them based on the highest utility (PEU), thereby reducing the generation of unpromising candidate patterns. At the same time, the algorithm employs a min heap to maintain the top-k patterns with the highest utility and dynamically adjusts the minimum utility threshold by continuously updating the top of the min heap, ensuring that only the most optimal patterns are retained. Building on this approach, we introduced the MAXG-MINUP strategy in TK-RNSP, which generates candidates using patterns with the highest support (MAXG) and maintains the top-k patterns with the highest support using a min heap. By continuously updating the top of the min heap, the minimum support is dynamically increased (MINUP), allowing for efficient pruning and pattern mining.

More importantly, under this strategy, the value of *Min-sup* is not fixed but dynamically adjusted in line with the continuous updates of the Top-K patterns. We call the support of the pattern with the smallest support in the final Top-K patterns as "*KMin-sup*". When the algorithm terminates, the *Min-sup* threshold will have been elevated to *KMin-sup*. Such a dynamic updating mechanism can quickly increase the *Min-sup* threshold, thereby effectively reducing the search space. Our experimental results (see Section 5.2) also demonstrate the superiority of this pruning strategy.

To implement the MAXG-MINUP strategy, the TK-RNSP algorithm employs two data structures: a max heap (SC-Maxheap) and a min heap (TOP-Minheap). The SC-Maxheap is used to store currently generated but untraversed candidates, ensuring that the highest support candidates are prioritized during pattern generation. The size of TOP-Minheap is set to K to maintain the current Top-K NSC and dynamically raise the *Min-sup* as the algorithm is running. For eligible newly generated pattern candidates, the saveSC method is invoked to add the sequence to SC-Maxheap and update the heap to keep the sequence with the maximum support at the top. The saveTOPK method is also called to add it to TOP-Minheap. The algorithm 1 displays the pseudocode of the saveTOPK method; if the TOP-Minheap is full, the top pattern is removed to make space for a new pattern, and the support of the new top pattern is set as *Min-sup*. If the heap is not full, the new pattern is directly added. After

inserting a new pattern, if the heap is full, then *Min-sup* is updated to the support of the top pattern in the current state.

4.5 Pattern search and support calculation

During the pattern generation process of the TK-RNSP algorithm, it simultaneously calculates the support for both RPSPs and RNSPs. The methods for searching and calculating the support for RPSPs are introduced in Section 4.5.1, and the methods for searching and calculating the support for RNSPs are detailed in Section 4.5.2.

4.5.1 Positive sequential pattern search and support calculation

We utilize the bitmap-based method from the SN-RNSP algorithm to calculate the support of RPSPs. This method offers high space efficiency, fast query speed, and efficient set operations, making it very effective in handling pattern search and support calculation, especially when dealing with RSPs. For detailed information on the method, please refer to [26]. To provide a more intuitive understanding of the pattern search process, we use a Nettree structure with multiple roots and nodes, allowing the same node to reappear at different levels to represent the pattern search process [12]. Each node has an ID corresponding to the position of elements in the sequence. The search tree illustrates the contiguous patterns explored through a depth-first search strategy, where each path from the root to the leaf node represents an occurrence of a pattern in the sequence. The notation n_k^h denotes a node with ID k at the h-th level of the search tree.

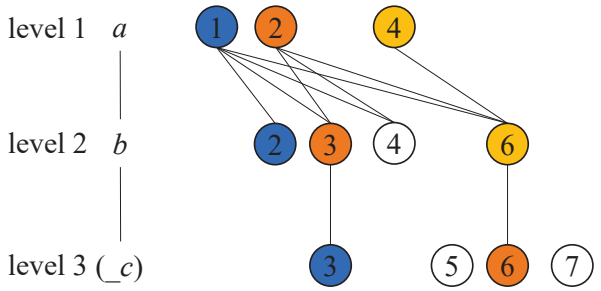


Fig. 1: Occurrence of $< a(bc) >$ in the sequence $s = \langle a(ab)(bc)(ab)c(bc)c \rangle$

Example 17 *Given a sequence* $s = \langle a(ab)(bc)(ab)c(bc)c \rangle$*, a PSP* $p = \langle a(bc) \rangle$*, the process of searching for* p *in* s *under nonoverlapping conditions is illustrated in Figure 1. First, the first level of the search tree is the node* a*, corresponding to the first element of pattern* p*. It can be observed that the element* a *is contained in* $e_1 = a$*,* $e_2 = (ab)$*, and* $e_4 = (ab)$ *of sequence* s*. Then, nodes* n_1^1*,* n_2^1*, and* n_4^1 *(representing the first, second, and fourth elements of the first level, respectively) are established, and the bitmap* 1101000000 *for* $\langle a \rangle$ *is created. Next, by performing a P-Sstep,* $\langle b \rangle$ *is added to* $\langle a \rangle$ *to form the pattern* $\langle ab \rangle$*. Therefore, the second level nodes for* b *are created at positions* n_2^2*,* n_3^2*,* n_4^2*, and* n_6^2*. The occurrences of pattern* $\langle ab \rangle$ *are* $\langle 1, 2 \rangle$*,* $\langle 2, 3 \rangle$*, and* $\langle 4, 6 \rangle$*. To obtain the leftmost support, the occurrences of appending element should be*

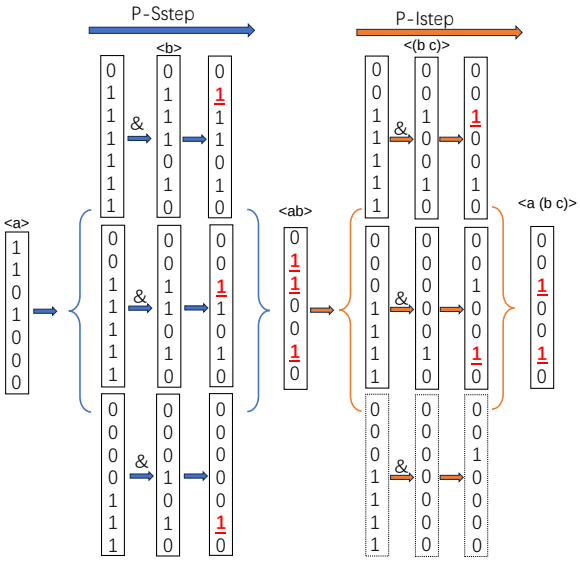


Fig. 2: The process of generating $s = \langle a(bc)\rangle$ in the s_2 of Table 1

rightmost, with each first unused element chosen to match an event. Since the bitmap for the pattern records only the position of the last element in each occurrence, the bitmap for $\langle ab\rangle$ *should be* 0110010000*. Finally, by using the P-Istep to add item* c *to itemset* (b)*, the pattern* $\langle a(bc)\rangle$ *is formed. Thus, the third level of the search tree represents the occurrence positions of node* c*. In sequence* s*, item* c *is contained in* $e_3 = (bc)$*,* $e_5 = c$*,* $e_6 = (bc)$*, and* $e_7 = c$*, resulting in the creation of nodes* n_3^3*,* n_5^3*,* n_3^6*, and* n_7^3*. Since* e_3 *and* e_6 *contain* (bc)*,* c *in the third level corresponds to the child nodes of the second level occurrences, with* n_3^3 *and* n_6^3 *being the child nodes of* n_3^2 *and* n_6^2*, respectively. As shown in Figure 2,* $\langle n_1^1, n_2^2, n_3^3\rangle$ *and* $\langle n_2^1, n_3^2, n_6^3\rangle$ *are two leftmost occurrences, namely* $\langle 1, 2, 3\rangle$*and* $\langle 2, 3, 6\rangle$*. The corresponding bitmap should have "1" in the third and sixth positions, i.e.,* 0010010000*.*

Example 17 uses a Nettree structure to explain the support search process in PSPs. The following sections detail the support calculation using bitmaps through the P-Istep and P-Sstep operations. For clarity, "&" denotes the "AND" operation, set(index) and clear(index) represent setting or clearing a bit at a specified position, respectively. The bitmap of $\langle a\rangle$ is represented by "$\langle a\rangle$.bm".

Example 18 *Suppose the DB contains only the sequence* $s = \langle a(ab)(bc)(ab)c(bc)c\rangle$ *from Figure 1. Figure 2 demonstrates the entire process of generating* $P = \langle a(bc)\rangle$ *in this sequence through P-Sstep and P-Istep. First,* $\langle ab\rangle$ *is generated via P-Sstep, then* $\langle a(bc)\rangle$ *is generated via P-Istep. In* s*, the bitmaps for* $\langle a\rangle$ *and* $\langle b\rangle$ *are* 1101000 *and* 0111010*, respectively.*

Step 1: Transform the position "1" in $\langle a\rangle$.bm and generate the transformed bitmap $\langle a\rangle trans.bm$: 0111111, indicating potential positions for element $\langle b\rangle$ in the sequence at positions 2, 3, 4, 5, 6, and 7. $\langle a\rangle trans.bm\&\langle b\rangle.bm$ yields 0111010, where the first bit set to "1" has an index (v), i.e., $v = 2$. Perform set$(v = 2)$ on $\langle ab\rangle$.bm to obtain 0100000.

Step 2: To avoid repetition, clear($v = 2$) on $\langle b \rangle$.bm, resulting in 0011010. Repeat similar steps for other positions in $\langle a \rangle$.bm, ultimately obtaining $\langle ab \rangle$.bm = 0110010. Then, perform P-Istep from $\langle ab \rangle$ to $\langle a(bc) \rangle$.

Step 3: First obtain the transformed bitmap $\langle ab \rangle trans.bm$:0011111 based on $\langle ab \rangle .bm.$, then $\langle b \rangle .bm \& \langle c \rangle .bm$ yields $\langle (bc) \rangle .bm$: 0010010. Perform $\langle ab \rangle trans.bm \& \langle (bc) \rangle .bm$ to get 0010010, and set($v = 3$) on $\langle a(bc) \rangle$.bm to obtain 0010000, corresponding to $\langle 1, 2, 3 \rangle$ in Figure 1.

Step 4: Similar to P-Sstep, perform clear(v) on $\langle (bc) \rangle$.bm. Continue steps 3 and 4 for other bits set to "1" in $\langle a(bc) \rangle$.bm, with subsequent iterations omitted. The final result is $\langle a(bc) \rangle$.bm = 0010010.

The clear bitmap transformations and generation of candidate sequences P_{temp} and their bitmaps $P_{temp}.bm$ can be interpreted as follows: (1) a "1" is set at the position index where the last element of the pattern occurrences; (2) a '1' represents one occurrence of the pattern, and the number of "1" indicates the support of the pattern.

4.5.2 Negative sequential pattern search and support calculation

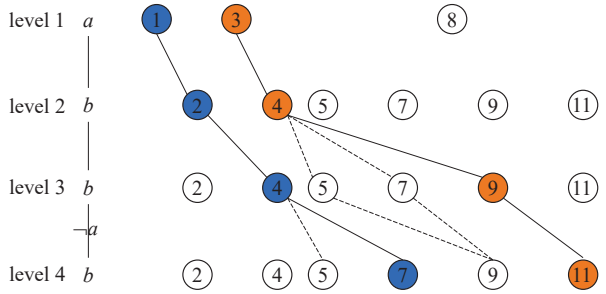


Fig. 3: Occurrence of $s = \langle abb\neg ab \rangle$ in the sequence $s = \langle ab(ad)bbdbabcb\neg ab \rangle$

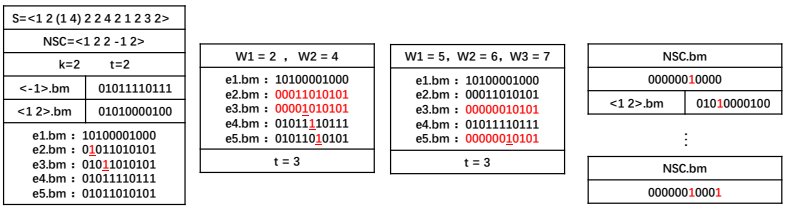


Fig. 4: Example of the bitmap calculation process for N-Extension

Next, we introduce the support calculation method for RNSPs. Unlike RPSPs, the search for RNSPs requires not only a depth-first search but also backtracking. To facilitate understanding, we use the Nettree structure in Example 19 to illustrate the support search process.

Example 19 *As shown in Figure 3, the search process for the NSP* $p = \langle abb\neg ab\rangle$ *in sequence* s *under nonoverlapping conditions using a Nettree structure is illustrated. This NSP* $p = \langle abb\neg ab\rangle$ *is formed by adding a negative itemset* $\neg\langle a\rangle$ *behind the last and second to last positive elements of the positive sequence pattern* $p = \langle abbb\rangle$*. Initially, using the method mentioned in Example 17, the Nettree for the pattern* $p = \langle abbb\rangle$ *is constructed, resulting in the leftmost set* $\langle 1, 2, 4, 5\rangle, \langle 3, 4, 5, 7\rangle$*. However,* $\langle 1, 2, 4, 5\rangle$ *is not a valid occurrence because there are no elements between the fourth and fifth positions in sequence* s*, thus further searching for the third occurrence* $\langle b\rangle$ *finds* $\langle 1, 2, 4, 7\rangle$ *to be a valid occurrence of* $p = \langle abbb\rangle$ *according to the definition in this paper, because position 7 is the rightmost occurrence that meets Definition 12. Since position 7 is used in* $\langle 1, 2, 4, 7\rangle$*, and due to the element* $\langle b\rangle$ *at that position, clearly* $\langle 3, 4, 5, 7\rangle$ *cannot be an occurrence again. Continuing the search for the third occurrence* $\langle b\rangle$*,* $\langle 3, 4, 5, 9\rangle$ *might be a possible next occurrence for* $p = \langle abbb\rangle$*, but the presence of element* $\langle a\rangle$ *between positions 5 and 9 clearly does not comply with Definition 12 indicating that node* n_3^5 *will not have children compliant with Definition 12. Hence, starting from position 4, searching for the second* $\langle b\rangle$ *results in possible occurrences* $\langle 3, 4, 7, 9\rangle$ *and* $\langle 3, 4, 9, 11\rangle$*, but since the element* $\langle a\rangle$ *appears between positions 7 and 9,* $\langle 3, 4, 7, 9\rangle$ *is not a valid occurrence, while* $\langle 3, 4, 9, 11\rangle$ *is. As illustrated in Figure 3, the root-leaf paths* $\langle n_1^1, n_2^2, n_4^3, n_7^4\rangle$ *and* $\langle n_3^1, n_4^2, n_9^3, n_{11}^4\rangle$*, correspond to two nonoverlapping occurrences* $\langle 1, 2, 4, 7\rangle$ *and* $\langle 3, 4, 9, 11\rangle$*, respectively. The corresponding bitmap for the NSP* $p = \langle abb\neg ab\rangle$ *is 00000010001.*

Based on the above examples, the core of the backtracking mechanism is to ensure that the appearance of negative elements does not violate specific sequence rules, i.e., the positive pattern corresponding to negative elements must not exist within a certain range of the sequence. When the algorithm detects the need to perform backtracking, it restores the bitmap state of the last positive element to its state at the start of the search.

Due to the uncertainty of negative elements' appearances, a negative element might appear in a specific segment of a sequence, but this does not necessarily mean that other negative elements are absent from that segment. In contrast, the appearance of positive elements is more certain. Thus, to avoid redundant support calculations, it is necessary to backtrack to the position of a previous definite positive element (the next position of this positive element is also a positive element). If no such position exists, return to the starting position of the sequence.

For NSCs generated by N-Sstep and N-Istep, the calRNSC method is used to search and record their occurrences, returning a bitmap representation of these occurrences in the sequence, with its pseudocode presented in Algorithm 2. The input consists of the NSC sequence generated by the N-Sstep and N-Istep, and the bitmap of the second-to-last element $e_{n-1}.bm$. The output is the bitmap representation of the NSC sequence $e_1e_2\ldots e_n$, indicating the occurrences of the NSC in the sequence. Below is a detailed introduction to the calRNSC algorithm:

Step 1: Find the starting element position (L1-9). Given an NSC $= \langle a_1\neg b_1a_2\neg b_2\ldots a_{j-1}\neg b_{j-1}a_j\rangle$, where the starting element meets one of the following

conditions: a represents the position of the corresponding positive element, b represents the position of the negative element, (1) a_1, if $\forall k \in [1, j-1], b_k \neq \emptyset$; (2) a_t, if $\forall k \in (t, j-1], b_k \neq \emptyset \wedge b_t = \emptyset$.

Step 2: From top to bottom in each sequence, search for the occurrence of each pattern (L10-47).

If the initial condition (1) from Step 1 is satisfied, the rightmost occurrence of b_1 is identified (L26), and it is determined whether the positive element corresponding to b_1 appears between a_1 and b_1 (L27-L40). If a positive element is found, the process backtracks to the position of a_1 and sets the position of a_1 to 0 (L38-L39). The reason for this is that since the current position of b_1 is at the rightmost end, the elements preceding this position must contain a positive element in the sequence.

If no positive element appears between a_1 and b_1, the process continues to check the interval between b_1 and a_2. The rightmost occurrence of a_2 is located (L28), and it is assessed whether the positive element corresponding to b_1 appears between the current positions of b_1 and a_2 (L29-L36). If a positive element is found, the process backtracks and sets the corresponding bit in the bitmap of a_1 to 0 (because the positive element corresponding to b_1 cannot appear between a_1 and a_2 (L22-28)). If no positive element corresponding to b_1 appears, the bitmap representation of a_1 is updated by clearing the positions that overlap with a_1 (L34-L35) (because the nonoverlapping condition requires that each item cannot be used at the same position repeatedly (L34)).

The search continues, and to satisfy the nonoverlapping condition, the bitmap of a_2 is updated (by setting the bits before the current position of a_2 to 0 (L35)), and the search for the occurrence of a_3 continues (L36).

If the initial condition (2) is satisfied, the rightmost occurrence of element a_{t+1} is found, and the search process is the same as in condition (1) (L42-45). Additionally, if backtracking is required, the bitmap of the last positive element needs to be restored (L15-20).

Step 3: When t equals n, it signifies that the last element position in the current sequence has been found, and this position is recorded in the NSC bitmap, which is then returned.

Step 4: For each bit set to "1" in $\langle e_1 \dots e_k \rangle$.bm, repeat Step 2 and Step 3, ultimately returning the NSC bitmap.

To better understand the workings of Algorithm 1, we will demonstrate its operational steps and effects in a practical application through a specific Example 20 . This will help us clearly see the process and results of the algorithm when processing data.

Example 20 *Assume there is a database containing the sequence* $s = \langle 12(14)22421232 \rangle$. *The corresponding bitmap is generated using the calRNSC method. Figure 3 shows how to generate* $NSC = \langle 122\neg 12 \rangle$. *This process corresponds to the search for* $NSC = \langle abb\neg ab \rangle$ *in the sequence* $s = \langle ab(ad)bbdbabcb \rangle$ *as depicted in Figure 4.*

Step 1: *Given the* $NSC < 1, 2, 2, -1, 2 >$, *we have* $n = 5$ *and* $k = n - 1 = 4$. *Since* e_4 *is a negative element, we update* $k = k - 2 = 4 - 2 = 2$, *and assign* k *to* t, *thus* $t = k = 2$. *It is observed that when performing pattern search on* $NSC < 1, 2, 2, -1, 2 >$, *the starting element position corresponds to the second case, where there exists a positive element followed by another positive element, and the information is obtained*

as shown in step (a). ***Step 2:*** *At this point, the loop condition $t \geq k$ and $k \leq n-2$ is satisfied. Given that $\langle ab \rangle.bm =$ 01010000100, it is known that positions 2, 4, and 9 contain a 1. First, a round of pattern search is conducted for the occurrence at position 2. Since $t = k = 2$, the element e_2 is located at position 2 in the NSC $\langle 122\neg 12 \rangle$, which is a positive element, and no backtracking is required. First, $e_2.bm =$ 01011010101 is obtained, and $e_t.bm$ is passed to $posEleBMSet$. If backtracking occurs in subsequent operations, $posEleBMSet$ can be directly invoked to restore the original state of $e_t.bm$. The index of the first bit set to 1 in $e_2.bm$ is recorded as $w_1 = 2$. The matching continues with e_{t+1}, i.e., e_3, where the element at position 3 in the NSC $\langle 122\neg 12 \rangle$ is a positive element. From $e_3.bm =$ 01011010101, the index of the first bit set to 1 after position w_1 is found and recorded as $w_2 = 4$. After updating the bitmaps using $e_t.bm.clear(w_1)$ and $e_{t+1}.bm.clear(0, w_2)$, the updated bitmaps are $e_2.bm =$ 00011010101 and $e_3.bm =$ 00001010101, and t is updated to $t = t + 1 = 3$. After updating the bitmap states, the information shown in (b) is obtained. At this point, $t = 3 = k = 3$, and since $t \neq n$, the loop condition $t \geq k$ and $k \leq n-2$ is still satisfied, so the current round of pattern search continues. Since $e_t = e_3$ and e_3 is a positive element, with $e_3.bm =$ 00001010101, $w_1 = 5$ is obtained. $e_{t+1} = e_4$, which is a negative element. The index of the first bit set to 1 after index w_1 in $e_4.bm$ is recorded as $w_2 = 6$. The positive counterpart of e_4 is not within the corresponding interval $\langle E_{w_1+1}...E_{w_2} \rangle$. $e_5.bm =$ 01011010101 is used to determine the position of e_5 in the sequence, recorded as w_3. The index of the first bit set to 1 after position w_2 in $e_5.bm$ is recorded as $w_3 = 7$. The positive counterpart of e_4 is also not within the corresponding interval $\langle E_{w_2}...E_{w_3-1} \rangle$. At this point, $t + 2 = 5 = n$, indicating that the last element of the NSC has been searched, and one occurrence of this NSC has been found in the current sequence.*

Step 3 : *Proceed to set the NSC.bm using Set(v), where $v = w_3$ represents the position of the last element of the current NSC. The updated NSC.bm is* 00000010000*, corresponding to one occurrence at positions 1, 2, 4, 7, and the bitmap is updated as shown in (c).*

Step 4: *For the remaining positions with 1 in $< ab > .bm =$* 01010000100*, repeat the operations in steps 2 and 3.*

Finally, the result 00000010001 *is obtained, corresponding to the two occurrences of NSC $\langle abb\neg ab \rangle$ in sequence $s = \langle ab(ad)bbdbabcb \rangle$ at positions {1, 2, 4, 7} and {3, 4, 9, 11}.*

Algorithm 1 *saveTopk*$(p, p.bm, S_n, I_n, NI_n)$

```
1: Input: p, p.bm, S_n, I_n, NI_n
2: Output: Updated top-K patterns and updated Min-sup
3: if |TOP-Minheap| = K then
4:     TOP-Minheap.remove(TOP-Minheap.root)
5:     TOP-Minheap.insert(p, p.bm, S_n, I_n, NI_n)
6:     Min-sup = sup(TOP-Minheap.root)
7: else
8:     TOP-Minheap.insert(p, p.bm, S_n, I_n, NI_n)
9:     if |TOP-Minheap| = K then
10:        Min-sup = sup(TOP-Minheap.root)
11:    end if
12: end if
```

Algorithm 2 $CalRNSC(e_1e_2 \dots e_{n-1}e_n, e_{n-1}.bm)$

Input: NSC $e_1e_2 \dots e_{n-1}e_n$ and the bitmap of the last negative element $e_{n-1}.bm$
Output: The bitmap of $e_1e_2 \dots e_{n-1}e_n$
$k \leftarrow n-1$
while $k > 1$ and e_k is negative **do**
    $k \leftarrow k-2$
end while
if $k < 1$ **then**
    $k \leftarrow 1$
end if
for each sequence $E_1E_2 \dots$ in DB **do**
    for each "1" in $e_1 \dots e_k.bm$ **do**
        $t \leftarrow k$
        $posEleBMSet \leftarrow \emptyset$
        while $t \geq k$ and $k \leq n-2$ **do**
            if e_t is positive **then**
                if backtracing **then**
                    $e_t.bm \leftarrow posEleBMSet$
                else
                    $posEleBMSet \leftarrow posEleBMSet \cup e_t.bm$
                end if
                $w_1 \leftarrow$ the index of first "1" in $e_t.bm$
            else
                $t \leftarrow t-1$ and backtracing
            end if
            if e_{t+1} is negative **then**
                $w_2 \leftarrow$ the index of first "1" after index w_1 in $e_{t+1}.bm$
                if $p(e_{t+1}) \notin \langle E_{w_1+1} \dots E_{w_2} \rangle$ **then**
                    $w_3 \leftarrow$ the index of first "1" after index w_2 in $e_{t+2}.bm$
                    if $p(e_{t+1}) \notin \langle E_{w_2} \dots E_{w_3-1} \rangle$ **then**
                        $e_t.bm.clear(w_1)$
                        $e_{t+2}.bm.clear(0, w_3)$
                        $t \leftarrow t+2$
                    else
                        $e_t.bm.clear(0, w_2)$
                        $t \leftarrow t-1$ and backtracing
                    end if
                else
                    $e_t.bm.clear(0, w_2-1)$
                    $t \leftarrow t-1$ and backtracing
                end if
            else
                $w_2 \leftarrow$ the index of first "1" after index w_1 in $e_{t+1}.bm$
                $e_t.bm.clear(w_1)$
                $e_{t+1}.bm.clear(0, w_2)$
                $t \leftarrow t+1$
            end if
        end while
        if $t == n$ **then**
            set one value in $e_1 \dots e_n.bm$ where the index of first "1" in $e_n.bm$
        end if
    end for
end for
return $e_1 \dots e_n.bm$

4.6 Algorithms for DFBS

This subsection introduces a bitmap-based Depth-First Backtracking Search (DFBS) strategy for pattern generation, pattern search, and support calculation. This approach handles both positive and negative sequence generation while leveraging a bitmap conversion mechanism to efficiently calculate RPSPs and RNSPs support. It is important to note that the basic methods for pattern generation, support pruning, and pattern search have been previously discussed. The following section will focus

Algorithm 3 $DFBS(p, p.bm, S_n, I_n, NI_n)$

```
1: Input: p, p.bm, S_n,I_n, NI_n
2: Output: Updated S_n, I_n, NI_n,TOP-Minheap,SC-Maxheap,Min-sup,and new candidate patterns
3: Positive sequence extension
4: Calculate support Iplus
5: if Iplus.cardinality() > Min-sup then
6:     Add item to S_temp
7:     saveSC and saveTopk
8: end if
9: Positive itemset extension
10: Calculate support Iplus
11: if Iplus.cardinality() > Min-sup then
12:     Add item to I_temp
13:     saveSC and saveTopk
14: end if
15: if p.size() > 1 then
16:     if not p.getLastElement().isNeg() then
17:         Negative sequence extension
18:         Calculate support Iplus
19:         if Iplus.cardinality() > Min-sup then
20:             Add item to NI_temp
21:             saveSC and saveTopk
22:         end if
23:     else
24:         Negative itemset extension
25:         Calculate support Iplus
26:         if Iplus.cardinality() > Min-sup then
27:             saveSC and saveTopk
28:         end if
29:     end if
30: end if
```

on the detailed implementation of the DFBS strategy, and Algorithm 3 provides the pseudocode.

By inputting all existing items and their bitmaps from the database, as well as the current extension pattern P and its bitmap $P.bm$, the output is the generated candidate patterns. If frequent, the SC-Maxheap and TOP-Minheap are updated, and the minimum support *Min-sup* is adjusted accordingly.

As shown in the pseudocode, the TK-RNSP algorithm first retrieves all existing items and their support from the database, S_n and I_n are used to store items that meet the conditions to be used for positive extensions. Due to the various constraints in selecting negative elements, such as the requirement for lexicographical order and the types of elements in the database, the NI_n set is used to record items that can appear in the same element and are used for negative extensions to generate NSPs. The detailed steps are as follows:

1. For each item in S_n, perform a positive sequence extension (P-Sstep) to generate the candidate P_{temp}. Calculate the support of P_{temp}, and if frequent, add the current item to the S_{temp} list, to be used as the new S_n in the next iteration. The pattern is then saved to SC-Maxheap and TOP-Minheap using the saveSC and saveTOPK methods, respectively, and updated accordingly.
2. For each item in I_n, depending on the positivity or negativity of the second-to-last position's itemset in the current P_{temp}, perform either a positive itemset extension (P-Istep) or a negative itemset extension (N-Istep). If the second-to-last element of P_{temp} is negative, perform an N-Istep, otherwise, perform a P-Istep. Check if

the newly generated P_{temp} is frequent; if so, add the current item to the I_{temp} list, to be used as the new I_n in subsequent iterations, and save the pattern. Update SC-Maxheap, TOP-Minheap, and *Min-sup* accordingly.

3. When the sequence size is two or greater, determine the type of extension based on the positivity or negativity of the second-to-last element. If the conditions are met, add the current item to the NI_{temp} list, to be used as the new NI_n in subsequent iterations, and save the pattern. Update SC-Maxheap, TOP-Minheap, and *Min-sup* accordingly.

4.7 TK-RNSP - Top-K repetitive negative sequential pattern mining algorithm

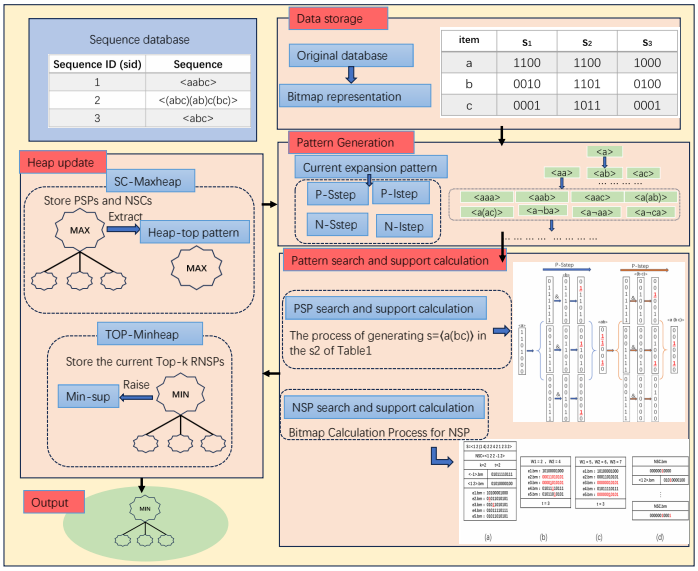


Fig. 5: The TK-RNSP algorithm's flowchart

The algorithm begins with processing the data from the database. It first extracts all the individual items present in the database and constructs a bitmap for each. Next, it generates candidate patterns through Positive Extension (including P-Sstep) and Negative Extension (including N-Sstep and N-Istep). For each generated candidate pattern, the algorithm performs pattern searching and calculates its support. If the support meets the minimum threshold (Min-sup), the pattern is stored in two heaps: the SC-Maxheap, a max-heap for storing PSPs and NSCs, and the TOP-Minheap, a min-heap for storing the current top-k RNSPs.

The heaps are then updated: the top pattern from the TOP-Minheap is used to raise the Min-sup threshold, and the top pattern from the SC-Maxheap is used to continue generating new patterns. This process repeats until the SC-Maxheap is empty or the top pattern's support in the SC-Maxheap falls below the Min-sup threshold.

At this point, the algorithm terminates, and the patterns in the TOP-Minheap are output as the final Top-k RNSPs.

Algorithm 4 $TK-RNSP(DB, K)$

```
1: construct TOP-Minheap whose size is K
2: construct SC-Maxheap
3: set Min-sup = 0
4: Add all items to S_n
5: for each i in S_1 do
6:     SC.insert(<< i >, i.bm, S_n, items from S_n that are greater than i, ∅ >)
7: end for
8: while |SC-Maxheap| > 0 do
9:     if |TOP-Minheap| = K and sup(SC-Maxheap.root) ≤ Min-sup then
10:        break
11:    end if
12:    Pattern Generation and Pruning (TOP-Minheap, SC-Maxheap)
13: end while
14: return Top-K RNSP
```

Figure 5 and Algorithm 4 present the overall framework of the TK-RNSP algorithm, with the specific steps described as follows:

Step 1: First, create a min-heap of size K, TOP-Minheap, to hold the current Top-K RNSPs, and a max-heap, SC-Maxheap, to store currently generated but unexplored candidate sequences. Next, traverse the database to obtain all single items and place them as individual elements into SC-Maxheap.

Step 2: From SC-Maxheap, select the pattern with the highest support as the Current Extension Sequence (CES). Generate patterns and calculate support for this sequence, ensuring that each extension is based on the most promising candidate pattern currently available.

Step 3: For each generated pattern, if it meets the current *Min-sup*, it is added to SCMax-heap and TOP-Minheap and updated to ensure that TOP-Minheap always holds the Top-K patterns and dynamically updates *Min-sup*.

Step 4: Repeat Steps 2 and 3 until the capacity of TOP-Minheap is satisfied, and either the support of the top element in SCMax-heap is less than *Min-sup*, or SCMax-heap is empty.

Step 5: Output the Top-K RNSPs stored in TOP-Minheap.

Next, we will demonstrate the steps of the TK-RNSP algorithm for mining Top-K RNSPs through a specific example.

Example 21 *Figure 6 shows a specific example of mining the Top-K RNSPs using the TK-RNSP algorithm. In this case, the database contains only a single sequence* $s = \langle 2(14)22421232 \rangle$*, and mining for the K=3 RNSPs is conducted under an initial minimum support threshold, Min-sup=0. A detailed analysis of the mining process follows:*

First, all single items and their occurrences are extracted from the database, and these 1-Size sequences are stored in SC-Maxheap.

Subsequently, the candidate with the highest support is selected for the first round of pattern generation, as shown in (a), where no new NSC is generated.

During the pattern generation process, as shown in (b), the algorithm continues to extract the current highest support candidate $\langle 22 \rangle$ *for candidate generation, at which*

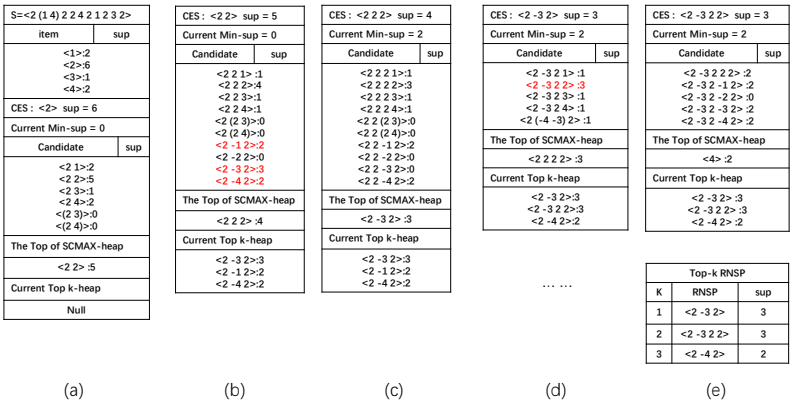


Fig. 6: Example demonstration of TK-RNSP algorithm

point an NSC is produced, and TOP-Minheap reaches full capacity for the first time, with the current Min-sup updated to 2.

Continuing pattern generation, as shown in (c), for the candidates generated from CES = ⟨222⟩, none are able to enter the current TOP-Minheap.

Subsequently, as shown in (d), the algorithm continues to extract the sequence with the highest support from SC-Maxheap for each round of pattern expansion. Pattern generation will cease when the candidate with the highest support in SC-Maxheap is less than or equal to the current Min-sup.

Finally, the algorithm returns the Top-K NSCs from the current TOP-Minheap as the generated Top-K RNSPs, as shown in (e). Moreover, from the candidate generation of each round, it is evident that the support for candidates generated by extending CES is less than the support of CES itself, confirming the anti-monotonicity characteristic of RNSP mining.

Based on the above description, the algorithm's termination and correctness are analyzed as follows. The termination of the algorithm is guaranteed by the heap update mechanism involving SC-Maxheap and TOP-Minheap. In each iteration, the algorithm selects the pattern with the highest support from SC-Maxheap for expansion, and adjusts *Min-sup* dynamically to prune patterns that do not meet the support threshold. The algorithm terminates when SC-Maxheap is empty or the support of the top pattern in SC-Maxheap falls below *Min-sup*. Thus, the algorithm ensures completion within a finite number of steps and outputs the current Top-K RNSPs that meet the criteria.

Additionally, the algorithm ensures the generation of all possible positive and negative sequential patterns through sequence and itemset extensions. The support for each pattern is efficiently calculated using bitmaps. The pruning strategy, based on the anti-monotonicity of *RNSP*, dynamically adjusts *Min-sup* to effectively prune patterns that do not meet the support requirement, ensuring that the remaining patterns satisfy the support threshold. As a result, the algorithm is able to correctly output the Top-K *RNSPs* while ensuring termination.

5 Experiments

To assess the efficiency of TK-RNSP, the following exploration questions (EQ) were considered:

- EQ1: Does the MAXG-MINUP strategy effectively reduce the candidate space and improve algorithm efficiency?
- EQ2: Can the DFBS strategy effectively identify the repeated occurrences of patterns?
- EQ3: Does the performance of the TK-RNSP algorithm surpass that of other advanced algorithms in completing the task of mining top-K RNSPs?
- EQ4: How does the value of parameter K affect the time performance of TK-RNSP?
- EQ5: How scalable is the TK-RNSP algorithm when applied to large-scale databases?

To answer EQ1 and verify the effectiveness of the MAXG-MINUP strategy, we test two variations of TK-RNSP, named Extension-MINEXTEND and Extension-FIXEDSUP in Section 5.2. For EQ2, we introduce a variation named TK-NSP to evaluate the search effectiveness of the DFBS strategy (see Section 5.2). To answer EQ3, we select the state-of-the-art E-RNSP method, namely SN-RNSP, as the competing algorithm to validate the efficiency of TK-RNSP in mining Top-K RNSPs (see Section 5.2). For EQ4, we examine the impact of different K value settings on TK-RNSP from multiple perspectives, and results are presented in Section 5.2. Lastly, for EQ5, we test the scalability of TK-RNSP using large-scale databases in Section 5.3.

5.1 Datebase and baseline

The experiments were carried out on eight distinct datasets. The selection of these datasets is based on their widespread use in related research. For instance, the SN-RNSP algorithm [26] was evaluated using DS1, DS3, DS5, and DS8. These datasets encompass both generated and real-world data, featuring diverse characteristics and structures. By using such varied datasets, we can more comprehensively evaluate the algorithm's performance across different scenarios.

- **Dataset 1 (DS1):** C5-T8-S8-I8-DB10K-N100-R0, features an average of 5 elements per sequence (C), 8 items in each element (T), 8 as the average size of maximal potentially large sequences (S), 8 items in the maximal potential PSP (I), 10K data sequences (DB), 100 divergent items (N), and a repetition level of 0 (R).
- **Dataset 2 (DS2):** C5-T8-S8-I8-DB10K-N100-R1, similar to DS1 but with a repetition level of 1 (R).
- **Dataset 3 (DS3):** C6-T8-S8-I8-DB10K-N100-R0, features an average of 6 elements per sequence (C), similar to DS1 in other parameters but differing in the average number of elements per sequence.
- **Dataset 4 (DS4):** C6-T8-S8-I8-DB10K-N100-R1, similar to DS3 but with a repetition level of 1 (R).
- **Dataset 5 (DS5):** OnlineRetail_II_best, comprises transactions from an online retail company in the UK, selling gifts from January 12, 2009, to September 12,

2011. It includes 4,383 sequences and 10,157 unique items, with an average of 9.0 items per itemset.

- **Dataset 6 (DS6):** E-SHOP, emanates from an online store catering to pregnant women, featuring clickstream data. Its sequences, composed of itemsets, average 9.0 items each.
- **Dataset 7 (DS7):** MicroblogPCU, sourced from Sina Weibo for identifying spammers within microblogs, consists of 221,579 instances. Constraints on sequence length are required for DS6 to facilitate the mining task.
- **Dataset 8 (DS8):** BIKE, encompasses sequences of shared bicycle parking locations within a city. Each item signifies a bike-sharing station, and each sequence denotes a bicycle's location over time. The DS5 dataset includes 21,078 sequences and 67 unique items, with an average sequence length of 7.27.

Datasets **DS1-4** were generated using the IBM data generator, while DS5-8 are real datasets downloaded from SPMF (www.philippe-fournier-viger.com/spmf/).

To evaluate the performance of RNP-Miner, we selected five competing algorithms:

1. **Extension-MINEXTEND (E-M) and Extension-FIXEDSUP (E-F):** To verify the effectiveness of the MAXG-MINUP strategy, we designed E-M and E-F. E-M aims to validate the effectiveness of the MAXG strategy in reducing the search space by generating candidates using minimal support patterns, which also serves to assess the improvement in algorithm efficiency. E-F focuses on validating the effectiveness of the MINUP strategy in narrowing the space by modifying the pruning method to use a fixed support threshold instead of dynamically increasing it, again to evaluate efficiency enhancements.
2. **TK-NSP:** To assess whether the DFBS strategy can effectively capture repeated occurrences of patterns, we propose TK-NSP. This algorithm mines NSPs without considering pattern repetitions, utilizing the pattern generation and pruning methods of TK-RNSP, but calculating support solely based on the number of sequences that contain the pattern (i.e., the support calculation does not count the frequency of the pattern's occurrences in the database but rather the number of sequences that include it).
3. **SN-RNSP and E-RNSP:** For comparative mining capabilities, we employed two classic state-of-the-art RNSP mining methods: SN-RNSP and E-RNSP. Prior to modification, these algorithms could not perform Top-K RNSP mining tasks, as they only output all RNSPs under a fixed threshold. To accomplish the Top-K RNSP task, we modified both algorithms, resulting in the E-RNSP* and SN-RNSP* versions, which can output Top-K RNSPs under fixed thresholds, similar to the original algorithms.

The experiments were conducted on two different machines, all running on Windows 11 (64-bit) and implemented in Java. The experiment in Section 5.2.1 was carried out on a machine with an Intel(R) Core(TM) i7-6700 CPU at 3.40 GHz, equipped with 64.0 GB of RAM. All other experiments were performed on a system featuring a 12th Gen Intel(R) Core(TM) i9-12900H 2.50 GHz processor with 16 GB of memory.

5.2 Mining Performance and Analysis

This section delves into four core questions (EQ1, EQ2, EQ3, EQ4) and addresses EQ1, EQ2, and EQ3 sequentially across three subsections, while also analyzing the related content of EQ4 within each subsection. Specifically, Subsection 5.2.1 will examine the effectiveness of the MAXG-MINUP strategy in reducing the candidate space and improving algorithm efficiency, validated through the algorithms E-M and E-F. Subsection 5.2.2 will assess the effectiveness of the DFBS strategy employed by TK-RNSP in pattern repetitiveness search using TK-NSP. In Subsection 5.2.3, we will compare TK-RNSP with E-RNSP and SN-RNSP to verify its performance in mining Top-K RNSPs. Furthermore, each subsection will analyze the impact of different K value settings on the performance of TK-RNSP. The experimental results demonstrate that TK-RNSP performs exceptionally well across multiple datasets, efficiently accomplishing Top-K RNSP mining.

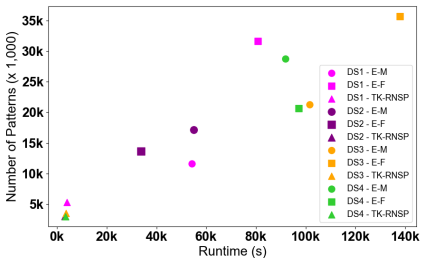


Fig. 7: Performance on DS1-DS4

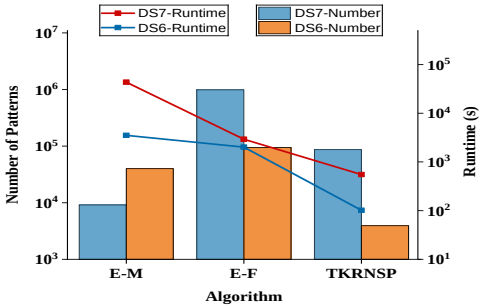


Fig. 8: Performance on DS6 and DS7

5.2.1 Analysis of the Effectiveness of the MAXG-MINUP Strategy

TK-RNSP outperforms E-M and E-F, indicating that the MAXG-MINUP strategy effectively reduces the number of candidate generated and enhances algorithm efficiency.

Figures 7 and 8 illustrate the experiments conducted with TK-RNSP, E-M, and E-F on datasets DS1-4, DS6, and DS7 when $K = 500$. It is found that Given that E-M and E-F generated a higher number of patterns and performed multiple support calculations, resulting in increased execution time, their initial support threshold was set to *Min-sup* = 1000 to balance experimental conditions and reduce runtime. In E-M, *Min-sup* dynamically increases as the algorithm progresses, whereas in E-F, it remains constant. The initial threshold for TK-RNSP is set at *Min-sup* = 0, which is its default setting, and it dynamically increases during execution. While this configuration may seem unfair, its purpose is to emphasize the performance comparison of TK-RNSP and accurately demonstrate its efficiency.

With regards to EQ1, in Figure 7 the same color represents experiments conducted on the same dataset, while specific shapes denote experiments using particular algorithms. For instance, the green triangle represents TK-RNSP experiments on the DS4 dataset. The results show that even starting from a lower initial support

of *Min-sup* = 0, TK-RNSP still achieves the shortest runtime. For example, on the DS2 dataset, the runtime of TK-RNSP was 3.271s, compared to 54.958s for E-M and 33.731s for E-F.

In Figure 8, the runtime for TK-RNSP on the DS6 dataset was only 101.649s, significantly lower than the 20,212.479s for E-F and 3,527.582s for E-M. This means that the runtime for TK-RNSP was approximately 0.5% of that for E-F and 2.9% of that for E-M. Furthermore, TK-RNSP generated 3,927 patterns, while E-M and E-F produced 40,034 and 94,211 patterns, respectively, approximately 10.2 times and 24 times the number generated by TK-RNSP.

The reason for this is that MAXG-MINUP rapidly narrows the search space based on the anti-monotonicity property. MAXG employs the candidate patterns with the highest support to generate candidates, which are more likely to satisfy the current support threshold conditions, while MINUP allows for dynamic increases in support, continuously narrowing the search space and enhancing algorithm efficiency.

For EQ4, Figure 9 displays the number of candidate patterns generated by the three algorithms (E-M, E-F, and TK-RNSP) at different K values on datasets DS1 and DS3. At $K = 10$, E-M generated 27,562 candidates, E-F generated 20,948, while TK-RNSP only generated 5,040. This is due to the effectiveness of the MAXG-MINUP pruning strategy, which significantly reduce the generation of invalid candidate patterns, thus compressing the search space.

As K increases, Figures 10, 11, and 12 show the number of mined patterns and runtime on DS2, DS4, and DS7, respectively. Overall, the advantages of TK-RNSP in terms of both pattern count and runtime become increasingly apparent as K increases. While the runtime and pattern count of all algorithms trend upward across all datasets, TK-RNSP consistently maintains lower runtimes. For instance, on the DS2 dataset at $K = 1000$, TK-RNSP's runtime was 4.828s, while E-M and E-F took 62.831s and 55.081s, respectively, making TK-RNSP approximately 13 times and 11.4 times faster than E-M and E-F. Simultaneously, TK-RNSP generated 4,841 patterns, while E-M and E-F produced 23,850 and 18,267 patterns, respectively. This indicates that TK-RNSP retains a significant advantage even at high K values, demonstrating its stability.

Figures 10 and 11 further confirm this trend. In Figure 12, when $K = 10$, TK-RNSP generated 86,094 patterns, whereas E-M and E-F produced 225 and 555,842 patterns, respectively. The runtime for E-M reached 365.87 times that of TK-RNSP at 83.249s, totaling 30,457.696s, while E-F's runtime was 17.83 times longer than that of TK-RNSP, totaling 1,484.471s.

These observations indicate that the pruning strategy employed by the TK-RNSP algorithm significantly reduces both the number of patterns and runtime when searching for patterns with varying K values, and the algorithm's performance remains stable despite fluctuations in K values.

5.2.2 Evaluation of the Effectiveness of the DFBS Strategy

To evaluate the effectiveness of the DFBS strategy in identifying pattern repetitiveness, we compared it with TK-NSP, which focuses solely on the number of sequences

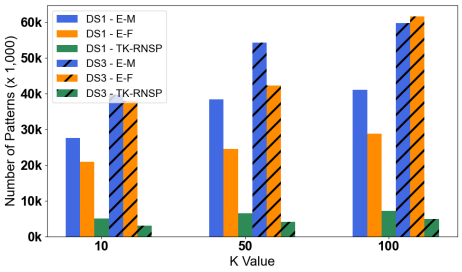


Fig. 9: Comparative Number of Candidate Patterns Generated on DS1 and DS3

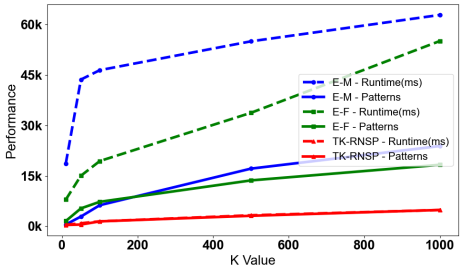


Fig. 10: Performance on DS2

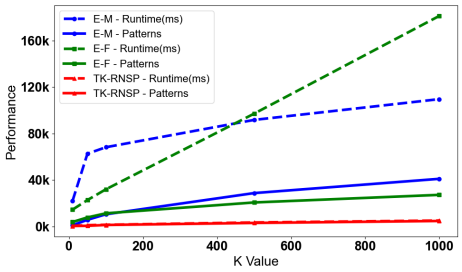


Fig. 11: Performance on DS4

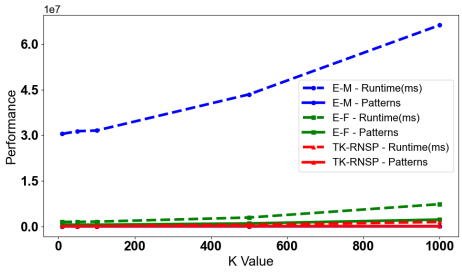


Fig. 12: Performance on DS7

containing the patterns. The results indicate that, across various datasets and different K values, TK-RNSP consistently achieves higher pattern support levels. This demonstrates that the DFBS strategy effectively recognizes the occurrence frequency of patterns within the database.

Table 4 presents a performance comparison of the TK-RNSP and TK-NSP algorithms in mining Top-K RNSPs across different K values, with a focus on analyzing the impact of the DFBS strategy on pattern repetitiveness (EQ2) and the effect of varying K values on algorithm performance (EQ4). The experimental results demonstrate that TK-RNSP significantly outperforms TK-NSP across multiple datasets, particularly in terms of the number of generated patterns and support levels.

First, regarding EQ2, the introduction of the BFDS strategy enables TK-RNSP to effectively identify repeated occurrences of patterns. For instance, in the DS1 dataset with $K = 10$, the support range for TK-RNSP is [1685, 1869], while that for TK-NSP is lower. This reflects TK-RNSP's capability to generate higher-support patterns when considering pattern repetitiveness. This is because TK-RNSP takes into account the repeated occurrences of patterns within sequences, where support refers to the number of times a pattern appears in the database. In contrast, TK-NSP does not consider pattern repetitiveness, and support indicates the number of sequences containing the pattern. Moreover, TK-NSP generates more patterns when completing the top-K mining task; for example, in DS5, when $K = 10$, TK-RNSP generates only 1,551 patterns, whereas TK-NSP generates 15,712 patterns, meaning TK-RNSP produces only about 10% of the patterns generated by TK-NSP. This is due to the slower

increase of the minimum support threshold and its relatively low value when pattern repetitiveness is not considered, resulting in less effective candidate space reduction.

Secondly, concerning EQ4, the results in Table 4 reflect the influence of different K value settings on the TK-RNSP algorithm. As K increases, TK-RNSP consistently outperforms TK-NSP in terms of both the number of generated patterns and the range of pattern support. For example, as K increases from 10 to 1000, the number of patterns generated by TK-RNSP in DS1 increases from 2,278 to 5,531, with a relatively smooth growth trend in both support range and pattern count, indicating that TK-RNSP maintains good performance stability at higher K values.

5.2.3 Performance Comparison of the TK-RNSP Algorithm

We modified E-RNSP and SN-RNSP to create the E-RNSP* and SN-RNSP* versions that can output Top-K results based on support size, similar to the original algorithms. The results indicate that TK-RNSP performs well across different K values.

Table 4 presents a comparative analysis of SN-RNSP*, E-RNSP*, and TK-RNSP across datasets DS1-3 and DS5, focusing on metrics such as execution time (T), number of patterns (N), and support range (R). The support thresholds for SN-RNSP* and E-RNSP* are consistent within the same datasets, set at 500, 200, 500, and 1000 for DS1-3 and DS5, respectively. In contrast, the initial support threshold for TK-RNSP is consistently set to 0. This difference highlights TK-RNSP's advantages in pattern support.

Regarding EQ3, Table 4 shows that the number of patterns generated (N) by TK-RNSP is significantly lower across all datasets compared to E-RNSP* and SN-RNSP*. For instance, in the DS1 dataset at $K = 50$, TK-RNSP generates 1,352 patterns, while SN-RNSP and E-RNSP generate 7,550 and 3,953 patterns, respectively. In terms of runtime (T), TK-RNSP also consistently outperforms the other algorithms. In DS1 at $K = 50$, TK-RNSP's runtime is 2.037s, compared to 5.827s for SN-RNSP* and 53.037s for E-RNSP*. However, in DS5, TK-RNSP's runtime is longer than that of E-RNSP*, as E-RNSP* generates very few patterns under the Min-sup=1000 condition. Additionally, TK-RNSP generally finds patterns with higher support at the same K value than SN-RNSP* and E-RNSP*, demonstrating stronger pattern repetitiveness capture. These results validate TK-RNSP's efficiency in handling Top-K RNSP tasks.

Table 4 also illustrates the impact of K values on the performance of TK-RNSP (EQ4). As the K value increases, both the number of patterns (N) and the runtime (T) for TK-RNSP also rise, although the increase is relatively gradual. This indicates that TK-RNSP maintains good performance and stability even when faced with larger K values. For instance, in DS3, as the K value increases from 50 to 1000, the number of patterns generated by TK-RNSP rises from 841 to 5,116. However, this figure remains significantly lower compared to other algorithms that must first generate all patterns before performing top-k selection under the same threshold conditions. Additionally, in DS3, the runtime for TK-RNSP increases from 1.411s to 6.062s , yet it still remains substantially lower than the execution times of other algorithms.

Finally, Figure 13 presents a performance comparison of three algorithms (E-RNSP*, SN-RNSP*, and TK-RNSP) under different *Min-sup* conditions, with a

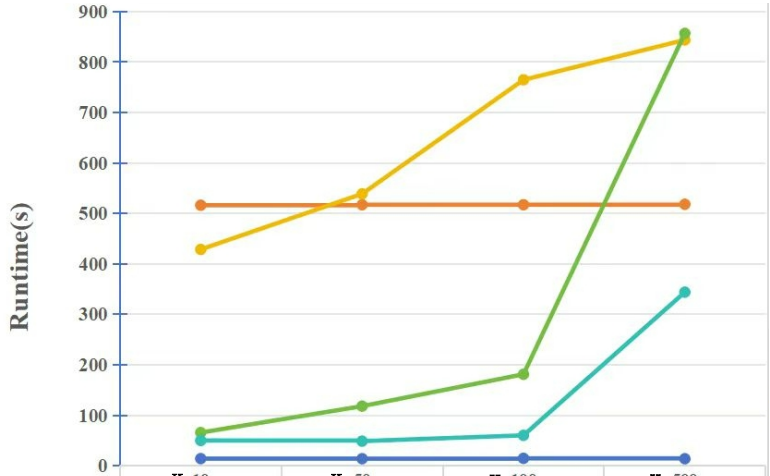
Table 4: Performance Comparison of TK-NSP and TK-RNSP Across DS1, DS3, DS5, and DS6

K	Method	Pattern number				Support range			
		DS1	DS3	DS5	DS6	DS1	DS3	DS5	DS6
10	TK-NSP	2819	2822	15712	56472	[1665,1669]	[2486,2492]	[2026,2026]	[13552,13552]
	TK-RNSP	1228	710	1551	26824	[2470,2477]	[4223,4247]	[13877,13877]	[96163,96163]
100	TK-NSP	4406	4544	16024	56573	[1642,1669]	[2455,2492]	[2026,2026]	[13552,13552]
	TK-RNSP	2278	1847	1806	27037	[2416,2477]	[4133,4247]	[13877,13877]	[96163,96163]
500	TK-NSP	7900	7137	18612	57008	[1606,1669]	[2395,2492]	[2026,2026]	[13551,13552]
	TK-RNSP	5351	3556	3927	27936	[2333,2477]	[3967,4247]	[13877,13877]	[96163,96163]
1000	TK-NSP	10058	8651	21460	57591	[1568,1669]	[2340,2492]	[2026,2026]	[13550,13552]
	TK-RNSP	6694	5116	6477	29036	[2247,2477]	[3801,4247]	[13877,13877]	[96163,96163]

particular focus on their runtime across varying K values. Specifically, the for E-RNSP* is set to 1000, while E-RNSP*' refers to the same algorithm but with *Min-sup* set to 500. Similarly, the *Min-sup* for S-RNSP* is also set to 1000. For TK-RNSP, the *Min-sup* is set to 0, whereas TK-RNSP' represents the version with the *Min-sup* set to 1000.

The figure shows that TK-RNSP (*Min-sup*=0) exhibits a clear advantage in runtime at lower K values. Specifically, at $K = 10$, the runtime is only 65.647s , while SN-RNSP* is at 428.514s , and E-RNSP* is at 13.977s , demonstrating TK-RNSP's ability to complete tasks quickly at lower K values. As K increases, the runtime of TK-RNSP (*Min-sup*=0) rises significantly, reaching 856.738s at $K = 500$. In contrast, SN-RNSP* has a runtime of 843.830s at $K = 500$, while E-RNSP* remains relatively low at 13.736s. This is because SN-RNSP* and E-RNSP* filter Top-K RNSP from a fixed number of patterns under a fixed threshold. Under *Min-sup*=1000, the number of patterns generated by E-RNSP is minimal (as shown in Table 4), resulting in faster runtime. Moreover, the runtime of TK-RNSP increases rapidly with higher K values, particularly at $K = 500$, where there is a significant increase and some variability. This fluctuation occurs because, as K increases, the relative value of *Min-sup* decreases, leading to a significant increase in the number of patterns traversed and the time required.

It is important to note that while this section aims to evaluate the algorithm's ability to rapidly raise the minimum support threshold to find the highest support Top-K RNSP, the described comparison method may have certain limitations due to differences in negative inclusion definitions, pattern generation strategies, and pruning strategies between SN-RNSP and E-RNSP. Additionally, in practical scenarios, we cannot use SN-RNSP and E-RNSP to set reasonable minimum support thresholds for Top-K RNSP without prior experience.



	K=10	K=50	K=100	K=500
E-RNSP*	13.977	13.865	14.509	13.736
E-RNSP*'	516.173	517.146	517.183	518.121
S-RNSP*	428.514	538.665	764.691	843.83
TK-RNSP	65.647	117.964	181.171	856.738
TK-RNSP'	49.884	48.642	60.083	343.901

Fig. 13: Algorithm performance testing on DS5

Table 4: Detailed Runtime, Pattern Number and Support Analysis for E-RNSP*, SN-RNSP*, and TK-RNSP.

Data Source	K	E-RNSP*			SN-RNSP*			TK-RNSP		
		T(ms)	N	R	T(ms)	N	R	T(ms)	N	R
DS1	50	53037	3953	[1838,2046]	5827	7550	[2030,2109]	2037	1352	[2442,2477]
	100	51987	3953	[1681,2046]	5744	7550	[1874,2109]	2471	2278	[2416,2477]
	500	48151	3953	[1118,2046]	5294	7550	[1333,2109]	3937	5351	[2333,2477]
	1000	47745	3953	[702,2046]	5229	7550	[1005,2190]	5329	6694	[2247,2477]
DS2	50	77843	20782	[2503,2585]	25164	134452	[2068,2523]	893	508	[3717,3750]
	100	79589	20782	[2397,2595]	24318	134452	[1998,2523]	1469	1454	[3695,3750]
	500	83265	20782	[1958,2595]	23089	134452	[1212,2523]	3271	3111	[3596,3750]
	1000	84957	20782	[1559,2595]	22564	134452	[636,2523]	4828	4841	[3490,3750]
DS3	50	273302	24000	[2320,2636]	26470	44732	[2786,2920]	1411	841	[5909,5964]
	100	269013	24000	[2088,2636]	26983	44732	[2662,2920]	1777	1847	[5875,5964]
	500	252816	24000	[1112,2636]	26171	44732	[2281,2920]	3704	3556	[5701,5964]
	1000	248625	24000	[512,2636]	25160	44732	[1921,2920]	6062	5116	[5526,5964]
DS5	10	13977	1094	[2086,2139]	428514	549222	[2567,2567]	65647	26824	[13877,13877]
	50	13865	1094	[1878,2139]	538665	549222	[2567,2567]	117964	26914	[13877,13877]
	100	14509	1094	[1729,2139]	764691	549222	[2567,2567]	118171	27037	[13877,13877]
	500	13736	1094	[960,2139]	843830	549222	[2566,2567]	156738	27936	[13877,13877]

5.3 Scalability experiments

To address EQ5, this section conducted scalability experiments to evaluate the performance of TK-RNSP on large-scale datasets, analyzing its stability and efficiency as data volume increases. This part of the experiment includes two datasets: DS1 and DS8. The sizes of the datasets increased from approximately 2.0 MB and 1.6 MB to a maximum of 84.8 MB and 40.9 MB, resulting in a 25-fold increase, with K set to 500 and the initial minimum support threshold set to 0. The test results indicate that TK-RNSP demonstrates strong performance when handling large-scale datasets, exhibiting good scalability and stability.

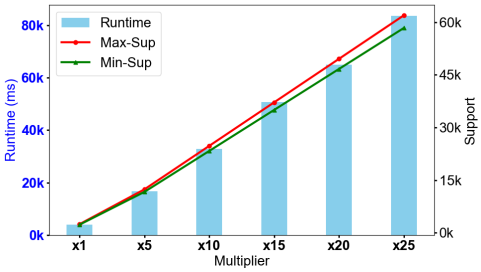


Fig. 14: Performance on DS1

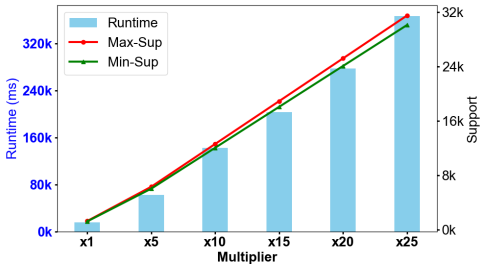


Fig. 15: Performance on DS8

Results are shown in Figure 14 for DS1 in terms of runtime and support as the dataset size is expanded. In the figure, *Max-sup* represents the highest support of patterns in the result set, and *Min-sup* represents the lowest support. It is observed that the runtime increased from 3.937s to 83.542s, displaying a linear growth trend. *Max-sup* increased from 2,477 to 61,925 , and *Min-sup* from 2,333 to 58,325 , also showing a linear growth trend. Figure 15 presents results for DS8. It is found that the algorithm's runtime increased from 15.506s to 366.920s as the dataset size increased, showing a relatively linear growth. *Max-sup* increased from 1,261 to 31,525 , and *Min-sup* from 1,205 to 30,125 , growing linear with data volume.

The experimental results from DS1 and DS8 indicate that the TK-RNSP algorithm maintains stable runtime performance, effectively handling large datasets without sudden drops or unstable fluctuations, demonstrating good scalability and stability.

5.4 Summary

1. Compared to two variants using the same pattern generation and NSP search methods but different pruning strategy, TK-RNSP exhibits significantly higher operational efficiency across multiple datasets, demonstrating the effectiveness of its pruning strategy;
2. Compared to the Top-K non-repetitive NSP mining algorithm, TK-NSP that use the same pattern generation strategy and pruning strategy, TK-RNSP is able to mine patterns with higher support across multiple datasets;
3. In Top-K frequent pattern mining tasks, TK-RNSP demonstrates superior adaptability, efficiently mining patterns even with a default support threshold of 0. This

capability renders TK-RNSP highly practical in real-world applications where prior knowledge is lacking;
4. TK-RNSP demonstrates excellent stability, operating stably on large datasets even for large K-value settings.

6 Conclusion and future work

We propose the TK-RNSP algorithm, which effectively accomplishes Top-K mining tasks for RNSPs, demonstrating high efficiency even when the preset support threshold is set to zero. This addresses the challenges users face in determining appropriate thresholds. Additionally, we have established that RNSP mining within the proposed framework adheres to the property of anti-monotonicity, enabling us to introduce the MAXG-MINUP strategy for pruning. Furthermore, we use DFBS strategy to achieve efficient pattern search and support calculation. Finally, experimental results across multiple datasets indicate that the TK-RNSP algorithm excels in both mining efficiency and result stability.

In the 5G era, social media platforms have rapidly evolved, generating a substantial amount of streaming data, and users' real-time interactions and content uploads present unprecedented challenges. Although the existing TK-RNSP algorithm demonstrates a certain level of efficiency in RNSP mining, it falls short in effectively handling real-time data streams. Therefore, future research should focus on enhancing the algorithm's capacity to process streaming data, particularly by optimizing the TK-RNSP algorithm to adapt to the dynamically changing data environment.With these improvements, the TK-RNSP algorithm is expected to play a significant role in social media data analysis and provide more comprehensive and robust support in emerging fields such as online behavior analysis. This will further advance the development of data mining technologies.

7 Acknowledgements

This paper was supported by the National Natural Science Foundation of China under Grant 62076143, National Natural Science Foundation of China under Grant 62202270,the Fundamental Research Promotion Plan of Qilu University of Technology (Shandong Academy of Sciences) under Grant 2021JC02009, and in part by the Shandong Excellent Young Scientists Fund (Oversea) under Grant 2022HWYQ-044. Additionally, this work received support from the Taishan Scholar Project of Shandong Province under Grant tsqn202306066.

References

1. Cao L, Philip SY (2012) Behavior computing: modeling, analysis, mining and decision. Springer Science & Business Media

2. Cao L, Philip SY, Kumar V (2015) Nonoccurring behavior analytics: A new area. IEEE Intelligent Systems 30(6):4–11

3. Zheng Z (2012) Negative sequential pattern mining. PhD thesis

4. Cao L, Dong X, Zheng Z (2016) e-nsp: Efficient negative sequential pattern mining. Artif Intell 235:156–182

5. Nawaz MS, Fournier-Viger P, Aslam M, et al (2023) Using alignment-free and pattern mining methods for sars-cov-2 genome analysis. Applied Intelligence 53(19):21920–21943

6. Cao L (2020) Health and medical behavior informatics. In: Biomedical Information Technology. Elsevier, p 735–761

7. Nawaz MS, Nawaz MZ, Fournier-Viger P, et al (2024) Analysis and classification of employee attrition and absenteeism in industry: A sequential pattern mining-based methodology. Computers in Industry 159:104106

8. Dong X, Gong Y, Cao L (2018) F-nsp+: A fast negative sequential patterns mining method with self-adaptive data storage. Pattern Recognit 84:13–27

9. Qiu P, Gong Y, Zhao Y, et al (2021) An efficient method for modeling nonoccurring behaviors by negative sequential patterns with loose constraints. IEEE Transactions on Neural Networks and Learning Systems

10. Gao X, Gong Y, Xu T, et al (2020) Toward to better structure and constraint to mine negative sequential patterns. IEEE Trans Neural Netw Learn Syst

11. Dong X, Gong Y, Cao L (2018) e-rnsp: An efficient method for mining repetition negative sequential patterns. IEEE Trans Cybern 50(5):2084–2096

12. Wu Y, Tong Y, Zhu X, et al (2017) Nosep: Nonoverlapping sequence pattern mining with gap constraints. IEEE Trans Cybern 48(10):2809–2822

13. Wu Y, Yuan Z, Li Y, et al (2022) Nwp-miner: Nonoverlapping weak-gap sequential pattern mining. Inf Sci 588:124–141

14. Shi Q, Shan J, Yan W, et al (2020) Netnpg: Nonoverlapping pattern matching with general gap constraints. Appl Intell 50(6):1832–1845

15. Wu Y, Zhu C, Li Y, et al (2020) Netncsp: Nonoverlapping closed sequential pattern mining. Knowl Based Syst 196:105812

16. Li Y, Zhang C, Li J, et al (2023) Maximal co-occurrence nonoverlapping sequential rule mining. arXiv preprint arXiv:230112630

17. Wu Y, Luo L, Li Y, et al (2021) Ntp-miner: Nonoverlapping three-way sequential pattern mining. ACM Transactions on Knowledge Discovery from Data (TKDD) 16(3):1–21

18. Wang Y, Wu Y, Li Y, et al (2021) Self-adaptive nonoverlapping sequential pattern mining. Appl Intell pp 1–16

19. Li Y, Zhang S, Guo L, et al (2022) Netnmsp: Nonoverlapping maximal sequential pattern mining. Appl Intell pp 1–24

20. Wu Y, Luo L, Li Y, et al (2021) Ntp-miner: Nonoverlapping three-way sequential pattern mining. ACM Trans Knowl Discov Data 16(3):1–21

21. Wu Y, Wang Z, Li Y, et al (2024) Co-occurrence order-preserving pattern mining. arXiv preprint arXiv:240419243

22. Geng M, Wu Y, Li Y, et al (2023) Rnp-miner: Repetitive nonoverlapping sequential pattern mining. IEEE Transactions on Knowledge and Data Engineering

23. Wu Y, Geng M, Li Y, et al (2021) Hanp-miner: High average utility nonoverlapping sequential pattern mining. Knowl Based Syst 229:107361

24. Wu Y, Lei R, Li Y, et al (2021) Haop-miner: Self-adaptive high-average utility one-off sequential pattern mining. Expert Syst Appl 184:115449

25. Wu Y, Chen M, Li Y, et al (2023) Onp-miner: One-off negative sequential pattern mining. ACM Trans Knowl Discov Data 17(3):1–24

26. Sun C, Gong Y, Guo Y, et al (2024) Sn-rnsp: Mining self-adaptive nonoverlapping repetitive negative sequential patterns in transaction sequences. Knowledge-Based Systems p 111449

27. Zhang C, Du Z, Gan W, et al (2021) Tkus: Mining top-k high utility sequential patterns. Information Sciences 570:342–359

28. Wu Y, Wang Y, Li Y, et al (2021) Top-k self-adaptive contrast sequential pattern mining. IEEE Trans Cybern

29. Pham TT, Do T, Nguyen A, et al (2020) An efficient method for mining top-k closed sequential patterns. IEEE Access 8:118156–118163

30. Lei M, Chu L, Wang Z, et al (2020) Mining top-k sequential patterns in transaction database graphs: A new challenging problem and a sampling-based approach. World Wide Web 23:103–130

31. Lei M, Zhang X, Yang J, et al (2019) Efficiently approximating top-k sequential patterns in transactional graphs. IEEE Access 7:62817–62832

32. Dong X, Qiu P, Lü J, et al (2019) Mining top-k useful negative sequential patterns via learning. IEEE Trans Neural Netw Learn Syst 30(9):2764–2778

33. Zheng Z, Zhao Y, Zuo Z, et al (2010) An efficient ga-based algorithm for mining negative sequential patterns. In: Advances in Knowledge Discovery and Data Mining: 14th Pacific-Asia Conference, PAKDD 2010, Hyderabad, India, June 21-24, 2010. Proceedings. Part I 14, Springer, pp 262–273

34. Qiu P, Zhao L, Chen W, et al (2018) Mining negative sequential patterns from infrequent positive sequences with 2-level multiple minimum supports. Filomat 32(5):1875–1885

35. Qiu P, Jiang X, Hao F, et al (2018) Mining negative sequential patterns from frequent and infrequent sequences based on multiple level minimum supports. Filomat 32(5):1765–1776

36. Zhang M, Xu T, Li Z, et al (2020) e-hunsr: An efficient algorithm for mining high utility negative sequential rules. Symmetry 12(8):1211

37. Zheng Z, Zhao Y, Zuo Z, et al (2009) Negative-gsp: An efficient method for mining negative sequential patterns. In: Conferences in Research and Practice in Information Technology Series

38. Guyet T, Quiniou R (2020) Negpspan: efficient extraction of negative sequential patterns with embedding constraints. Data Min Knowl Discov 34(2):563–609

39. Wang W, Cao L (2021) Vm-nsp: Vertical negative sequential pattern mining with loose negative element constraints. ACM Trans Inf Syst 39(2):1–27

40. Xu T, Dong X, Xu J, et al (2017) E-msnsp: Efficient negative sequential patterns mining based on multiple minimum supports. International Journal of Pattern Recognition and Artificial Intelligence 31(02):1750003

41. Gong Y, Xu T, Dong X, et al (2017) e-nspfi: Efficient mining negative sequential pattern from both frequent and infrequent positive sequential patterns. International Journal of Pattern Recognition and Artificial Intelligence 31(02):1750002

42. Hsueh SC, Lin MY, Chen CL (2008) Mining negative sequential patterns for e-commerce recommendations. In: 2008 IEEE Asia-Pacific Services Computing Conference, IEEE, pp 1213–1218

43. Srikant R, Agrawal R (1996) Mining sequential patterns: Generalizations and performance improvements. In: International conference on extending database technology, Springer, pp 1–17

44. Pei J, Han J, Mortazavi-Asl B, et al (2004) Mining sequential patterns by pattern-growth: The prefixspan approach. IEEE Transactions on knowledge and data engineering 16(11):1424–1440

45. Huang JW, Jaysawal BP, Chen KY, et al (2019) Mining frequent and top-k high utility time interval-based events with duration patterns. Knowledge and Information Systems 61:1331–1359

46. Davashi R (2023) Itufp: A fast method for interactive mining of top-k frequent patterns from uncertain data. Expert Systems with Applications 214:119156

47. Wang JZ, Huang JL, Chen YC (2016) On efficiently mining high utility sequential patterns. Knowledge and Information Systems 49:597–627

48. Kieu T, Vo B, Le T, et al (2017) Mining top-k co-occurrence items with sequential pattern. Expert Systems with Applications 85:123–133

49. Huang S, Gan W, Miao J, et al (2023) Targeted mining of top-k high utility itemsets. Engineering Applications of Artificial Intelligence 126:107047

50. Nguyen LT, Vo B, Nguyen LT, et al (2018) Etarm: an efficient top-k association rule mining algorithm. Applied Intelligence 48:1148–1160