TaSPM: Targeted Sequential Pattern Mining

GENGSEN HUANG and WENSHENG GAN, Jinan University, Guangzhou, China
PHILIP S. YU, University of Illinois Chicago, Chicago, USA

Sequential pattern mining (SPM) is an important technique in the field of pattern mining, which has many applications in reality. Although many efficient SPM algorithms have been proposed, there are few studies that can focus on targeted tasks. Targeted querying of the concerned sequential patterns can not only reduce the number of patterns generated, but also increase the efficiency of users in performing related analysis. The current algorithms available for targeted sequence querying are based on specific scenarios and can not be extended to other applications. In this article, we formulate the problem of targeted sequential pattern mining and propose a generic algorithm, namely TaSPM. What is more, to improve the efficiency of TaSPM on large-scale datasets and multiple-item-based sequence datasets, we propose several pruning strategies to reduce meaningless operations in the mining process. Totally four pruning strategies are designed in TaSPM, and hence TaSPM can terminate unnecessary pattern extensions quickly and achieve better performance. Finally, we conducted extensive experiments on different datasets to compare the baseline SPM algorithm with TaSPM. Experiments show that the novel targeted mining algorithm TaSPM can achieve faster running time and less memory consumption.

CCS Concepts: • Information systems → Data mining; • Theory of computation → Sorting and searching;

Additional Key Words and Phrases: Data mining, targeted querying, sequential pattern, target sequence

ACM Reference Format:
Gengsen Huang, Wensheng Gan, and Philip S. Yu. 2024. TaSPM: Targeted Sequential Pattern Mining. ACM Trans. Knowl. Discov. Data. 18, 5, Article 114 (February 2024), 18 pages. https://doi.org/10.1145/3639827

1 INTRODUCTION

In the data mining area, frequent pattern mining (FPM) is a well established topic, which aims at discovering all frequent and valuable patterns from the database provided by the user. As for how to specify whether a pattern is frequent or not, FPM is done by comparing the number of its occurrences appearing in the database with the minimum support (minsup) threshold specified by the user. For a frequent pattern, the items that appear in the pattern can be considered disordered. This means that \(<A, B>\) and \(<B, A>\) are the same, because FPM does not consider the order in which \(A\) and \(B\) appear in the transaction record. FPM is widely applied to various services, and it can also be used to develop reasonable solutions and decisions based on different types of data. For
example, by processing the data of a specific breakfast store, two frequent patterns \(<\text{bread}>\) and \(<\text{bread}, \text{milk}>\) can be obtained. And we can learn that most people would make these two combinations of merchandise purchases. From these two frequent patterns, another thing we can know is that the main items sold in this breakfast store are bread and milk. Therefore, the merchants can put bread and milk on top of the same shopping shelf, which may make more customers buy bread and milk at the same time. According to the survey, only those who have bought bread will continue to buy milk, but we can not know this relationship from the pattern \(<\text{bread}, \text{milk}>\). One reason for this may be that bread is a staple food and milk is just a drink. In cases where customers mainly buy bread, some of them will buy milk. On the contrary, few customers buy milk separately or continue to buy bread after buying milk, as inferred by the infrequent pattern \(<\text{milk}>\). Based on the above analysis, the merchants can set a slight increase in the price of bread without increasing the price of the combination of bread and milk. This decision may better promote sales. With this simple breakfast store case, we can clearly see the deficiency of FPM, which ignores the order of items appearing in the pattern.

In order to solve the problem discussed above, sequential pattern mining (SPM) [2, 13] was proposed. SPM can find more valuable and meaningful patterns than FPM. It takes into account the relationship between where each item appears in the data, resulting in \(<\text{A}, \text{B}>\) and \(<\text{B}, \text{A}>\) forming two different patterns. There are many researchers have proposed a series of approaches and algorithms [13, 19, 30] to improve the mining efficiency. Besides, some researchers focus on specific research contexts in order to solve practical problems in different scenarios. Examples include user behavior analysis in business [37], DNA analysis in biology [8], and click-stream analysis on web pages [12]. Considering the format requirements of the discovered sequential patterns, there are also relevant studies introduced, such as closed sequential pattern [24, 44], multidimensional sequential pattern [32, 36], and nonoverlapping sequential pattern [42, 43]. Traditional SPM algorithms are based on the metric frequency, and they aim at discovering sequential patterns that satisfy the \(\text{minsup}\) threshold. Those discovered sequential patterns can be considered as frequent sub-sequences of sequence records. The sequence record in a database can be classified into single-item-based sequence and multiple-item-based sequence depending on whether the itemset of the sequence contains more than one item. In practical applications, multiple-item-based sequences are often more valuable. For example, in a breakfast store, if a customer buys bread and milk simultaneously, the sequence should be represented as \(<\text{(bread, milk)}>\), not \(<\text{(bread), (milk)}>\). The former means that the customer buys bread and milk at one time, while the latter means that the customer buys milk after buying bread.

Obviously, these SPM algorithms will be very inefficient when dealing with a small \(\text{minsup}\) setting or performing query functions. This is due to the large number of meaningless sequential patterns that are generated during the mining process. There are two common querying including top-\(k\) sequential pattern querying [10, 40] and targeted sequential pattern querying [4–6, 47]. The former aims at mining the top-\(k\) most critical sequential patterns, while the latter mines the results that users interested in. Targeted SPM aims at discovering the patterns required by the user, thus avoiding the generation of irrelevant patterns. For example, in a supermarket, there will be a large number of transaction records. If we want to know which frequent sequential patterns are related to pencils, then other frequent sequential patterns such as \(<\text{(pen)}\) should not be generated. Although pens are also a common product for sale. In addition, web search is also a good example for targeted sequential pattern querying. Users often care about their search results, and when a user enters keywords in the browser through the keyboard, the search engine returns relevant content to the user. Although some frequency-based targeted SPM algorithms [4–6] have been proposed, they are either insufficient or not general. These algorithms focus only on those sequential patterns where the last itemset is the query itemset (or sequential patterns that contain a query item).
Such restrictions are clearly not suitable for web search and transaction log analysis. For example, if we want to query a page with a certain key verb, we may get very few results. This is because the verb does not usually appear at the end of the sentence. Furthermore, it is extremely unreasonable to focus on only one item or itemset. In transaction data, these algorithms are unable to provide relevant queries if we focus on the sequential patterns of users purchasing both books and pencils in order. This means some target sequential patterns like \(<(book), (pencil)\) may not be mined.

To address these issues, in this article, we redefine the concept of target sequential pattern and then formulate the problem of targeted sequential pattern mining. Our definition allows the mining tasks to acquire patterns as much as possible while avoiding useful patterns that are ignored. We also present a post-processing technique as well as four strategies: the unpromising transaction filtering pruning strategy (UTFP), the unpromising prefix item pruning strategy (UPIP), the unpromising $S$-Extension item pruning strategy (USIP), and the unpromising $I$-Extension item pruning strategy (UIIP). Based on these pruning strategies, we propose the TaSPM algorithm and its variants. The major contributions of our work in this article are as follows:

- Targeted mining of results that users are interested in makes more sense than mining all patterns. In view of the problems existing in the current targeted sequential pattern mining/searching, we redefine the target sequential pattern and formalize the mining problem of it in this article.
- For generalized SPM algorithms, we first introduce a post-processing technique to save all qualified target sequential patterns in their mining process. And then, an efficient algorithm called TaSPM is also proposed. Based on the filtering approach, we propose UTFP to filter unrelated transactions for TaSPM at the beginning of mining.
- Inspired by the idea of using location information to improve efficiency, we propose UPIP, USIP, and UIIP to reduce unnecessary operations in the mining process. These strategies can well address the efficiency problem of TaSPM on dense, large, and multiple-item-based sequence datasets.
- The real and synthetic datasets are used in the experiments. Experiments show that different variants of TaSPM (adopting different strategies) can achieve different degrees of optimized performance compared to the baseline algorithm.

The remaining part of this article is organized as follows. In Section 2, we briefly review and summarize the previous related work. We present the definitions and formalize the problem of targeted sequential pattern mining in Section 3. And then, we propose the TaSPM algorithm in Section 4. The experimental results are shown and discussed in Section 5. Finally, in Section 6, we draw conclusions and look forward to future work.

2 RELATED WORK

In this section, we discuss SPM in Section 2.1 and targeted pattern querying (TPQ) in Section 2.2.

2.1 Sequential Pattern Mining

SPM was originally proposed by Agrawal and Srikant [2] and can discover all sub-sequences that satisfy the $\minsup$ threshold from a database. There are many algorithms proposed for SPM to discover a complete set of sequential patterns. Some of the better known algorithms are GSP [37], SPADE [46], PrefixSpan [25], SPAM [3], LAPIN [45], FAST [34], CM-SPADE [9], CM-SPAM [9], and Tree-Miner [33]. GSP [37] is a violent algorithm and utilizes the downward-closure property of patterns to prune search space. If a sequential pattern is infrequent, then all of its super-sequences
are infrequent too. Therefore, if the support of a candidate is less than \( \minsup \), then the extension operations or growth process associated with it can be terminated. This property is used in almost all traditional SPM algorithms. The search method of GSP is breadth-first that first scans the database to find all frequent 1-sequences, and then uses these frequent 1-sequences to find longer sequences [37]. Besides, a hash tree can be used to store the data, thus reducing indexing time. However, in any case, this process of merging generates a large number of candidates, making the GSP algorithm very inefficient. Subsequently, the pattern-growth-based PrefixSpan [25] was proposed. With the projection mechanism and two extension approaches, PrefixSpan can use the projected database to achieve far better efficiency than GSP. However, PrefixSpan faces the dilemma of excessive memory usage when dealing with large datasets. To address this issue, some vertical-data-structure-based algorithms, such as SPADE [46] and SPAM [3], are designed with a better database representation to save memory. SPADE reduces the number of database scans by using ID-List, and SPAM also reduces that by using Bitmap. To solve the problems on dense datasets, an algorithm called LAPIN [45] focuses on the position of the last item, allowing the algorithm to achieve excellent results on dense datasets. FAST [34] has improved the data structure by introducing indexed sparse id-lists, which reduce memory consumption by quickly calculating the support of candidates in candidate generation. Furthermore, co-occurrence Map (CMAP) is used in CM-SPADE [9] and CM-SPAM [9] to reduce meaningless operations on specific datasets. Tree-Miner [33] introduced a tree data structure called SP-Tree, and enhanced the performance of the mining tasks.

In addition to the traditional SPM algorithms discussed above, some other types of SPM algorithms have been proposed. **Maximal sequential pattern mining (MSPM)** and **closed sequential pattern mining (CSPM)** are two of these types of algorithms. Compared to MSPM, the patterns obtained from CSPM are lossless. For a sequence \( s \), if there is no super-sequence with the same support as it and containing it, then \( s \) is a closed sequence. For MSPM, it does not require that the support of super-sequence of \( s \) must be the same. Therefore, the set of sequential patterns mined by MSPM can be considered as a subset of the set of sequential patterns mined by CSPM. DIMASP [23], MaxSP [16], and VMSP [15] are the main algorithms of MSPM. And CSPM algorithm has some famous algorithms, including CloSpan [44], ClaSP [24], CM-ClaSP [9], and CloFAST [17], and so on. In addition to constraining the format of sequential patterns to reduce the number of sequential patterns obtained, there are some types of algorithms that discover the user’s desired patterns over the set of all sequential patterns. **Top-k sequential pattern mining (TSPM)** and **targeted sequential pattern mining (TaSPM)** are these types of algorithms, which discover the sequential patterns that meet the user’s requirements. Compared to MSPM and CSPM, TSPM and TaSPM are more from the user’s point of view. TSPM discovers the top-k sequential patterns that are most valuable to the user, and its main algorithms are TSP [40], TKS [10], and SkOPUS [31]. For the existing TaSPM algorithms [4–6], they all focus on those sequential patterns that contain the query item or itemset. Obviously, these SPM algorithms have some drawbacks. Generally speaking, these algorithms were proposed to solve the problem that traditional SPM would discover a large number of sequential patterns in large datasets or under too small \( \minsup \) settings. Too many sequential patterns are not conducive to user analysis and many of them are redundant. In addition to these types of SPM, there are also high-utility SPM [21, 38, 39], sequential generator pattern mining [11, 22], non-overlapping SPM [42, 43], compressing SPM [27], constraint SPM [41], and negative SPM [7].

### 2.2 Targeted Pattern Querying

To reduce the number of irrelevant patterns obtained and improve analytic efficiency, targeted queries play an important role in this process. The earliest mining task that used targeted
querying was the association mining task. Kubat et al. [26] designed the Itemset-Tree to mine which rules with a user-specified target itemset and also discussed three query cases, the most useful of which is that the query is performed by entering the target itemset. In these queries, the system discovers all the rules with the target itemset in their antecedent and that satisfy the minimum support threshold or the minimum confidence threshold. In order to better improve the efficiency of Itemset-Tree, MEIT [14] was proposed. MEIT is a better data structure that reduces unnecessary nodes in the Itemset-Tree, thus improving the efficiency of queries in itemset mining and association rule mining. In order to solve the problem that the Itemset-Tree cannot utilize Apriori property and many invalid operations in the mining process, Lewis et al. [28] designed a fast generation process to obtain query responses and improve efficiency. In addition, query-constraint-based ARM (QARM) [1] was introduced to analyze a wide variety of clinical datasets in the National Sleep Research Resource. Recently, Shabtay et al. [35] proposed GFP-Growth to deal with the problem of multitude-targeted itemsets mining in big data. In utility mining [18], TargetUM [29] was the first algorithm that could find the high-utility target itemsets from the transaction database. Based on a tree-based data structure called the lexicographic querying tree, TargetUM proposed three effective pruning strategies to achieve better efficiency.

All of the above algorithms focus on targeted itemset mining or targeted association rule mining from the transaction database. As mentioned before, sequence data is quite different. In an earlier work, Chiang et al. [5] introduced the definition of Goal-oriented sequential pattern, i.e., a sequential pattern in which any itemset of it contains the target item. An inefficient multi-phase-based algorithm, as well as concepts such as loss, time windows, and normalization are also proposed. Chuah [6] reversed and filtered the database and considered time-intervals to discover all sequential patterns where the last itemset is the query itemset. By considering the old customer recall strategy and RFM model, Chand et al. [4] proposed a PrefixSpan-based algorithm to discover the target sequences that satisfy multiple constraints. These targeted SPM algorithms concern only on an item or itemset of the pattern, and are not hold in many applications. Unlike the previous algorithms for mining target frequent sequences, the TUSQ algorithm [47] is designed based on the utility-driven applications [18, 20]. Inspired by these research works, we aim at providing a generic problem of targeted sequential pattern mining and propose a new efficient algorithm.

3 PRELIMINARIES AND PROBLEM FORMULATION

In this section, we define some fundamental concepts and notations about target patterns. Subsequently, we formalize the problem of targeted sequential pattern mining.

In this article, we use letters to denote the basic units, and each item can be considered as a map of the real items. Let \( I = \{i_1, i_2, \cdots, i_N\} \) be an finite set of distinct items occurring in a database. An itemset is a subset of \( I \) and is denoted as \( X (X \subseteq I) \). All items in an itemset are sorted according to the alphabetical order, which is presented as \( >_{\text{lex}} \). For an itemset \( X \), its size is the number of items that appear in it, denoted as \( |X| \). For a sequence \( S = <X_1, X_2, \cdots, X_m> \), its size is the number of itemsets it contains, and the length is the total number of items in it. For another sequence \( T = <Y_1, Y_2, \cdots, Y_n> \), if \( S \) is a sub-sequence of \( T (S \subseteq T) \), then \( \exists 1 \leq k_1 < k_2 < \cdots < k_m \leq n \) such that \( \forall 1 \leq v \leq m, X_v \subseteq Y_{k_v} \). For example, given two sequences \( S_1 = <f, g, f> \) and \( S_2 = <f, h, f, g, f> \), the size and length of \( S_1 \) are 3 and 4, respectively. In addition, \( S_1 \) is a sub-sequence of \( S_2 \), because that is \( S_1 \subseteq S_2 \).

**Definition 3.1 (Sequence database).** For a sequence (transaction record) database \( D \), it is a set of tuples \( <sid, S> \) in which \( sid \) is the identifier of the sequence \( S \). The size of this set is also the size of the database \( D \), denoted as \(|D|\).
Table 1. Sequence Database

| sid | Sequence              |
|-----|-----------------------|
| s₁  | ⟨⟨g⟩⟩, ⟨a, d⟩⟩        |
| s₂  | ⟨⟨g⟩⟩, ⟨a, b, c, d⟩⟩, ⟨b⟩, ⟨f⟩⟩ |
| s₃  | ⟨⟨g⟩⟩, ⟨a, b, c, d⟩⟩, ⟨a, b⟩, ⟨e⟩⟩ |
| s₄  | ⟨⟨a⟩⟩, ⟨b, c, d, e⟩⟩, ⟨e⟩, ⟨f⟩⟩ |

To better illustrate the relevant definitions and concepts that follow, the sequence database we use as an example database is shown in Table 1. We can see that this example database has four sequences, and each of them has a corresponding unique identifier.

**Definition 3.2 (Support and minimum support).** For an item e or an itemset I, its support can be denoted as sup(e / I) and defined as sup(e / I) = |T | e / I ∈ T ∧ T ∈ D|. Similarly, the support value of a sequence S can be denoted as sup(S) and defined as sup(S) = |T | S ⊆ T ∧ T ∈ D|. The minimum support is defined in advance by the user and is denoted as minsup. For an item e or a sequence S, if it satisfies sup(e / S) ≥ minsup, it is called a frequent item or sequential pattern.

For example, in Table 1, if minsup is set to 2, then all items are frequent items. And ⟨⟨g⟩⟩, ⟨a⟩⟩ is a frequent sequential pattern, because that is sup(⟨⟨g⟩⟩, ⟨a⟩⟩) = 3 ≥ 2.

**Definition 3.3 (Target sequential pattern and targeted sequential pattern mining).** Given a query sequence, called qs, the target sequential pattern ts of qs satisfies qs ⊆ ts and sup(ts) ≥ minsup, i.e., qs is the sub-sequence of the frequent sequence ts. The goal of targeted sequential pattern mining is to find a complete set of target sequential patterns for the query sequence qs. When qs is an empty sequence, this means that all frequent sequential patterns in the database are its target sequential patterns.

For example, in Table 1, given a query sequence qs = ⟨⟨a⟩⟩, ⟨b⟩⟩ and minsup = 2, then ts₁ = ⟨⟨g⟩⟩, ⟨a⟩⟩, ⟨b⟩⟩ is one of the target sequential patterns of qs, because of qs ⊆ ts₁ and sup(ts₁) = 2 ≥ minsup. While ts₂ = ⟨⟨g⟩⟩, ⟨a⟩⟩, ⟨b⟩⟩, ⟨e⟩⟩ is not, because of sup(ts₂) = 1 < minsup. Based on the above concepts, a generic problem of targeted sequential pattern mining can be formulated below.

**Problem statement:** Given a sequence database D, a query sequence qs, and a predefined minimum support threshold minsup, the goal of targeted sequential pattern mining is to discover the entire set of target sequential patterns ts of qs, and they satisfy qs ⊆ ts and sup(ts) ≥ minsup.

4 THE TASPM ALGORITHM

In this section, we first present some important definitions and several pruning strategies. Consequently, we propose an efficient algorithm, called TaSPM, which is based on CM-SAPM [9].

4.1 Definitions and Pruning Strategies

**Definition 4.1 (Extension).** Extension is an important operation of the pattern-growth-based mining algorithms [3, 25, 46]. In the mining process, two types of extensions are used: S-Extension and I-Extension. The S-Extension adds a new itemset to the end of the current sequence, and that new itemset contains the expanded item. As for I-Extension, it adds an expanded item to the last itemset of the current sequence. Obviously, with these two extensions, the sequence can be used to grow.

**Definition 4.2 (Post-processing technique).** As we discussed above, there is no algorithm for the new definition of targeted sequential pattern mining. If we use a traditional SPM algorithm to
perform some query tasks, then the post-processing technique can assist in work. This technique works when the SPM algorithm saves a sequential pattern during the mining process. It scans the sequential pattern generated and determines if the sequential pattern contains the query sequence. If the pattern contains the query sequence and is frequent, it is a target sequential pattern and can be saved, otherwise it is filtered.

Definition 4.3 (Itemset match position and item match position). In the mining process, for a current sequential pattern, we specify that the corresponding item at the matching position of the query sequence is called the current query item, denoted as $q_i$. This means that the items before $q_i$ in the query sequence can be matched in the current pattern. During the pattern growth, we use two flags to record the current position match on $q_i$. The first flag is called an itemset match position, denoted as IMatch. IMatch is used to record the position of the itemset currently matched to the query sequence. The second flag is called item match position and is denoted as IIMatch, which is used to match the position of the item in the itemset currently matched to the query sequence. In addition, both flags start from 0.

For example, given a query sequence $<(g), (a, d)>$ and a current sequential pattern $<(g), (b)>$, we can know that the matching sequence is $<(g)>$ and $q_i$ is $a$. Therefore, IMatch is set to 1 and IIMatch is set to 0.

Definition 4.4 (Bitmap representation of a sequence). For a sequence $s$, if we represent it using a bitmap representation, it can be denoted as $[S_{11}, S_{12}, ..., S_{21}, ..., ...]$. In a sequence database, we use $S_{ij}$ to associate with the $j$th itemset in the $i$th sequence of the database. If the sequence $s$ can be matched by $S_i$ and the matching position is $j$, then $S_{ij}$ is set to 1.

For example, in Table 1, given a sequence $s = <(a), (b)>$, if we use a bitmap to represent this query sequence, then it would be $[0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0]$. This is because $S_2$, $S_3$, and $S_4$ can all match $s$ and their positions are 2, 2, and 1, respectively.

Question 1. When does it mean that the query sequence has been matched? We initialize IMatch and IIMatch to 0 when the sequence is <>. As the sequence is used to grow, these two flags are updated. When IMatch is equal to the size of the query sequence, it means that the query sequence is matched exactly. After that, we can stop updating IMatch and IIMatch and start saving the target sequential patterns as well.

Question 2. When will IMatch and IIMatch be updated and not be updated? We can decide whether to update IMatch and IIMatch based on whether the extended item of the current sequence matches to $q_i$. Initially, both IMatch and IIMatch are initialized to 0.

When the current sequence is nothing, i.e., the current sequence is <> and $q_i$ is the first item of the query sequence. For an extended item $e$, if it does not match $q_i$, then both IMatch and IIMatch remains 0. If the item $e$ matches $q_i$, then IIMatch is updated to 1. Subsequently, if IIMatch equals the size of the first itemset in the query sequence, then IMatch is updated to IIMatch + 1 and IIMatch is updated to 0. As the sequence is used to grow, each extended item is compared with $q_i$ of the query sequence. If the extended item $e$ is the same as $q_i$, then IIMatch is updated to IIMatch + 1. After IIMatch is updated, if IIMatch satisfies IIMatch equals the size of the IMatch-th itemset of the query sequence, then IMatch is updated to IIMatch + 1 and IIMatch is updated to 0.

There is another situation where IIMatch needs to be updated. When IIMatch is not 0, but the current sequence performs S-Extension, this indicates that we need to reset the matching position, i.e., update IIMatch to 0. If the extended item matches $q_i$ after updating the position of the query sequence, then update IIMatch to 1 and decide whether we need to continue updating IMatch and IIMatch.
As for the sequence performs I-Extension, if the extended item \( e \) is less than \( q_i \), we will not update IMatch and IIMatch. If the extended item \( e \) and \( q_i \) are the same, then we update IIMatch, and then repeat the above judgment process to update IMatch. If the item \( e \) is greater than \( q_i \), the IIMatch has to be updated to 0 since all the items in the itemset are sorted according to alphabetical order. In this case, the sequence continues to perform I-Extension is unable to match the itemset located at the position IMatch of the query sequence.

Consider the following scenario: the current sequence is \((a, b)\), and the query sequence is \((a, b, c, e)\). When we add the item \( c \) to the current sequence using I-Extension, the sequence becomes \((a, b, c)\), and it can continually perform I-Extension. Since the extended item is \( c \) is consistent with the item to be queried, then IIMatch is updated and causes IMatch to be updated. Obviously, if we continue to perform I-Extension with item \( e \), IIMatch will be updated. In this way, the matching position is wrong. This is because \((a, b, c, e)\) and \((a, b, c, e)\) are not the same. The former can not match the query sequence, while the latter can. To solve this problem, we use the flag NotUpdate to determine if updates are not performed on IMatch and IIMatch. If NotUpdate is true, we will not update the IMatch and IIMatch. If the growth of the sequence can still continue by performing I-Extension after IMatch is updated, NotUpdate is updated to true. When the sequence performs S-Extension, NotUpdate is updated to false. There is also another case where we will not update IMatch and IIMatch, i.e., IMatch is equal to the size of the query sequence, which means that the query sequence has been completely matched.

**Strategy 1 (Unpromising transaction filtering pruning strategy, UTFP).** For a query sequence \( qs \), if a transaction record in the database \( D \) does not contain \( qs \), the transaction record is filtered. It is quite obvious that transaction records that do not contain \( qs \) do not effect the support calculation of target sequential patterns. By removing these irrelevant transaction records, TaSPM not only reduces the memory used to store these records, but also improves efficiency. If the size of the database \( D \) is smaller than \( minsup \) after the database \( D \) is filtered, it means that no frequent target sequential patterns can be mined.

For example, in Table 1, given a query sequence \( qs = \langle a \rangle \), if we use a bitmap to represent this query sequence, then it would be \([0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1]\). Obviously, we should filter the sequences \( s_1 \) and \( s_3 \) to obtain a more concise representation, i.e., \([0, 0, 0, 1, 0, 0, 0, 1]\). If we do not use UTFP to filter these two sequences, when \( minsup = 2 \) and the current sequence is \( \langle g \rangle \), the support of \( \langle g \rangle \) is 3 and it can be extended recursively. In this way, invalid operations become more numerous because the support of the target sequential pattern with \( g \) is equal to 1 at the highest. As another example, if the query pattern is \( \langle a \rangle, (e), (f) \rangle \), after the database \( D \) is filtered, then \( D \) contains only \( s_4 \). In this case, \(|D|\) is less than \( minsup \), then no frequent target sequential patterns are generated.

**Strategy 2 (Unpromising prefix item pruning strategy, UPnP).** In the initial stage of the most sequential pattern mining algorithms, recursive growth is generally performed using frequent items. For a prefix item \( e \), we compare the bitmap of \( e \) with the bitmap of \( q_i \) and count the number of sequences where the position of \( q_i \) appears after the frequent item \( e \). This number is denoted as \( fn \). In this case, \( q_i \) is also the first item in the query sequence. If item \( e \) is not \( q_i \) and \( fn \) is less than \( minsup \), we do not use item \( e \) to continue to grow pattern. This is because the maximum support of any target sequential pattern beginning with \( e \) is less than \( minsup \), and we can safely assume that \( e \) is not a beginning item used to grow patterns.

For example, in Table 1, given a query sequence \( qs = \langle a \rangle, (b) \rangle \) and \( minsup \) is set to 2, we can know that the item \( f \) is a frequent item because \( sup(f) = 2 \geq minsup \). Then, we do a bitmap
comparison using UP. For this query sequence \( q_s, q_i \) is \( a \) and its bitmap is \([0 \ 1 \ 0 \ 0, 0 \ 1 \ 0 \ 0, 0 \ 1 \ 0 \ 0] \). For the frequent item \( f \), its bitmap is \([0 \ 0 \ 0 \ 1, 0 \ 0 \ 0 \ 0, 0 \ 0 \ 0 \ 1] \). Obviously, according to UIIP, \( fn \) is equal to 0. We do not need to use the frequently occurring item \( f \) to grow the pattern. In fact, item \( f \) can be used as a part of the target sequential pattern, such as \(<(a), (b), (f)>, sup(<(a), (b), (f)>) = 2.\)

**Strategy 3 (Unpromising S-Extension item pruning strategy, USIP).** We propose USIP to decide which sequence can be used to grow pattern recursively after being extended by S-Extension. We compare the bitmap of the sequence carrying the extended item \( e \) with the bitmap of \( q_i \) and count the number of sequences where the position of \( q_i \) appears after the position of \( e \). This number is denoted as \( fn \). If \( fn \) is less than \( minsup \), we do not use the item \( e \) as an extended item to grow pattern. This is because the maximum support of the target sequential pattern with the extended sequential pattern as a prefix can not be greater than \( minsup \). If the extended item \( e \) matches \( q_i \), then \( fn \) is changed to the number of sequences where the position of \( q_i \) is the same as or after the position of \( e \).

For example, in Table 1, given a query sequence \( q_s = <(a), (b)>, a \) current sequence \( s = <(g)>, and minsup = 2, the item \( b \) is an extended item that can be extended by S-Extension, and the extended sequence \( s' = <(g), (b)>\) is a frequent pattern. In this case, \( q_i = a \). The bitmap of \( s' = [0 \ 1 \ 1 \ 0, 0 \ 1 \ 0 \ 0, 0 \ 0 \ 0 \ 0] \) and the bitmap of \( q_i \) is \([0 \ 1 \ 0 \ 0, 0 \ 1 \ 1 \ 0, 1 \ 0 \ 0 \ 0] \). Therefore, \( fn \) is equal to 1. \( s' \) is not used to grow pattern. In fact, \( s' \) can continue to be extended, but the sequential pattern obtained by its extension will not belong to the target sequential pattern of the query sequence \( q_s \). If the extended item is \( a \), then the bitmap of \( s' = [0 \ 1 \ 0 \ 0, 0 \ 1 \ 1 \ 0, 0 \ 0 \ 0 \ 0] \) and \( fn \) is equal to 2 \( \geq minsup \). Therefore, \(<(g), (a)>, can be used to grow pattern. Finally, we can obtain a sequence \( t = <(g), (a), (b)>\) that is a target sequential pattern w.r.t. the query sequence \( q_s \).

**Strategy 4 (Unpromising I-Extension item pruning strategy, UIIP).** It is sufficient to use only USIP on the single-item-based sequence dataset, because the sequence can only perform S-Extension to grow pattern. However, on multiple-item-based sequence datasets, sequences can be performed I-Extension, which requires UIIP to reduce meaningless extension operations. We propose UIIP to decide which sequences can continue to grow pattern recursively after being extended by I-Extension. As we discussed above in Question 2, we do not need to update \( IMatch \) and \( II\) \( IMatch \) if the extended item is less than \( q_i \). Otherwise, we need to update \( II\) \( IMatch \) and \( IMatch \). If the match position is updated, \( q_i \) may also have been changed. For an extended item, there are two situations that we need to consider. The first one is that the item \( e \) is less than or equal to \( q_i \). In this case, we use \( fn \) to count the number of sequences where the position of \( q_i \) is equal to or greater than the position of \( e \). The second case is when the item \( e \) is greater than \( q_i \), and \( II\) \( IMatch \) is updated to 0. We use \( fn \) to count the number of sequences where the position of \( q_i \) greater than the position of \( e \). For these two cases, if \( fn \) is less than \( minsup \), we do not use the item \( e \) as an extended item to grow a pattern.

For example, in Table 1, given a query sequence \( q_s = <(g), (a), (b)>, a \) current sequence \( s = <(g), (a)>, and minsup = 2, items \( b, c, \) and \( d \) are extended items that can be extended by I-Extension. For the item \( b \), it is smaller than \( q_i \) and \( q_i \) will not be updated. The bitmap of extended sequence \( s' = [0 \ 1 \ 0 \ 0, 0 \ 1 \ 0 \ 0] \) and the bitmap of \( q_i \) is \([0 \ 1 \ 0 \ 0, 0 \ 1 \ 0 \ 0] \). In this case, \( fn \) is equal to 2 and \( s' = <(g), (a), (b)>\) can continually to grow pattern. For the item \( c \), it matches \( q_i \) and \( q_i \) is updated to \( b \). The bitmap of extended sequence \( s' = <(g), (a), (c)> \) is \([0 \ 0 \ 0 \ 0, 0 \ 1 \ 0 \ 0] \) and the bitmap of \( q_i \) is \([0 \ 0 \ 1 \ 0, 0 \ 1 \ 0 \ 0] \). In this case, \( fn \) is equal to 2 and \( s' \) can continually to grow pattern. Since the last item \( d \) is greater than \( q_i \), \( q_i \) is updated to \( a \). The bitmap of the extended sequence \( s' = <(g), (a), (d)> \) is \([0
1 0 0, 0 1 0 0] and the bitmap of qi is [0 0 0, 0 0 1 0]. Therefore, fn is equal to 1 and s’ is not used to grow pattern.

4.2 Proposed TaSPM Algorithm

TaSPM utilizes a bitmap representation for all items and sequences, which is similar to CM-SPAM [9]. Based on four novel pruning strategies, we propose the TaSPM algorithm for the targeted discovery of sequential patterns. TaSPM is a depth-first-search algorithm, and the main procedure is shown in Algorithm 1.

TaSPM first scans the database D to filter transaction records that do not contain the query sequence (Line 1). After the database D is filtered, TaSPM scans the database again to get all frequent items (denoted as F1), removes all infrequent items from D, creates the bitmap for all frequent items, and constructs the CMAP for co-occurrence pruning (Line 2). TaSPM initializes the relevant variables qi, IMatch, and IIMatch for the next operations (Line 3). For all frequent items in F1, TaSPM gets fn and uses UPIP to decide whether such prefix patterns can continually grow (Lines 4–8). If a frequent prefix pattern satisfies the minsup, TaSPM uses the function UpdateMatch to update the matching position (Line 9). The details of the function UpdateMatch can be referred to the above discussion, and we do not describe them here. Finally, TaSPM performs the recursive procedures to grow patterns (Line 10).

ALGORITHM 1: The TaSPM algorithm

Input: D: a sequence database; qs: a query sequence provided by users; minsup: a minimum support threshold defined by users.

Output: all target sequential patterns w.r.t. the query sequence qs.

1. filter sequences that do not contain qs in D; (UTFP Strategy)
2. scan D again to 1) discover all frequent items (F1); 2) remove all infrequent items from D; 3) create the bitmap for all frequent items; and 4) construct the CMAP;
3. initialize qi ← the first item in qs, IMatch ← 0, IIMatch ← 0;
4. for <f> ∈ F1 do
   5. compute fn ← the result by comparing the bitmap of f with the bitmap of qi;
   6. if fn < minsup then
      7. continue; (UPIP Strategy)
   end
   8. IMatch, IIMatch ← call UpdateMatch();
   9. call SEARCH(<f >, F1, {e | e ∈ F1 ∧ e ≻ lex f }, qs, minsup, IMatch, IIMatch).
10. end

The SEARCH procedure is shown in Algorithm 2. It first uses IMatch to determine whether the current pattern exactly matches the query sequence, and outputs this frequent pattern if it does (Lines 1–3). Next, it initializes Stemp, Pr temp, newlMatch, newllMatch, qi, s’, and fn (Lines 4–6). Stemp and Pr temp record those items that can be extended by S-Extension or I-Extension, respectively. newlMatch and newllMatch record the new match position of qi. s’ is used to represent the extended sequence for s, and fn is used by USIP and UIIP. And then, the SEARCH procedure finds out all items which are not skipped by co-occurrence pruning (Lines 7–10). SEARCH uses these items to create s’ and judges whether s’ is frequent (Line 11). If s’ occurs frequently, the extended item f is added to Stemp (Lines 12–14). For all extended items in Stemp, SEARCH updates newlMatch and newllMatch, and obtains fn (Lines 16–19). By utilizing USIP, the SEARCH procedure can terminate some meaningless operations in S-Extension of s (Lines 20–22). Subsequently, more target
ALGORITHM 2: The SEARCH procedure

Input: s: the current sequence; SE: a set containing all items that can be performed by S-Extention; IE: a set containing all items that can be performed by I-Extention; qs: a query sequence provided by users; minsup: a minimum support threshold defined by users; IMatch: the matching position of itemset of the query sequence; II_Match: the matching position of item in the matching itemset.

1. if IMatch == qs.size() then
2. output s;
3. end
4. initialize S_temp ← ∅, I_temp ← ∅;
5. initialize newIMatch ← IMatch, newIIMatch ← II_Match;
6. initialize qi, s′ ← ∅, fn ← 0;
7. for f ∈ SE do
8. if CMAP(s, f) < minsup then
9. continue; (Co-occurrence Pruning)
10. end
11. s′ ← s extend f by S-Extention;
12. if sup(s′) ≥ minsup then
13. S_temp ← S_temp ∪ f;
14. end
15. end
16. for f ∈ S_temp do
17. s′ ← s extend f by S-Extention;
18. newIMatch, newIIMatch ← call UpdateMatch();
19. fn ← the result by comparing the bitmap of s′ with the bitmap of qi;
20. if fn < minsup then
21. continue; (USIP Strategy)
22. end
23. call SEARCH(s′, S_temp, {e | e ∈ S_temp ∧ e > lex f}, qs, minsup, newIMatch, newIIMatch);
24. end
25. for f ∈ IE do
26. if CMAP(s, f) < minsup then
27. continue; (Co-occurrence Pruning)
28. end
29. s′ ← s extend f by I-Extention;
30. if sup(s′) ≥ minsup then
31. I_temp ← I_temp ∪ f;
32. end
33. end
34. for f ∈ I_temp do
35. s′ ← s extend f by I-Extention;
36. newIMatch, newIIMatch ← call UpdateMatch();
37. fn ← the result by comparing the bitmap of s′ with the bitmap of qi;
38. if fn < minsup then
39. continue; (UIIP Strategy)
40. end
41. call SEARCH(s′, S_temp, {e | e ∈ I_temp ∧ e > lex f}, qs, minsup, newIMatch, newIIMatch);
42. end
sequential patterns are discovered recursively by the search procedure (Line 23). The $I$-Extension process of $s$ is similar to that of $S$-Extension (Lines 25–42).

5 EXPERIMENTS

In this section, we designed two variants of the TaSPM algorithm to evaluate the effectiveness and efficiency of the proposed strategies. The first variant is called TaSPM$_V^1$ which only uses the UTFP and a post-processing technique. TaSPM$_V^2$ is another variant that utilizes all the proposed strategies, including UTFP, UPIP, USIP, and UIIP. In TaSPM$_V^2$, partial strategies are used along with the matching positions of the query sequence. Therefore, TaSPM$_V^2$ does not need to utilize the post-processing technique to save target patterns. To ensure that the number of obtained patterns is correct, we used the CM-SPAM algorithm [9] as a comparison, which also makes use of the post-processing technique. TaSPM$_V^1$ and TaSPM$_V^2$ were compared with CM-SPAM in terms of running time, memory consumption, and the number of bitmap intersections.

Our experimental equipment is a bare computer with a 64-bit Windows 10 operating system. It has an AMD Ryzen 5 3600 CPU and 8 GB RAM. All algorithms are implemented in the Java language. The details of the experimental results are given below.

5.1 Datasets for the Experiment

We used six datasets for our experiments, and three of them are real datasets, including Bible, Sign, and Accidents. They are converted from books or real-life application records. Each itemset of these three real datasets has only one item, which means they are all single-item-based sequence datasets. These datasets are available for download from the open source platform SPMF. Bible is a moderately dense dataset and transformed from the book Bible. Sign is a dense sign language dataset that has a few hundred items but many long sequences. Accidents is an anonymized traffic accident dataset. We chose a version of the accident dataset containing 10 K records in our experiments. For synthetic datasets Syn10k, Syn20k, and Syn40k, they are all multiple-item-based sequence datasets and generated from the IBM data generator [2].

These synthetic datasets have many moderately long sequences, and can simulate some records of realistic applications very well. The details of all datasets used in experiments are shown in Table 2. For convenience, $|\mathcal{D}|$ is the size of dataset $\mathcal{D}$, $|\mathcal{I}|$ is the number of distinct items in $\mathcal{D}$, $\text{avg}(\mathcal{I})$ is the average number of distinct items in an itemset of $\mathcal{D}$, $\text{avg}(\mathcal{S})$ is the average length of sequences of $\mathcal{D}$, $\text{min}(\mathcal{S})$ and $\text{max}(\mathcal{S})$ are respectively the minimum and maximum size of sequences of $\mathcal{D}$.

For six different datasets, we have selected six query sequences with size 3 for our experiments. They are $\text{query}_{\text{Bible}} = <(10), (46), (38)>$, $\text{query}_{\text{Sign}} = <(17), (117), (144)>$, $\text{query}_{\text{Accidents}} = <(16), (31), (43)>$, $\text{query}_{\text{Syn10k}} = <(196), (8845, 9250), (4010)>$, $\text{query}_{\text{Syn20k}} = <(1801), (842, 4616), (7752)>$, and $\text{query}_{\text{Syn40k}} = <(196), (8845, 9250), (4010)>$.

Table 2. Details of Different Experimental Datasets

| Dataset    | $|\mathcal{D}|$ | $|\mathcal{I}|$ | $\text{avg}(\mathcal{I})$ | $\text{avg}(\mathcal{S})$ | $\text{min}(\mathcal{S})$ | $\text{max}(\mathcal{S})$ |
|------------|----------------|----------------|---------------------------|---------------------------|--------------------------|--------------------------|
| Bible      | 36,369         | 13,905         | 1.0                       | 21.64                     | 9                        | 100                      |
| Sign       | 730            | 267            | 1.0                       | 51.99                     | 18                       | 94                       |
| Accidents10k | 10,000        | 310            | 1.03                      | 33.92                     | 23                       | 45                       |
| Syn10k     | 10,000         | 7,312          | 4.35                      | 27.11                     | 1                        | 18                       |
| Syn20k     | 20,000         | 7,442          | 4.33                      | 26.97                     | 1                        | 18                       |
| Syn40k     | 40,000         | 7,537          | 4.32                      | 26.84                     | 1                        | 18                       |

1http://www.philippe-fournier-viger.com/spmf/
query_{Syn40k} = \langle 8496, 5926, 6384, 9737 \rangle$. These sequences are randomly selected from the complete mining set. The query length that is too long makes the number of target sequential patterns particularly small, and the gap between each algorithm is not obvious. Too short a query length makes too many target sequential patterns generated, which is not conducive to analysis. On the multiple-item-based synthetic datasets, the length of these query sequences is 4. This can ensure that the target patterns need to be extended through both $I$-Extension and $S$-Extension. Note that if a test case was run for more than 10,000 minutes in an experiment, we considered it unable to discover patterns. Correspondingly, its memory consumption is not displayed.

### 5.2 Efficiency Analysis

In this subsection, we conducted a lot of experiments by adjusting $\text{minsup}$. To better view the gap between algorithms, we presented the logarithm of the runtime values. The experimental results are shown in Figure 1. From the results, it is clear that variants of the TaSPM algorithm can achieve better performances compared to the baseline CM-SPAM algorithm. Even better, TaSPM$V_2$ can reduce the running time by more than one order of magnitude.

Since TaSPM$V_2$ uses the most pruning strategies, it has the fastest running time. On the Bible dataset, the runtime of TaSPM$V_1$ is 150% to 200% of TaSPM$V_2$. This difference is very significant on the other datasets. When $\text{minsup}$ is set to 40, the baseline CM-SPAM algorithm can not mine any target pattern. Even with $\text{minsup}$ set to 20, both TaSPM$V_1$ and TaSPM$V_2$ are able to mine all patterns in less than 30 minutes. On the Sign dataset, the baseline CM-SPAM algorithm does not run when $\text{minsup}$ is 5. The runtime difference between the baseline CM-SPAM algorithm and TaSPM$V_1$ on the Accidents dataset is not as large as on the other datasets. This is because many sequences in the database contain the query sequence, and the database filters out few sequences. On the Syn10k dataset, the baseline CM-SPAM algorithm can not mine any target sequential pattern when $\text{minsup}$ is set to 5, and TaSPM$V_1$ can not mine any target sequential patterns when $\text{minsup}$ is set to 2. However, TaSPM$V_2$ is able to finish this task in about 200 seconds. Note that some runtime lines do not show a linear decrease because when the $\text{minsup}$ is small, the change in $\text{minsup}$ does not make a significant change in the number of generated target sequential patterns. This also occurs in the other two synthetic datasets, but not obviously.

In addition, we also used the number of intersections to compare the efficiency of each algorithm, and the results are shown in Table 3. Intersection is the main operation of the mining process, which largely affects the runtime. On the Bible dataset, it is obvious that CM-SPAM has 10 times
Table 3. Number of Intersections under Various \textit{minsup}

| Dataset    | Algorithm | \textit{minsup}_1 | \textit{minsup}_2 | \textit{minsup}_3 | \textit{minsup}_4 | \textit{minsup}_5 | \textit{minsup}_6 |
|------------|-----------|-------------------|-------------------|-------------------|-------------------|-------------------|-------------------|
| Bible      | CM-SAPM   | /                 | 29,730,973        | 11,373,572        | 5,775,770         | 3,422,826         | 2,232,202         |
|            | TaSPM\textsubscript{V1} | 10,696,148        | 3,796,616         | 1,205,362         | 536,414           | 291,818           | 177,875           |
|            | TaSPM\textsubscript{V2} | 4,613,035         | 1,714,605         | 550,128           | 249,246           | 136,309           | 83,073            |
| Sign       | CM-SAPM   | /                 | 326,469,305       | 93,050,762        | 38,131,279        | 19,044,929        | 10,709,256        |
|            | TaSPM\textsubscript{V1} | 777,682,420       | 66,036,945        | 16,115,832        | 5,730,301         | 2,510,663         | 1,248,830         |
|            | TaSPM\textsubscript{V2} | 61,293,211        | 5,464,723         | 1,426,235         | 544,759           | 261,804           | 144,437           |
| Accidents\textsubscript{10k} | CM-SAPM   | /                 | 5,411,453         | 1,678,640         | 590,381           | 243,176           | 111,509           |
|            | TaSPM\textsubscript{V1} | 4,613,035         | 1,048,936         | 330,199           | 124,854           | 51,979            | 23,290            |
|            | TaSPM\textsubscript{V2} | 807,694           | 257,933           | 94,368            | 39,531            | 17,953            | 8,428             |
| Syn\textsubscript{10k} | CM-SAPM   | /                 | 387,870,990       | 150,708,918       | 70,466,773        | 45,694,609        | 19,593,591        |
|            | TaSPM\textsubscript{V1} | 266,338,270       | 66,846,686        | 24,004,574        | 16,920,542        | 4,081,633         | 2,240,481         |
|            | TaSPM\textsubscript{V2} | 16,649,265        | 4,181,041         | 1,503,409         | 1,060,657         | 257,823           | 142,751           |
| Syn\textsubscript{20k} | CM-SAPM   | /                 | 100,042,878       | 27,263,239        | 19,970,108        | 13,631,370        | 7,414,568         |
|            | TaSPM\textsubscript{V1} | 33,554,406        | 7,024,614         | 6,029,286         | 4,860,006         | 2,248,310         | 500,573           |
|            | TaSPM\textsubscript{V2} | 2,144,150         | 486,038           | 602,930           | 350,750           | 176,126           | 51,871            |
| Syn\textsubscript{40k} | CM-SAPM   | /                 | 387,870,990       | 150,708,918       | 70,466,773        | 45,694,609        | 19,593,591        |
|            | TaSPM\textsubscript{V1} | 266,338,270       | 66,846,686        | 24,004,574        | 16,920,542        | 4,081,633         | 2,240,481         |
|            | TaSPM\textsubscript{V2} | 16,649,265        | 4,181,041         | 1,503,409         | 1,060,657         | 257,823           | 142,751           |

Fig. 2. Memory usage under various \textit{minsup}.

more intersections than TaSPM\textsubscript{V1} and 20 times more intersections than TaSPM\textsubscript{V2}. On the \textit{Sign} dataset, the number of intersections of CM-SPAM is tens of times greater than that of TaSPM\textsubscript{V1}. On the \textit{Accidents\textsubscript{10k}} dataset, the difference between CM-SPAM and TaSPM\textsubscript{V1} is more than double. On the synthetic dataset, for each algorithm, the number of intersections still differs greatly. TaSPM\textsubscript{V2} that uses the most pruning strategies always performs the least number of intersections, regardless of any dataset. This performance is more evident on some datasets.

5.3 Memory Evaluation

The results of each algorithm in terms of memory consumption are shown in Figure 2. We can see from the \textit{Bible} dataset that CM-SPAM consumes more memory than TaSPM\textsubscript{V1} and TaSPM\textsubscript{V2}. This gap is slowly decreasing as the \textit{minsup} increases. Besides, in all test cases, TaSPM\textsubscript{V1} uses a little more memory than TaSPM\textsubscript{V2}. We can also see that the memory consumption shows a linear
relationship, with \( \text{minsup} \) increasing and plateauing at the end. On the \( \text{Sign} \) dataset, TaSPM\(_V1\) uses more memory when \( \text{minsup} \) is greater than or equal to 20. As expected, TaSPM\(_V2\) uses the least amount of memory. The memory consumption also decreases with the increase of \( \text{minsup} \). On the \( \text{Accidents10k} \) dataset, the effect of \( \text{minsup} \) is almost the same as on the \( \text{Bible} \) dataset. However, the difference in memory consumption is not significant, and there are a few more unstable test cases. This is because the dataset filters a few sequences. On the three synthetic datasets, we can see that CM-SPAM uses several times more memory. This is because there are many records that do not contain the query sequence that are not filtered, increasing the memory consumption. In addition, the memory is consumed in a fluctuating range of stable values. This is because the \( \text{minsup} \) of the experiments is not changed very much. In brief, TaSPM\(_V2\) is still the most memory-efficient algorithm.

5.4 Scalability
In this subsection, we used 10 datasets to evaluate the scalability of each algorithm. We used \( \text{qs} = \left< (196), (8845, 9250), (4010) \right> \) as the query sequence and set \( \text{minsup} \) to 8. The size of the dataset grows from 6,000 sequences to 15,000 sequences. We mainly evaluate the two aspects of runtime and memory consumption, and the experimental results are shown in Figure 3. It is clear that as the size of the dataset increases, the runtime of CM-SPAM increases linearly. We can see that TaSPM\(_V1\) and TaSPM\(_V2\) do not change much in some test cases as the dataset size increases, and the number of frequent target sequential patterns remains the same. This is because the size of the dataset affects the runtime of CM-SPAM, whereas the number of target sequential patterns affects the runtimes of TaSPM\(_V1\) and TaSPM\(_V2\).

In terms of memory consumption, CM-SPAM uses several times more memory compared to TaSPM\(_V1\) and TaSPM\(_V2\). The general trend of memory consumption for CM-SPAM also increases with dataset size. When the dataset size is between 8 k and 15 k, both TaSPM\(_V1\) and TaSPM\(_V2\) consume approximately 160 MB of memory. In addition, the number of frequent target sequential patterns does not increase linearly. It remains the same or suddenly increases abruptly as the dataset size increases.

5.5 Target Sequence Analysis
In Section 5.2, the query sequence considers \( I \)-Extension and \( S \)-Extension, but does not consider some special query cases. To better evaluate the performance of TaSPM, we selected different query sequences \text{w.r.t.} the target sequences to perform relevant experiments on the \( \text{Syn10k} \) dataset. The query sequences are \( \text{qs}_1 = \left< (1069) \right> \), \( \text{qs}_2 = \left< (1069), (8808), (9661) \right> \), and \( \text{qs}_3 = \left< (1069, 2594, 5375) \right> \). Note that \( \text{qs}_1 \) represents the case of querying only one item, \( \text{qs}_2 \) represents the case of querying only sequences that are extended using \( S \)-Extension, and \( \text{qs}_3 \) represents the case of querying only sequences that are extended using \( I \)-Extension. The experimental results are shown in Figures 4 and 5, respectively.
Fig. 4. Runtime of different query sequences.

Fig. 5. Memory of different query sequences.

From these results, we can see that, under the three query sequences, CM-SPAM is still the least efficient. It fails to discover the interesting patterns when $\minsup$ is reduced to 5. We can see that as the length of a query pattern increases, the running time of TaSPM$_{V2}$ on most datasets decreases, but the running time of TaSPM$_{V1}$ does not change significantly. In terms of memory consumption, as analyzed in Section 5.3, TaSPM$_{V2}$ uses the least amount of memory. The memory consumed by the baseline CM-SPAM algorithm does not fluctuate much under different query sequences. In conclusion, TaSPM$_{V2}$ is the best algorithm among these three comparison algorithm for targeted sequential pattern mining.

6 CONCLUSIONS

In this article, to address a series of problems incurred by the current sequential pattern mining algorithms and targeted sequence querying approaches, we redefine the concept of target sequential pattern and propose an efficient algorithm called TaSPM to quickly finish mining tasks. Targeted mining makes more sense than mining all patterns, since the obtained results are more specific and meaningful for users. We introduce a post-processing technique for traditional SPM algorithms to ensure that the mined patterns meet various requirements. Moreover, we propose four pruning strategies to reduce meaningless operations of TaSPM in the mining process. Finally, experiments show that TaSPM can achieve better performance than the baseline CM-SPAM algorithm for different datasets and different query sequences. In the future, we can propose different effective algorithms for specific applications based on the TaSPM algorithm. In addition, some data structures, such as trees and hyperlinks, can also be applied to the task of targeted sequential pattern mining. Collecting target sequences from big data or distributed data is also a worthwhile research direction.

REFERENCES

[1] Rashmie Abeysinghe and Licong Cui. 2017. Query-constraint-based association rule mining from diverse clinical datasets in the national sleep research resource. In Proceedings of the 11th IEEE International Conference on Bioinformatics and Biomedicine. IEEE, 1238–1241.
[2] Rakesh Agrawal and Ramakrishnan Srikant. 1995. Mining sequential patterns. In Proceedings of the 11th International Conference on Data Engineering. IEEE, 3–14.

[3] Jay Ayres, Jason Flannick, Johannes Gehrke, and Tomi Yu. 2002. Sequential pattern mining using a bitmap representation. In Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. 429–435.

[4] Chetna Chand, Amit Thakkar, and Amit Ganatra. 2012. Target oriented sequential pattern mining using recency and monetary constraints. International Journal of Computer Applications 45, 10 (2012), 12–18.

[5] Ding-An Chiang, Yi-Fan Wang, Shao-Lun Lee, and Cheng-Jung Lin. 2003. Goal-oriented sequential pattern for network banking churn analysis. Expert Systems with Applications 25, 3 (2003), 293–302.

[6] Hao-En Chueh. 2010. Mining target-oriented sequential patterns with time-intervals. arXiv:1009.0929. Retrieved from https://arxiv.org/abs/1009.0929

[7] Xiangjun Dong, Yongshun Gong, and Longbing Cao. 2018. F-NSP+: A fast negative sequential patterns mining method with self-adaptive data storage. Pattern Recognition 84 (2018), 13–27.

[8] Avrilia Floratou, Sandeep Tata, and Jignesh M. Patel. 2011. Efficient and accurate discovery of patterns in sequence data sets. IEEE Transactions on Knowledge and Data Engineering 23, 8 (2011), 1154–1168.

[9] Philippe Fournier-Viger, Antonio Gomariz, Manuel Campos, and Rincy Thomas. 2014. Fast vertical mining of sequential patterns using co-occurrence information. In Proceedings of the 18th Pacific-Asia Conference on Knowledge Discovery and Data Mining. Springer, 40–52.

[10] Philippe Fournier-Viger, Antonio Gomariz, Ted Gueniche, Espérance Mwamikazi, and Rincy Thomas. 2013. TKS: Efficient mining of top-\(k\) sequential patterns. In Proceedings of the 9th International Conference on Advanced Data Mining and Applications. Springer, 109–120.

[11] Philippe Fournier-Viger, Antonio Gomariz, Michal Šebek, and Martin Hlosta. 2014. VGEN: Fast vertical mining of sequential generator patterns. In Proceedings of the 16th International Conference on Database Warehousing and Knowledge Discovery. Springer, 476–488.

[12] Philippe Fournier-Viger, Ted Gueniche, and Vincent S. Tseng. 2012. Using partially-ordered sequential rules to generate more accurate sequence prediction. In Proceedings of the 8th International Conference on Advanced Data Mining and Applications. Springer, 431–442.

[13] Philippe Fournier-Viger, Jerry Chun-Wei Lin, Rage Uday Kiran, Yun Sing Koh, and Rincy Thomas. 2017. A survey of sequential pattern mining. Data Science and Pattern Recognition 1, 1 (2017), 54–77.

[14] Philippe Fournier-Viger, Espérance Mwamikazi, Ted Gueniche, and Usel Faghhi. 2013. MEIT: Memory efficient itemset tree for targeted association rule mining. In Proceedings of the 9th International Conference on Advanced Data Mining and Applications. Springer, 95–106.

[15] Philippe Fournier-Viger, Cheng-Wei Wu, Antonio Gomariz, and Vincent S. Tseng. 2014. VMSP: Efficient vertical mining of maximal sequential patterns. In Proceedings of the 27th Canadian Conference on Artificial Intelligence. Springer, 83–94.

[16] Philippe Fournier-Viger, Cheng-Wei Wu, and Vincent S. Tseng. 2013. Mining maximal sequential patterns without candidate maintenance. In Proceedings of the 9th International Conference on Advanced Data Mining and Applications. Springer, 169–180.

[17] Fabio Fumarola, Pasqua Fabiana Lanotte, Michelangelo Ceci, and Donato Malerba. 2016. CloFAST: Closed sequential pattern mining using sparse and vertical id-lists. Knowledge and Information Systems 48, 2 (2016), 429–463.

[18] Wensheng Gan, Jerry Chun-Wei Lin, Philippe Fournier-Viger, Han-Chieh Chao, Vincent S. Tseng, and Philip S. Yu. 2021. A survey of utility-oriented pattern mining. IEEE Transactions on Knowledge and Data Engineering 33, 4 (2021), 1306–1327.

[19] Wensheng Gan, Jerry Chun-Wei Lin, Philippe Fournier-Viger, Han-Chieh Chao, and Philip S. Yu. 2019. A survey of parallel sequential pattern mining. ACM Transactions on Knowledge Discovery from Data 13, 3 (2019), 1–34.

[20] Wensheng Gan, Jerry Chun-Wei Lin, Jieexiong Zhang, Han-Chieh Chao, Hamido Fujita, and Philip S. Yu. 2020. ProUM: Projection-based utility mining on sequence data. Information Sciences 513 (2020), 222–240.

[21] Wensheng Gan, Jerry Chun-Wei Lin, Jieexiong Zhang, Hongzhi Yin, Philippe Fournier-Viger, Han-Chieh Chao, and Philip S. Yu. 2021. Utility mining across multi-dimensional sequences. ACM Transactions on Knowledge Discovery from Data 15, 5 (2021), 1–24.

[22] Chuancong Gao, Jianyong Wang, Yukai He, and Lizhu Zhou. 2008. Efficient mining of frequent sequence generators. In Proceedings of the 17th International Conference on World Wide Web. 1051–1052.

[23] René Arnulfo García-Hernández, José Francisco Martínez-Trinidad, and Jesús Ariel Carrasco-Ochoa. 2006. A new algorithm for fast discovery of maximal sequential patterns in a document collection. In Proceedings of the 7th International Conference on Intelligent Text Processing and Computational Linguistics. Springer, 514–523.

[24] Antonio Gomariz, Manuel Campos, Roque Marin, and Bart Goethals. 2013. ClaSP: An efficient algorithm for mining frequent closed sequences. In Proceedings of the 17th Pacific-Asia Conference on Knowledge Discovery and Data Mining. Springer, 50–61.
[25] Jiawei Han, Jian Pei, Behzad Mortazavi-Asl, Helen Pinto, Qiming Chen, Umeshwar Dayal, and Mei-Chun Hsu. 2001. PrefixSpan: Mining sequential patterns efficiently by prefix-projected pattern growth. In Proceedings of the 17th International Conference on Data Engineering. IEEE, 215–224.

[26] Miroslav Kubat, Aladdin Hafez, Vijay V. Raghavan, Jayakrishna R. Lekkala, and Wei-Kian Chen. 2003. Itemset trees for targeted association querying. IEEE Transactions on Knowledge and Data Engineering 15, 6 (2003), 1522–1534.

[27] Hoang Thanh Lam, Fabian Mörchen, Dmitriy Fradkin, and Toon Calders. 2014. Mining compressing sequential patterns. Statistical Analysis and Data Mining 7, 1 (2014), 34–52.

[28] Jay Lewis, Ryan G. Benton, David Bourrie, and Jennifer Lavergne. 2019. Enhancing itemset tree rules and performance. In Proceedings of the 7th IEEE International Conference on Big Data. IEEE, 1143–1150.

[29] Jinbao Miao, Shicheng Wan, Wensheng Gan, Jiayi Sun, and Jiashui Chen. 2021. Targeted high-utility itemset querying. In Proceedings of the 9th IEEE International Conference on Big Data. IEEE, 5534–5543.

[30] Carl H. Mooney and John F. Roddick. 2013. Sequential pattern mining-approaches and algorithms. Computing Surveys 45, 2 (2013), 1–39.

[31] François Petitjean, Tao Li, Nikolaj Tatti, and Geoffrey I. Webb. 2016. SkOPUS: Mining top-\(k\) sequential patterns under leverage. Data Mining and Knowledge Discovery 30, 5 (2016), 1086–1111.

[32] Helen Pinto, Jiawei Han, Jian Pei, Ke Wang, Qiming Chen, and Umeshwar Dayal. 2001. Multi-dimensional sequential pattern mining. In Proceedings of the 10th International Conference on Information and Knowledge Management. 81–88.

[33] Redwan Ahmed, Mohammad Arefin, and Chowdhury Farhan Ahmed. 2020. Tree-Miner: Mining sequential patterns from SP-Tree. In Proceedings of the 24th Pacific-Asia Conference on Knowledge Discovery and Data Mining. Springer, 44–56.

[34] Eliana Salvemini, Fabio Fumarola, Donato Malerba, and Jiawei Han. 2011. Fast sequence mining based on sparse id-lists. In Proceedings of the 19th International Symposium on Methodologies for Intelligent Systems. Springer, 316–325.

[35] Lior Shabtay, Philippe Fourrier-Viger, Rami Yaari, and Itai Dattner. 2021. A guided FP-Growth algorithm for mining multiple-targeted item-sets and class association rules in imbalanced data. Information Sciences 553 (2021), 353–375.

[36] Petre Tzvetkov, Xifeng Yan, and Jiawei Han. 2005. TSP: Mining top-\(k\) closed sequential patterns. Knowledge and Information Systems 7, 4 (2005), 438–457.

[37] Zhenglu Yang, Yitong Wang, and Masaru Kitsuregawa. 2007. LAPIN: Effective sequential pattern mining algorithms by last position induction for dense databases. In Proceedings of the 12th International Conference on Database Systems for Advanced Applications. Springer, 1020–1023.

Received 17 January 2022; revised 10 March 2023; accepted 15 December 2023