High-utility sequential pattern mining (HUSPM) is a significant and valuable activity in knowledge discovery and data analytics with many real-world applications. In some cases, HUSPM can not provide an excellent measure to predict what will happen. High-utility sequential rule mining (HUSR) discovers high utility and high confidence sequential rules, so it can solve the issue in HUSPM. However, all existing HUSR algorithms aim to find high-utility partially-ordered sequential rules (HUSRs), which are not consistent with reality and may generate fake HUSRs. Therefore, in this article, we formulate the problem of high-utility totally-ordered sequential rule mining and propose a novel algorithm, called TotalSR, which aims to identify all high-utility totally-ordered sequential rules (HTSRs). TotalSR introduces a left-first expansion strategy that can utilize the anti-monotonic property to use a confidence pruning strategy. TotalSR also designs a new utility upper bound: \( RSPEU \), which is tighter than the existing upper bounds. TotalSR can drastically reduce the search space with the help of utility upper bounds pruning strategies, avoiding much more meaningless computation. To effectively compute the information, TotalSR proposes an auxiliary antecedent record table that can efficiently calculate the antecedent’s support and utility prefix sum list that can compute the upper bound in \( O(1) \) time for a sequence. Finally, there are numerous experimental results on both real and synthetic datasets demonstrating that TotalSR is more efficient than the existing algorithms.

CCS Concepts: • Information systems → Data mining;

Additional Key Words and Phrases: Data mining, utility mining, knowledge discovery, sequence rule, totally-ordered.

ACM Reference format:
Chunkai Zhang, Maohua Lyu, Wensheng Gan, and Philip S. Yu. 2024. Totally-ordered Sequential Rules for Utility Maximization. ACM Trans. Knowl. Discov. Data. 18, 4, Article 80 (February 2024), 23 pages.
https://doi.org/10.1145/3628450

1 INTRODUCTION

In the age of rapid data generation, how to extract valuable information from data is a challenging task. Sequential pattern mining (SPM) [1, 9, 15, 26] with many real-life applications such as
biological medicine diagnosis [25, 32], web-page click-stream analysis [3, 6], market basket analysis [30], and e-learning [44] is such a method that helps people discover valuable information from data.

SPM discovers patterns as the frequency of the sub-sequences in the sequence database, which only captures the bald sequential patterns but cannot provide a good measure to predict what will happen and what the probability of the sequences is if they appear after a specific sequence. Zaki et al. [37] proposed sequential rule mining (SRM) that uses the additional concept of confidence to find sequential rules. SRM can provide a probability from one sequence to the other sequence, which can be used in sequence prediction [5] and recommendation [19]. The goal of SRM is to identify all sequential rules (SRs) that satisfy the minimum support threshold (minsup) and minimum confidence threshold (minconf). Generally speaking, SR is represented as $\langle X \rangle \rightarrow \langle Y \rangle$, where $X$ represents the antecedent of SR and $Y$ stands for its consequent. SR can be viewed as the probability of $Y$ occurring under the condition that $X$ appears. For example, an SR $\langle \{\text{television, refrigerator}\} \rangle \rightarrow \langle \text{air-conditioning} \rangle$ with confidence equals 0.6 means that there is a probability of 60% that customers will buy air-conditioning after purchasing television and refrigerator.

However, both SPM and SRM output result in the frequency of sequence occurrence in the database, which treats all items in the database as the same weight and does not consider the number of occurrences of the same item in an itemset. It is not committed to reality. For instance, in the selling data of a supermarket, we should not only consider the quantity of the commodities (items) but also the price of each commodity, because both the commodity’s quantity and price matter, which can bring high profit to people. To solve the limitation mentioned above, utility-driven pattern mining has been proposed [3, 15, 33, 36]. The utility can take both the quantity (internal utility) and unit utility (external utility) of each item into account. For example, in retail scenarios, the internal utility of the item is the purchased quantity and the external utility of that item is its price. Furthermore, the utility of one specific item equals its internal utility multiplied by its external utility.

**High-utility sequential rule mining (HUSRM)** [43] was proposed to discover the more valuable SRs. There are two types of sequential rules: partially-ordered and totally-ordered. The partially-ordered SR consists of two unordered itemsets, i.e., both the antecedent and the consequent contain only one itemset each. The antecedent and consequent are all sequences in totally-ordered SR, i.e., they can contain multiple itemsets. HUSRM discovers the partially-ordered HUSR, which only requires that items in the antecedent of a HUSR occur earlier than the consequent. HUSRM does not consider the inner order of the antecedent or consequent of a HUSR. However, the sequential relationship between each item does matter. Thinking about two sequences $S_1$: $\langle \text{heart attack, emergent measures, go to hospital, survival} \rangle$ and $S_2$: $\langle \text{heart attack, go to hospital, emergent measures, death} \rangle$, HUSRM may output a rule: $r = \langle \text{heart attack, emergent measures, go to hospital} \rangle \rightarrow \langle \text{survival} \rangle$, where the order between heart attack, emergent measures, and, go to hospital has been ignored. When there is a sequence like $S_2$, if we use $r$ to predict, then we will get a result: survival. However, in the practical sequence, the following result is death. Thus, HUSRM cannot precisely handle this issue. More general, Table 1(a) is an example sequence database, and Table 1(b) are some of the partially-ordered HUSRs output from Table 1(a). Note that we assume the external utility of all items is 1. The rule $\langle a, b, c \rangle \rightarrow \langle e \rangle$ cannot be output precisely in HUSRM, since its confidence is lower than minconf. But in the totally-ordered SRM, the totally-ordered rule $r = \langle a, c, b \rangle \rightarrow \langle e \rangle$ with utility equals 17 and confidence equals 1 will be output. CMRules [5] mentioned that the totally-ordered SR is so specific but infrequent that the usage of rules is low. Nevertheless, in the utility-driven SRM, the infrequent SR can also be high-utility. Especially, in anomaly detection by using utility-driven SRM, anomaly SR like $r$ will escape. Besides, in the totally-ordered SRM, $R_2$ is a low-utility SR, since $R_2$ is merged from the rules $\langle f, g \rangle \rightarrow \langle h,
Table 1. The Example Sequence Database that Reveals the Problem in Partially-ordered HUSR

| SID | Sequence               | HUSRs                      | utility | confidence |
|-----|------------------------|----------------------------|---------|------------|
| s_1 | \langle (a, 2), (b, 2), (c, 1) \rangle \langle d, 3 \rangle | R_1 = \langle [a, b], c \rangle \rightarrow \langle d \rangle | 15      | 0.67       |
| s_2 | \langle (a, 2), (c, 1), (b, 2) \rangle \langle r, 12 \rangle | R_2 = \langle [f, g] \rangle \rightarrow \langle [h, j] \rangle | 15      | 1          |

The example sequence database (a) contains 5 sequences: $s_1, s_2, s_3, s_4, s_5$. Each sequence is a set of itemsets, where each itemset is a pair of item and its frequency. The utility and confidence of each sequence are calculated based on the itemsets present in the sequence.

In this example, $j > f, g \rightarrow j, h$, causing the utility of $R_2$ increased. Thus, if we only simplify the sequence into two parts such that in each part we do not care about the inner order, then some infrequent but valuable HUSRs will be filtered and some fake HUSRs like $R_2$ will be output. Therefore, it is necessary to formulate an algorithm for discovering high-utility totally-ordered sequential rules.

However, there are many challenges in high-utility totally-ordered sequential rules mining (ToSRM). First, unlike HUSRM, they use bit vectors of items to calculate the support of the antecedent easily. For example, in Table 1(a), the bit vector of $f$ and $g$ is 00011 and 00011. To get the support of antecedent of $R_2$, HUSRM only needs to intersect the bit vectors of $f$ and $g$ and then count the number of 1 in the result bit vector. However, the bit vectors cannot be used in ToSRM; since there is a chronological order between items, we must design an additional data structure to maintain the antecedent projected database to get the support of the antecedent correctly. Second, the search space is tremendous, since the combination of items is explosive especially when we consider the totally-ordered chronological relationship. Finally, the existing utility upper bounds used in HUSRM are not tight enough, making the pruning strategies used in HUSRM not work well in totally-ordered rule mining. Thus, developing an excellent pruning strategy remains a challenge.

To the best of our knowledge, there is no work that discovers high-utility totally-ordered sequential rules. For the sake of the limitations of high-utility partially-ordered sequential rule mining, in this article, we formulate the problem of high-utility totally-ordered sequential rule mining and propose an algorithm called TotalSR, which aims to find all high-utility totally-ordered sequential rules (HTSRs) in the sequence database. To solve the challenges in ToSRM, in TotalSR, we design two data structures: the utility table and the auxiliary antecedent record table (ART). They can compute the confidence of HTSRs in $O(1)$ time. Inspired by the remaining utility [16, 17, 33], in this article, we propose a novel utility upper bound, named the reduced sequence prefix extension utility (RSPEU). It is tighter than the utility upper bounds used in HUSRM [43] and US-Rule [20]. In addition, to fast compute the utility upper bound for a given sequence, we design a data structure called utility prefix sum list (UPSL), which can calculate the utility upper bound of the given sequence in $O(1)$ time. Based on RSPEU, we can prune the search space efficiently. Moreover, we use a left-first expansion strategy to utilize the confidence to prune search space, which can make great use of the anti-monotonic property [2] to avoid the invalid expansion of low-confidence HTSR. The main contributions of this work can be outlined as follows:

- We formulate the problem of high-utility totally-ordered sequential rule mining and propose an algorithm, TotalSR, which can find all HTSRs in a given sequence database efficiently.
- New utility upper bound RSPEU is proposed, which is tighter than the existing utility upper bounds used in high-utility partially-ordered SRM. A left-first expansion strategy is introduced, which can utilize the anti-monotonic property of confidence to prune the search space.
The designed data structures: utility table, ART, and UPSL, can maintain the information of the candidate HTSRs. With the help of these data structures, TotalSR can efficiently compute the value of utility and confidence of each candidate HTSR.

Experiments on both real and synthetic datasets show that TotalSR with all optimizations is much more efficient compared to those algorithms that use only a few optimizations.

The rest of this article is organized as follows: In Section 2, we briefly review the related work on HUSPM, SRM, and HUSRM. The basic definitions and the formal high-utility totally-ordered sequential rule mining problem are introduced in Section 3. The proposed algorithm, TotalSR, as well as the corresponding pruning strategies, are provided in Section 4. In Section 5, we discuss the experimental results of both real and synthetic datasets. Finally, the conclusions of this article and future work are discussed in Section 6.

2 RELATED WORK

There is a lot of work on high-utility sequential pattern mining (HUSPM) and sequential rule mining (SRM), but the research work on high-utility sequential rule mining (HUSRM) is few. In this section, we separately review the prior literature on HUSPM, SRM, and HUSRM.

2.1 High-utility Sequential Pattern Mining

To make the found sequential patterns meet the different levels of attention of users, utility-oriented sequential pattern mining, which mines the patterns that satisfy a minimum utility threshold defined by users, has been widely developed in recent years. However, since utility is neither monotonic nor anti-monotonic, the utility-oriented pattern mining method does not possess the Apriori property [2], which makes it difficult to discover patterns in the utility-oriented framework compared to the frequency-based framework. Ahmed et al. [3] developed the sequence weighted utilization (SWU) to prune the search space. They also designed two tree structures, UWAS-tree and IUWAS-tree, to discover high-utility sequential patterns (HUSPs) in web log sequences. With the help of SWU, which has a downward closure property based on the utility upper bound SWU, the search space can be pruned like the Apriori property. After that, UtilityLevel and UtilitySpan [4] were proposed based on the SWU, in which they first generated all candidate patterns and then selected the HUSPs. Therefore, they are time-consuming and memory-costing algorithms. USMP [29] applied HUSPM to analyze mobile sequences. However, all these algorithms [3, 4, 29] assume that each itemset in a sequence only contains one item. Therefore, the applicability of these algorithms is hard to expand. After that, USpan [36] introduced a data structure called utility-matrix, which can discover HUSPs from the sequences consisting of multiple items in each itemset, to help extract HUSPs. In addition, USpan utilized the upper bound SWU to efficiently find HUSPs. However, there is a big gap between the SWU and the exact utility of a HUSP, which means the SWU upper bound will produce too many unpromising candidates. To solve the problem of SWU, HUS-Span [33], which can discover all HUSPs by generating fewer candidates, introduced two other utility upper bounds, prefix extension utility (PEU) and reduced sequence utility (RSU). However, the efficiency of HUS-Span is still not good enough. To more efficiently discover all HUSPs, Gan et al. [16] proposed a novel algorithm ProUM that introduces a projection-based strategy and a new data structure called the utility array. ProUM can extend a pattern faster and take up less memory based on the projection-based strategy. HUSP-ULL [17] introduced a data structure called UL-list, which can quickly create the projected database according to the prefix sequence. Besides, HUSP-ULL also proposed the irrelevant item pruning strategy that can remove the unpromising items in the remaining sequences to reduce the remaining utility, i.e., to generate a tighter utility-based upper bound.
In addition to improving the efficiency of the HUSPM algorithms, there are also many algorithms that apply HUSPM to some specific scenarios. OSUMS [41] integrated the concept of on-shelf availability into utility mining for discovering high-utility sequences from multiple sequences. To get the fixed numbers of HUSPs and avoid setting the minimum utility threshold, which is difficult to determine for different datasets, TKUS [40] was the algorithm that only mined top-$k$ numbers of HUSPs. CSPM [38] was the algorithm that required the itemset in the pattern to be contiguous, which means that the itemsets in the pattern found in CSPM occur consecutively. To acquire HUSPs consisting of some desired items, Zhang et al. [39] proposed an algorithm called TUSQ for targeted utility mining. By integrating the fuzzy theory, PGFUM [13] was proposed to enhance the explainability of the mined HUSPs.

2.2 Sequential Rule Mining

To successfully predict the probability of the occurrence of the next sequence while mining the pattern, sequential rule mining (SRM) was proposed as a complement to sequential pattern mining (SPM) [1, 9, 15, 26, 34, 35]. Differing from SPM, a sequential rule (SR) counts the additional condition of confidence, which means an SR should not be less than both the conditions of $\text{minsup}$ and $\text{minconf}$. An SR is defined as $<X> \rightarrow <Y>$ and $X \cap Y = \emptyset$, where $X$ and $Y$ are subsequences from the same sequence. In general, according to the SR forming style, there are two types of SRM: partially-ordered SRM and totally-ordered SRM. The first type of rule indicates that both antecedent and consequent in an SR are unordered sets of items [7, 10, 11]. But the items that appear in the consequent must be after the items in the antecedent, which means a partially-ordered SR consists of only two itemsets formed by disorganizing the original itemsets in the given sequence. The second type of rule states that both antecedent and consequent are sequential patterns [24, 27]. In other words, both the antecedent and consequent follow the original order in the given sequence. There are lots of algorithms for sequential rule mining. Sequential rule mining was first proposed by Zaki et al. [37]. They first mined all sequential patterns (SPs) and then generated SRs based on SPs, which is inefficient. Since it mines all SPs as the first step, the SRs they obtained belonged to the totally-ordered SR. To improve the efficiency of the SRM, CMRules [5] introduced the left and right expansion strategies. After that, RuleGrowth [10] and TRuleGrowth [11], which discovered partially-ordered sequential rules, were proposed, in which they used the left and right expansions to help the partially-ordered SR growth just like PrefixSpan [18]. The authors only extracted partially-ordered sequential rules, because they explained that some totally-ordered SRs are too specific to infrequent. Since the partially-ordered SRM does not care about the ordering in the inner antecedent or consequent, it can simplify the mining process. Therefore, ERMiner [7], which makes great use of the property of partially-ordered, proposed a data structure, Sparse Count Matrix, to prune some invalid rules generation and improve efficiency. Lo et al. [24] proposed a non-redundant SRM algorithm that discovers the non-redundant SR, in which each rule cannot be a sub-rule of the other rule. Pham et al. [27] enhanced the efficiency of non-redundant SRM based on the idea of prefix-tree. Gan et al. [14] proposed an SRM algorithm that can discover target SRs.

2.3 High-utility Sequential Rule Mining

Although SRM [5, 7, 10, 11] can provide the probability of the next sequence to users, it just finds the rules that satisfy the frequency requirement, which may omit some valuable but infrequent rules. Therefore, Zida et al. [43] introduced the utility concept into SRM and proposed a utility-oriented sequential rule mining algorithm called HUSRM. Similarly to References [5, 7, 10, 11], HUSRM also used the partially-ordered SRM method and introduced a data structure called the utility table to maintain the essential information about candidate rules for expansion. Besides,
HUSRM designed a bit map to calculate the support value of the antecedent and made some optimizations to improve efficiency. Afterward, Huang et al. [20] proposed an algorithm called US-Rule to enhance the efficiency of the algorithm. Inspired by $PEU$ and $RSU$ from HUSPM in US-Rule, to remove useless rules, they proposed four utility-based upper bounds: left and right expansion estimated utility ($LEEU$ and $REEU$) and left and right expansion reduced sequence utility ($LERSU$ and $RERSU$). There are also some extensions to high-utility sequential rule mining. DUOS [12] extracted unusual high-utility sequential rules, i.e., to detect the anomaly in sequential rules. DOUS is the first work that links anomaly detection and high-utility sequential rule mining. Zhang et al. [42] addressed HUSRM with negative sequences and proposed the e-HUNSR algorithm. HAUSRules [28] introduced the high average-utility concept into sequential rule mining and found all high average-utility sequential rules in the gene sequences. However, all of these approaches address partially-ordered high-utility sequential rules.

3 DEFINITIONS AND PROBLEM DESCRIPTION

In this section, we first introduce some significant definitions and notations used in this article. Then, the problem of high-utility totally-ordered sequential rule mining is formulated.

3.1 Preliminaries

Definition 3.1 (Sequence Database). Let $I = \{i_1, i_2, \ldots, i_q\}$ be a set of distinct items. An itemset (also called element) $I_k$ is a nonempty subset of $I$, that is, $I_k \subseteq I$. Note that each item in an itemset is unordered. Without loss of generality, we assume that every item in the same itemset follows the lexicographical order $\geq_{lex}$, which means $a < b < \cdots < z$. Besides, we will omit the brackets if an itemset only contains one item. A sequence $s = \langle e_1, e_2, \ldots, e_m \rangle$, where $e_i \subseteq I$ $(1 \leq i \leq m)$, is consisted of a set of ordered itemsets. A sequence database $D$ consists of a list of sequences, $D = \langle s_1, s_2, \ldots, s_p \rangle$, where $s_i$ $(1 \leq i \leq p)$ is a sequence and each sequence has a unique identifier (SID). Each distinct item $i \in I$ has a positive number that represents its external utility and is designated as $iu(i)$. In addition, each item $i$ in a sequence $s_k$ has an internal utility that is represented by a positive value and is designated as $q(i, s_k)$. Similar to HUSRM [43] and US-Rule [20], in this article, we also assume that each sequence can only contain the same item at most once.

As the sequence database illustrated in Table 2, which will be the running example used in this article, there are four sequences with $s_1$, $s_2$, $s_3$, and $s_4$ as their SID, respectively. In Table 3, we can see that the external utility of $a, b, c, d, e, f, g,$ and $h$ is $2, 1, 3, 1, 2, 3, 2,$ and $1$, respectively. For example, in $s_2$, there is an item $(a, 1)$ then we can know that $iu(a) = 2$ and $q(a, s_2) = 1$.

Definition 3.2 (Position and Index of Item). Let $i$ be the $n$th item in sequence $s_k$, $e_m$ be the $n$th itemset in sequence $s_k$. The itemset $e_m$ contains item $i$. Thus, the position of item $i$ is $m$, and the index of item $i$ is $n$.

For example, the positions of items $a, b, e, f, d, c, h$ in the sequence $s_2$ are $1, 1, 2, 2, 3, 4, 5$, respectively, and the indices of items $a, b, e, f, d, c, h$ in the sequence $s_2$ are $1, 2, 3, 4, 5, 6, 7$, respectively.
Definition 3.3 (Totally-ordered Sequential Rule). A totally-ordered sequential rule (ToSR) \( r = X \rightarrow Y \) is defined as a relationship between two sequences \( X \) and \( Y \), where \( X \) is the antecedent of ToSR \( r \) and \( Y \) is the consequent of ToSR \( r \). We also stipulate that \( X \cap Y = \emptyset \), which means any item appears in \( X \) will not occur in \( Y \).

For totally-ordered SRM, a ToSR can be interpreted as under the condition that a sequence \( X \) occurs, what probability of the sequence \( Y \) will occur?

Definition 3.4 (The Size of a Totally-ordered Sequential Rule). The size of a ToSR \( r = X \rightarrow Y \) is denoted as \( k * m \), where \( k \) denotes the number of items that show in the antecedent of \( r \) and \( m \) denotes the number of items that appear in \( r \)’s consequent. Note that \( k * m \) only reveals the length of the antecedent and consequent of \( r \). Moreover, a rule \( r_1 \) with size \( g * h \) is smaller than a rule \( r_2 \) with size \( f * l \) if and only if \( g \leq f \) and \( h < l \), or \( g < f \) and \( h \leq l \).

Considering the rule \( r_3 = \{e, f\} \rightarrow \{c\} \) and \( r_4 = \{e, f\} \rightarrow \{c, b\} \), we can get them from Table 4, as the example, in which the size of \( r_3 \) is \( 2 * 1 \) and \( r_4 \) is \( 2 * 2 \). Thus, it is said that \( r_3 \) is smaller than \( r_4 \).

Definition 3.5 (Sequence/Rule Occurrence). Given two sequences \( s_1 = \{e_1, e_2, \ldots, e_p\} \) and \( s_2 = \{E_1, E_2, \ldots, E_n\} \), it is said that \( s_1 \) occurs in \( s_2 \) (denoted as \( s_1 \subseteq s_2 \)) if and only if \( \exists 1 \leq j_1 < j_2 < \ldots < j_p \leq n \) such that \( e_1 \subseteq E_{j_1}, e_2 \subseteq E_{j_2}, \ldots, e_p \subseteq E_{j_p} \). A rule \( r = X \rightarrow Y \) is said to occur in \( s_2 \) if and only if there exists an integer \( k \) such that \( 1 \leq k < n, X \subseteq \{E_1, E_2, \ldots, E_k\} \) and \( Y \subseteq \{E_{k+1}, \ldots, E_n\} \). In addition, we denote the set of sequences that contain \( r \) as \( \text{seq}(r) \) and the set of sequences that contain the antecedent as \( \text{ant}(r) \).

For example, a ToSR \( r_4 = \{e, f\} \rightarrow \{c, b\} \) occurs in \( s_3 \) and \( s_4 \) and its antecedent occurs in \( s_2 \), \( s_3 \), and \( s_4 \). Therefore, \( \text{seq}(r) \) and \( \text{ant}(r) \) are \{\( s_3, s_4 \)\} and \{\( s_2, s_3, s_4 \)\}, respectively.

Definition 3.6 (Support and Confidence). Let \( r \) be a ToSR and \( D \) be a sequence database. We use the value of \( |\text{seq}(r)| / |D| \) to represent the support value of ToSR \( r \), i.e., \( \text{sup}(r) \). It implies that the number of sequences containing ToSR \( r \) divided by the total number of sequences in \( D \) gives \( r \)’s support value. The confidence of ToSR \( r \) is defined as \( \text{conf}(r) = |\text{seq}(r)|/|\text{ant}(r)| \), which means that the confidence value of ToSR \( r \) equals the number of sequences that \( r \) appear divides by the number of sequences in which the antecedent \( X \) appears.

Definition 3.7 (Utility of an Item/itemset in a Sequence). Given an item \( i \), an itemset \( I \), and a sequence \( s_k \), the utility of an item is equal to its internal utility multiplied by its external utility. Let \( u(i, s_k) \) denote the utility of item \( i \) in sequence \( s_k \) and be defined as \( q(i, s_k) \times iu(i) \). The utility of the itemset \( I \) in the sequence \( s_k \) is designated as \( u(I,s_k) \) and defined as \( u(I,s_k) = \sum_{i \in I} q(i,s_k) \times iu(i) \).

Definition 3.8 (Utility of a Totally-ordered Sequential Rule in a Sequence). Let \( r \) be a ToSR and \( s_k \) be a sequence. We use \( u(r,s_k) \) to represent the utility of ToSR \( r \) in sequence \( s_k \). Then, \( u(r,s_k) \) is defined as \( u(r,s_k) = \sum_{i \in r \land s_k \subseteq \text{seq}(r)} q(i,s_k) \times iu(i) \).

Definition 3.9 (Utility of a Totally-ordered Sequential Rule in a Database). Given a ToSR \( r \) and a sequence database \( D \), we use \( u(r) \) to denote the utility of ToSR \( r \) in the sequence database \( D \). Then \( u(r) \) is defined as \( u(r) = \sum_{s_k \in \text{seq}(r) \land s_k \subseteq D} u(r,s_k) \).

| Item | a | b | c | d | e | f | g | h |
|------|---|---|---|---|---|---|---|---|
| Unit utility | 2 | 1 | 3 | 1 | 2 | 3 | 2 | 1 |

Table 3. External Utility Table
Table 4. HTSRs in Table 2 when \( \text{minutil} = 25 \) and \( \text{minconf} = 0.5 \)

| ID | HTSR                        | Support | Confidence | Utility |
|----|-----------------------------|---------|------------|---------|
| r₁ | \(<e, f>, c> \rightarrow <b>\) | 0.5     | 0.67       | 28      |
| r₂ | \(<e> \rightarrow <c>\)      | 0.75    | 1.0        | 25      |
| r₃ | \(<e, f> \rightarrow <c>\)    | 0.75    | 1.0        | 34      |
| r₄ | \(<e, f> \rightarrow <c, b>\)| 0.5     | 0.67       | 28      |

For example, a ToSR \( r₄ = \langle e, f \rangle \rightarrow <c, b> \) occurs in \( s₃ \) and \( s₄ \), and its \( \text{seq}(r₄) = \{s₃, s₄\} \) and \( \text{ant}(r₄) = \{s₂, s₃, s₄\} \). Thus, the support value of rule \( r₄ \) is \( \text{sup}(r₄) = |\text{seq}(r₄)| / |\mathcal{D}| = 2 / 4 = 0.5 \), and the confidence value of rule \( r₄ \) is \( \text{conf}(r₄) = |\text{seq}(r₄)| / |\text{ant}(r₄)| = 2 / 3 = 0.67 \). The utility of item \( e \) in sequence \( s₃ \) is \( u(e, s₃) = q(e, s₃) \times iu(e) = 2 \times 2 = 4 \) and the utility of rule \( r₄ \) in sequence \( s₃ \) is \( u(r₄, s₃) = \sum_{i \in r₄ \land s₃ \subseteq \text{seq}(r₄)} q(i, s₃) \times iu(i) = 2 \times 2 + 1 \times 3 + 3 \times 3 + 1 \times 1 = 4 + 3 + 9 + 1 = 17 \). Correspondingly, the utility of rule \( r₄ \) in sequence \( s₄ \) is \( u(r₄, s₄) = 11 \). Therefore, the utility of rule \( r₄ \) is \( u(r₄) = \sum_{sₖ \in \text{seq}(r₄)} \left( u(r₄, sₖ) \right) = u(r₄, s₃) + u(r₄, s₄) = 17 + 11 = 28 \).

### 3.2 Problem Description

**Definition 3.10 (High-utility Totally-ordered Sequential Rule Mining).** Given a sequence database \( \mathcal{D} \), a positive minimum utility threshold \( \text{minutil} \) and a minimum confidence threshold \( \text{minconf} \) between 0 and 1, a ToSR is called **high-utility totally-ordered sequential rule (HTSR)** if and only if it satisfies both the minimum utility and confidence thresholds simultaneously, i.e., \( u(r) \geq \text{minutil} \) and \( \text{conf}(r) \geq \text{minconf} \). Thus, the problem of high-utility totally-ordered sequential rule mining is to identify and output all ToSRs that satisfy both the conditions of \( \text{minutil} \) and \( \text{minconf} \). For high-utility totally-ordered sequential rule mining, the HTSR can be viewed as under the condition of antecedent \( X \) occurs what the probability of consequent \( Y \) appears and what the profit of the entire rule is? Note that, in this article, we use ToSR to represent the candidate HTSR, and \( \text{minutil} \) can be absolute or relative minimum utility threshold.

For example, if we specify that \( \text{minutil} = 25 \) and \( \text{minconf} = 0.5 \), then we will discover four HTSRs, shown in Table 4. From the result, we can find that two ToSRs \( \langle a, b \rangle \rightarrow <c, d> \) and \( \langle a, b \rangle \rightarrow <d, c> \) with utility 15 and 10, respectively. Both are low-utility and will not be output. Using a partially-ordered SRM algorithm, the two ToSRs will be merged as high-utility SR \( \{a, b\} \rightarrow \{c, d\} \) with a utility of 25. However, it is not committed to reality.

**Definition 3.11 (I-expansion and S-expansion).** Let \( s = \langle e₁, e₂, \ldots, e_k \rangle \) be a sequence and \( i \in I \) be an item. The I-expansion is defined as \( \langle e₁, e₂, \ldots, e_k \cup \{i\} \rangle \), where \( i \) should be occurring simultaneously with items in itemset \( e_k \) and greater than the items in \( e_k \) according to the \( \text{lex} \). Given a sequence \( s = \langle e₁, e₂, \ldots, e_k \rangle \) and an item \( i \in I \), the S-expansion is defined as \( \langle e₁, e₂, \ldots, e_k, \{i\} \rangle \), where \( i \) should be occurring after the items in itemset \( e_k \) of a sequence.

**Definition 3.12 (The Expansion of a Totally-ordered Sequential Rule).** Similar to RuleGrowth [11], in this article, TotalSR implements left and right expansion to grow a ToSR. Given a rule \( r = X \rightarrow Y \), where \( X = \langle e₁, e₂, \ldots, e_k \rangle \) and \( Y = \langle e_m, e_{m+1}, \ldots, e_n \rangle \) \((k < m \leq n)\), the left expansion is defined as \( \langle e₁, e₂, \ldots, e_k \rangle \circ i \rightarrow \langle e_m, e_{m+1}, \ldots, e_n \rangle \), where \( \circ \) represents the expansion can be both I-expansion and S-expansion and item \( i \) should not be in \( Y \), i.e., \( i \notin Y \). Correspondingly, the right expansion is defined as \( \langle e₁, e₂, \ldots, e_k \rangle \rightarrow \langle e_m, e_{m+1}, \ldots, e_n \rangle \circ i \), where \( \circ \) represents the expansion can be both I-expansion and S-expansion and item \( i \notin X \).

A ToSR can be formed by first performing a left expansion and then a right expansion or performing a right expansion and then a left expansion. To avoid generating the same rules, in TotalSR,
unlike References [20, 43], we stipulate that a ToSR cannot implement the left expansion after it performs a right expansion, which means a left-first expansion.

4 THE PROPOSED ALGORITHM

In this section, we will present the algorithm of TotalSR. To discover HTSRs efficiently, we will first introduce the utility upper bounds and pruning strategies. Then the data structures are presented for fast calculating the information such as utility upper bounds, utility, and confidence. Third, the pseudocode of TotalSR is given. Finally, the computational complexity of the algorithm TotalSR is analyzed.

4.1 Upper Bounds and Pruning Strategies

In this article, we also use some extraordinary techniques that are utilized in the utility-driven SRM field. Sequence estimated utility (SEU) can help us to only keep useful items in the database, which can be referred to Reference [43] to get a detailed description. SEU is the same as the upper bound SWU used in HUSPM [36]. The prefix extension utility (PEU) and reduced sequence utility (RSU) are from References [16, 33, 36]. Although US-Rule [20] designed some new utility upper bounds based on PEU and RSU, the definition of these utility upper bounds is redundant. Thus, in this article, we directly use PEU and RSU and add a condition to them to make they can adapt for TotalSR. In addition to the existing utility upper bounds, we introduce a novel utility upper bound: reduced sequence prefix extension utility (RSPEU), which does not consider the invalid items between expansion position and expansion item. Thus, RSPEU is tighter than PEU and RSU. Based on the utility upper bounds, we designed several pruning strategies. Since there are two measures, utility and confidence, to evaluate a candidate HTSR, we also proposed a confidence pruning strategy based on left-first expansion.

**Definition 4.1 (Sequence Estimated Utility of Item/ToSR).** Let a be an item and \( \mathcal{D} \) be a sequence database. The sequence estimated utility (SEU) of an item \( a \) is designated as \( \text{SEU}(a) \) and defined as \( \text{SEU}(a) = \sum_{i \in \mathcal{S}_k \wedge i \in \mathcal{S}_k \wedge a \in \mathcal{D}} u(i, s_k) \), where \( s_k \) is the sequence that contains the item \( a \) and \( i \) is the item that appears in sequence \( s_k \). Note that \( i \) can be any item in the sequence \( s_k \). Correspondingly, given a ToSR \( r \), the sequence estimated utility of \( r \) is denoted as \( \text{SEU}(r) \) and defined as \( \text{SEU}(r) = \sum_{i \in \mathcal{S}_k \wedge i \in \text{seq}(r)} u(i, s_k) \), where \( i \) is the item that occurs in the sequence \( s_k \).

**Strategy 1 (Unpromising Items Pruning Strategy).** Given a sequence database \( \mathcal{D} \), TotalSR will remove all unpromising items from \( \mathcal{D} \). For an unpromising item \( i \), the SEU of \( i \) is smaller than \( \text{minutil} \). As a result, the utility of any ToSR that contains the item \( i \) will not exceed the \( \text{minutil} \). In other words, the unpromising item \( i \) will not be contained in an HTSR, which means the item \( i \) is useless for HTSRs. We can remove the item from \( \mathcal{D} \) directly.

In Table 2, for instance, if we set \( \text{minutil} = 25 \), the SEU of \( a \) is \( \text{SEU}(a) = \sum_{a \in \mathcal{S}_k \wedge i \in \mathcal{S}_k \wedge a \in \mathcal{D}} u(i, s_k) = \sum_{a \in \mathcal{S}_k \wedge i \in \mathcal{S}_k \wedge a \in \mathcal{D}} u(i, s_1) + \sum_{a \in \mathcal{S}_k \wedge i \in \mathcal{S}_k \wedge a \in \mathcal{D}} u(i, s_2) = 21 + 16 = 37 \), then item \( a \) is a promising item. However, the SEU of item \( h \) is \( \text{SEU}(h) = \sum_{h \in \mathcal{S}_k \wedge i \in \mathcal{S}_k \wedge a \in \mathcal{D}} u(i, s_k) = \sum_{h \in \mathcal{S}_k \wedge i \in \mathcal{S}_k \wedge a \in \mathcal{D}} u(i, s_1) = 16 \). Thus, there is an unpromising item \( h \) and we can delete item \( h \) from \( \mathcal{D} \).

**Upper Bound 1 (Prefix Extension Utility).** We use \( \text{PEU}(r, s) \) to represent \( \text{PEU} \) of a ToSR \( r \) in sequence \( s \). \( \text{PEU}(r, s) \) is defined as:

\[
\text{PEU}(r, s) = \begin{cases} 
  u(r, s) + \text{ERU}(r, s), & \text{ERU}(r, s) > 0 \\
  0, & \text{otherwise}.
\end{cases}
\]

Note that \( \text{ERU} \) means extendable remaining utility, and the value of \( \text{ERU}(r, s) \) depends on the type of expansion. For example, consider the HTSR \( r_3 = \langle e, f \rangle \rightarrow \langle c \rangle \) in Table 4. If \( r_3 \)
performs a left expansion, then $ERU(r_3, s_2)$ equals 2, since there is only one extendable item $d$ that can be extended into the antecedent of $r_3$ in the sequence $s_2$. If $r_3$ performs a right expansion, then $ERU(r_3, s_4)$ equals 10, which is equal to the sum of utility of the items $d$, $g$, and $b$, since only these items can be extended into consequent of $r_3$ in $s_4$.

We use $PEU(r)$ to denote the $PEU$ of a ToSR $r$ in $D$. Then, $PEU(r)$ is defined as:

$$PEU(r) = \sum_{s_k \in \text{seq}(r) \setminus \text{seq}(r) \subseteq D} PEU(r, s_k).$$

**Theorem 4.2.** Given a ToSR $r = X \rightarrow Y$ and the other ToSR $r'$, where $r'$ is extended from $r$ by performing $I$- or $S$-expansion, we have $u(r') \leq PEU(r)$.

**Proof.** Given an item $i$, a ToSR $r$, and a sequence $s$, where $i$ can be extended into $r$ to form the other ToSR $r'$, according to the definition of $ERU(r, s)$ in upper bound $PEU$, we have $u(i, s) \leq ERU(r, s)$. Then, we have $u(r', s) = u(r, s) + u(i, s) \leq u(r, s) + ERU(r, s) = PEU(r, s)$. Therefore, $u(r') \leq PEU(r)$. \hfill $\square$

For brevity, we use $LEPEU(r)$ to represent the $PEU(r)$ of left expansion and $REPEU(r)$ to represent the $PEU(r)$ of right expansion.

In Table 4, for instance, consider HTSR $r_3 = \{c, f\} \rightarrow \{c\}$. There are three sequences, $s_2$, $s_3$, and $s_4$, that contain $r_3$. In sequence $s_2$, item $d$ can be extended into the antecedent of $r_3$, so $LEPEU(r_3, s_2) = 18$. However, in $s_4$, there is no item to be extended. Therefore, $LEPEU(r_3, s_4) = 0$. Finally, $LEPEU(r_3) = LEPEU(r_3, s_2) + LEPEU(r_3, s_3) + LEPEU(r_3, s_4) = 10 + 18 + 0 = 28$. In sequence $s_2$, there is no item to be extended into the consequent of $r_3$ (note that item $h$ has been removed from $D$, since it is an unpromising item). Thus, $REPEU(r_3, s_2) = 0$. In sequence $s_3$, there is an item $b$ that can be extended into the consequent of rule $r_3$. Thus, $REPEU(r_3, s_3) = 17$. In $s_4$, $REPEU(r_3, s_4) = 20$. Finally, $REPEU(r_3) = REPEU(r_3, s_2) + REPEU(r_3, s_3) + REPEU(r_3, s_4) = 0 + 17 + 20 = 37$.

**Upper bound 2 (Reduced Sequence Utility).** Let $\xi$ be a ToSR that can extend an item $w$ to form the other ToSR $r$. In a sequence $s$, we use $RSU(r, s)$ to represent the reduced sequence utility (RSU) of $r$. $RSU(r, s)$ is defined as:

$$RSU(r, s) = \begin{cases} PEU(\xi, s), & s \in \text{seq}(r) \\ 0, & \text{otherwise}. \end{cases}$$

Correspondingly, we use $RSU(r)$ to present the $RSU$ of $r$ in $D$. Then, $RSU(r)$ is defined as:

$$RSU(r) = \sum_{s \in D} RSU(r, s).$$

**Theorem 4.3.** Given a ToSR $r = X \rightarrow Y$ and the other ToSR $r'$, where $r'$ is extended with item $i$ from $r$ by performing $I$- or $S$-expansion, we have $u(r') \leq RSU(r)$.

**Proof.** Let $s$ be a sequence, $r$ be a ToSR generated from ToSR $\xi$ and $i$ be an item that can be extended into $r$ to form a rule $r'$. According to the definition of upper bound $RSU$ and theorem 4.2, each sequence $s \in \text{seq}(r')$, we have $u(r', s) \leq PEU(r, s)$ and $PEU(r, s) \leq PEU(\xi, s) = RSU(r, s)$. Thus, $u(r', s) \leq RSU(r, s)$. Finally, $\sum_{s \in D} u(r', s) \leq \sum_{s \in D} RSU(r, s)$, i.e., $u(r') \leq RSU(r)$. \hfill $\square$

Since when we use $RSU$, it just utilizes the corresponding value of $PEU$ of the sequences that can extend with a specific item $i$. However, there is a little useless utility in the remaining utility. For example, if we extend the item $g$ in the $s_4$ into $r_3$, then the item $d$ that is between item $c$ and $g$ is invalid for the subsequent expansions. Thus, we design a tighter utility upper bound $RSPEU$,  

ACM Transactions on Knowledge Discovery from Data, Vol. 18, No. 4, Article 80. Publication date: February 2024.
which only utilizes the real remaining utility. Because when we extend an item \( i \) into a ToSR, the items before \( i \) are useless for subsequent expansion.

**Upper bound 3 (Reduced Sequence Prefix Extension Utility).** For a ToSR \( r \) that can implement an expansion with an item \( i \) to generate the other ToSR \( r' \), \( RSPEU(r', s) \) denotes the reduced sequence prefix extension utility (RSPEU) of \( r' \) in sequence \( s \). \( RSPEU(r', s) \) is defined as:

\[
RSPEU(r', s) = \begin{cases} 
  u(r, s) + RU(r, i, s), & s \in \text{seq}(r') \\
  0, & \text{otherwise}.
\end{cases}
\]

Where \( RU(r, i, s) \) implies the total utility from item \( i \) to the last extendable item in sequence \( s \) that can be extended into \( r \). Note that the last extendable item in a sequence \( s \) that can be extended into \( r \) is not the end item of the sequence \( s \). The last extendable item depends on the type of expansion.

Correspondingly, we use \( RSPEU(r') \) to signify the \( RSPEU \) of a ToSR \( r' \) in \( D \). Accordingly, \( RSPEU(r') \) is defined as:

\[
RSPEU(r') = \sum_{s_k \in \text{seq}(r') \land s_k \subseteq D} RSPEU(r', s_k).
\]

**Theorem 4.4.** Given a ToSR \( r = X \rightarrow Y \) and the other ToSR \( r' \), where \( r' \) is extended with an item \( i \) from \( r \) by performing I- or S-expansion, we have \( u(r') \leq RSPEU(r') \).

**Proof.** Let \( s \) be a sequence, \( r \) be a ToSR and \( i \) be an item that can be extended into \( r \) to form rule \( r' \). According to the definition of \( RU(r, i, s) \) in upper bound \( RSPEU \), we have \( u(i, s) \leq RU(r, i, s) \). Then, we have \( u(r', s) = u(r, s) + u(i, s) \leq u(r, s) + RU(r, i, s) = RSPEU(r', s) \). Therefore, \( u(r') \leq RSPEU(r') \).

We use \( LERSPEU \) and \( RERSPEU \) to distinguish the \( RSPEU \) of left expansion and right expansion.

For example, for \( LERSPEU \), consider the HTSR \( r_1 = \{e, f\}, c \rightarrow \{b\} \) in Table 4 and a ToSR \( r = \{e, f\} \rightarrow \{b\} \). HTSR \( r_1 \) can be generated from \( r \) by extending an item \( c \) to its antecedent and \( \text{seq}(r_1) = \{s_3, s_4\} \). In \( s_3 \) there exists a useless item \( g \). Thus, \( LERSPEU(r_1, s_3) = 8 + 9 = 17 \). In \( s_4 \), there is no useless item, so \( LERSPEU(r_1, s_4) = 8 + 12 = 20 \). In total, \( LERSPEU(r_1) = LERSPEU(r_1, s_3) + LERSPEU(r_1, s_4) = 17 + 20 = 37 \).

For \( RERSPEU \), consider the HTSR \( r_3 = \{e, f\} \rightarrow \{c\} \) and \( r_4 = \{e, f\} \rightarrow \{c, b\} \) in Table 4. HTSR \( r_3 \) can be generated from \( r_4 \) by extending an item \( b \) to its consequent and \( \text{seq}(r_4) = \{s_3, s_4\} \). In \( s_3 \) there is no useless item. Thus, \( RERSPEU(r_4, s_3) = 16 + 1 = 17 \). However, in \( s_4 \), there are two items, \( d \) and \( g \), before item \( b \), i.e., they are useless. Therefore, \( RERSPEU(r_4, s_4) = 10 + 1 = 11 \). In total, \( RERSPEU(r_4) = RERSPEU(r_4, s_3) + RERSPEU(r_4, s_4) = 17 + 11 = 28 \).

Since, in utility-driven mining, there is no ideal anti-monotonic property, to avoid the severe combinatorial explosion problem of the search space when we set a lower \( \text{minutil} \), we used several utility upper bound pruning strategies to cope with the combinatorial explosion problem. In addition, since confidence value is not related to utility, it is only generated from support value. In other words, the confidence value of a ToSR satisfies the anti-monotonic property. Thus, we can design a pruning strategy based on confidence. But it requires a left-first expansion to ensure that the anti-monotonic property can work correctly. The pruning strategies are given below.

**Strategy 2 (Prefix Extension Utility Pruning Strategy).** Given a ToSR \( r \), according to the Theorem 4.2, when \( r \) implements an expansion to form the other ToSR \( r' \), the utility of \( r' \) will not exceed the upper bound \( PEU(r) \), i.e., \( u(r') \leq PEU(r) \). If \( PEU(r) < \text{minutil} \), then we can know that any ToSR extending from \( r \) will not be an HTSR, i.e., \( u(r') < \text{minutil} \). Thus, we can stop further expansion.
Strategy 3 (Reduced Sequence Prefix Extension Utility Pruning Strategy). Given a ToSR $r$, according to Theorem 4.4, when $r$ implements an expansion with a specific item $i$ to generate a ToSR $r'$, the utility of $r'$ will not exceed the upper bound $RSPEU$, i.e., $u(r') \leq RSPEU(r')$. If $RSPEU(r') < \minutil$, then we can know that any ToSR extended from $r$ cannot be an HTSR. Thus, we can stop the subsequent expansion.

Since we stipulate that a left expansion will not follow a right expansion to avoid generating the same ToSR more times, this expansion mode satisfies the anti-monotonic property. We give a corresponding proof, shown later. Note that this expansion style is different from HUSRM [43], US-Rule [20] and the other SRM algorithms like TRuleGrowth [11], where they use a right-first expansion to avoid the repeated generation of a rule.

Proof. Let $r = X \rightarrow Y$ be a ToSR. Since we stipulate that a left expansion will not follow a right expansion, the support value of the sequence $X$ is fixed when we perform a right expansion. In this case, the support of the entire ToSR $r$ still satisfies the anti-monotonic property, i.e., the support value of $r$ will remain constant or decrease. Thus, we can utilize this property to prune the search space. However, if we use a right-first expansion, then the support value of sequence $X$ varies, as does the support value of rule $r$. Therefore, the confidence value can get smaller, the same, or greater, i.e., it is unknown. That is why we begin with a left-first expansion. But when we perform a left expansion, the confidence value of a ToSR is unknown. Therefore, we can only use the anti-monotonic property to prune when we perform the right expansion.

Strategy 4 (Confidence Pruning Strategy). Given a ToSR $r$, when $r$ implements a right expansion with a specific item $i$ to generate the other ToSR $r'$, then the confidence value of $r'$ will be less or equal to $r$’s confidence value, i.e., $\text{conf}(r') \leq \text{conf}(r)$. If $\text{conf}(r) < \minconf$, then we have $\text{conf}(r') < \minconf$, too. Thus, we can stop extending further.

4.2 Data structures

For mining rules effectively and efficiently, in HUSRM [43] and US-Rule [20], they proposed a data structure called a utility table, which can maintain the necessary information about the candidate rules. In this article, we redesigned the utility table to make it adapt to TotalSR. The utility table only keeps the projected sequence regarding the ToSR. We can not know the support of the antecedent. In HUSRM [43] and US-Rule [20], they used bit vectors to efficiently calculate the support of the antecedent. In TotalSR, we introduce an auxiliary antecedent record table (ART) to count the sequences that only contain the antecedent of a ToSR. Besides, we also propose a data structure called utility prefix sum list (UPSL) for fast calculating LEPEU, REPEU, LERSPEU, and RERSPEU of a ToSR.

Definition 4.5 (LE-utility Table in $D$). Given a ToSR $r$ and a sequence database $D$, the LE-utility table contains six domains: $\langle\text{SID}, \text{Utility}, \text{LEPEU}, \text{REPEU}, \text{Positions}, \text{Indices}\rangle$. SID is the sequence identifier that contains $r$. Utility is the utility of $r$ in a sequence. LEPEU means the left expansion $\text{PEU}$ of $r$. REPEU is the right expansion $\text{PEU}$ of $r$. Positions is a $3$-tuple $(\alpha, \beta, \gamma)$, where $\alpha$ is the last item’s position of antecedent of $r$, $\beta$ is the first item’s position of consequent of $r$, and $\gamma$ is the last item’s position of consequent of $r$. Indices is a $2$-tuple $(\alpha’, \gamma’)$, where $\alpha’$ is the index of the first item that can be extended into antecedent and $\gamma’$ is the index of last item in a sequence.

For example, Table 5 shows the complete LE-utility table of rule $r_1$. For the second entry, we have $u(r_1, s_4) = 11$, $\text{LEPEU}(r_1, s_4) = 20$, $\text{REPEU}(r_1, s_4) = 0$. In $s_4$ the position of item $c$ is 2 and $b$ is 4, so $\text{Positions} = (2, 4, 4)$. Besides, $\text{Indices} = (4, 6)$. Thus, the total entry is $\langle s_4, 11, 20, 0, (2, 4, 4), (4, 6)\rangle$. 

ACM Transactions on Knowledge Discovery from Data, Vol. 18, No. 4, Article 80. Publication date: February 2024.
Note that we stipulate a left-first expansion, i.e., the right expansion will follow the left expansion. Thus, we should record the \( \text{REPEU} \) for the right expansion in \( \text{LE-utility table} \). According to the \( \text{LE-utility table} \), we only know the support of \( \text{ToSR} \). The support of the antecedent is unknown. Thus, we cannot directly derive the confidence of a \( \text{ToSR} \). Unlike \( \text{HUSRM} \) \cite{43} and \( \text{US-Rule} \) \cite{20}, they can use bit vectors to compute the support of the antecedent of a rule, as they do not care about the order of items in the antecedent. In \( \text{TotalSR} \), we use an auxiliary \( \text{ART} \) to record the sequences that only contain the antecedent.

Definition 4.6 (Auxiliary Antecedent Record Table). Given a \( \text{ToSR} \) \( r = X \rightarrow Y \) and a sequence database \( \mathcal{D} \), the auxiliary antecedent record table of \( r \) is defined as \( \text{ART}(r) = \{ \text{key: value} \} \), where the key is the antecedent of \( r \), i.e., \( X \), and value is the set of sequences that only contain the antecedent of \( r \) in \( \mathcal{D} \). Note that the auxiliary antecedent record table only records the sequences that cannot form a \( \text{ToSR} \).

For example, consider the \( \text{HTSR} \) \( r_1 = \langle \{e, f\}, c \rangle \rightarrow \langle b \rangle \) in Table 4. The sequence \( s_2 \) only contains the antecedent of \( r_1 \). Thus, \( \text{ART}(r_1) = \{ \langle \{e, f\}, c \rangle \rightarrow \langle s_2 \rangle \} \). With the help of \( \text{LE-utility table} \) and \( \text{ART} \), we can quickly calculate the support of the antecedent.

Definition 4.7 (\( \text{RE-utility Table} \) in \( \mathcal{D} \)). Given a \( \text{ToSR} \) \( r \) and a sequence database \( \mathcal{D} \), the \( \text{RE-utility table} \) contains five domains: \( \langle \text{SID}, \text{Utility}, \text{REPEU}, \text{Position}, \text{Indices} \rangle \). \( \text{SID} \) is the sequence identifier that contains \( r \). \( \text{Utility} \) is the utility of \( r \) in a sequence. \( \text{REPEU} \) is the right expansion \( \text{PEU} \) of \( r \) in a specific sequence. \( \text{Position} \) is the last item’s position of consequent of \( \text{ToSR} \) \( r \). \( \text{Index} \) is the index of the last item in a sequence.

Note that we stipulate a left-first expansion, i.e., the left expansion will not be conducted after the right expansion. Therefore, some information, like \( \text{LEPEU} \), is not necessary to be recorded in \( \text{RE-utility table} \). Besides, when the right expansion is executed, the support of the antecedent is fixed. So, we do not need to maintain and update \( \text{ART} \).

Definition 4.8 (Utility Prefix Sum List in a Sequence). Given a sequence \( s \) with the largest index equals to \( k \), the utility prefix sum list of the sequence \( s \) is denoted as \( \text{UPSL}(s) \) and defined as \( \text{UPSL}(s) = \langle u_{s1}, u_{s2}, \ldots, u_{sk} \rangle \), where \( u_{si} \) \((1 \leq i \leq k)\) is the utility prefix sum of the first \( i \) items in the sequence \( s \).

As an example, Table 6 shows the \( \text{UPSL} \) of sequence \( s_4 \). We can calculate the \( \text{LEPEU}(r_1, s_4) = u(r_1, s_4) + u_{s5} - u_{s3} = 11 + 19 - 10 = 20 \). In general, we will scan \( \mathcal{D} \) once after we remove all unpromising items from \( \mathcal{D} \) to get the \( \text{UPSL} \) of each sequence. With the help of \( \text{UPSL} \) of a sequence, we can compute the \( \text{LEPEU} \) and \( \text{REPEU} \) in \( O(1) \) time, which in the past will cost \( O(k) \) and \( O(l) \) time, respectively, where \( k \) and \( l \) are the average numbers of the items that can extend to antecedent and consequent, respectively. Similarly, \( \text{LERSPEU} \) and \( \text{RERSPEU} \) can be computed in \( O(1) \) time, too.

### 4.3 \( \text{TotalSR Algorithm} \)

Based on the pruning strategies and the data structures mentioned above, the high-utility totally-ordered sequential rule mining algorithm, \( \text{TotalSR} \), is proposed in this subsection. To avoid generating a rule twice and applying the confidence pruning strategy, we designed a left-first expansion procedure. The main pseudocode of the \( \text{TotalSR} \) algorithm is shown in Algorithms 1, 2, and 3.
Table 6. The UPSL of Sequence $s_4$ in Table 2

| item | e | f | c | d | g | b |
|------|---|---|---|---|---|---|
| index| 1 | 2 | 3 | 4 | 5 | 6 |
| $u_s$index| 4 | 7 | 10 | 13 | 19 | 20 |

Algorithm 1: TotalSR algorithm

**Input:** $D$: a sequence database, $\text{minconf}$: the minimum confidence threshold, $\text{minutil}$: the minimum utility threshold.

**Output:** HTSRs: all high-utility totally-ordered sequential rules.

1. scan $D$ to compute $\text{SEU}(i)$ for each item $i$ in $D$;
2. delete all unpromising items from $D$;
3. scan $D$ to identify all antecedent items set $A$ with size 1 and compute UPSL;
4. for $a \in A$ do
   5. scan $D$ to identify all consequent items set $C$ corresponding to $a$;
   6. for $c \in C$ do
      7. generate candidate ToSR $r = a \rightarrow c$, create its utility table $UT(r)$ and ART($r$);
      8. scan $UT(r)$ to compute $u(r)$ and $\text{conf}(r)$;
      9. if $u(r) \geq \text{minutil}$ and $\text{conf}(r) \geq \text{minconf}$ then
         10. update HTSRs $\leftarrow$ HTSRs $\cup$ $r$;
      end
      11. if $\text{LEPEU}(r) + \text{REPEU}(r) - u(r) \geq \text{minutil}$ then
         12. call ruleGrowth($r$, $UT(r)$, ART($r$), $sup$);
      end
   9. end
end
end

Algorithm 2: The ruleGrowth procedure

**Input:** $r$: a ToSR, $UT(r)$: utility table of $r$, ART($r$): $r$’s ART, $sup$: the support of antecedent.

1. scan $UT(r)$ to get all items set $A$ and all items set $C$ that can be extended into antecedent and consequent, respectively;
2. for $a \in A$ do
   3. $t \leftarrow a$ extended into $r$’s antecedent;
   4. call ruleJudge($t$, $UT(r)$, ART($r$), $sup$);
end
   5. for $c \in C$ do
      6. $t \leftarrow c$ extended into $r$’s consequent;
      7. call ruleJudge($t$, $UT(r)$, ART($r$), $sup$);
end
end

TotalSR takes a sequence database $D$, a minimum confidence threshold ($\text{minconf}$), and a minimum utility threshold ($\text{minutil}$) as its inputs and then outputs all high-utility totally-ordered sequential rules. TotalSR first scans $D$ to compute the $\text{SEU}$ of all items in $D$ and then deletes the unpromising item from $D$ (Lines 1–2). After that, TotalSR scans $D$ again to identify all items set
Algorithm 3: The ruleJudge procedure

Input: a ToSR extended from \( r \), \( UT(r) \): utility table of \( r \), \( ART(r) \): \( r \)'s \( ART \), \( sup \): the support of antecedent.
1. scan \( UT(r) \) to calculate \( RSPEU(t) \), \( u(t) \), and \( conf(t) \);
2. if \( u(t) \geq minutil \) and \( conf(t) \geq minconf \) then
   3. update HTSRs \( \leftarrow \) HTSRs \( \cup t \);
   4. end if
5. if \( RSPEU(t) \geq minutil \) then
   6. construct \( ART(t) \) and utility table of \( t \): \( UT(t) \);
   7. if \( sup = 0 \) and \( LEPEU(t) + REPEU(t) - u(t) \geq minutil \) then
      8. call \( ruleGrowth(t, UT(t), ART(t), 0) \);
      9. if \( conf(t) \geq minconf \) and \( REPEU(t) \geq minutil \) then
         10. call \( ruleGrowth(t, UT(t), null, ART(t).length + UT(t).length) \);
   11. end if
   12. end if
13. if \( sup \neq 0 \) and \( conf(t) \geq minconf \) and \( REPEU(t) \geq minutil \) then
   14. call \( ruleGrowth(t, UT(t), null, sup) \);
   15. end if
16. end if
17. end

A that can become the antecedent of a ToSR and compute the UPSL of the processed \( D \) (Line 3). Next, for each item \( a \in A \), TotalSR will scan \( D \) to obtain all consequent items set \( C \) corresponding to item \( a \) (Lines 4–5). For each item \( c \in C \), TotalSR forms candidate ToSR \( r = a \rightarrow c \), creates \( r \)'s utility table \( UT(r) \) and \( ART(r) \). Then, TotalSR will scan \( UT(r) \) to compute \( u(r) \) and \( conf(r) \) (Lines 6–8). If ToSR \( r \) satisfies \( minutil \) and \( minconf \), then TotalSR will update the HTSRs set with \( r \) (Lines 9–11). If \( LEPEU(r) + REPEU(r) - u(r) \) exceeds \( minutil \), then TotalSR will implement ruleGrowth to extend the antecedent. Note that TotalSR will perform right expansion after left expansion. Thus, it is necessary to add \( REPEU(r) \). Because both \( LEPEU(r) \) and \( REPEU(r) \) include \( u(r) \), we should subtract \( u(r) \) one time to get the correct utility upper bound (Lines 12–14). Subsequently, according to the confidence pruning strategy, if \( conf(r) \geq minconf \) and \( REPEU(r) \geq minutil \), then TotalSR will call ruleGrowth to extend consequent (Lines 15–17).

In Algorithm 2, i.e., ruleGrowth, TotalSR will grow the ToSR. The ruleGrowth procedure takes a ToSR \( r \), a utility table \( UT(r) \) of ToSR \( r \), \( r \)'s \( ART \), and the support of antecedent (\( sup \)) as its input. First, ruleGrowth will scan \( UT(r) \) to get all items set \( A \) and all items set \( C \) that can be extended into antecedent and consequent, respectively (Line 1). Then for each \( a \in A \), ruleGrowth will extend \( a \) into \( r \)'s antecedent to get a candidate ToSR \( t \) and call ruleJudge procedure to determine whether \( t \) is a HTSR or a promising HTSR (Lines 2–5). Similarly, for each item \( c \in C \), ruleGrowth will extend \( c \) into \( r \)'s consequent to get a candidate ToSR \( t \) and call ruleJudge procedure to determine whether \( t \) is a HTSR or a promising HTSR (Lines 6–9).

In Algorithm 3, i.e., ruleJudge, TotalSR will update the HTSRs set and call ruleGrowth to extend the rule further. The ruleJudge procedure takes a ToSR \( t \) that extended from \( r \), a utility table \( UT(r) \) of \( r \), \( r \)'s \( ART \), and the support of antecedent (\( sup \)) as its input. The ruleJudge procedure will first scan \( UT(r) \) to calculate \( RSPEU(t) \), \( u(t) \), and \( conf(t) \) (Line 1). Note that \( RSPEU(t) \) can be \( LERSPEU(t) + RERSPEU(t) - u(t) \) or \( RERSPEU(t) \) according to the extending type of \( t \). Then, if both utility and confidence exceed the thresholds, ruleJudge will update the HTSRs with \( t \) (Lines 2–4). Subsequently, if \( RSPEU(t) \geq minutil \), then ruleJudge will construct \( ART(t) \) and the utility table \( UT(t) \) of \( t \) (Lines 5–6). Then, if \( sup \) equals 0, which means \( t \) is generated from \( r \) by performing a left
expansion, and $\text{LEPEU}(t) + \text{REPEU}(t) - u(t) \geq \minutil$, a ruleGrowth will be called for left expansion (Lines 7–8). Next if $\text{conf}(t) \geq \minconf$ and $\text{REPEU}(t) \geq \minutil$, then the other ruleGrowth will be implemented for right expansion (Lines 9–11). Finally, if $\text{sup}$ is greater than 0, then $\text{conf}(t) \geq \minconf$, and $\text{REPEU}(t) \geq \minutil$, ruleJudge will call ruleGrowth for right expansion (Lines 13–15). Note that we determine the type of expansion based on the value of $\text{sup}$. When $\text{sup}$ is greater than 0, ruleGrowth is a right expansion, otherwise ruleGrowth is a left expansion.

4.4 Computational Complexity Analysis

Assume that there are $|D|$ sequences in database $D$. We use $L$ to represent the longest length of the sequence in $D$ and $I$ to be the distinct items appearing in $D$. TotalSR first scans the database to delete the unpromising items and construct the UPSL data structure (Algorithm 1, Lines 1, 3). Both the time complexity and space complexity are $O(|D|L)$. Because there are $|D|$ sequences and the longest length of the sequence is $L$, it needs to search the whole database to identify all items that can become antecedent or consequent. Identifying all antecedent items in $D$ will cost $O(|D|L)$ time complexity (Algorithm 1, Line 3). The time complexity of identifying the consequent items is also $O(|D|L)$ (Algorithm 1, Line 5). The space complexity is $O(I)$, since there are at most $O(I)$ items that can be maintained in the memory. The first loop also runs at most $O(I)$ times. The time complexity of creating a utility table for a ToSR is $O(|D|)$, since a ToSR appears in at most $O(|D|)$ sequences. Also, the space complexity is $O(|D|)$, too. As we mentioned in Section 4.2, the utility upper bound calculation can be computed in $O(1)$. Thus, the time complexity of calculation $u(r)$ and $\text{conf}(r)$ is $O(|D|)$ as a utility table has at most $O(|D|)$ entries. Thus, in the initial phase, the time complexity is $O(|D|)(2L + 2) + |D|L)$ and the space complexity is $O(|D|)(L + 1) + I)$.

In Algorithm 2 line 1, the time complexity of obtaining all items set $A$ and items set $C$ is $O(|D|L)$, since the sequences in $UT(r)$ is at most $O(|D|)$ and the longest sequence length is $O(L)$. For each item, it will connect to a ruleJudge. For ruleJudge (Algorithm 3), the time complexity of calculation $RSPEU(t)$, $u(t)$, and $\text{conf}(t) = O(|D|)$. The time complexity of construction $ART(t)$ and $UT(t)$ (Algorithm 3, Line 6) is $O(|D|)$. Assume that the expansion times are $C$ in the stage of rule growth. Finally, we can get the time complexity of extending ToSR is $O(|D|(L + 2))$. Different from US-Rule and HUSRM, where they used breadth-first search strategy, TotalSR uses depth-first search strategy. Thus, the space complexity of the search process (Algorithm 2) depends on the height of the search space. In Algorithm 2, the space complexity is $O(I)$, since there are at most $I$ items that can be extended into a ToSR. Because the sequence maintained in the utility table will not appear in $ART$ and the sum number of entries in utility table and $ART$ at most is $O(|D|)$. Thereby, the space complexity is $O(|D| + I)$ of rule growth stage. Assume the height of the search space is $H$, and the space complexity of extending ToSR is $O(H)$. Finally, the space complexity is $O(H(|D| + I))$.

In summary, the time complexity of TotalSR is $O(|D|(2I(L + 1) + L + C(L + 2)))$ and the space complexity is $O(|D| + I)(H + 1) + |D|L)$.

5 EXPERIMENTS

In this section, we designed several experiments to verify the performance of the algorithm, TotalSR. Since the result generated from TotalSR and HUSRM is different and there is no existing work about mining high-utility totally-ordered sequential rules, we cannot directly use HUSRM and US-Rule as the comparison algorithms for TotalSR. Besides, the calculation of the support of the antecedent is also different. In TotalSR, the combination of items is much more complicated such that we cannot use bit vectors to get the support of the antecedent. However, the pruning strategies used in HUSRM and US-Rule can also be used in high-utility totally-ordered sequential rule mining. Thus, to demonstrate pruning strategies proposed in TotalSR are more effective, we
Table 7. Description of the Datasets

| Dataset       | | | avg(S) | max(S) | avg(Seq) | avg(Ele) |
|---------------|---|---|--------|--------|----------|---------|
| BIBLE         | 36,369 | 13,905 | 21.64  | 100    | 21.64    | 1.00    |
| Kosarak10k    | 10,000 | 10,094 | 8.14   | 608    | 8.14     | 1.00    |
| Leviathan     | 5,834  | 9,025  | 33.81  | 100    | 33.81    | 1.00    |
| SIGN          | 730    | 267    | 52.00  | 94     | 52.00    | 1.00    |
| SynDataset10k | 10,000 | 7,312  | 6.22   | 18     | 26.99    | 4.35    |
| SynDataset20k | 20,000 | 7,442  | 6.20   | 18     | 26.97    | 4.35    |
| SynDataset40k | 40,000 | 7,584  | 6.19   | 18     | 26.74    | 4.32    |
| SynDataset80k | 79,718 | 7,584  | 6.20   | 18     | 26.78    | 4.32    |
| SynDataset160k| 159,501| 7,609  | 6.19   | 18     | 26.74    | 4.32    |
| SynDataset240k| 239,211| 7,617  | 6.19   | 18     | 26.74    | 4.32    |
| SynDataset320k| 318,889| 7,620  | 6.19   | 18     | 26.74    | 4.32    |
| SynDataset400k| 398,716| 7,621  | 6.19   | 18     | 26.74    | 4.32    |

design two comparators: TotalSR$_{V_1}$ and TotalSR$_{V_2}$, where TotalSR$_{V_1}$ used the pruning strategies proposed in HUSR and TotalSR$_{V_2}$ used the pruning strategies proposed in US-Rule. We also conducted the ablation study to demonstrate the effect of the pruning strategies and to prove the effectiveness of each pruning strategy simultaneously. We designed an algorithm TotalSR$^-$ that only used RSPEU to shrink the search space. To validate the memory consumption effectiveness, TotalSR$_{V_1}$, TotalSR$_{V_2}$, and TotalSR$^-$ use breadth-first search strategy just like HUSR and US-Rule, but TotalSR uses depth-first search strategy. To verify TotalSR’s ability to mine HTSRs in large-scale datasets, we also conduct a scalability test on the size of the datasets varying from 10k to 400k. All algorithms are implemented in Java, and the machine used for all experiments has a 3.8 GHz Intel Core i7-10700K processor, 32 GB of RAM, and a 64-bit version of Windows 10. All the source code and datasets are available at GitHub.

Evaluation metrics. We will compare the widely used metrics, such as runtime and memory consumption, to evaluate the efficiency of TotalSR. To validate the effectiveness of the pruning strategies proposed in this article, we also compare the number of candidate HTSRs generated from TotalSR$_{V_1}$, TotalSR$_{V_2}$, TotalSR$^-$, and TotalSR.

5.1 Data Description

The performance of the two proposed algorithms is assessed using multiple datasets, comprising four real-life datasets and eight synthetic datasets. The four real-life datasets, including BIBLE, Kosarak10k, Leviathan, and Sign are generated from a book, click-stream, a book, and sign language, respectively. All real-life datasets can be downloaded from the open-source website SPMF [8]. The remaining datasets are synthetic generated from the IBM data generator [1]. Moreover, we use the same simulation model that was universally used in References [16, 22, 23, 31] to generate the internal utility and the external utility for each dataset. The descriptions of these datasets are listed in Table 7. $|\mathcal{D}|$ represents the number of sequences in each dataset. The distinct quantity of items in each dataset is denoted as $|I|$. The average number and the maximum number of itemsets in each dataset are denoted as avg(S) and max(S), respectively. The average length of the sequence in each dataset is expressed as avg(Seq). avg(Ele) means the average items in each itemset of the sequence in every dataset.

1https://github.com/DSI-Lab1/TotalSR
5.2 Efficiency Analysis

In this subsection, we will analyze the efficiency of the proposed algorithm, TotalSR, in terms of execution time. To keep track of the results that we want to compare with the different algorithms in the last subsection we designed, we set the minimum confidence threshold to 0.6, but the minimum utility threshold will be varied according to the characteristics of the different datasets. All experimental results are illustrated in Figure 1.

From Figure 1, we can find that on all datasets and under any minimum utility threshold, TotalSR can achieve the best performance compared to all the other algorithms, demonstrating that TotalSR is the best method in terms of execution time. Especially in long-sequence datasets such as SIGN, BIBLE, Leviathan, and synthetic datasets, the running time of TotalSR is obviously less than TotalSR\textsubscript{V\textsubscript{1}} and TotalSR\textsubscript{V\textsubscript{2}}. On the dataset, Kosarak10k, as Figure 1(b) shows, the execution times of different algorithms are almost the same. This is because the average length of the sequence in Kosarak10k is 8.14, as Table 7 shows, resulting in a small depth of search space. Thus, the number of candidate HTSRs is also small. An interesting result is that in some datasets such as BIBLE, the running time of TotalSR\textsubscript{V\textsubscript{2}} is slightly longer than TotalSR\textsubscript{V\textsubscript{1}}. While the pruning strategies used in TotalSR\textsubscript{V\textsubscript{2}} are tighter than TotalSR\textsubscript{V\textsubscript{1}}, the time to calculate the utility upper bound also increases, which may cause the cost of calculating the upper bound to be larger than the profit of pruning the search space. But the result of TotalSR\textsuperscript{−} slightly outperforms TotalSR\textsubscript{V\textsubscript{1}} and TotalSR\textsubscript{V\textsubscript{2}}. For example, in Figures 1(a), (b), and (c), TotalSR\textsuperscript{−} uses a bit less time to complete the mining process compared to TotalSR\textsubscript{V\textsubscript{1}}. Especially in Figures 1(e) and (f), the gap between TotalSR\textsuperscript{−}, TotalSR\textsubscript{V\textsubscript{2}}, and TotalSR\textsubscript{V\textsubscript{1}} is apparent. The results reveal that the upper bound RSPEU proposed in this article can effectively work. Comparing the curves of TotalSR\textsuperscript{−} and TotalSR, we can find that the confidence pruning strategy is an excellent optimization that can shrink the execution time in HTSR discovery greatly.

In summary, the abundant results demonstrate that TotalSR using the two novel pruning strategies can discover all HTSRs in a satisfactory time and also validate that TotalSR is superior to TotalSR\textsubscript{V\textsubscript{1}} and TotalSR\textsubscript{V\textsubscript{2}}.

5.3 Effectiveness of the Pruning Strategies

To validate the effectiveness of the two novel pruning strategies, we compared the number of candidate HTSRs generated from TotalSR\textsubscript{V\textsubscript{1}}, TotalSR\textsubscript{V\textsubscript{2}}, TotalSR\textsuperscript{−}, and TotalSR in this subsection. The results are shown in Figure 2. Note that the left Y axis is the number of candidate HTSRs using...
Totally-ordered Sequential Rules for Utility Maximization

Fig. 2. The generated candidates and HTSRs under different pruning strategies and various minimum utility thresholds.

As Figure 2 shown with the minutil increases, the number of corresponding candidates decreases, but the extent of the decrease of different methods is distinct. TotalSR$_{V1}$ has the highest number of candidates across all datasets, since the pruning strategies in TotalSR$_{V1}$ are loose. With the help of the RSU pruning strategy, the number of candidates for TotalSR$_{V2}$ decreases sharply on all datasets. Moreover, the number of candidates generated by RSPEU on all datasets is smaller than the traditional pruning strategies based on PEU and RSU. Even if the reduction in the number of candidates is not particularly noticeable, it still has an effect, which can achieve less running time in discovering HTSRs. We can validate this in Figure 1. The other important point is that the quantity of candidates produced by the confidence pruning strategy drops further on each dataset. For example, on BIBLE, the candidate generated from TotalSR on the large minutil is very small, contributing to a small execution time. Note that different from the RSPEU, the confidence pruning does need an extra data structure to maintain. Thus, the effect on the execution time of the confidence pruning strategy is more notable. The result of other datasets also can prove that the confidence pruning strategy is highly effective.

5.4 Memory Evaluation

In addition to analyzing the running time, memory consumption is also a crucial measure of the effectiveness of the algorithm. In this subsection, we will analyze the memory consumption of the different versions of the algorithms. The details are presented in Figure 3.

On the BIBLE dataset, TotalSR$_{V1}$ and TotalSR$_{V2}$ cost similar memory, while the memory consumption of TotalSR$^-$ and TotalSR uses fewer. In particular, TotalSR takes up the least memory. On the Kosarak10k dataset, the memory usage of the algorithms TotalSR$_{V1}$, TotalSR$_{V2}$, and TotalSR$^-$ is almost the same. But the memory consumption of TotalSR is much less than others under most minutil thresholds. This is because the space complexity of the depth-first search strategy depends on the height of the search space, but the space complexity of the breadth-first search strategy is exponential. Similar results can be found in the synthetic datasets, i.e., Figures 3(e) and (f), where the difference in memory consumption between TotalSR and the other algorithms is more apparent. On Leviathan, the difference between algorithms is obvious under small minutil. When the minutil is above 2,500, the memory usage of the algorithms is similar. This may be
Fig. 3. The memory usage of the proposed methods under various minimum utility thresholds.

because in TotalSR\textsubscript{V1}, TotalSR\textsubscript{V2}, and TotalSR\textsuperscript{−} the SEU pruning strategy is also used to filter the ToSR with size $1 \times 1$, contributing to fewer data structures being created. But TotalSR always uses the least memory, since fewer candidates are generated by TotalSR. In Figure 3(d), a weird result is illustrated. The memory consumption of TotalSR is larger than the left algorithms. Since the distinct items in SIGN dataset are small, the number of $1 \times 1$ ToSRs is small, which does not need much memory to maintain. Besides, TotalSR\textsubscript{V1}, TotalSR\textsubscript{V2}, and TotalSR\textsuperscript{−} used SEU to prune all un-promising $1 \times 1$ ToSRs, resulting in the candidate HTSR’s data structures generated from TotalSR\textsubscript{V1}, TotalSR\textsubscript{V2}, and TotalSR\textsuperscript{−} are less than TotalSR. Last, the memory usage of TotalSR on dataset SIGN is larger than the other algorithms.

5.5 Scalability Test

To verify the robustness of the proposed algorithms, we conducted several experiments on the datasets [1] with sizes ranging from 10k to 400k, and the minimum utility threshold was fixed to one-thousandth of the total utility of each dataset. For example, if the total utility is 10,000, then the minutil is $10,000 \times 0.001 = 10$.

In Figure 4(a), as the dataset’s size increases, the execution time of each algorithm will increase gradually. TotalSR always takes the shortest time, followed by TotalSR\textsuperscript{−}, TotalSR\textsubscript{V2}, and TotalSR\textsubscript{V1}. When the size of the dataset reaches 320k, only TotalSR can complete the search task, and the other algorithms will not perform due to the out-of-memory error. Although the running time of TotalSR\textsuperscript{−} is only slightly shorter than that of TotalSR\textsubscript{V2}, this can also indicate that RSPEU is more effective than RSU. We can prove it in Figure 4(c), where the candidates generated by TotalSR\textsuperscript{−} are less than TotalSR\textsubscript{V2}. From Figure 4(b), we can observe that the memory consumption difference between TotalSR and the other algorithms is obvious. Thus, TotalSR can effectively cope with large-scale datasets.

6 CONCLUSION

In this work, we formulate the problem of high-utility totally-ordered sequential rule mining and propose an algorithm TotalSR to discover the complete HTSRs in a given dataset. To verify the effectiveness of the pruning strategies proposed in this article, we conducted several experiments with TotalSR\textsubscript{V1}, TotalSR\textsubscript{V2}, and TotalSR\textsuperscript{−}. TotalSR\textsubscript{V1} and TotalSR\textsubscript{V2} used the pruning strategies proposed in HUSRM and US-Rule, respectively. We cannot use HUSRM and US-Rule as the baseline algorithms, since HUSRM and US-Rule cannot output the same result as TotalSR. The experimental
results showed that the pruning strategies can make a great contribution to the reduction of the search space, which can improve the efficiency of the algorithm. Finally, extensive experiments conducted on different datasets showed that the proposed algorithm can not only effectively and efficiently discover all HTSRs but also have outstanding scalability.

In the future, we are looking forward to developing several algorithms based on TotalSR that can discover target rules [14] or process more complicated data like interval-based events [21]. All these fields would be interesting to be exploited.

REFERENCES

[1] Rakesh Agrawal and Ramakrishnan Srikant. 1995. Mining sequential patterns. In Proceedings of the 11th International Conference on Data Engineering. IEEE, 3–14.

[2] Rakesh Agrawal, Ramakrishnan Srikant, et al. 1994. Fast algorithms for mining association rules. In Proceedings of the 20th International Conference Very Large Data Bases. Morgan Kaufmann, 487–499.

[3] Chowdhury Farhan Ahmed, Syed Khairuzzaman Tanbeer, and Byeong-Soo Jeong. 2010. Mining high utility web access sequences in dynamic web log data. In Proceedings of the 11th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing. IEEE, 76–81.

[4] Chowdhury Farhan Ahmed, Syed Khairuzzaman Tanbeer, and Byeong-Soo Jeong. 2010. A novel approach for mining high-utility sequential Patterns in Sequence Databases. ETRI J. 32, 5 (2010), 676–686.

[5] Philippe Fournier-Viger, Usef Faghihi, Roger Nkambou, and Engelbert Mephu Nguifo. 2012. CMRules: Mining sequential rules common to several sequences. Knowl.-based syst. 25, 1 (2012), 63–76.

[6] 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 International Conference on Advanced Data Mining and Applications. Springer, 431–442.

[7] Philippe Fournier-Viger, Ted Gueniche, Souleymane Zida, and Vincent S. Tseng. 2014. ERMiner: Sequential rule mining using equivalence classes. In Proceedings of the International Symposium on Intelligent Data Analysis. Springer, 108–119.

[8] Philippe Fournier-Viger, Jerry Chun-Wei Lin, Antonio Gomariz, Ted Gueniche, Azadeh Soltani, Zhihong Deng, and Hoang Thanh Lam. 2016. The SPMF open-source data mining library version 2. In Proceedings of the Joint European Conference on Machine Learning and Knowledge Discovery in Databases. Springer, 36–40.

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

[10] Philippe Fournier-Viger, Roger Nkambou, and Vincent Shin-Mu Tseng. 2011. RuleGrowth: Mining sequential rules common to several sequences by pattern-growth. In Proceedings of the ACM Symposium on Applied Computing. 956–961.

[11] Philippe Fournier-Viger, Cheng-Wei Wu, Vincent S. Tseng, Longbing Cao, and Roger Nkambou. 2015. Mining partially-ordered sequential rules common to multiple sequences. IEEE Trans. Knowl. Data Eng. 27, 8 (2015), 2203–2216.

[12] Wensheng Gan, Lili Chen, Shicheng Wan, Jiahui Chen, and Chien-Ming Chen. 2021. Anomaly rule detection in sequence data. IEEE Trans. Knowl. Data Eng. (2021).

[13] Wensheng Gan, Zilin Du, Weiping Ding, Chunkai Zhang, and Han-Chieh Chao. 2021. Explainable fuzzy utility mining on sequences. IEEE Trans. Fuzzy Syst. 29, 12 (2021), 3620–3634.

[14] Wensheng Gan, Gengsen Huang, Jian Weng, Tianlong Gu, and Philip S. Yu. 2022. Towards target sequential rules. arXiv preprint arXiv:2206.04728 (2022).

[15] 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 Trans. Knowl. Discov. Data 13, 3 (2019), 1–34.
[16] Wensheng Gan, Jerry Chun-Wei Lin, Jiexiong Zhang, Han-Chieh Chao, and Philip S. Yu. 2020. ProUM: Projection-based utility mining on sequence data. Inf. Sci. 513 (2020), 222–240.

[17] Wensheng Gan, Jerry Chun-Wei Lin, Jiexiong Zhang, Philippe Fournier-Viger, Han-Chieh Chao, and Philip S. Yu. 2020. Fast utility mining on sequence data. IEEE Trans. Cybern. 51, 2 (2020), 487–500.

[18] Jiawei Han, Jian Pei, Behzad Mortazavi-Asl, Helen Pinto, Qiming Chen, Umeshwar Dayal, and Meichun Hsu. 2001. PrefixSpan: Mining sequential patterns efficiently by prefix-projected pattern growth. In Proceedings of the 17th International Conference on Data Engineering. Citeseer, 215–224.

[19] M. Han, Z. Wang, and J. Yuan. 2013. Mining constraint based sequential patterns and rules on restaurant recommendation system. J. Comput. Inf. Syst. 9, 10 (2013), 3901–3908.

[20] Gengsen Huang, Wensheng Gan, Jian Weng, and Philip S. Yu. 2023. US-Rule: Discovering utility-driven sequential rules. ACM Trans. Knowl. Discov. Data 17, 1 (2023), 1–22.

[21] Yong Joon Lee, Jun Wook Lee, Duck Jin Chai, Bu Hyun Hwang, and Keun Ho Ryu. 2009. Mining temporal interval relational rules from temporal data. J. Syst. Softw. 82, 1 (2009), 155–167.

[22] Jerry Chun-Wei Lin, Wensheng Gan, Philippe Fournier-Viger, Tzung-Pei Hong, and Han-Chieh Chao. 2017. FDHUP: Fast algorithm for mining discriminative high utility patterns. Knowl. Inf. Syst. 51, 3 (2017), 873–909.

[23] Mengchi Liu and Junfeng Qu. 2012. Mining high utility itemsets without candidate generation. In Proceedings of the 21st ACM International Conference on Information and Knowledge Management. 55–64.

[24] David Lo, Siu-Cheng Khoo, and Limsoon Wong. 2009. Non-redundant sequential rules—Theory and algorithm. Inf. Syst. 34, 4-5 (2009), 438–453.

[25] Sinkon Nayak, Mahendra Kumar Gourisaria, Manjusha Pandey, and Siddharth Swarup Rautaray. 2019. Heart disease prediction using frequent item set mining and classification technique. Int. J. Inf. Eng. Electron. Bus. 11, 6 (2019).

[26] Jian Pei, Jiawei Han, B. Mortazavi-Asl, Jianyong Wang, H. Pinto, Qiming Chen, U. Dayal, and M. C. Hsu. 2004. Mining sequential patterns by pattern-growth: The PrefixSpan approach. IEEE Trans. Knowl. Data Eng. 16, 11 (2004), 1424–1440.

[27] Thi-Thiet Pham, Jiawei Luo, Tzung-Pei Hong, and Bay Vo. 2014. An efficient method for mining non-redundant sequential rules using attributed prefix-trees. Eng. Appl. Artif. Intell. 32 (2014), 88–99.

[28] Alberto Segura-Delgado, Augusto Anguita-Ruiz, Rafael Alcalá, and Jesús Alcalá-Fdez. 2022. Mining high average-utility sequential rules to identify high-utility gene expression sequences in longitudinal human studies. Expert Syst. Appl. 193 (2022), 116411.

[29] Bai-En Shie, Hui-Fang Hsiao, Vincent S. Tseng, and Philip S. Yu. 2011. Mining high utility mobile sequential patterns in mobile commerce environments. In Proceedings of the International Conference on Database Systems for Advanced Applications. Springer, 224–238.

[30] Ramakrishnan Srikant and Rakesh Agrawal. 1996. Mining sequential patterns: Generalizations and performance improvements. In Proceedings of the International Conference on Extending Database Technology. Springer, 1–17.

[31] Vincent S. Tseng, Bai-En Shie, Cheng-Wei Wu, and Philip S. Yu. 2012. Efficient algorithms for mining high utility itemsets from transactional databases. IEEE Trans. Knowl. Data Eng. 25, 8 (2012), 1772–1786.

[32] Jianyong Wang, Jiawei Han, and Chun Li. 2007. Frequent closed sequence mining without candidate maintenance. IEEE Trans. Knowl. Data Eng. 19, 8 (2007), 1042–1056.

[33] Jun-Zhe Wang, Jiun-Long Huang, and Yi-Cheng Chen. 2016. On efficiently mining high utility sequential patterns. Knowl. Inf. Syst. 49, 2 (2016), 597–627.

[34] Youxi Wu, Lanfang Luo, Youxi Wu, Yuehua Wang, Yan Li, Xingquan Zhu, and Xindong Wu. 2021. NTP-Miner: A Fast algorithm for mining discriminative high utility sequential patterns. IEEE Trans. Cybern. 52, 11 (2021), 11819–11833.

[35] Junfu Yin, Zhigang Zheng, and Longbing Cao. 2012. USpan: An efficient algorithm for mining high utility sequential patterns. IEEE Trans. Big Data 9, 2 (2023), 512–527.

[36] Chunkai Zhang, Zilin Du, Wensheng Gan, and Philip S. Yu. 2021. TKUS: Mining top-k high utility sequential patterns. Inf. Sci. 570 (2021), 342–359.
[41] Chunkai Zhang, Zilin Du, Yuting Yang, Wensheng Gan, and Philip S. Yu. 2021. On-shelf utility mining of sequence data. *ACM Trans. Knowl. Discov. Data* 16, 2 (2021), 1–31.

[42] Mengjiao Zhang, Tiantian Xu, Zhao Li, Xiqing Han, and Xiangjun Dong. 2020. e-HUNSR: An efficient algorithm for mining high utility negative sequential rules. *Symmetry* 12, 8 (2020), 1211.

[43] Souly Zida, Philippe Fournier-Viger, Cheng-Wei Wu, Jerry Chun-Wei Lin, and Vincent S. Tseng. 2015. Efficient mining of high-utility sequential rules. In *Proceedings of the International Workshop on Machine Learning and Data Mining in Pattern Recognition*. Springer, 157–171.

[44] Sabrina Ziebarth, Irene-Angelica Chounta, and H. Ulrich Hoppe. 2015. Resource access patterns in exam preparation activities. In *Proceedings of the European Conference on Technology Enhanced Learning*. Springer, 497–502.

Received 22 September 2022; revised 7 August 2023; accepted 26 September 2023