Utility-driven Mining of Contiguous Sequences

Chunkai Zhang[a,b], Jun Zhu[a], Hanyu Zhang[c], Wensheng Gan[d], Philip S. Yu[e]

[a]*Department of Computer Science and Technology, Harbin Institute of Technology (Shenzhen), Shenzhen, 518055, Guangdong, China*
[b]*Guangdong Provincial Key Laboratory of Intelligent System Research Center, Guangzhou, 510000, Guangdong, China*
[c]*New York Stock Exchange LLC, New York, 10005, New York, United States*
[d]*College of Cyber Security, Jinan University, Guangzhou, 510632, Guangdong, China*
[e]*Department of Computer Science, University of Illinois at Chicago, Chicago, 60607-7053, Illinois, United States*

Abstract

Recently, contiguous sequential pattern mining (CSPM) gained interest as a research topic, due to its varied potential real-world applications, such as web log and biological sequence analysis. To date, studies on the CSPM problem remain in preliminary stages. Existing CSPM algorithms lack the efficiency to satisfy users' needs and can still be improved in terms of runtime and memory consumption. In addition, existing algorithms were developed to deal with simple sequence data, working with only one event at a time. Complex sequence data, which represent multiple events occurring simultaneously, are also commonly observed in real life. In this paper, we propose a novel algorithm, fast utility-driven contiguous sequential pattern mining (FUCPM), to address the CSPM problem. FUCPM adopts a compact sequence information list and instance chain structures to store the necessary information of the database and candidate patterns. For further efficiency, we develop the global unpromising items pruning and local unpromising items pruning strategies, based on sequence-weighted utilization and item-extension utilization, to reduce the search space. Extensive experiments on real-world and synthetic datasets demonstrate that FUCPM outperforms the state-of-the-art algorithms and is scalable enough to handle complex sequence data.

Keywords: Sequence data, Utility mining, Sequential pattern, Contiguous.

1. Introduction

In the era of big data, large volumes of data are produced every day, including a large amount of sequence data. Sequence data consist of a series of elements (also called *itemsets*) that are arranged in a specific order, such as chronological order (e.g., web access logs, vehicle trajectories) and biological order (e.g., DNA sequences). Each itemset is a set of one or more events (also called *items*). Sequential pattern mining (SPM) technology [1, 2] extracts the useful underlying knowledge contained in sequence data. SPM's objective is to discover all the frequent sequences from a sequence database, as sequential patterns, where the frequency (also known as *support*) of a sequence is measured by its occurrence times in the database, with respect to a user-defined minimum support threshold. It is well-recognized that SPM facilitates a variety of applications, including key-phrase extraction [3], medical early alarm systems [4], outlying behavior detection [5], and recommendation systems [6, 7]. Amazon utilizes SPM to recommend products with the slogan "customer bought something always then bought this."

SPM assumes that high-frequency sequential patterns are meaningful and interesting. However, such an assumption is impractical in several real-world situations. For example, an SPM-based traffic flow analysis system assesses congestion by mining vehicle trajectory databases. Routes containing heavily used trunk roads are discovered as frequent patterns and are regarded as congested. However, the frequent routes are not necessarily congested because trunk roads often contain more lanes to accommodate more vehicles, while side roads are relatively narrow to form traffic jams. In this case, the route frequency is not a suitable measurement for congestion analysis, while a vehicle's average speed while traveling the roads may be better. A retailer developing bundled sales strategies to maximize revenue provides another example of this technology in use. SPM technology can only tell the retail which commodity patterns are hot selling but cannot provide any information about profit. In general, the frequency cannot represent importance in some scenarios, and infrequent sequences may contain crucial information. To address this problem, SPM was generalized to obtain a new study field called high-utility sequential pattern mining (HUSPM) [8, 9, 10].

HUSPM incorporates the concept of utility that reveals the relative importance of the items. The mining objectives of HUSPM are quantitative sequence databases, wherein each item is associated with an integer called internal utility, which represents the quantities of an item in an itemset.

Moreover, each distinct item has an external utility that measures its significance (e.g., unit profit, interest, and satisfaction). HUSPM aims to find all sequences whose utility is no less than the user-defined minimum utility threshold in a sequence database. The discovered sequences are called high-utility sequential patterns (HUSPs) [10]. Unlike SPM, HUSPM can find more valuable patterns because the utility is more related to realistic business needs than frequency [11]. In recent years, HUSPM has been a popular topic in knowledge discovery in databases (KDD) [12] due to its practicality. Some studies [13, 14, 15, 16] were conducted to improve the efficiency of mining HUSPs, while others explored the practical application of HUSPM in various domains, including purchase behavior analysis [17], web access analysis [18], and mobile computing [19].

Although SPM and HUSPM can excavate useful information from databases, they suffer from a large number of discovered sequential patterns, especially when processing a large-scale database with a low threshold. In addition, some patterns may be meaningless in certain scenarios. For example, when mining the vehicle trajectory database, the sequential patterns that maintain the continuity of items with respect to the original trajectories are more meaningful than those that are not contiguous because the former correspond to real-world routes, whereas the latter jump from place to place [20]. In the task of DNA sequence analysis, ensuring that the nucleotide arrangement in the patterns is consistent with that of DNA sequences is important because skip nucleotides are difficult to interpret [21]. In summary, the inherent continuous order of items in the sequence data is non-negligible under certain circumstances. This problem introduces a new research task, called contiguous sequential pattern mining (CSPM) [22]. CSPM aims to discover contiguous sequential patterns (CSPs), in which the items must maintain the adjacent relation that is defined in the sequence data [23]. In other words, a CSP must be a continuous sub-sequence of some sequences in the database. For example, in Figure 1, a person goes to *Bryant Park* from the *Times Square* through the street sequence <*7th Ave, W 43rd St, 6th Ave, W 42nd St, 5th Ave*> that is marked in green, and sequential patterns, such as <*7th Ave, W 43rd St, 6th Ave*> and <*6th Ave, W 42nd St, 5th Ave*> are CSPs, while <*7th Ave, 6th Ave, 5th Ave*> is not a CSP. In addition, the streets in <*7th Ave, 6th Ave, 5th Ave*> are not connected directly to each other; such a pattern may not be useful for applications such as path planning. To date, CSPM has attracted much attention and has been successfully applied in many fields [24, 25].

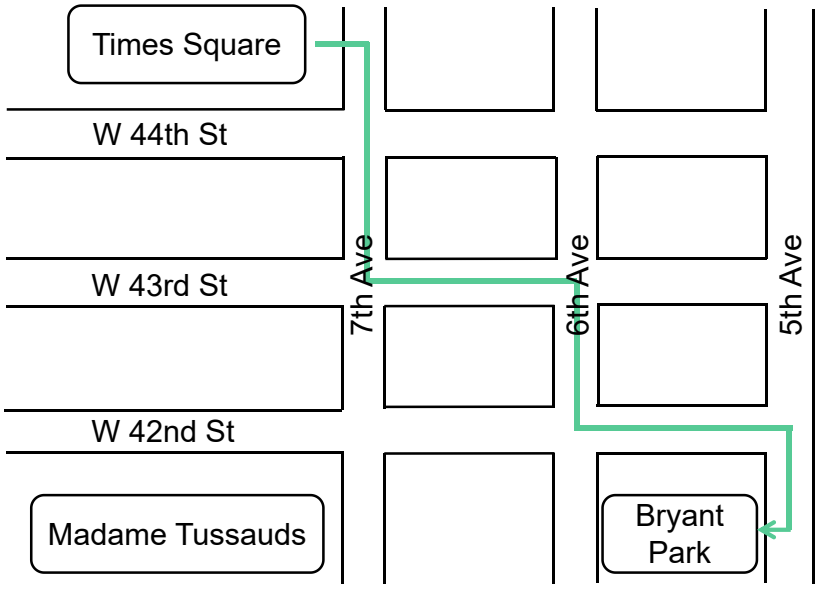

Figure 1: Example of vehicle trajectories

Previous studies on CSPM are mostly based on frequency, while only a few studies [26, 27, 28] have been conducted to address the problem of utility-driven contiguous sequential pattern mining (UCSPM). UCSPM aims to discover high-utility contiguous sequential patterns (HUCSPs) from sequence databases. In fact, UCSPM can be used in a wide range of applications. For example, in [26] and [27], HUCSPs were discovered from users' web browsing sequences, where each web page was regarded as an item and the browsing time was regarded as utility. The discovered patterns can reveal users' browsing preferences and help provide proper navigation suggestions for users. Because HUCSPs maintain the adjacent relationship between the web pages browsed by users, they can also help optimize the website architecture, for example, providing efficient access between highly correlated web pages [27]. Other potential applications of UCSPM may include vehicle trajectory analysis [20] and next-items recommendation [29].

To date, research on UCSPM remains in its early stages. It is noteworthy that almost all existing CSPM and UCSPM methods can only handle simple sequences whose itemsets contain one single item [28]. This type of sequence is called single-item-based sequences. In the age of big data, the volume and complexity of data grow rapidly. Complex sequence data,

whose itemsets may contain multiple items (also called multi-items-based sequences), are also commonly observed in real-life scenarios where multiple events occur simultaneously. For example, in Figure 2, a tourist expenditure at an attractions is <{*Entrance*}, {*Mountain, Telpher*}, {*Zoo, Souvenir Shop, Lounge*}, {*Circus Performance, Souvenir Shop*}, {*Pageant*}, {*Export*}>, which is a multi-items CSP. <{*Entrance*}, {*Virgin Forest, Waterfall*}, {*Spanish Village, Lounge*}, {*Gateway Foodcourt*}, {*Museum, Souvenir Shop*}, {*Export*}> and <{*Entrance*}, {*Fork Performance*}, {*Opera House, Gateway Foodcourt*}, {*Museum*}, {*Pageant*}, {*Export*}> are also possible sequences. In the context of sequence database constructed with the tourist expenditures from different visitors as mentioned above, we can mine multi-items HUCSPs for decision-maker to optimize travel recommendation plans. For example, considering the importance of profit, a pattern like <{*Mountain, Telpher*}, {*Zoo, Souvenir Shop, Lounge*}, {*Circus Performance, Souvenir Shop*}, {*Pageant*}> may be a multi-items HUCSP that we could mine. Another noticeable problem is that existing algorithms are not efficient enough to satisfy the requirements of users to quickly discover useful information in large-scale databases.

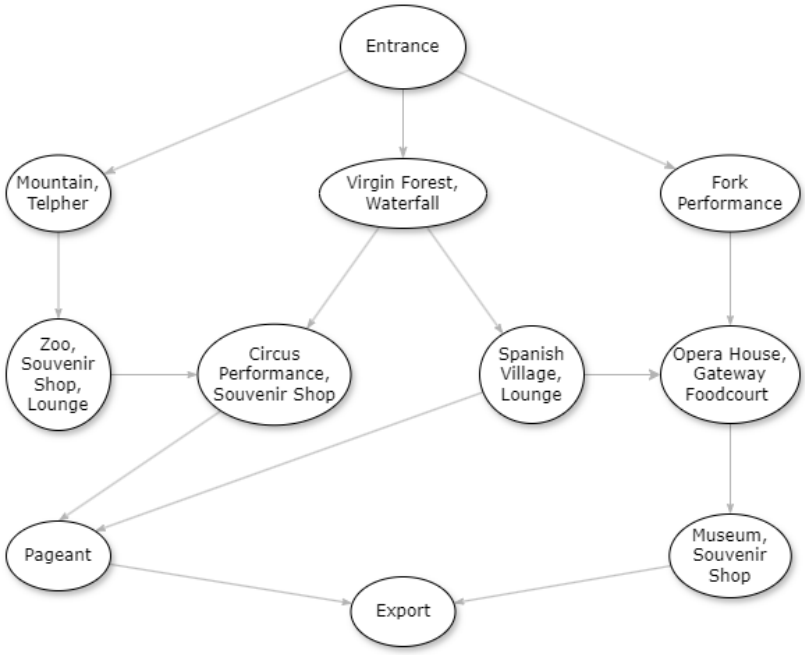

Figure 2: Example of Attraction

In general, several challenges remain in this field, which can be described as follows. First, existing UCSPM approaches [26, 27, 28] only consider single-item-based sequence data but none of them can handle multi-items-based sequence data. Therefore, developing an efficient algorithm that is scalable to large-scale and multi-items-based sequences is an urgent need. Second, UCSPM can easily encounter the combinatorial explosion of the search space. This is because the inherent ordering of itemsets in sequence data generates various compositions of candidate patterns. In particular, when handling complex multi-items-based sequences, the number of candidates can be extremely large. The integration of the contiguous constrain to design powerful pruning strategies is crucial. Third, the downward closure property [30] of frequency does not hold for utility; that is, no exact relationship exists between the utility of a sequence and that of its sub-sequences or super-sequences. Therefore, the pruning strategies of frequency-based SPM cannot be directly applied to UCSPM. Fourth, to identify HUCSPs, the utility values and positions of the candidate patterns must be stored during the mining process. Because numerous candidates can be generated in the mining process, designing compact and accessible data structures for storing these rich information is important.

Motivated by these challenges, in this paper, we propose a novel algorithm called FUCPM to mine HUCSPs more efficiently. The major contributions of our study are summarized as follows:

- We formalized the problem of UCSPM and then proposed an efficient and scalable algorithm called FUCPM. Unlike the previous UCSPM algorithm, FUCPM can further handle multi-items-based sequence data.
- We designed two compact data structures called sequence information list (SIL) and instance chain to facilitate the utility calculation of candidate patterns.
- We proposed a novel utility upper bound called item-extension utilization (IEU) and two search space pruning strategies: global unpromising items pruning (GUIP) and local unpromising items pruning (LUIP).
- After conducting substantial experiments on real-world and synthetic datasets, FUCPM shows its superiority in terms of runtime, memory consumption, and unpromising candidates pruning compared to the

state-of-the-art algorithms. In addition, it demonstrates high scalability in handling large-scale sequence datasets (either single-item-based or multi-items-based).

The remainder of this paper is organized as follows. Section 2 reviews related works. Section 3 presents the basic preliminaries and problem statements. The proposed FUCPM algorithm with several data structures and pruning strategies is discussed in Section 4. The experimental results and analysis are presented in Section 5. Finally, Section 6 concludes the paper and provides prospects for future work.

2. Related Work

In this section, we review prior works from the fields of SPM, HUSPM, CSPM, and UCSPM.

2.1. SPM and HUSPM

Frequent itemset mining (FIM) [30] aims to discover frequent itemsets from transaction databases. Compared to FIM, SPM additionally considers the sequential ordering of items in the sequences; thus, it is a more complicated task. In general, SPM algorithms can be categorized into three types: breadth-first, depth-first, and pattern-growth algorithms [31]. Inspired by the Apriori algorithm [30] for FIM, Agrawal *et al.* proposed the first SPM algorithm called AprioriAll [1] and GSP [2]. Both AprioriAll and GSP are breadth-first; that is, they generate k-sequences (i.e., sequences with k items) based on (k-1)-sequences and scan the database repeatedly to calculate the supports of all k-sequences. The algorithms do not terminate until no candidate sequences can be generated. The breadth-first algorithms suffer from multiple database scans, which are time-consuming. To address this problem, a depth-first algorithm, SPADE [32], is proposed. By utilizing a vertical database representation called IDList, which facilitates the calculation of support, SPADE avoids scanning the original database repeatedly. Due to the high efficiency of IDList, it has been used in several subsequent depth-first algorithms (e.g., SPAM [33], CM-SPAM [34], and CM-SPADE [34]). Although depth-first algorithms are more efficient, they generate many candidate patterns that do not exist in the database. To reduce the search space, Pei *et al.* [35] proposed a pattern-growth algorithm called PrefixSpan, which only considers candidate patterns appearing in the database. PrefixSpan introduced

an important concept, called the projected database, to reduce the cost of database scans. More detailed overviews of SPM can be found in [31, 36] .

Frequency-based SPM has its limitations because frequency cannot reflect importance under many circumstances. To address this problem, Ahmed *et al.* [9] first incorporated the concept of utility into SPM and proposed two different algorithms called utility-level (UL) and utility-span (US), and both of them adopt an upper bound on utility called sequence-weighted utilization (SWU) to prune the search space. UL and US have two main disadvantages. First, SWU is quite loose, so the algorithms still generate numerous candidates. Second, it costs much time to calculate the exact utility of candidates because multiple database scans are required. In view of this, many later HUSPM algorithms [10, 37, 13, 14, 15] have focused on proposing tighter upper bounds to further prune the search space and design efficient data structures to facilitate the calculation of utility and upper bound value. Among them, USpan [10] adopts the utility matrix to store the utility information of quantitative sequences. Two upper bounds (SWU and sequence-projected utilization (SPU)) are utilized to eliminate the candidates. Based on the framework of USpan, Alkan *et al.* [37] proposed a complete algorithm called HuspExt. The HuspExt algorithm uses a tight upper bound called Cumulate Rest of Match to reduce unpromising candidates. HUS-Span [13] is another famous HUSPM algorithm that utilizes two novel upper bounds (i.e., prefix extension utility and reduced sequence utility). In HUS-Span, a projected database called utility chain is designed to store the utility information of candidates. Recently, Gan *et al.* proposed ProUM [14] and HUSP-ULL [15], which use efficient data structures called utility array and utility-linked-list, respectively. Lin *et al.* [38] proposed the novel sequence-utility (SU)-chain structure to store information of patterns. The SU-chain-based algorithm outperforms HUSP-ULL in some cases. More comprehensive overview of HUSPM can be found in the latest literature reviews [39, 8, 40].

2.2. CSPM and UCSPM

The methods mentioned in subsection 2.1 aim to find the complete set of sequential patterns; however, not all discovered patterns are useful under certain circumstances. In addition, the volume of mining results can be quite large, making it difficult for users to analyze and utilize them. To address this problem, several constraints (such as contiguous [22], top-k [41], and closed [42] constraints) have been proposed to obtain a concise and more meaningful subset of the complete set of sequential patterns. Among them,

the contiguous constraint is significant when the underlying ordering of items in the sequence data must be emphasized.

Pan *et al.* [43] are among the earliest researchers to investigate the problem of CSPM. They developed two algorithms called MacosFSpan and MacosVSpan, which are inspired by the PrefixSpan algorithm, to mine frequent concatenate sequences (i.e., CSPs) from biological datasets. Kang *et al.* [44] proposed the fixed-length spanning tree structure that is used to store the fixed-length contiguous sub-sequences and their supports of given sequence data. The formal definition of CSP was provided by Chen *et al.* [22]. To discover CSPs from web access logs, they designed the UpDown tree to store sequences that contain a given item, which benefits the mining process. Chen *et al.* [45] latter proposed an improved version of the UpDown tree, which can handle the sequence data where each itemset may contain more than one item. Currently, CSPM remains a research hotspot. Several studies [23, 46] combined closed and contiguous constraints for SPM to obtain a more concise pattern set without loss of information. Other recent studies applied CSPM to a wide range of realistic applications, such as vehicle trajectory analysis [20, 25], protocol specification extraction [47], and biological sequence analysis [48, 21]. Hu *et al.* [49] proposes a novel problem of target-oriented contiguous sequential pattern mining and introduces the algorithm called TCSPM.

Frequency-based CSPM has been extensively studied; however, very few studies focused on utility-driven CSPM. Zhou *et al.* [26] first introduced utility into CSPM and developed a two-phase algorithm to mine web path traversal patterns, which are HUCSPs actually. This algorithm is similar to the UL algorithm mentioned previously; both generate candidate patterns in a breadth-first fashion and calculate the SWU of these candidates in the first phase. In the second phase, the algorithm scans the database repeatedly to calculate the exact utility value of candidates with high SWU. An improved algorithm for web path traversal pattern mining, called EUWPTM [27], was proposed later. To reduce a large number of candidates, EUWPTM uses a pattern-growth mechanism to generate candidate patterns. These two algorithms mainly adopt the loose upper bound SWU to prune the search space. Huang *et al.* [28] proposed a more efficient algorithm HUCP-Miner with a tight upper bound called remaining utility upper-bound ($RUUB$) that is actually equivalent to the prefix extension utility in HUS-Span. In addition, they designed a data structure called the UL-list to store the utility and position information of candidate patterns. Thus far, HUCP-Miner is the state-

of-the-art algorithm for UCSPM. However, *RUUB* is not tight enough to filter out unpromising patterns during the early stage of the mining process. In addition, all the aforementioned algorithms can only handle single-item-based sequences, whereas a large volume of multi-items-based sequences is produced and needs to be analyzed. These problems motivated us to develop a more efficient and scalable algorithm for the UCSPM task.

3. Preliminaries

This section introduces the basic concepts and definitions used in this paper. Then, the formal problem statement of the UCSPM is provided.

3.1. Concepts and Definitions

Let $I = \{i_1, i_2, \cdots, i_n\}$ be a set of distinct items appearing in the database. An itemset X is a nonempty subset of I, that is, $X \subseteq I$. The size of X is defined as the number of items it contains and is denoted by $|X|$. Without loss of generality, the items contained in an itemset are arranged in a lexicographical order hereinafter. A sequence $S = <X_1, X_2, \cdots, X_m>$ is an ordered list of itemsets, where $X_k \subseteq I$ for $1 \leq k \leq m$. We define $|S| = \sum_{k=1}^{m} |X_k|$ as the length of S. A sequence is called an l-sequence if its length is l. For example, given a set $I = \{a, b, c, d, e, f\}$, $X = \{cef\}$ is an itemset with a size of three, and $S = <\{a\}, \{bcf\}, \{ab\}>$ is a 6-sequence as it contains six items.

Definition 1 (contiguous sub-sequence and super-sequence). *A sequence $S = <X_1, X_2, \cdots, X_m>$ is a contiguous sub-sequence of another sequence $S' = <X'_1, X'_2, \cdots, X'_n>$, $m \leq n$, which is denoted by $S \subseteq S'$, if there exists an integer k, $1 \leq k \leq n - m + 1$, such that $X_1 \subseteq X'_k$, $X_2 \subseteq X'_{k+1}$, $\cdots$, $X_m \subseteq X'_{k+m-1}$. In addition, S' is called a super-sequence of S.*

For example, given three sequences $S = <\{a\}, \{af\}>$, $S' = <\{c\}, \{af\}>$, and $S'' = <\{c\}, \{ab\}, \{aef\}>$, we say S is a contiguous sub-sequence of S'', while S' is not a contiguous sub-sequence of S''.

Definition 2 (quantitative sequence database). *The database processed by UCSPM is a quantitative (abbreviated as q- in the following) sequence database. In a q-sequence database, each item is called a q-item, which is represented as a tuple (i:q), where $i \in I$ and q is the internal utility of item*

i. In addition, each distinct item i has an external utility p(i). A q-itemset Y with m q-items is denoted as $Y = \{(i_1{:}q_1)\ (i_2{:}q_2)\ \cdots\ (i_m{:}q_m)\}$. *A q-sequence* $Q = <Y_1, Y_2, \cdots, Y_n>$ *is an ordered list of n q-itemsets. Further, a q-sequence database is composed of a series of q-sequences with a unique identifier SID.*

Table 1 shows a q-sequence database with five q-sequences and six distinct items. In this study, we use this q-sequence database as the running example. Table 2 lists the external utility of each item. In S_1, the first q-item (b:2) represents that the internal utility of b is 2. The first itemset of S_1 (i.e., $\{(b{:}2)\ (f{:}4)\}$) is a q-itemset containing two q-items: (b:2) and (f:4). S_1 = <{(b:2) (f:4)},{(a:2) (e:2)},{(c:2) (e:1)}> is a q-sequence containing three q-itemsets: {(b:2) (f:4)}, {(a:2) (e:2)}, and {(c:2) (e:1)}.

Definition 3 (utility of q-item, q-itemset and q-sequence). *Given a q-sequence* $Q = <Y_1, Y_2, \cdots, Y_n>$, *we denote the internal utility of q-item i in* Y_j *as* $q(i,\ j,\ Q)$, *then the utility of i in* Y_j *is defined as* $u(i,\ j,\ Q) = q(i,\ j,\ Q) \times p(i)$, *where* $1 \leq j \leq n$. *In addition, the utility of the q-itemset* Y_j *is defined as* $u(Y_j, Q) = \sum_{i \in Y_j} u(i,\ j,\ Q)$. *The utility of Q is* $u(Q) = \sum_{k=1}^{n} u(Y_k, Q)$. *Finally, the utility of a q-sequence database D is* $u(D) = \sum_{Q \in D} u(Q)$.

For example, the utility of (b:2) within the first q-itemset of S_1 is $u(b,\ 1,\ S_1) = q(b,\ 1,\ S_1) \times p(b) = 2 \times 2 = 4$. Meanwhile, the utility of the first q-itemset of S_1 is $u(X_1, S_1) = 4 + 4 = 8$, and that of S_1 is $u(S_1) = 8 + 8 + 7 = 23$. Finally, the utility of the the database D shown in Table 1 is $u(D) = 23 + 18 + 19 + 21 + 25 = 106$.

SID	**q-sequence**
S_1	<{(b:2) (f:4)}, {(a:2) (e:2)}, {(c:2) (e:1)}>
S_2	<{(a:1)}, {(c:2) (d:1)}, {(a:1) (b:1) (e:2)}>
S_3	<{(b:2) (f:2)}, {(f:2)}, {(a:3) (d:1)}>
S_4	<{(d:1)}, {(b:4) (f:5)}, {(c:1) (e:2)}, {(f:1)}>
S_5	<{(a:2)}, {(a:1) (c:3)}, {(c:1) (f:2)}, {(b:1)}>

Table 1: Running example of a q-sequence database

Item	a	b	c	d	e	f
External utility	3	2	3	2	1	1

Table 2: External utility table

Definition 4 (matching). *Given an itemset $X = \{i_1, i_2, \cdots, i_m\}$ and a q-itemset $Y = \{(j_1{:}q_1)\ (j_2{:}q_2) \cdots\ (j_m{:}q_m)\}$, we say that X is the matching of Y, denoted by $X \sim Y$, if and only if $i_k = j_k$ for $1 \leq k \leq m$. Similarly, given a sequence $S = <X_1, X_2, \cdots, X_n>$, and a q-sequence $Q = <Y_1, Y_2, \cdots, Y_n>$, we say that S is the matching of Q, denoted by $S \sim Q$, if and only if $X_k \sim Y_k$ for $1 \leq k \leq n$.*

Definition 5 (instance). *Given a sequence $S = <X_1, X_2, \cdots, X_m>$ and a q-sequence $Q = <Y_1, Y_2, \cdots, Y_n>$, where $m \leq n$. If there exists an integer p, $m \leq p \leq n$, such that $X'_k \sim Y_{p-m+k} \wedge X_k \subseteq X'_k$ for $1 \leq k \leq m$ (i.e., S is the contiguous sub-sequence of the matching of Q), we say that Q has an instance of S at ending position p. It should be noted that Q may have several instances of S, which correspond to different ending positions. We denote the set of these ending positions as $EP(S, Q)$. In addition, we say that Q contains S if Q has at least one instance of S, which is denoted by $S \sqsubseteq Q$.*

For example, itemset $\{bf\}$ is the matching of $\{(b{:}2)\ (f{:}4)\}$. Sequence $<\{bf\}, \{ae\}, \{ce\}>$ is the matching of S_1. Given a sequence $S = <\{a\}, \{c\}>$, S_5 has instances of S at ending positions 2 and 3, respectively. Therefore, $EP(S,\ S_5) = \{2, 3\}$. In addition, we say that S_5 contains S and denote it as $S \sqsubseteq S_5$.

Definition 6 (utility of instance). *Given an itemset X, a sequence $S = <X_1, X_2, \cdots, X_m>$, and a q-sequence $Q = <Y_1, Y_2, \cdots, Y_n>$,where $m \leq n$. The utility of X in the j-th q-itemset in Q is defined as $u(X, j, Q) = \sum_{i \in X} u(i,\ j,\ Q)$. Assuming that Q has an instance of S at the ending position p, the utility of this instance is calculated as $u(S,\ p,\ Q) = \sum_{j=1}^{m} u(X_j,\ p - m + j,\ Q)$. The utility of S in Q is defined as the maximal utility among all instances of S in Q; that is, $u(S, Q) = \max\{u(S,\ p,\ Q) | \forall p \in EP(S, Q)\}$. Finally, the utility of S in database D is defined as $u(S) = \sum_{Q \in D} u(S, Q)$.*

For example, the utility of $\{ab\}$ in the third q-itemset of S_2 is $u(\{ab\}, 3, S_2) = 3 + 2 = 5$. Given a sequence $S = <\{a\}, \{c\}>$, $u(S, 2, S_5) = 6 + 9 = 15$ and $u(S, 3, S_5) = 3 + 3 = 6$ can be obtained. Therefore, $u(S, S_5) = \max\{15, 6\} = 15$. Finally, considering the q-sequence database given in Table 1, we have $u(S) = 12 + 9 + 15 = 36$.

3.2. Problem Statement

Definition 7 (high-utility contiguous sequential pattern). *In a q-sequence database D, a sequence S is called a HUCSP, if it is the contiguous sub-sequence of some sequences in D and satisfies that u(S) ≥ ξ × u(D), where ξ is the minimum utility threshold that is given as a percentage.*

Problem Statement Given a q-sequence database D, an external utility table and a minimum utility threshold ξ, the problem of UCSPM is to identify the complete set of HUCSPs in D.

To illustrate the problem of UCSPM clearly, an example is given as follows: Considering the q-sequence database D given in Table 1, we have $u(D) = 106$. When $\xi = 25\%$, the minimum utility threshold value is $25\% \times 106 = 26.5$, and the discovered HUCSPs are $<\{bf\}>$ and $<\{a\}, \{c\}>$ with a utility of 27 and 36, respectively. Evidently, $<\{bf\}>$ is a contiguous sub-sequence of S_1, S_3, S_4, and $<\{a\}, \{c\}>$ is a contiguous sub-sequence of S_1, S_2, and S_5.

4. Proposed Method

This section describes the proposed FUCPM algorithm for mining HUCSPs in a pattern-growth manner. FUCPM utilizes the SIL and instance chain data structures to avoid scanning the q-sequence database repeatedly when calculating the utility of candidate patterns. Two powerful pruning strategies (i.e., GUIP and LUIP) are used to reduce the search space. The details of the above data structures, pruning strategies, and the FUCPM algorithm are provided in the following section. To facilitate the discussion in this section, we first define the following.

Definition 8 (extension). *Given a sequence S and an item i, the I-extension sequence of S, denoted by $<S \bigoplus i>$, appends i to the last itemset of S. The S-extension sequence of S, denoted by $<S \bigotimes i>$, adds i to a new itemset and then appends the new itemset to the end of S. For brevity, the I-extension*

sequence and S-extension sequence are collectively referred to as extension sequences hereinafter.

For example, given a sequence $S = <\{a\}, \{c\}>$, and an item d, then $<\{a\}, \{cd\}>$ is the I-extension sequence of S, and $<\{a\}, \{c\}, \{d\}>$ is the S-extension sequence of S.

Definition 9 (extension item). *Given a sequence S whose last item is i, a q-sequence Q, and a q-sequence database D, we assume that Q has n instances of S and $EP(S, Q) = \{ep_1, ep_2, \cdots, ep_n\}$. The set of I-extension items of S in Q, denoted by $Iitem(S, Q)$, comprises the items that appear in the ep_1, ep_2, $\cdots$, ep_n-th itemset of Q and are lexicographically larger than i. The set of S-extension items of S in Q, denoted by $Sitem(S, Q)$, is composed of the items appearing in the $(ep_1 + 1)$, $(ep_2 + 1)$, $\cdots$, $(ep_n + 1)$-th itemset of Q. Furthermore, the set of I-extension/S-extension items of S in D is defined as $Iitem(S) = \bigcup_{Q \in D} Iitem(S, Q)$ and $Sitem(S) = \bigcup_{Q \in D} Sitem(S, Q)$, respectively.*

For example, the sets of I-extension/S-extension items of $<\{a\}>$ in S_2 are $Iitem(<\{a\}>, S_2) = \{b, e\}$, and $Sitem(<\{a\}>, S_2) = \{c, d\}$. Furthermore, given the database shown in Table 1, we have $Iitem(<\{a\}>) = \{b, c, d, e\}$ and $Sitem(<\{a\}>) = \{a, c, d, e, f\}$.

Definition 10 (remaining sequence and remaining utility). *Given a sequence S and a q-sequence Q, we assume that Q has an instance of S at the ending position p. The remaining sequence of Q with respect to such an instance is denoted by $Q/_{(S,p)}$ and is defined as a suffix sequence of Q, which begins from the item after the last item of the instance in Q to the end of Q. Furthermore, the utility of the remaining sequence is called the remaining utility and is defined as $ru(Q/_{(S,p)}) = \sum_{i \in Q/_{(S,p)}} u(i)$.*

For example, the remaining sequence of the instance of $<\{a\}, \{c\}>$ at ending position 2 in S_5 is $S_5/_{(<\{a\},\{c\}>,2)} = <\{(c{:}1)\ (f{:}2)\}, \{(b{:}1)\}>$, and the corresponding remaining utility is $u(S_5/_{(<\{a\},\{c\}>,2)}) = 3 + 2 + 2 = 7$.

4.1. Data Structures

As mentioned in the introduction, the main challenge of UCSPM is the existence of numerous candidate patterns. In addition, each candidate pattern may appear multiple times in a q-sequence. Therefore, to calculate

the utility of a candidate pattern, the whole q-sequence database should be scanned to find all its instances in each q-sequence. Clearly, the scanning process requires a long execution time. To address this problem, we proposed using SIL to represent the original database and using instance-chain (IChain) to store the utility and positions of instances of a candidate pattern. The details of SIL and IChain are described below.

SIL stores information of the q-sequence, including the utility of each q-item and the remaining utility related to each q-item. Table 3 shows SIL of S_1 in Table 1. In SIL of S_1, each curly bracket ({}) corresponds to a q-itemset. The element (b, 4, 19) within the first curly bracket indicates that the utility of item b is four, and the remaining utility of S_1 with respect to this item is 19. The storage of utility and remaining utility in the SIL benefits the calculation of utility and upper bound value of candidate patterns, which will be discussed later.

IChain stores the utility and ending positions of all instances of a candidate pattern. To facilitate the following description, the q-sequence Q is assumed to have n instances of candidate pattern S. The set of these ending positions is $EP(S, Q) = \{ep_1, ep_2, \cdots, ep_n\}$. IChain comprises several instance lists, each of which corresponds to a q-sequence containing the candidate pattern. For the example of S and Q, the instance list contains the SID of Q, as well as a list of n elements. The i-th element contains two fields: (1) *EPos*, which is the ending position of the i-th instance of S in Q (i.e., ep_i) and (2) *Utility*, which is the utility of the i-th instance of S in Q. In the following, we use a tuple (*EPos*, *Utility*) to represent an element in instance list. The IChain of all 1-sequences can be constructed by scanning the SIL once. Figure 3 shows the IChain of sequence $<\{a\}>$ in the q-sequence database given in Table 1.

SID	Sequence information list
S_1	$<\{(b, 4, 19)\ (f, 4, 15)\}, \{(a, 6, 9)\ (e, 2, 7)\}, \{(c, 6, 1)\ (e, 1, 0)\}>$

Table 3: Sequence information list of S_1

The IChain for k-sequences, where $k > 1$, is constructed recursively based on the IChain of its prefix with a length of k-1. Specifically, given sequences S and $S' = <S \bigoplus i'>$, to construct the IChain of S', we must scan the IChain of S using the following method. For a certain element (*EPos*, *Utility*) in the

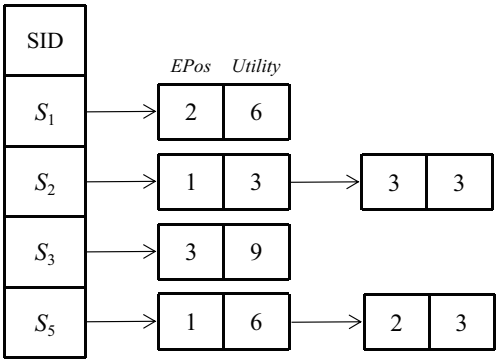

Figure 3: The IChain of $<a>$

instance list of S corresponding to a q-sequence Q, we check the SIL of Q whether item i' exists in the $EPos$-th itemset. If there exists, we construct a new element ($EPos$, $Utility + u(i', EPos, Q)$) and insert it into the instance list of S' corresponding to Q. After traversing all elements of the IChain of S in the same manner, the IChain of S' can be built. For an S-extension sequence $S'' = <S \bigotimes i''>$, the construction method of the IChain of S'' is similar to that of S'. The only difference is that we check whether i'' exists in the $(EPos + 1)$-th itemset of Q. If there exists, we construct an element ($EPos + 1$, $Utility + u(i'', EPos + 1, Q)$) and insert it to the instance list of S''. Figure 4 shows the IChain of sequence $<\{a\}, \{c\}>$ in the q-sequence database given in Table 1.

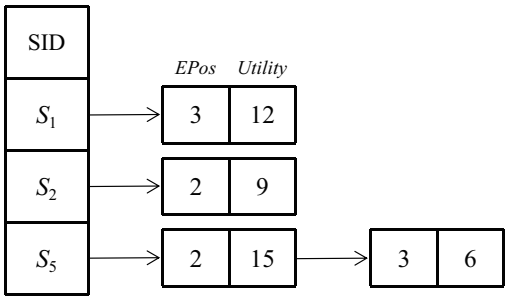

Figure 4: The IChain of $<\{a\}, \{c\}>$

4.2. Pruning Strategies

In the SPM and HUSPM tasks, the combinatorial explosion of the search space is a classic problem, which is caused by the sequential ordering of itemsets. Many SPM algorithms adopt the downward closure property to prune the search space. This property states that, if a sequence is infrequent, then all its super-sequences are also infrequent. However, this property only holds for the frequency, but does not hold for utility. To address this problem, researchers have proposed using upper bounds on utility to prune the search space in HUSPM. An upper bound on utility must satisfy the following two properties: (1) overestimate property (i.e., the upper bound value of a sequence overestimates the utility of this sequence) and (2) the downward closure property (i.e., the upper bound value of a sequence is no less than that of its extension sequences). Sequence-weighted utilization (SWU) [10] is a commonly used upper bound in HUSPM. Inspired by SWU, we develop a GUIP strategy to prune low-utility 1-sequences (also called global unpromising items) in the early stage of mining HUCSPs. Furthermore, we proposed a novel upper bound called item-extension utilization (IEU) that considers the contiguous constraint and a corresponding pruning strategy to eliminate local unpromising items for each candidate during the entire mining process. Subsequently, we introduce the details of these two upper bounds and the corresponding pruning strategies.

Definition 11 (sequence-weighted utilization). *Given a sequence S and a q-sequence database D, the sequence-weighted utilization (SWU) of S in D is defined as $SWU(S) = \sum_{S \sqsubseteq Q \wedge Q \in D} u(Q)$.*

For example, in Table 1, $SWU(a) = u(S_1) + u(S_2) + u(S_3) + u(S_5) = 23 + 18 + 19 + 25 = 85$, and $SWU(d) = u(S_2) + u(S_3) + u(S_4) = 58$.

Theorem 1 (overestimate property of SWU). *Given a sequence S and a q-sequence database D, it can be obtained that $u(S) \leq SWU(S)$.*

Proof. For any q-sequence Q, we can obtain that $u(S, Q) \leq u(Q)$. Therefore, $u(S) = \sum_{S \sqsubseteq Q \wedge Q \in D} u(S, Q) \leq \sum_{S \sqsubseteq Q \wedge Q \in D} u(Q) = SWU(S)$.

Theorem 2 (downward closure property of SWU). *Given two sequences S and S', and a q-sequence database D, if S' is a super-sequence of S (i.e., $S \subseteq S'$), then $SWU(S') \leq SWU(S)$.*

PROOF. Because $S \subseteq S'$, we have $\{Q|S' \sqsubseteq Q \wedge Q \in D\} \subseteq \{Q|S \sqsubseteq Q \wedge Q \in D\}$. Therefore, we can obtain that $SWU(S') = \sum_{S' \sqsubseteq Q \wedge Q \in D} u(Q) \leq \sum_{S \sqsubseteq Q \wedge Q \in D} u(Q) = SWU(S)$.

Some existing HUSPM algorithms [14, 15] utilize SWU to prune low-utility 1-sequence. That is, if the SWU value of a 1-sequence is less than the minimum utility threshold, then this 1-sequence and all its super-sequences cannot be HUSPs. In other words, the unique item in the low-utility 1-sequence is a global unpromising item, and the sequences containing this item cannot be HUSPs. Numerous unpromising candidate patterns can be pruned using the SWU. However, existing algorithms only use the strategy once, which cannot eliminate global unpromising items thoroughly. Hence, we propose a more effective pruning strategy called GUIP.

Strategy 1 (GUIP strategy). *The GUIP strategy is a recurrent process: (1) for each item i in the q-sequence database, if $SWU(i) < \xi \times u(D)$, then remove i from the database; (2) update the utility of each q-sequence and the SWU of the remaining items; (3) go to (1) until, for any remaining item i, $SWU(i) \geq \xi \times u(D)$.*

In addition to the GUIP strategy, we further propose the IEU upper bound and the corresponding LUIP strategy to prune the local unpromising items of each candidate pattern.

Definition 12 (item-extension utilization). *Given a sequence α and a q-sequence Q containing α, assume that S is the extension sequence of α where the extension item is i. The p-th itemset of Q is denoted by Q^p.*

i) For I-extension (i.e., $S = \alpha \bigoplus i$), the IEU of S in Q with respect to the ending position p, $p \in EP(\alpha, Q)$, is defined as

$$IEU(S, p, Q) = u(\alpha, p, Q) + u(i, p, Q) + ru(Q/_{(i,p)}) \tag{1}$$

if and only if $i \in Q^p$; otherwise, $IEU(S, p, Q) = 0$.

ii) For S-extension (i.e., $S = \alpha \bigotimes i$), the IEU of S in Q with respect to the ending position p, $p \in EP(\alpha, Q)$, is defined as

$$IEU(S, p, Q) = u(\alpha, p, Q) + u(i, p+1, Q) + ru(Q/_{(i,p+1)}) \tag{2}$$

if and only if $i \in Q^{p+1}$; otherwise, $IEU(S, p, Q) = 0$.

iii) For both I-extension and S-extension, the IEU of S in Q is defined as

$$IEU(S,Q) = \max_{p \in EP(\alpha,Q)} IEU(S,p,Q) \tag{3}$$

The IEU of S in q-sequence database D is defined as

$$IEU(S) = \sum_{S \sqsubseteq Q \wedge Q \in D} IEU(S,Q) \tag{4}$$

For example, in Table 1, consider the sequence $\alpha = <a>$, I-extension item e and S-extension item c. Then, for the I-extension sequence $S = <\{ae\}>$, we have $IEU(S,2,S_1) = 6 + 2 + 7 = 15$, $IEU(S,2,S_2) = 3 + 2 + 0 = 5$. Finally, $IEU(S) = IEU(S,S_1) + IEU(S,S_2) = 15 + 5 = 20$.

For the S-extension sequence $S' = <\{a\},\{c\}>$, we have $IEU(S',2,S_1) = 6 + 6 + 1 = 13$, $IEU(S',1,S_2) = 3 + 6 + 9 = 18$, $IEU(S',1,S_5) = 6 + 9 + 7 = 22$ and $IEU(S',2,S_5) = 3 + 3 + 4 = 10$. The IEU value of S' in S_5 is $IEU(S',S_5) = \max\{22,\ 10\} = 22$. Finally, $IEU(S') = IEU(S',S_1) + IEU(S',S_2) + IEU(S',S_5) = 13 + 18 + 22 = 53$.

Theorem 3 (overestimate property of *IEU*). *Given a sequence S and a q-sequence database D, we can obtain that $u(S) \leq IEU(S)$.*

Proof. It is assumed that S is the extension sequence of α where the extension item is i.

i) For I-extension, we have

$$\begin{aligned} u(S,Q) &= \max_{p \in EP(\alpha,Q)} u(S,p,Q) \\ &= \max_{p \in EP(\alpha,Q)} u(\alpha,p,Q) + u(i,p,Q) \\ &\leq \max_{p \in EP(\alpha,Q)} u(\alpha,p,Q) + u(i,p,Q) + ru(Q/_{(i,p)}) \\ &= IEU(S,Q). \end{aligned}$$

ii) For S-extension, we have

$$\begin{aligned} u(S,Q) &= \max_{p \in EP(\alpha,Q)} u(S,p+1,Q) \\ &= \max_{p \in EP(\alpha,Q)} u(\alpha,p,Q) + u(i,p+1,Q) \\ &\leq \max_{p \in EP(\alpha,Q)} u(\alpha,p,Q) + u(i,p+1,Q) + ru(Q/_{(i,p+1)}) \\ &= IEU(S,Q). \end{aligned}$$

iii) For both I-extension and S-extension, $u(S) = \sum_{S \sqsubseteq Q \wedge Q \in D} u(S, Q) \leq \sum_{S \sqsubseteq Q \wedge Q \in D} IEU(S, Q) = IEU(S)$ can be obtained.

Theorem 4 (downward closure property of *IEU*). *Given two sequences S and S', and a q-sequence database D, if S' is the extension sequence of S, then $IEU(S') \leq IEU(S)$ can be obtained.*

Proof. It is assumed that S' is the extension sequence of S, where the extension item is i', and S is the extension sequence of α, where the extension item is i. According to Definition 9, i' is arranged after i in any q-sequence Q that contains both S and S'.

i) First, if $S = \alpha \bigoplus i$, for I-extension (i.e., $S' = S \bigoplus i'$), we have

$$\begin{aligned} IEU(S', Q) &= \max_{p' \in EP(S,Q)} u(S, p', Q) + u(i', p', Q) + ru(Q/_{(i',p')}) \\ &\leq \max_{p' \in EP(S,Q)} u(S, p', Q) + ru(Q/_{(i,p')}) \\ &= \max_{p \in EP(\alpha,Q)} u(\alpha, p, Q) + u(i, p, Q) + ru(Q/_{(i,p)}) \\ &= IEU(S, Q). \end{aligned}$$

ii) For S-extension (i.e., $S' = S \bigotimes i'$), we have

$$\begin{aligned} IEU(S', Q) &= \max_{p' \in EP(S,Q)} u(S, p', Q) + u(i', p' + 1, Q) \\ &\qquad + ru(Q/_{(i',p'+1)}) \\ &\leq \max_{p' \in EP(S,Q)} u(S, p', Q) + ru(Q/_{(i,p')}) \\ &= \max_{p \in EP(\alpha,Q)} u(\alpha, p, Q) + u(i, p, Q) + ru(Q/_{(i,p)}) \\ &= IEU(S, Q). \end{aligned}$$

Similarly, it can be proved that $IEU(S', Q) \leq IEU(S, Q)$ if $S = \alpha \bigotimes i$. Because S' is the extension sequence of S, we have $\{Q | S' \sqsubseteq Q \wedge Q \in D\} \subseteq \{Q | S \sqsubseteq Q \wedge Q \in D\}$. Therefore, $IEU(S') = \sum_{S' \sqsubseteq Q \wedge Q \in D} IEU(S', Q) \leq \sum_{S' \sqsubseteq Q \wedge Q \in D} IEU(S, Q) \leq \sum_{S \sqsubseteq Q \wedge Q \in D} IEU(S, Q) = IEU(S)$.

Strategy 2 (LUIP strategy). *Let S be a candidate pattern; if $IEU(S) < \xi \times u(D)$, then S and all its extension sequences can be pruned from the search space. That is, assume that S is the extension sequence of sequence α, where the extension item is i, if $IEU(S) < \xi \times u(D)$, then i is a local unpromising item for α and can be pruned.*

4.3. Proposed FUCPM Algorithm

Based on the aforementioned data structures and pruning strategies, the proposed FUCPM algorithm is described as follows. Algorithm 1 shows the pseudocode of the main procedure of FUCPM, which takes a q-sequence database D, an external utility table EUT, and a minimum utility threshold ξ as the inputs. FUCPM first scans D to calculate the utility of each q-sequence in D and the utility of D. Further, FUCPM follows the GUIP strategy to delete global unpromising items and obtain the revised database D' (line 1). Then, FUCPM scan D' to construct the SIL of each q-sequence and IChain of each 1-sequence (line 2). The utility of each 1-sequence is obtained from its IChain, and FUCPM determines whether the 1-sequence is a HUCSP (lines 4–6). Subsequently, FUCPM calls the Recursive-Search procedure to discover longer HUCSPs recursively (line 7). Finally, the algorithm returns the complete set of HUCSPs (line 9).

Algorithm 1 FUCPM algorithm

Input: D: a q-sequence database; EUT: external utility table; ξ: minimum utility threshold.
Output: *HUCSPs*: the set of high-utility contiguous sequential patterns.
1: scan D to:
(1) calculate $u(Q)$ for each $Q \in D$ and calculate $u(D)$;
(2) calculate $SWU(i)$ for each item i recurrently and obtain the revised database D' by deleting the item whose SWU is less than $\xi \times u(D)$; // *The GUIP strategy*
2: scan D' to:
(1) construct the SIL of each q-sequence;
(2) construct the IChain of each 1-sequence;
3: **for** each $S \in$ 1-sequences **do**
4: **if** $u(S) \geq \xi \times u(D)$ **then**
5: $HUCSPs \leftarrow HUCSPs \cup S$;
6: **end if**
7: call ***Recursive-Search***$(S, SIL, S.$***IChain***$)$;
8: **end for**
9: **return** *HUCSPs*

Algorithm 2 presents the details of the Recursive-Search procedure that recursively excavates HUCSPs in a depth-first search manner. It takes a prefix sequence S, the SIL of all q-sequences, and the IChain of S as the inputs. First, the procedure scans the IChain of S and SIL to obtain the sets

Algorithm 2 Recursive-Search

Input: S: a sequence as the prefix; SIL: the SIL of all q-sequences; $S.IChain$: the IChain of S.

```
1: for each instance list il ∈ S.IChain do
2:    obtain SIL of the q-sequence whose SID is il.SID;
3:    scan SIL to obtain the set of I-extension items Iitem(S);
4:    scan SIL to obtain the set of S-extension items Sitem(S);
5: end for
6: for each item i ∈ Iitem(S) do
7:    S' ← <S ⊕ i>;
      // The LUIP strategy
8:    if IEU(S') ≥ ξ × u(D) then
9:       construct the IChain of S';
10:      if u(S') ≥ ξ × u(D) then
11:         HUCSPs ← HUCSPs ∪ S';
12:      end if
13:      call Recursive-Search(S', SIL, S'.IChain);
14:   end if
15: end for
16: for each item i ∈ Sitem(S) do
17:   S' ← <S ⊗ i>;
      // The LUIP strategy
18:   if IEU(S') ≥ ξ × u(D) then
19:      construct the IChain of S';
20:      if u(S') ≥ ξ × u(D) then
21:         HUCSPs ← HUCSPs ∪ S';
22:      end if
23:      call Recursive-Search(S', SIL, S'.IChain);
24:   end if
25: end for
```

of I-extension and S-extension items of S in D (i.e., $Iitem(S)$ and $Sitem(S)$ (lines 1–5)). Next, the procedure processes each item i in $Iitem(S)$ (lines 6–16). The extension sequence of S is generated by performing I-extension on S with i (line 7). Next, the IEU value of S' is calculated. Note that $IEU(S')$ has three components: (1) utility of S, (2) utility of i, and (3) utility of the remaining sequence with respect to i. $IEU(S')$ is easy to calculate because (1) is stored in the IChain of S, and (2) and (3) can be obtained directly from the SIL. If $IEU(S') \geq \xi \times u(D)$, we construct the IChain of S' based

on the IChain of S(line 9). Then, S' is evaluated to determine whether it is a HUCSP (lines 10–12). Note that $u(S')$ can be obtained from the IChain of S'. Finally, the procedure invokes itself recursively to generate and examine new candidate patterns with S' as their prefix (line 13). The items in *Sitem* can be processed using a similar procedure (lines 15–25).

Complexity analysis. Let $|D|$ denote the number of q-sequences in a database D, L denote the average length of the q-sequence in D, F denote the average number of items following the certain prefix in a q-sequence in D, M denote the number of itemsets of the q-sequence containing the most items in D, and $|I|$ denote the number of promising items. The first step of the FUCPM algorithm is to scan D to calculate $u(Q)$ and SWU(i), then scan D' to construct the SIL and IChain, whose time complexity is $O(|D|L)$. Then, we analyse the time and memory complexity of the following deeply recursive process. For each Recursive-Search, we need to finish the following steps: (1) Scanning the Iitem and SItem of this prefix, which time complexity is $O(|I|F)$; (2) Using the LUIP strategy to cut each unpromising candidate in Iitem and SItem and deeply call Recursive-Search. The worst-case time complexity of a Recursive-Search is $O(|D||I|M + |I|F)$ because there are at most $|D||I|$ items in IChain. In general, the time complexity of the whole algorithm is $O(|D|L + |D||I|^2M + |I|^2F)$ and the memory complexity is $O(|D||L| + |D|F(|I| + 1))$.

5. Experiments

This section presents the experimental results of the compared algorithms to show the advantages of FUCPM in the UCSPM task. The following algorithms were selected as baseline algorithms.

1. HUCP-Miner [28]: This algorithm adopts the remaining utility upper-bound ($RUUB$) to prune the search space and a data structure called UL-list to store necessary information of candidate patterns. HUCP-Miner is currently the most efficient algorithm for the UCSPM task. However, it can only handle the simple single-item-based sequences.
2. HUSP-ULL*: This is the adapted version of the state-of-the-art HUSPM algorithm HUSP-ULL [15]. We adapted HUSP-ULL by including some constraints while performing extension operations, making it possible to discover HUCSPs from databases. HUSP-ULL* utilizes a compressed utility-linked list structure to store information of patterns. Be-

sides, two pruning strategies called look-ahead removing (LAR) and irrelevant item pruning (IIP) are used to eliminate low-utility sequences.

All the above algorithms were implemented in Java. We conducted substantial experiments on both real-world and synthetic datasets for the following purposes.

1. Compare the efficiency of the compared algorithms in terms of runtime, memory consumption, and candidates pruning.
2. Evaluate the effectiveness of the proposed GUIP and LUIP strategies.
3. Verify the scalability of the compared algorithms on large-scale multi-items-based sequence datasets.

The experiments were performed on a personal computer equipped with an Intel(R) Core(TM) i7-8700 CPU @3.20 GHz processor and 16 GB of RAM, running a 64-bit Microsoft Windows 10 operating system.

Dataset	***#Seq***	***#Item***	***maxLen***	***avgLen***	***avgItem***
Bible	36,369	13,905	100	21.64	1.00
Leviathan	5,834	9,025	100	33.81	1.00
BMS	77,512	3,340	267	4.62	1.00
MSNBC	31,790	17	100	13.33	1.00
Kosarak10k	10,000	10,094	608	8.14	1.00
FIFA	20,450	2,990	100	34.74	1.00
Syn40k	40,000	7,584	18	6.19	4.32
Syn80k	79,718	7,584	18	6.19	4.32
Syn160k	159,501	7,609	20	6.19	4.32
Syn240k	239,211	7,617	20	6.19	4.32
Syn320k	318,889	7,620	20	6.19	4.32
Syn400k	398,716	7,621	20	6.18	4.32

Table 4: Features of the datasets

5.1. Dataset Description

In the experiments, six real-world datasets and six synthetic datasets were used to evaluate the performance of the algorithms. The detailed characteristics of the datasets are listed in Table 4. Note that *#Seq* is the number of

sequences; *#Item* is the number of distinct items; *maxLen* and *avgLen* are the maximum and average length of the sequences, respectively; and *avgItem* is the average number of items contained in each itemset. We can observe that the *avgItem* of the six real-world datasets are all equal to 1 because these datasets are composed of single-item-based sequences. Conversely, the synthetic datasets are composed of multi-items-based sequences, where each itemset contains more than four items on average. The detailed descriptions of the datasets are as follows. (1) *Bible* and *Leviathan* are conversions of the Bible and novel Leviathan. Each word in the books is transformed into an item, and each sentence is treated as a sequence. (2) *BMS*, *MSNBC*, *Kosarak10k*, and *FIFA* are clickstream data from an e-commerce website, the MSNBC website, the website of FIFA World Cup 98, and a Hungarian news portal, respectively. (3) The synthetic datasets (from *Syn40k* to *Syn400k*) were generated by the IBM Quest Dataset Generator [50]. We assigned utility values to the synthetic datasets according to the simulation method used in the previous study [15]. All the real-world datasets were obtained from the open-source data mining library SPMF [1]. It is noted that the Leviathan dataset does not contain utility values and thus we also used the simulation method to assign utility for this dataset.

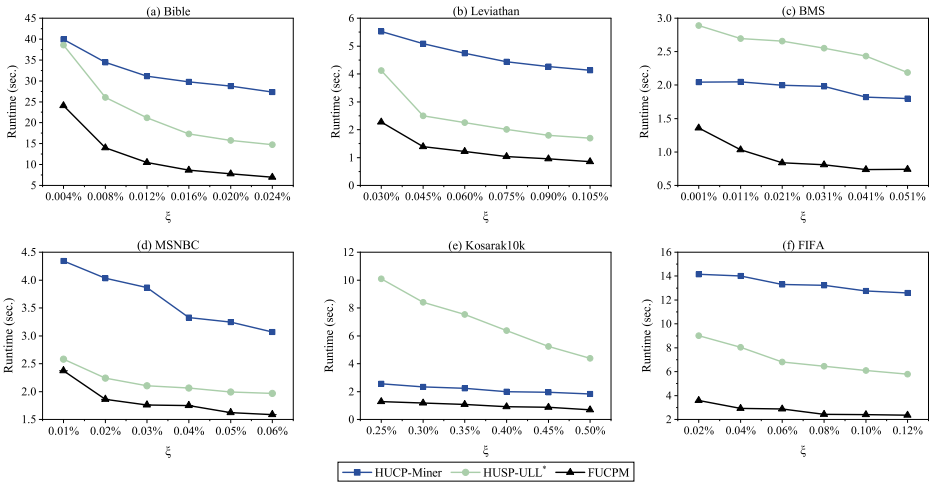

Figure 5: Runtime of the compared algorithms under various minimum utility thresholds

[1]http://www.philippe-fournier-viger.com/spmf/

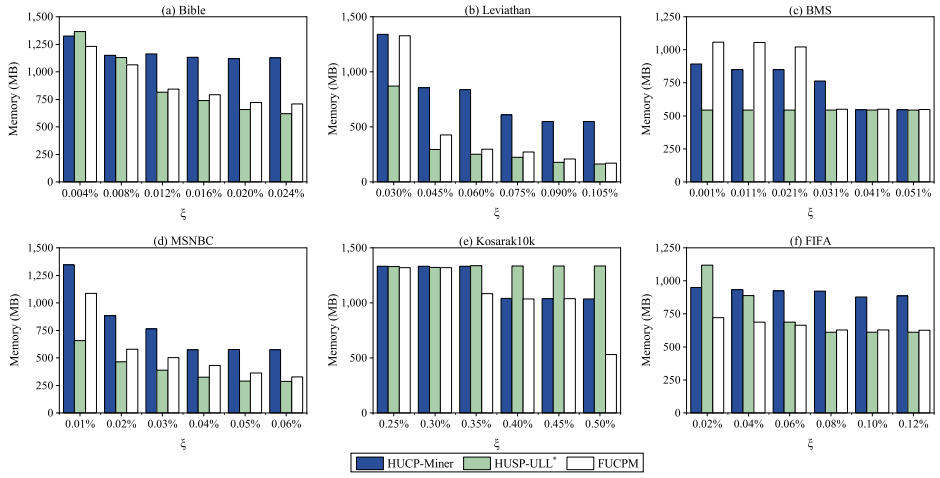

Figure 6: Memory consumption of the compared algorithms under various minimum utility thresholds

5.2. Runtime Analysis

Figure 5 shows the runtime of the compared algorithms under various threshold values on the six real-world datasets. As it can be observed, FUCPM outperformed the two baselines for all datasets under various parameter settings. The advantage of FUCPM is clearer on *Bible* and *FIFA* datasets. For example, in Figure 5 (a), FUCPM only took 7.5 seconds to obtain the results when the threshold is set to 0.024%. In contrast, the runtime of HUSP-ULL* and HUCP-Miner were twice and four times more than that of FUCPM. The reason for this is that the average length of q-sequences of *Bible* and *FIFA* is relatively longer, and the proposed GUIP and LUIP strategies can prune unpromising items earlier to prevent the candidate sequences from growing too long. On the other four datasets, although all the three algorithms completed the mining process within ten seconds, FUCPM still demonstrated a 10%-50% improvement in runtime. Furthermore, we can observe that the runtime decreased as the threshold value increased for all the algorithms. This is because, under larger thresholds, more candidate sequences can be pruned earlier given that their upper bound value does not exceed the thresholds. In general, the prominent advantage in terms of runtime reveals that FUCPM benefits from the proposed pruning strategies and the designed data structure. Besides, FUCPM is good at handling datasets that contain long sequences.

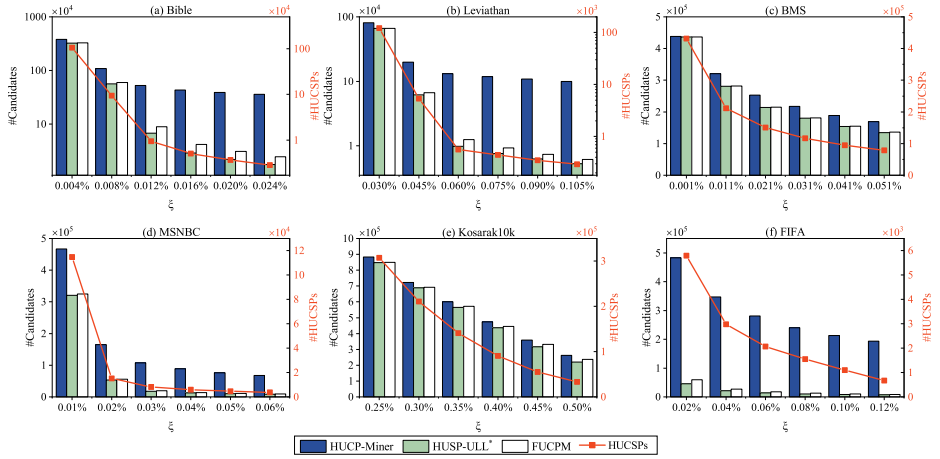

Figure 7: Candidates and patterns of the compared algorithms under various minimum utility thresholds

5.3. Memory Analysis

The memory usage of FUCPM and HUCP-Miner is displayed in Figure 6. As it can be observed, as the threshold value increased, the memory consumption of the three algorithms showed a decreasing trend. This is because as the threshold increased, fewer candidate sequences were generated, and the memory consumed on the data structure for storing information of candidates also reduced. From Figure 6, it is clear that the HUCP-Miner algorithm had the worst memory performance in most cases. For example, on the *Bible* dataset, the memory consumption of HUCP-Miner was large and steady as the threshold increased, while the memory consumed by HUSP-ULL* and FUCPM gradually decreased.

It is an interesting phenomenon that HUSP-ULL* outperformed the other two algorithms in terms of memory on many datasets. For example, on the *Leviathan* dataset, FUCPM consumed approximately 400 MB less memory than HUCP-Miner under most threshold settings, while HUSP-ULL* consumed even less memory than FUCPM. The similar case also occurred on *Bible*, *BMS* and *MSNBC*. This is because HUSP-ULL* adopts a compressed utility-linked list data structure. However, we can observe that FUCPM defeated the two baselines on *Kosarak10k* and *FIFA*. It is also interesting that FUCPM used more memory than the two baselines when the thresholds were relatively low on *BMS*. We speculate that this is because FUCPM required more memory to store the SIL structure of a large number of q-sequences in

BMS, although it reduced the memory consumed on the instance chains of fewer candidates. In summary, by adopting efficient pruning strategies and compact data structures, FUCPM can achieve better performance in terms of memory consumption than HUCP-Miner and is superior to HUSP-ULL* on some datasets.

5.4. Candidate Analysis

The number of candidate patterns generated by the algorithm is an important measure of the size of the search space, which reflects the ability of the pruning strategies. In Figure 7, histograms are used to represent the number of generated candidate patterns and broken lines are used to represent the number of discovered HUCSPs. As shown in Figure 7, the number of candidates generated by HUCP-Miner was always the highest among the three algorithms. Especially on *BIBLE*, *Leviathan* and *FIFA*, the advantage of HUSP-ULL* and FUCPM was remarkable. This can explain why HUSP-ULL* and FUCPM performed much better in terms of runtime and memory consumption than HUCP-Miner on these datasets. Moreover, the number of candidates generated by HUSP-ULL* and FUCPM generated was almost the same in many cases, which indicates that the GUIP and LUIP strategies used in FUCPM have similar effects compared to the LAR and IIP strategies used in HUSP-ULL*. However, on *BMS* and *Kosarak10k*, the number of candidates generated by the compared algorithms did not show a significant difference. We infer that this is because the data volume of these two datasets, which is calculated as $\#Seq \times avgLen$, is relatively small, so the effect of pruning strategies is not quite evident.

We further introduce a new metric called the effective search rate (*ESR*), which is defined as $(\#HUCSPs \div \#Candidates) \times 100\%$, to evaluate the search efficiency of the algorithms. Clearly, the algorithm with a higher *ESR* value can reduce the unnecessary search for low-utility candidates. The *ESR* values of the compared algorithms are listed in Table 5. We can observe that the *ESR* of FUCPM was close to that of HUSP-ULL*. In addition, the *ESR* of FUCPM was approximately 10% higher than that of HUCP-Miner under most threshold settings on *Bible*, *BMS*, and *FIFA*. The advantage of FUCPM was more apparent on *MSNBC*, whereas on *Leviathan* and *Kosarak10k*, the *ESR* of FUCPM was slightly higher than that of HUCP-Miner.

From the above discussions, we can conclude that the proposed GUIP and LUIP strategies can prune the search space more effectively than the *RUUB*-based strategy used in HUCP-Miner, and have similar effect compared to the

Bible	ξ	0.004%	0.008%	0.012%	0.016%	0.020%	0.024%
	HUCP-Miner (%)	27.86	8.67	1.81	1.17	0.94	0.79
	HUSP-ULL* (%)	32.64	16.78	14.02	17.74	16.99	16.15
	FUCPM (%)	32.24	15.81	10.72	12.28	12.02	11.67
Leviathan	ξ	0.030%	0.045%	0.060%	0.075%	0.090%	0.105%
	HUCP-Miner (%)	14.98	2.68	4.27	3.74	3.23	2.92
	HUSP-ULL* (%)	18.32	8.62	5.73	5.81	5.57	5.52
	FUCPM (%)	18.17	8.04	4.51	4.75	4.74	4.72
BMS	ξ	0.001%	0.011%	0.021%	0.031%	0.041%	0.051%
	HUCP-Miner (%)	98.42	65.86	59.40	53.90	50.29	46.74
	HUSP-ULL* (%)	98.93	75.25	70.28	65.02	61.88	58.54
	FUCPM (%)	98.93	74.94	69.92	64.65	61.46	58.04
MSNBC	ξ	0.01%	0.02%	0.03%	0.04%	0.05%	0.06%
	HUCP-Miner (%)	24.57	9.17	7.56	6.63	5.91	5.50
	HUSP-ULL* (%)	35.77	28.26	45.40	45.74	44.68	44.85
	FUCPM (%)	35.30	27.00	41.94	42.30	41.19	41.36
Kosarak10k	ξ	0.25%	0.30%	0.35%	0.40%	0.45%	0.50%
	HUCP-Miner (%)	34.84	29.21	23.42	19.09	15.42	12.53
	HUSP-ULL* (%)	36.30	30.65	24.90	20.76	17.51	15.00
	FUCPM (%)	36.18	30.51	24.65	20.32	16.68	13.91
FIFA	ξ	0.02%	0.04%	0.06%	0.08%	0.10%	0.12%
	HUCP-Miner (%)	1.20	0.86	0.74	0.65	0.52	0.35
	HUSP-ULL* (%)	12.61	13.70	14.42	14.47	12.67	9.22
	FUCPM (%)	9.79	10.97	11.76	11.96	10.58	7.80

Table 5: Effective search rate of the compared algorithms

strategies used in HUSP-ULL*.

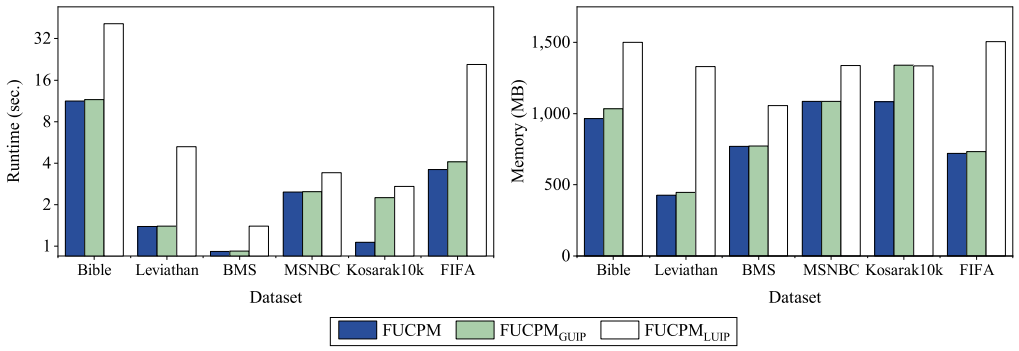

Figure 8: Effectiveness of pruning strategies

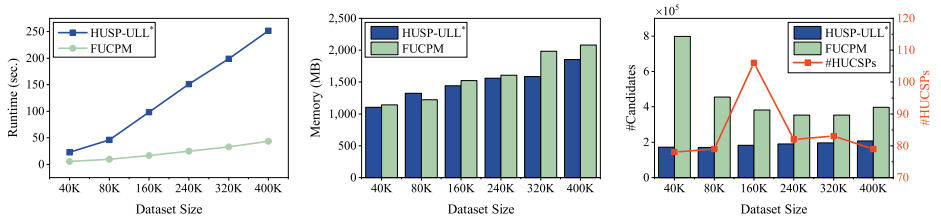

Figure 9: Scalability test of HUSP-ULL* and FUCPM

5.5. Effectiveness of Pruning Strategies

An ablation experiment was conducted on six real-world datasets to evaluate the effectiveness of the proposed pruning strategies. Figure 8 shows the performance in terms of runtime and memory consumption of the FUCPM and its two variants. Note that FUCPM$_{\text{GUIP}}$ and FUCPM$_{\text{LUIP}}$ are the algorithms that remove GUIP and LUIP strategies from FUCPM, respectively. The minimum utility thresholds were set to 0.01% for the six datasets. As shown in Figure 8, FUCPM is far more superior to FUCPM$_{\text{LUIP}}$ in terms of runtime for all datasets. In particular, compared to FUCP$_{\text{LUIP}}$, the advantage of FUCPM is much more apparent on *Bible*, *Leviathan*, and *FIFA*, which occupy a larger *avgLen*. This result demonstrates that the LUIP strategy is suitable for handling long sequences. From Figure 8, we can also observe that FUCPM consumed the least memory for each dataset. However, the runtime and memory consumption of FUCPM and FUCPM$_{\text{GUIP}}$ were almost equal

for *Leviathan*, *BMS*, and *MSNBC*. This is mainly because only a few global unpromising items exist and can be pruned on these datasets. In conclusion, the results of the ablation experiment present the contributions of the proposed GUIP and LUIP strategies to improve the efficiency of FUCPM, and the LUIP strategy is more effective than the GUIP strategy for pruning the search space.

5.6. Scalability Test

The scalability of the compared algorithms was tested on six synthetic datasets containing multi-items-based sequences with different sizes varying from 40 K to 400 K. HUCP-Miner was not tested in this experiment because it can not handle the multi-items-based sequences. Figure 9 displays the results in terms of runtime, memory consumption, and the number of candidates and HUCSPs when $\xi = 0.1\%$ for each test.

As shown in Figure 9, the runtime and memory consumption increased almost linearly as the dataset size increased for both algorithms. However, the runtime of FUCPM was far less than that of HUSP-ULL* and increased smoothly. In contrast, the runtime of HUSP-ULL* increased more dramatically. As for memory performance, the memory consumed by FUCPM was slightly more than that of HUSP-ULL* on different datasets. Furthermore, the number of candidate sequences generated by HUSP-ULL* was less than that of FUCPM. This is because HUSP-ULL* adopted two powerful pruning strategies to eliminate low-utility candidates. It is also interesting that the number of HUCSPs on *Syn160k* was the greatest. This phenomenon can be attributed to the distinctive utility distribution of this dataset. In summary, because the runtime and memory consumption were approximately linear with the dataset size, we can conclude that FUCPM has good scalability to handle large-scale multi-items-based sequence datasets.

6. Conclusion

HUSPM has been a problem of great interest in the domain of KDD. Nevertheless, HUSPM may suffer from discovering redundant and meaningless patterns. Recent research has recognized the critical role played by contiguous sequential patterns in some fields, such as vehicle trajectory analysis [20] and network protocol specification extraction [47]. In this paper, we focused on the problem of UCSPM, which aims to discover HUCSPs from sequence databases. To resolve this problem, we proposed an efficient and scalable

algorithm called FUCPM. Specifically, two compact data structures(i.e., sequence information list and instance chain) were designed to facilitate the calculation of utility and upper bound values of the candidate patterns. To further improve the efficiency, we proposed the GUIP and LUIP strategies to prune the search space, which are based on the *SWU* and the novel *IEU* upper bounds, respectively. Extensive experimental results on both real-world and synthetic datasets demonstrate that FUCPM outperforms the state-of-the-art algorithms for UCSPM and is scalable to large-scale and complex multi-items-based sequence datasets.

In the future, several extensions of FUCPM can be further researched, such as using FUCPM to discover on-shelf patterns [51], and designing the distributed and parallel version of FUCPM to better handle big data [11]. This work also provides the following insights for exploring the applications of FUCPM in various fields. Given that HUCSPs maintain the contiguous order of items in sequence data, FUCPM can be applied for next-items recommendation [29]. For some practical issues, such as text representation [52] and biological sequence discovery [53], where the adjacent relationship of items is significant, FUCPM can also come in handy.

Acknowledgements

This work was supported by Natural Science Foundation of Guangdong Province, China (Grant NO. 2024A1515010242), Guangdong Provincial Key Laboratory of Novel Security Intelligence Technologies, China (Grant NO. 2022B1212010005) and Shenzhen Research Council (Grant NO. Grant NO. GXWD20220811702530022).

References

[1] R. Agrawal, R. Srikant, Mining sequential patterns, in: Proceedings of the 11th International Conference on Data Engineering, IEEE, 1995, pp. 3–14.

[2] R. Srikant, R. Agrawal, Mining sequential patterns: Generalizations and performance improvements, in: Proceedings of the 5th International Conference on Extending Database Technology, Springer, 1996, pp. 1–17.

[3] Q. Wang, V. Sheng, X. Wu, Keyphrase extraction with sequential pattern mining, in: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 31, 2017.

[4] X. Yu, N. Ma, T. Yang, Y. Zhang, Q. Miao, J. Tao, H. Li, Y. Li, Y. Yang, A multi-level hypoglycemia early alarm system based on sequence pattern mining, BMC Medical Informatics and Decision Making 21 (1) (2021) 1–11.

[5] T. Wang, L. Duan, G. Dong, Z. Bao, Efficient mining of outlying sequence patterns for analyzing outlierness of sequence data, ACM Transactions on Knowledge Discovery from Data 14 (5) (2020) 1–26.

[6] H. H. Le, Y. Horino, T. Yamazaki, K. Araki, H. Yokota, Sequential pattern mining of large combinable items with values for a set-of-items recommendation, in: IEEE 34th International Symposium on Computer-Based Medical Systems, IEEE, 2021, pp. 56–61.

[7] C. Bin, T. Gu, Y. Sun, L. Chang, A personalized POI route recommendation system based on heterogeneous tourism data and sequential pattern mining, Multimedia Tools and Applications 78 (24) (2019) 35135–35156.

[8] W. Gan, J. C.-W. Lin, P. Fournier-Viger, H.-C. Chao, V. S. Tseng, P. S. Yu, A survey of utility-oriented pattern mining, IEEE Transactions on Knowledge and Data Engineering 33 (4) (2021) 1306–1327.

[9] C. F. Ahmed, S. K. Tanbeer, B.-S. Jeong, A novel approach for mining high-utility sequential patterns in sequence databases, ETRI Journal 32 (5) (2010) 676–686.

[10] J. Yin, Z. Zheng, L. Cao, USpan: an efficient algorithm for mining high utility sequential patterns, in: Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2012, pp. 660–668.

[11] M. Zihayat, Z. Z. Hut, A. An, Y. Hut, Distributed and parallel high utility sequential pattern mining, in: Proceedings of IEEE International Conference on Big Data, IEEE, 2016, pp. 853–862.

[12] M.-S. Chen, J. Han, P. S. Yu, Data mining: an overview from a database perspective, IEEE Transactions on Knowledge and Data Engineering 8 (6) (1996) 866–883.

[13] J.-Z. Wang, J.-L. Huang, Y.-C. Chen, On efficiently mining high utility sequential patterns, Knowledge and Information Systems 49 (2) (2016) 597–627.

[14] W. Gan, J. C.-W. Lin, J. Zhang, H.-C. Chao, H. Fujita, P. S. Yu, ProUM: Projection-based utility mining on sequence data, Information Sciences 513 (2020) 222–240.

[15] W. Gan, J. C.-W. Lin, J. Zhang, P. Fournier-Viger, H.-C. Chao, P. S. Yu, Fast utility mining on sequence data, IEEE transactions on Cybernetics 51 (2) (2021) 487–500.

[16] X. Zhang, F. Lai, G. Chen, W. Gan, Mining high-utility sequences with positive and negative values, Information Sciences 637 (2023) 118945. doi:https://doi.org/10.1016/j.ins.2023.118945.

[17] W. Gan, J. C.-W. Lin, P. Fournier-Viger, H.-C. Chao, H. Fujita, Extracting non-redundant correlated purchase behaviors by utility measure, Knowledge-Based Systems 143 (2018) 30–41.

[18] C. F. Ahmed, S. K. Tanbeer, B.-S. Jeong, A framework for mining high utility web access sequences, IETE Technical Review 28 (1) (2011) 3–16.

[19] B.-E. Shie, H.-F. Hsiao, V. S. Tseng, Efficient algorithms for discovering high utility user behavior patterns in mobile commerce environments, Knowledge and Information Systems 37 (2) (2013) 363–387.

[20] L. Bermingham, I. Lee, Mining distinct and contiguous sequential patterns from large vehicle trajectories, Knowledge-Based Systems 189 (2020) 105076.

[21] M. S. Nawaz, P. Fournier-Viger, A. Shojaee, H. Fujita, Using artificial intelligence techniques for covid-19 genome analysis, Applied Intelligence (2021) 1–18.

[22] J. Chen, T. Cook, Mining contiguous sequential patterns from web logs, in: Proceedings of the 16th International Conference on World Wide Web, 2007, pp. 1177–1178.

[23] J. Zhang, Y. Wang, D. Yang, CCSpan: Mining closed contiguous sequential patterns, Knowledge-Based Systems 89 (2015) 1–13.

[24] S. Jawahar, P. Sumathi, An efficient contiguous pattern mining technique to predict mutations in breast cancer for dna data sequences, International Journal of Bioinformatics and Biological Science 6 (1) (2018) 35–41.

[25] C. Yang, G. Gidófalvi, Mining and visual exploration of closed contiguous sequential patterns in trajectories, International Journal of Geographical Information Science 32 (7) (2018) 1282–1304.

[26] L. Zhou, Y. Liu, J. Wang, Y. Shi, Utility-based web path traversal pattern mining, in: Proceedings of the 7th IEEE International Conference on Data Mining Workshops, IEEE, 2007, pp. 373–380.

[27] C. F. Ahmed, S. K. Tanbeer, B.-S. Jeong, Y.-K. Lee, Efficient mining of utility-based web path traversal patterns, in: Proceedings of the 11th International Conference on Advanced Communication Technology, Vol. 3, IEEE, 2009, pp. 2215–2218.

[28] G. Huang, R. Gao, J. Wang, J. Yan, J. Ren, An efficient algorithm for mining high utility contiguous patterns from software executing traces, International Journal of Innovative Computing Infomation and Control 12 (3) (2016) 959–971.

[29] G.-E. Yap, X.-L. Li, P. S. Yu, Effective next-items recommendation via personalized sequential pattern mining, in: International Conference on Database Systems for Advanced Applications, Springer, 2012, pp. 48–64.

[30] R. Agrawal, R. Srikant, et al., Fast algorithms for mining association rules, in: Proceedings of the 20th International Conference on Very Large Data Bases, Morgan Kaufmann, 1994, pp. 487–499.

[31] P. Fournier-Viger, J. C.-W. Lin, R. U. Kiran, Y. S. Koh, R. Thomas, A survey of sequential pattern mining, Data Science and Pattern Recognition 1 (1) (2017) 54–77.

[32] M. J. Zaki, SPADE: An efficient algorithm for mining frequent sequences, Machine Learning 42 (1-2) (2001) 31–60.

[33] J. Ayres, J. Flannick, J. Gehrke, T. Yiu, Sequential pattern mining using a bitmap representation, in: Proceedings of the eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2002, pp. 429–435.

[34] P. Fournier-Viger, A. Gomariz, M. Campos, R. Thomas, 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, 2014, pp. 40–52.

[35] J. Pei, J. Han, B. Mortazavi-Asl, J. Wang, H. Pinto, Q. Chen, U. Dayal, M.-C. Hsu, Mining sequential patterns by pattern-growth: The PrefixSpan approach, IEEE Transactions on Knowledge and Data Engineering 16 (11) (2004) 1424–1440.

[36] W. Gan, J. C.-W. Lin, P. Fournier-Viger, H.-C. Chao, P. S. Yu, A survey of parallel sequential pattern mining, ACM Transactions on Knowledge Discovery from Data 13 (3) (2019) 1–34.

[37] O. K. Alkan, P. Karagoz, CRoM and HuspExt: Improving efficiency of high utility sequential pattern extraction, IEEE Transactions on Knowledge and Data Engineering 27 (10) (2015) 2645–2657.

[38] J. C.-W. Lin, Y. Li, P. Fournier-Viger, Y. Djenouri, J. Zhang, Efficient chain structure for high-utility sequential pattern mining, IEEE Access 8 (2020) 40714–40722.

[39] T. Truong-Chi, P. Fournier-Viger, A survey of high utility sequential pattern mining, in: High-Utility Pattern Mining, Springer, 2019, pp. 97–129.

[40] C. Zhang, M. Lyu, W. Gan, P. S. Yu, Totally-ordered sequential rules for utility maximization, ACM Trans. Knowl. Discov. Data 18 (4) (2024). doi:10.1145/3628450.

[41] C. Zhang, Z. Du, W. Gan, P. S. Yu, TKUS: Mining top-k high utility sequential patterns, Information Sciences 570 (2021) 342–359.

[42] M. Ceci, P. F. Lanotte, Closed sequential pattern mining for sitemap generation, World Wide Web 24 (1) (2021) 175–203.

[43] J. Pan, P. Wang, W. Wang, B. Shi, G. Yang, Efficient algorithms for mining maximal frequent concatenate sequences in biological datasets, in: Proceedings of the fifth International Conference on Computer and Information Technology, IEEE, 2005, pp. 98–104.

[44] T. H. Kang, J. S. Yoo, H. Y. Kim, Mining frequent contiguous sequence patterns in biological sequences, in: Proceedings of the IEEE 7th International Symposium on BioInformatics and BioEngineering, IEEE, 2007, pp. 723–728.

[45] J. Chen, S. Shankar, A. Kelly, S. Gningue, R. Rajaravivarma, A two stage approach for contiguous sequential pattern mining, in: Proceedings of the IEEE International Conference on Information Reuse and Integration, IEEE, 2009, pp. 382–387.

[46] Y. Abboud, A. Boyer, A. Brun, CCPM: a scalable and noise-resistant closed contiguous sequential patterns mining algorithm, in: Proceedings of the 13th International Conference on Machine Learning and Data Mining in Pattern Recognition, Springer, 2017, pp. 147–162.

[47] Y.-H. Goo, K.-S. Shim, M.-S. Lee, M.-S. Kim, Protocol specification extraction based on contiguous sequential pattern algorithm, IEEE Access 7 (2019) 36057–36074.

[48] J. Zhang, Y. Wang, C. Zhang, Y. Shi, Mining contiguous sequential generators in biological sequences, IEEE/ACM Transactions on Computational Biology and Bioinformatics 13 (5) (2015) 855–867.

[49] K. Hu, W. Gan, S. Huang, H. Peng, P. Fournier-Viger, Targeted mining of contiguous sequential patterns, Information Sciences 653 (2024) 119791. doi:https://doi.org/10.1016/j.ins.2023.119791.

[50] R. Agrawal, R. Srikant, Quest synthetic data generatorhttp://www.Almaden.ibm.com/cs/quest/syndata.html (1994).

[51] C. Zhang, Z. Du, Y. Yang, W. Gan, P. S. Yu, On-shelf utility mining of sequence data, ACM Transactions on Knowledge Discovery from Data 16 (2) (2021) 1–31.

[52] S. Alias, S. K. Mohammad, G. K. Hoon, T. T. Ping, A text representation model using sequential pattern-growth method, Pattern Analysis and Applications 21 (1) (2018) 233–247.

[53] E. Stamoulakatou, A. Gulino, P. Pinoli, DLA: A distributed, location-based and Apriori-based algorithm for biological sequence pattern mining, in: IEEE International Conference on Big Data, IEEE, 2018, pp. 1121–1126.