Knowledge-Based Systems 229 (2021) 107361

Contents lists available at ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

HANP-Miner: High average utility nonoverlapping sequential pattern mining

Youxi Wu [a,d,1], Meng Geng [a,1], Yan Li [b,*], Lei Guo [c], Zhao Li [e], Philippe Fournier-Viger [f], Xingquan Zhu [g], Xindong Wu [h,i]

[a] School of Artificial Intelligence, Hebei University of Technology, Tianjin 300401, China
[b] School of Economics and Management, Hebei University of Technology, Tianjin 300401, China
[c] State Key Laboratory of Reliability and Intelligence of Electrical Equipment, Hebei University of Technology, Tianjin 300401, China
[d] Hebei Data Driven Industrial Intelligent Engineering Research Center, Tianjin 300401, China
[e] Alibaba Group, Hangzhou, Zhejiang Province, 310000, China
[f] School of Humanities and Social Sciences, Harbin Institute of Technology (Shenzhen), Shenzhen, China
[g] Department of Computer & Electrical Engineering and Computer Science, Florida Atlantic University, FL, 33431, USA
[h] Key Laboratory of Knowledge Engineering with Big Data (Hefei University of Technology), Ministry of Education, Hefei 230009, China
[i] Mininglamp Academy of Sciences, Mininglamp Technology, Beijing 100084, China

ARTICLE INFO

Article history:
Received 31 December 2020
Received in revised form 17 July 2021
Accepted 2 August 2021
Available online 9 August 2021

Keywords:
Sequential pattern mining
High average utility
Depth-first search
Pattern join

ABSTRACT

Nonoverlapping sequential pattern mining (SPM) is a data analysis task, which aims at identifying repetitive sequential patterns with gap constraint in a set of discrete sequences. Nonoverlapping means that any character in the sequence can be rematched by characters at different positions in the pattern, while overlapping means that any character can be reused at the same position, which can be regarded as no condition. Compared with overlapping SPM, nonoverlapping SPM has more strict constraint on the occurrences of pattern, and meets the Apriori property. However, current algorithms mine frequent or closed patterns, resulting in some low-frequency but extremely important patterns being ignored. To tackle this issue, this paper proposes to mine high average utility nonoverlapping sequential patterns (HANP). An efficient algorithm called HANP-Miner is proposed, which involves two key steps: support calculation and candidate pattern reduction. To calculate the support (the occurrence frequency of a pattern), depth-first search and backtracking strategies based on the simplified Nettree structure are adopted, an approach that effectively reduces the time and space complexities of the algorithm. To effectively reduce the number of candidate patterns, a pattern join strategy based on an upper bound on the average utility is proposed. Experiments on biological sequences and real sales sequences are carried out to verify the efficiency of HANP-Miner and the superiority of HANPs. The results demonstrate that HANP-Miner is not only more efficient, but that the HANPs mined in this way are also more valuable than existing frequent patterns. The algorithms and datasets can be downloaded from https://github.com/wuc567/Pattern-Mining/tree/master/HANP-Miner.

1. Introduction

Sequential pattern mining (SPM) [1] as an important research topic, aims to mine sub-sequences (patterns) in sequence datasets. It has been widely used for analyzing biological sequence [2], customer purchase behavior [3], time series [4,5], inspection reports [6,7], etc. To meet different requirements, many types of patterns have been proposed, such as high utility pattern [8,9], high average utility pattern [10], and closed pattern [11]. To mine these types of patterns, various mining methods have been developed, such as high utility pattern mining [12–14], high average utility pattern mining [15], maximal frequent pattern mining [16,17], tri-partition pattern mining [18], negative sequential pattern mining [19,20], co-location pattern mining [21], outlying pattern mining [22], and closed pattern mining [23,24]. Traditional SPM describes a sequence such as <c(abd)(bd)(cd)> as an ordered list of sets of items (i.e. a list sets of symbols or characters). For instance, this sequence indicates that item c is followed by a, b, and d, followed by b and d, and then followed by c and d. Traditional methods only consider whether or not a pattern occurs in a sequence, but ignore how many times the pattern occurs in it. To solve this problem, a sequence can be instead

* Corresponding author.
E-mail address: lywuc@163.com (Y. Li).
[1] Both authors contributed equally to this research.

https://doi.org/10.1016/j.knosys.2021.107361
0950-7051/

described as a list of single items such as <cabdbdcd>, and all occurrences of patterns can be counted.

Furthermore, to avoid mining useless patterns, SPM with gap constraint [25,26] was proposed to make the patterns more flexible, and a pattern with gap constraint is expressed as $\mathbf{p} = p_1[min_1, max_1]p_2 \ [min_2, max_2] \ \cdots p_{j-1}[min_{j-1}, \ max_{j-1}]p_j \ \cdots p_{m-1}[min_{m-1}, max_{m-1}]p_m$, where min_{i-1} and max_{i-1} mean that at least min_{i-1} and at most max_{i-1} characters occur between p_{i-1} and p_i, respectively [27]. Compared with traditional SPM, SPM with gap constraint is difficult to be solved, and has three forms: no condition [5,28], one-off condition [29,30], [31], and nonoverlapping condition [27,32,33]. Previous research work has shown that, unlike the no condition case, the nonoverlapping condition avoids producing many redundant patterns, and unlike the one-off condition case, it also avoids overlooking valuable patterns [34]. In addition, nonoverlapping SPM (or SPM under the nonoverlapping condition) is a complete pattern mining method that satisfies the Apriori property.

Unfortunately, current research studies based on nonoverlapping SPM only take the occurrence frequencies of patterns into account [34], and do not consider other factors that can help evaluate the importance of the patterns, such as purchase quantity, unit profit of item [35], and the interest and weight of each item. As a result, the information extracted by traditional nonoverlapping SPM algorithms is insufficient for many applications. For example, in biological sequences, the frequency may not be enough to discovery a gene sequence related to a certain disease since although a gene may not occur frequently, its high expression may mean that it is very significant. Conversely, an inhibitory gene may occur frequently, but not have a strong effect. Researchers have therefore proposed a more general problem called high utility SPM [36] that incorporates the external utility values of items into traditional SPM, to comprehensively take into consideration the frequencies of pattern occurrences and the importance of each item. The goal is to find all sequences having a utility (importance) that is no less than a minimum utility threshold. An illustrative example is as follows.

Example 1. Suppose we have a sequence $\mathbf{s} = s_1s_2s_3s_4s_5s_6$ = CTCTTG, and a pattern $\mathbf{p} = p_1[0, 2]p_2[0, 2]p_3$ = C[0,2]T[0,2]T, where the utilities of C, G, and T are 8, 8, and 2, respectively, and that the minimum utility threshold is set to 20. Fig. 1 shows all occurrences of pattern **p** in sequence **s**.

As shown in Fig. 1, there are four occurrences of pattern **p** in sequence **s** under the no condition: <1,2,4>, <1,2,5>, <1,4,5>, and <3,4,5>, where the number indicates the position in sequence **s**. The nonoverlapping condition means that any character in the sequence can be rematched, but not at the same position. In this example, <1,2,4> and <3,4,5> are two nonoverlapping occurrences, since s_4 matches p_3 and p_2, respectively. Hence, pattern **p** occurs twice under the nonoverlapping condition. According to the utility values, $HU(\mathbf{p}) = (8 + 2 + 2) \times 2 = 24$, which is greater than the minimum utility threshold 20. Hence, pattern **p** is a high utility pattern.

From the above example, it can be seen that the utility of a pattern is the product of the sum of the utility of each item and its support (the number of occurrences). However, a serious flaw of this method is that it does not take the length of the pattern into account, meaning that it is easy to mine long but valueless patterns. For example, pattern $\mathbf{p}'$ = C[0,2]T[0,2]C[0,2]T[0,2]T[0,2]G occurs only once in the sequence, but the utility of $\mathbf{p}'$ is 30, which is greater than 20. Thus, $\mathbf{p}'$ is also a high utility pattern. From this example, we can see that it is unfair to measure the importance of patterns with different lengths using the same minimum utility threshold, since the longer the pattern length, the higher the utility. To address this problem, inspired by the concept of high average utility in SPM [37], we propose a novel method called high average utility nonoverlapping sequential pattern (HANP) mining. There are three main differences between the previous work [34] and our problem: the mined patterns, the support calculation strategies, and the candidate pattern reduction strategies. More detail will be shown in Related Work section. The main contributions are as follows.

	1	2	3	4	5	6		
s =	C	T	C	T	T	G		The sequence **s**
	C	T		T			<1, 2, 4>	First occurrence of **p**
	C	T			T		<1, 2, 5>	Second occurrence of **p**
	C			T	T		<1, 4, 5>	Third occurrence of **p**
			C	T	T		<3, 4, 5>	Fourth occurrence of **p**

Fig. 1. All occurrences of pattern **p** in sequence **s**.

(1) We address the problem of HANP mining, and propose an efficient mining algorithm called HANP-Miner which has two essential steps: support calculation and candidate pattern reduction.
(2) To efficiently calculate the support, we propose a depth-first online matching (DFOM) algorithm, which adopts a depth-first online matching strategy and employs a simplified Nettree data structure [38].
(3) We derive a strategy based on an upper bound on the average utility and combine it with the pattern join strategy to generate candidate patterns. These strategies can efficiently reduce the number of candidate patterns.
(4) Experimental results on the DNA, VIRUS, and sales datasets verify that not only does HANP-Miner outperform other competitive algorithms, but also the HANPs mined in this way are significant.

The structure of this paper is as follows. Section 2 introduces related work. Section 3 defines the problem. Section 4 proposes the HANP-Miner algorithm, which employs a depth-first online matching strategy to calculate the support and an upper bound on the average utility to reduce the number of candidate patterns, and presents a complexity analysis. Section 5 reports the results of comparative experiments on DNA, VIRUS and sales datasets, and analyzes these results. Section 6 concludes this paper.

2. Related work

SPM has been widely used in many fields, and various mining methods have been proposed. For example, web log mining [39] and transaction flow mining [40] were developed for different types of datasets. Van et al. [41–43] considered the problem of mining sequential patterns with itemset constraints which mined the user access behavior on the web. Pseudo-IDList data structure was proposed [44], which is more suitable for mining clickstream patterns. Based on this structure, a vertical format algorithm named CUP was proposed. To solve the problem of clickstream pattern mining based on the average weight measure, Huynh et al. [45] described an optimized data structure called a Weighted ID-Compact Value List (WICList) and proposed Compact-SPADE algorithm. Rare SPM [46], maximum SPM [16], and closed SPM [32,47] were designed for compress the frequent patterns based on different pattern features. Tran et al. [48] proposed CloFS-DBV algorithm, which used a dynamic bit vector (DBV) structure combined with location information to mine frequent closed sequences. Wu et al. [32] addressed closed pattern mining under the nonoverlapping condition and proposed Net-NCSP algorithm based on the Nettree structure and three kinds

of pruning strategies, inheriting, predicting, and determining. This algorithm can discover more closed patterns with good compressibility. Tri-partition alphabets SPM [49–51], negative SPM [52], and high utility SPM [53,54] were designed for different mining tasks. Qiu et al. [52] and Dong et al. [55] considered negative SPM which played more important role in many applications. HUNSP algorithm and e-RNSP algorithm were proposed for mining high utility negative patterns and repetition negative patterns. Truong et al. [53] introduced two upper bounds and a weak upper bound on the average-utility measure to speed up high average utility mining.

High utility SPM and high average utility SPM have been widely used in many important fields [56], such as customer purchase behavior analysis [57], disease diagnosis [58], event log discovery [59], and sentiment classification [60]. Nam et al. [61] proposed the DHUP algorithm which applied a damped window model concept to increase memory usage. Kim et al. [62] addressed a list structure to store information of patterns more compactly. The mining method with list structure only needs to scan the database once, and make it possible to search and utilize information quickly with index value of each item, which can improve the efficiency, memory utilization, and scalability of the algorithm. However, the list structure only records whether the patterns occur or not, it is difficult to count all occurrences in a sequence. Therefore, list structure is not suitable for repetitive SPM.

One of the disadvantages of traditional SPM is that it neglects the repetition of patterns in a sequence. To overcome this drawback, SPM with gap constraint was proposed, which is a kind of repetitive SPM [34] that can be divided into three types: no condition (or overlapping condition) [28], one-off condition [29], and nonoverlapping condition [32]. An illustrative example is given below.

Example 2. A comparison of the occurrences under different conditions of pattern **p** = C[0,2]T[0,2]T in sequence **s** = CTCTTG is shown in Table 1.

No condition means that any character can be reused at the same position. As can be seen from Table 1, there are four no condition occurrences, i.e. <1,2,4>, <1,2,5>, <1,4,5>, and <3,4,5>. s_1 is used multiple times by p_1 in the occurrences of <1,2,4>, <1,2,5>, and <1,4,5>, which are three overlapping occurrences. Hence, no condition can be regarded as overlapping condition. Unfortunately, overlapping SPM does not satisfy the Apriori property [63]. For example, in Example 2, pattern "C" has two occurrences, i.e. <1> and <3>. However, its super-pattern "C[0,2]T" has four no condition (overlapping) occurrences, i.e. $\langle 1, 2\rangle$, $\langle 1, 4\rangle$, $\langle 3, 4\rangle$, and $\langle 3, 5\rangle$. Thus, an Apriori-like property was employed to find all frequent patterns, which enlarged the search space [64]. The one-off condition means that any character in the sequence can be used only once at most [65]. In this example, there is only one occurrence under the one-off condition, i.e. <1,2,4>. Although the support under this condition satisfies the Apriori property, calculating the support is an NP-hard problem [66], meaning that an approximate pattern mining method must be developed. The nonoverlapping condition means that any character in the sequence can be matched multiple times, but not at the same position. There are therefore two occurrences of **p** in **s** under the nonoverlapping condition, i.e. <1,2,4> and <3,4,5>. This example shows that the nonoverlapping condition is stricter than the no condition and looser than the one-off condition. Not only does nonoverlapping SPM satisfy the Apriori property, but it also is possible to develop a complete mining algorithm. However, one of the shortcomings of nonoverlapping SPM [34] is that it does not consider the weight of each item. To address this issue, the HANP mining problem is explored in this paper. Table 2 gives a comparison of related studies.

The above researches adopted different measure standards for pattern mining, such as frequency, utility, sequential utility, average utility, and average sequential utility. For example, Li et al. [67], Xie et al. [30], and Wu et al. [34] mined frequent sequential patterns with different constraints. Nam et al. [61], Truong et al. [53], and Kim et al. [69] mined high utility and high average utility itemset patterns in transaction databases, while Srivastava et al. [68], Wu et al. [70], and Lin et al. [15] mined high utility and high average utility sequential patterns in sequence databases. The nonoverlapping SPM [34] is the closest scheme. The differences are shown as follows.

(1) The mined patterns are different. The nonoverlapping SPM [34] focused on mining frequent patterns, while this paper investigates mining HANPs. We show that HANP mining has more actual significance than frequent SPM.
(2) The support calculation strategies are different. NETGAP algorithm [34] was proposed to calculate the support which creates a whole Nettree initially and then prunes the invalid nodes after obtaining a nonoverlapping occurrence. To improve the efficiency, this paper proposes the DFOM algorithm which employs depth-first search and backtracking strategies to calculate the support without creating a whole Nettree. The time and space complexities of DFOM are therefore lower than those of NETGAP.
(3) Candidate patterns reduction strategies are different. Nonoverlapping SPM satisfies the Apriori property [34]. However, HANP-Miner does not satisfy the Apriori property, and we therefore propose an Apriori-like strategy that uses an upper bound on the average utility.

3. Problem definition

A sequence with length n is denoted as $\mathbf{s} = s_1 s_2 \cdots s_n$, where $s_i (0 \le i \le n) \in \Sigma$, Σ represents the set of items in sequence **s**, and the size of Σ can be expressed as $|\Sigma|$. For example, in a DNA sequence, Σ is {A,T,C,G} and $|\Sigma| = 4$.

Definition 1 (*Pattern*)**.** A pattern **p** with length m is denoted as $\mathbf{p} = p_1[a, b]p_2 \cdots [a, b]p_m$ (or abbreviated as $\mathbf{p} = p_1 p_2 \cdots p_m$ with $gap = [a, b]$), where a and b $(0 \le a \le b)$ are integers indicating the minimum and maximum wildcards between p_j and p_{j+1}, respectively.

Definition 2 (*Occurrence and Nonoverlapping Occurrence*)**.** Suppose we have a sequence $\mathbf{s} = s_1 s_2 \cdots s_n$ and pattern $\mathbf{p} = p_1[a, b]p_2 \cdots [a, b]p_m$. $L = \langle l_1, l_2, \ldots, l_m\rangle$ is an occurrence of pattern **p** in sequence **s** if and only if $p_1 = s_{l_1}$, $p_2 = s_{l_2}$, $\cdots$, $p_m = s_{l_m}$ $(0 \le l_1 \le l_2 \le \cdots \le l_m \le n)$ and $a \le l_j - l_{j-1} - 1 \le b$. Suppose there is another occurrence $L' = \langle l'_1, l'_2, \ldots, l'_m\rangle$. L and L' are two nonoverlapping occurrences if and only if $\forall 1 \le j \le m$ and $l_j \ne l'_j$.

Definition 3 (*Support*)**.** The support of pattern **p** in sequence **s** is the number of nonoverlapping occurrences, represented by $sup(\mathbf{p},\mathbf{s})$. A sequence database with N sequences is denoted by $SDB = \{\mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_N\}$. The support of pattern **p** in SDB is the sum of the supports in each sequences, i.e. $sup(\mathbf{p}, SDB) = \sum_{k=1}^{N} sup(\mathbf{p}, \mathbf{s}_k)$.

Example 3. Suppose we have $SDB = \{\mathbf{s}_1 = s_1s_2s_3s_4s_5s_6 = \text{CTCTTG}, \mathbf{s}_2 = s_1s_2s_3s_4s_5s_6 = \text{CCTTGT}\}$, $\mathbf{p} = p_1[0, 2]p_2[0, 2]p_3 = C[0, 2]T[0, 2]T$. In Example 1, all occurrences of pattern **p** in sequence $\mathbf{s}_1$ are <1,2,4>, <1,2,5>, <1,4,5>, and <3,4,5>. The

Table 1
Comparison of occurrences under different conditions.

Condition	Support	Occurrences
No condition (Overlapping condition)	4	<1,2,4>, <1,2,5>, <1,4,5>, <3,4,5>
One-off condition	1	<1,2,4>
Nonoverlapping condition	2	<1,2,4>, <3,4,5>

Table 2
Comparison of related studies.

Literature	Measure standard	Type of condition	Pruning strategy	Gap constraint
Li et al. [67]	Frequency	No condition	Apriori-like	Periodic
Xie et al. [30]	Frequency	One-off condition	Apriori	Periodic
Nam et al. [61]	Utility	Without repetition	Other	No
Srivastava et al. [68]	Sequential utility	Without repetition	Upper-bounds on utility	No
Truong et al. [53]	Average utility	Without repetition	Depth Pruning	No
Kim et al. [69]	Average utility	Without repetition	Upper-bounds on utility	No
Wu et al. [70]	Average sequential utility	One-off condition	Apriori-like	Self-adaptive
Lin et al. [15]	Average sequential utility	Without repetition	Downward closure	No
Wu et al. [34]	Frequency	Nonoverlapping condition	Apriori	Periodic
This paper	Average sequential utility	Nonoverlapping condition	Apriori-like	Periodic

Table 3
All HANPs and its support and average utility.

Pattern	Support	Average utility
C	4	4 × 8/1 = 32
G	2	2 × 8/1 = 16
CC	2	2×(8+8)/2 = 16
CT	4	4×(8+2)/2 = 20
CG	2	2×(8+8)/2 = 16
CCG	2	2×(8+8+8)/3 = 16
CTT	4	4×(8+2+2)/3 = 16

nonoverlapping occurrences are <1,2,4> and <3,4,5>. Hence, $sup(\mathbf{p}, \mathbf{s}_1)$ is 2. The support of pattern $\mathbf{p}$ in SDB is $sup(\mathbf{p}, SDB) = \sum_{k=1}^{2} sup(\mathbf{p}, \mathbf{s}_k) = sup(\mathbf{p}, \mathbf{s}_1) + sup(\mathbf{p}, \mathbf{s}_2) = 2 + 2 = 4$.

Definition 4 (*Utility and Average Utility*)**.** The utility of a pattern with length m is the product of the sum of the utilities of each item and its support, and is denoted by:

$$PU(\mathbf{p}, SDB) = \sum_{j=1}^{m} U(p_j) \times sup(\mathbf{p}, SDB)\ . \tag{1}$$

The average utility is the ratio of the utility to the pattern length, denoted by:

$$PAU(\mathbf{p}, SDB) = \frac{PU(\mathbf{p}, SDB)}{m} = \frac{\sum_{j=1}^{m} U(p_j) \times sup(\mathbf{p}, SDB)}{m}\ . \tag{2}$$

Example 4. In Example 3, the support of $\mathbf{p}$ = CTT with gap [0,2] in SDB is 4, and the utilities of C and T are 8 and 2, respectively. We therefore know that $PU(\mathbf{p}, SDB) = \sum_{j=1}^{m} U(p_j) \times sup(\mathbf{p}, SDB) = (8 + 2 + 2) \times 4 = 48$. Hence, $PAU(\mathbf{p}, SDB) = \frac{PU(p, SDB)}{m} = \frac{48}{3} = 16$.

Definition 5 (*HANP*)**.** If the average utility of pattern $\mathbf{p}$ is no less than a minimum threshold *minpau*, then pattern $\mathbf{p}$ is an HANP, otherwise it is not an HANP.

Our problem involves mining all HANPs in the input database.

Example 5. Suppose we have $SDB = \{\mathbf{s}_1 = s_1s_2s_3s_4s_5s_6 =$ CTCTTG, $\mathbf{s}_2 = s_1s_2s_3s_4s_5s_6 =$ CCTTGT$\}$, gap = [0, 2], *minpau* = 16, and the utilities of C, G, and T are 8, 8, and 2, respectively. We aim to find all HANPs which are shown in Table 3.

It can be seen from Table 3 that there are 7 HANPs. For example, "CCG" is an HANP since it has only two occurrence, i.e. <1,3,6> in $\mathbf{s}_1$, and <1,2,5> in $\mathbf{s}_2$. Its support is 2, and its average utility is 2×(8+8+8)/3 = 16, which is no less than *minpau* 16.

In Example 5, we set the utilities of each items according to the level of gene expression, and 7 HANPs are found. This example shows that the biological characteristics mainly come from these 7 highly expressed gene subsequences. HANP can be applied in many fields, such as gene difference analysis and product recommendation. More details of real application will be given in Section 5.7.

4. Proposed algorithm

The main factors affecting the performance of HANP mining are the calculation of the average utility and the generation of candidate patterns. The core difficulty in calculating the average utility lies in calculating the support. In Section 4.1, we therefore present the DFOM algorithm, which employs depth-first search and backtracking strategies to reduce the time and space complexities. To effectively generate candidate patterns, we apply a pattern join strategy that involves an upper bound on the average utility, as described in Section 4.2. Based on these approaches, we propose the HANP-Miner algorithm in Section 4.3, and analyzes its time and space complexities.

4.1. Average utility calculation

When calculating the average utility of a pattern, the key issue is the calculation of the support of the pattern (the number of occurrences of the pattern). In [68], list structure was used to record the occurrences and utility of the pattern in the database. However, list structure only records whether the pattern occurs or not, and it is difficult to deal with the repetitive SPM issue which calculates the number of occurrences of a pattern. Wu et al. [71] proposed the NETGAP algorithm based on Nettree, which is a complete method for calculating the support. However, the weakness of NETGAP is its low efficiency. We therefore design a more efficient algorithm called DFOM for calculating the pattern support which employs a simplified Nettree structure. Example 6 and 7 illustrate the principles of NETGAP and DFOM, respectively.

Example 6. Given a sequence $\mathbf{s} = s_1s_2s_3s_4s_5s_6s_7s_8s_9s_{10}s_{11}s_{12}s_{13}$ = GTGAGTT TCGCGC, a pattern $\mathbf{p} = p_1p_2p_3p_4$ = GTGC, and gap = [0,3], we illustrate the principle of NETGAP as follows.

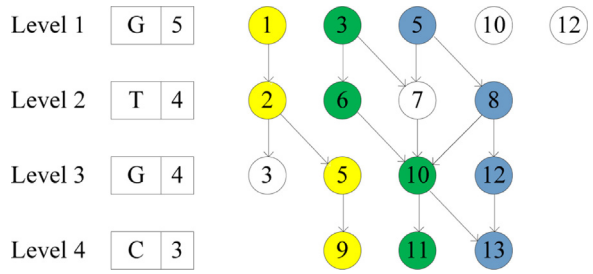


Fig. 2. Nettree for pattern **p** in sequence **s**. According to NETGAP, there are four levels, since the pattern length is four. Nodes n_1^1, n_1^3, n_1^5, n_1^{10}, and n_1^{12} are roots, and nodes n_4^9, n_4^{11}, and n_4^{13} are leaves. The same node ID can occur at different levels. For example, n_1^3 and n_3^3 are two nodes with the same ID 3, and are created in the first and third levels, respectively. Some nodes have more than one parent. For example, node n_2^7 has two parents, n_1^3 and n_1^5.

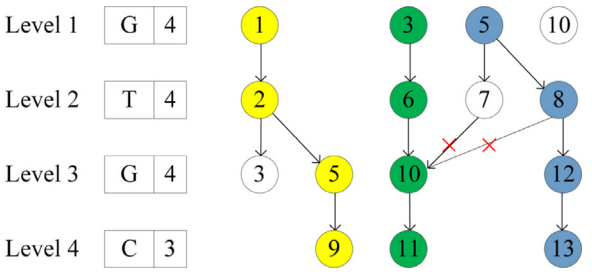


Fig. 3. Nettree for pattern **p** in sequence **s**. According to DFOM, the Nettree is a simplified Nettree which can be seen as a forest since each node has only one parent. For example, node n_3^{10} has only one parent n_2^6.

(1) Create a Nettree. First, create the roots of the Nettree which are the items in sequence **s** that can match p_1, i.e. s_1, s_3, s_5, s_{10}, and s_{12}. Iteratively create the next level nodes based on the previous level nodes. Suppose we have the second level nodes: n_2^2, n_2^6, n_2^7, and n_2^8. Thus we create the third level nodes based on the second level nodes. We take s_{10} as an example. Since $s_{10} = p_3$ = G and 10-6-1 = 3 satisfy the gap constraint [0, 3], node n_3^{10} is created and node n_2^6 is a parent of n_3^{10}. Meanwhile, nodes n_2^7 and n_2^8 are other two parents of n_3^{10} since 10-7-1 = 2 and 10-8-1 = 1 also satisfy the gap constraint [0, 3], Hence, node n_3^{10} has three parents: n_2^6, n_2^7, and n_2^8. Similarly, other nodes can be created and the Nettree for pattern **p** in sequence **s** is shown in Fig. 2.

(2) The next step is to prune invalid nodes, which are non-leaf nodes without children. In this example, the invalid nodes are n_3^3, n_1^{10}, and n_1^{12}.

(3) Then, start from its leftmost root n_1^1, and find the leftmost child n_2^2, until the leaf is found. It is easy to obtain the first path $\langle n_1^1, n_2^2, n_3^5, n_4^9\rangle$, which is marked in yellow in Fig. 2, and the corresponding occurrence is <1,2,5,9>.

(4) Nodes n_1^1, n_2^2, n_3^5, and n_4^9 are pruned. NETGAP then prunes invalid nodes in the new Nettree. In this example, there are no invalid nodes.

(5) Iterate Steps 3 and 4, and NETGAP obtains the second occurrence, <3,6,10,11>, which is marked in green. Nodes n_1^3, n_2^6, n_3^{10}, and n_4^{11} are pruned, and NETGAP then prunes invalid node n_2^7 in the new Nettree. Finally, the third occurrence <5,8,12,13> marked in blue is found. Hence, NETGAP finds three nonoverlapping occurrences.

From Example 6, it can be seen that the NETGAP algorithm is inefficient for two reasons: (i) a whole Nettree needs to be created; and (ii) it is necessary to prune invalid nodes after finding an occurrence. The time and space complexities of NETGAP are $O(m \times m \times n \times W)$ and $O(m \times n \times W)$ [71], respectively, where m, n, and W are the pattern length, sequence length, and gap constrain W = b-a+1 (gap=[a, b]), respectively.

To improve the efficiency of NETGAP, we propose the DFOM algorithm, which employs depth-first search and backtracking strategies to find nonoverlapping occurrences with no need to create a whole Nettree. The principle of DFOM is as follows.

Step 1. Find a root. If s_i ($1 \le i \le n - m + 1$) satisfies $s_i = p_1$, then s_i can be created on the first level, i.e. n_1^i.

Step 2. Adopt a depth-first search strategy to find the child node of the current node in an iterative way. Suppose n_{j-1}^t is the current node and n_j^k is the last node on the jth level. If s_i ($k \le i \le n$), n_{j-1}^t satisfies $s_i = p_j$, and $a \le i - t - 1 \le b$, then node n_j^i is created as a child node of n_{j-1}^t, and node n_j^i is selected as the current node. If there is no child node of n_{j-1}^t, then we select the parent node of n_{j-1}^t as the current node, and find its other child nodes.

Step 3. Obtain an occurrence. If we create a node on the mth level, then the path from the root to its mth level node is an occurrence.

Step 4. Iterate Steps 1, 2, and 3 until no root can be created.

Example 7. In this example, we use the sequence and pattern in Example 6 to illustrate the principle of DFOM.

We take n_1^1 as an example. From Step 1, we find that n_1^1 is the first root. DFOM then adopts a depth-first search strategy to iteratively create the child node of the current node. It is easy to see that n_2^2 is the child node of n_1^1, and n_3^3 is the child node of n_2^2. Since n_3^3 has no child, DFOM backtracks to n_2^2 and selects its other child n_3^5. Similarly, n_4^9 is the child node of n_3^5. According to Step 3, when n_4^9 is created, DFOM finds the first occurrence, <1,2,5,9>. Then, it is easy to see that <3,6,10,11> is the second occurrence. DFOM then finds the third root n_1^5, and selects n_2^7 as the first child. The first child node of n_2^7 is n_3^{10}, but n_3^{10} is occupied by the previous occurrence, and the parent–child relationship thus cannot be created. Since n_2^7 has no other child, DFOM backtracks to n_1^5 and selects its other child n_2^8. Hence, DFOM obtains the third occurrence, <5,8,12,13>. From Step 1, we find that n_1^{10} is the fourth root, but no child node can be created, and no node can be created as a root. Hence, DFOM finds three nonoverlapping occurrences. The Nettree created by DFOM is shown in Fig. 3.

Pseudocode for DFOM is given in Algorithm 1, which calculates the support of a pattern in a *SDB*. Moreover, DFOM adopts DFB algorithm which employs depth-first and backtracking strategies to obtain a nonoverlapping occurrence, as shown in Algorithm 2.

Algorithm 1 DFOM

Input: Sequence database *SDB*, pattern **p** with length m, gap=[a, b]
Output: sup(**p**,*SDB*)

```
1:  sup(p,SDB) ← 0;
2:  for each sequence s in SDB do
3:      n ← s.length();
4:      for i = 1 to n − m + 1 do
5:          if s_i = p_1 then
6:              Create root n_1^i;
7:              occ ← DFB(n_1^i, 1, s, p, gap);
8:              if occ ≠ NULL then
9:                  sup(p,SDB) ++;
10:             end if
11:         end if
12:     end for
```

Table 4
Utility of each item.

Item	A	C	G	T
Utility value	3	5	8	2

13: **end for**
14: return *sup*(**p**,*SDB*);

Algorithm 2 DFB

Input: Node *nd*, level *L*, sequence **s** with length *n*, pattern **p** with length *m*, $gap = [a, b]$
Output: *occ*

```
1: if L = m then
2:     Return occ;
3: else if L = 0 then
4:     Return NULL;
5: else
6:     child ← find the first child of nd according to the gap;
7:     if child = NULL then
8:         cparent ← occ[L − 1];
9:         DFB(cparent, L − 1, s, p, gap);
10:    else
11:        occ[L + 1] ← child;
12:        DFB(child, L + 1, s, p, gap);
13:    end if
14: end if
```

Theorem 1. *The space and time complexities of DFOM are $O(m \times n_1)$ and $O(m \times L)$, respectively, where m, n_1, and L are the length of the pattern, the maximum sequence length in SDB, and the total length of SDB, respectively.*

Proof. It is easy to see that the Nettree generated by DFOM is a forest, which means that each node has only one parent. For a sequence with length n, we know that the Nettree has m levels and each level has no more than n nodes. Hence, the space complexity of DFOM for a sequence is $O(m \times n)$. Since each node in the Nettree is visited only once, the time complexity of DFOM for a sequence is also $O(m \times n)$. We know that DFOM creates a simplified Nettree for each sequence and releases the space of the Nettree after the calculation of the sequence. Therefore, the space and time complexities of DFOM for a database are $O(m \times n_1)$ and $O(m \times L)$, respectively, where L is the sum of each sequence.

From Example 6 and 7, it can be seen that both NETGAP and DFOM achieve the same results, although the time and space of complexities of DFOM are lower than those of NETGAP. Hence, DFOM outperforms NETGAP.

4.2. Candidate pattern generation

In this section, we introduce our method of candidate pattern generation based on an upper bound on the average utility.

We know that nonoverlapping SPM satisfies the Apriori property [34], but both utility and average utility of a pattern do not satisfy the antimonotonic property. An illustrative example is shown in Example 8.

Example 8. Consider database $SDB = \{\mathbf{s} = \text{GTGAGTTTCGCGC}\}$, pattern $\mathbf{p}_1$ = TT, $\mathbf{p}_2$ = TTG, gap = [0,3], and $minpau$ = 8. The utilities are shown in Table 4.

Table 5 shows the comparison of occurrences, support, utility and average utility of pattern $\mathbf{p}_1$, and its super-pattern $\mathbf{p}_2$.

From Table 5, we know that the supports of patterns $\mathbf{p}_1$ and $\mathbf{p}_2$ are $sup(\mathbf{p}_1, SDB) = 3$ and $sup(\mathbf{p}_2, SDB) = 2$, respectively. Thus, the utilities of them are $3 \times (2+2) = 12$ and $2 \times (2+2+8) = 24$, respectively. Hence, $PAU(\mathbf{p}_1, SDB)$ and $PAU(\mathbf{p}_2, SDB)$ are $12/2 = 6$ and $24/3 = 8$, respectively. $\mathbf{p}_1$ is not an HANP, since $PAU(\mathbf{p}_1, SDB) < minpau$. Although $\mathbf{p}_2$ is a super-pattern of $\mathbf{p}_1$, $\mathbf{p}_2$ is an HANP, since $PAU(\mathbf{p}_2, SDB) \geq minpau$. This example shows that both utility and average utility do not satisfy the antimonotonic property.

Since the mining method does not satisfy the Apriori property, only the enumeration tree strategy can be used to generate candidate patterns. In Example 8, $\mathbf{p}_1$ = TT can generate four candidate patterns: TTA, TTT, TTC, and TTG. However, an enumeration tree strategy will generate exponential number of candidate patterns. To tackle this issue, we propose an Apriori-like property that employs a pattern join strategy with an upper bound on the average utility. A definition of the upper bound on the average utility is given below.

Definition 6 (*Upper Bound on the Average Utility and High Average Utility Upper Bound Pattern (HABP).*) The upper bound on the average utility of pattern **p** is the product of its support and the maximum utility shown as follows.

$$SPU(\mathbf{p}, SDB) = sup(\mathbf{p}, SDB) \times U_{max}, \tag{3}$$

where U_{max} represents the maximum utility of each item. If $SPU(\mathbf{p}, SDB)$ is no less than $minpau$, then pattern **p** is a HABP.

Example 9. In this example, we calculate the upper bound on the average utility of patterns $\mathbf{p}_1$ and $\mathbf{p}_2$ in Example 8. According to Table 4, $U_{max} = 8$. From Eq. (3), $SPU(\mathbf{p}_1, SDB) = 3 \times 8 = 24$ and $SPU(\mathbf{p}_2, SDB) = 2 \times 8 = 16$.

Theorem 2. *If pattern **p** is not an HABP, then it is not an HANP.*

Proof. We know that $\sum_{i=1}^{m} U(p_i) \leq m \times U_{max}$. Hence, $PAU(\mathbf{p}, SDB) = \frac{\sum_{i=1}^{m} U(p_i) \times sup(\mathbf{p}, SDB)}{m} \leq \frac{m \times U_{max} \times sup(\mathbf{p}, SDB)}{m} = sup(\mathbf{p}, SDB) \times U_{max} = SPU(\mathbf{p}, SDB)$. $SPU(\mathbf{p}, SDB) < minpau$ since pattern **p** is not an HABP. Thus, $PAU(\mathbf{p}, SDB)$ is less than $minpau$, and pattern **p** is not an HANP.

Lemma 1. *The support satisfies the Apriori property. i.e. $sup(\mathbf{q}, SDB) \geq sup(\mathbf{p}, SDB)$, where pattern **q** is a sub-pattern of pattern **p** [34].*

Proof. Suppose pattern **p** is $p_1 p_2 \cdots p_m$. Since pattern **q** is a sub-pattern of **p**, **q** can be written as $p_a p_{a+1} \cdots p_b$, where $a \geq 1$ and $m \geq b$. Suppose $sup(\mathbf{p}, SDB) = k$, which means that there are k nonoverlapping occurrences for pattern **p** in database SDB, i.e. $\langle l_{1,1}, l_{1,2}, \ldots, l_{(1,m)}\rangle$, $\langle l_{2,1}, l_{2,2}, \ldots, l_{(2,m)}\rangle$, $\cdots$, $\langle l_{(k,1)}, l_{(k,2)}, \ldots, l_{(k,m)}\rangle$. Therefore, we can say that the k occurrences $\langle l_{(1,a)}, l_{(1,a+1)}, \ldots, l_{(1,b)}\rangle$, $\langle l_{(2,a)}, l_{(2,a+1)}, \ldots, l_{(2,b)}\rangle$, $\cdots$, $< l_{(k,a)}, l_{(k,a+1)}, \ldots, l_{(k,b)} >$ are nonoverlapping. Thus, there are at least k nonoverlapping occurrences for pattern **q** in database SDB. Hence, $sup(\mathbf{q}, SDB) \geq sup(\mathbf{p}, SDB)$.

Theorem 3. *The upper bound on the average utility satisfies the Apriori property.*

Proof. Suppose we have a sequence **s**, a pattern **p**, and its prefix sub-pattern **l**, suffix sub-pattern **r**. It can be seen that $sup(\mathbf{l}, SDB) \geq sup(\mathbf{p}, SDB)$ and $sup(\mathbf{r}, SDB) \geq sup(\mathbf{p}, SDB)$ according to Lemma 1. Eq. (3) shows that the upper bound on the average utility of a pattern is its support multiplied by a fixed value. Hence, $SPU(\mathbf{l}, SDB) \geq SPU(\mathbf{p}, SDB)$ and $SPU(\mathbf{r}, SDB) \geq SPU(\mathbf{p}, SDB)$. Thus, if **l** is not an HABP, i.e., $SPU(\mathbf{l}, SDB) < minpau$, then $SPU(\mathbf{p}, SDB)$ is less than $minpau$. Hence, **p** is also not an HABP. Similarly, if **r** is not an HABP, then **p** is also not an HABP. Therefore, the upper bound on the average utility satisfies the Apriori property.

Table 5
Comparison of occurrences, support, utility and average utility of pattern $\mathbf{p}_1$ and its super-pattern $\mathbf{p}_2$.

Item	Occurrences	Support	Utility	Average utility
Pattern $\mathbf{p}_1$ = TT	$< 2, 6 >, < 6, 7 >, < 7, 8 >$	3	$3 \times (2 + 2) = 12$	$12/2 = 6$
Super-pattern $\mathbf{p}_2$ = TTG	<2,6,10>, <6,8,12>	2	$2 \times (2 + 2 + 8) = 24$	$24/3 = 8$

Based on Theorem 2 and 3, candidate patterns can be generated by employing a pattern join strategy.

Definition 7 (*Maximum Prefix Pattern, Maximum Suffix Pattern, and Pattern Join*)**.** Suppose we have pattern $\mathbf{p} = p_1[M, N]p_2 \ldots [M, N]p_m$, and items r and l. If $\mathbf{q} = \mathbf{p}r = p_1[M, N]p_2 \ldots [M, N]p_m[M, N]r$, then $\mathbf{p}$ is called the maximum prefix pattern of $\mathbf{q}$, and is denoted as $Prefix(\mathbf{q}) = \mathbf{p}$. Similarly, if $\mathbf{d} = l\mathbf{p} = l[M, N]p_1[M, N]p_2 \ldots [M, N]p_m$, then $\mathbf{p}$ is called the maximum suffix pattern of $\mathbf{d}$, and is denoted as $Suffix(\mathbf{d}) = \mathbf{p}$. We obtain a new pattern $\mathbf{t}$ of length m+2, denoted as $\mathbf{t} = \mathbf{d} \oplus \mathbf{q} = l\mathbf{p}r$, and this process is called pattern join.

Example 10. Given patterns $\mathbf{p_1}$ = GTG and $\mathbf{p_2}$ = TGC with $gap = [0, 3]$, we know that $Prefix(\mathbf{p_2}) = TG$ and $Suffix(\mathbf{p_1}) = TG$. Since $Prefix(\mathbf{p_2}) = Suffix(\mathbf{p_1})$, pattern $\mathbf{p} = \mathbf{p_1} \oplus \mathbf{p_2} = GTGC$ can be generated by pattern join.

Theorem 4. *Suppose HU_m is the HABP set. The candidate pattern set C_{m+1} can be generated from HU_m using the pattern join strategy. We can safely say that $HU_{m+1} \subseteq C_{m+1}$.*

Proof. 4 The proof is by contradiction. Suppose $\mathbf{p}_{m_{ij}}$ is an HABP, but $\mathbf{p}_{m_{ij}}$ is not in the set C_{m+1}. The maximum prefix and suffix patterns of $\mathbf{p}_{m_{ij}}$ are $\mathbf{p}_{m_i}$ and $\mathbf{p}_{m_j}$, respectively, i.e. $Prefix(\mathbf{p}_{m_{ij}}) = \mathbf{p}_{m_i}$ and $Suffix(\mathbf{p}_{m_{ij}}) = \mathbf{p}_{m_i}$. According to Theorem 3, $\mathbf{p}_{m_i}$ and $\mathbf{p}_{m_i}$ are two HABPs. From Definition 7, $\mathbf{p}_{m_{ij}} = \mathbf{p}_{m_i} \oplus \mathbf{p}_{m_j}$. Thus, $\mathbf{p}_{m_{ij}}$ is in the set C_{m+1}, which contradicts the hypothesis.

From Theorem 4, we know that all candidate patterns can be generated by the pattern join strategy. The following example shows that the pattern join strategy outperforms the enumeration tree strategy.

Example 11. In this example, we use the database and parameters in Example 8 to illustrate.

The number of HABPs with length three is 9. The enumeration tree strategy will generate 9 × 4 = 36 candidate patterns, since each HABP will generate four candidate patterns. However, the pattern join strategy generates only 14 candidate patterns. From Table 6, we can see that the number of candidate patterns generated by the pattern join strategy is much smaller than that by the enumeration tree strategy. Hence, the pattern join strategy is more efficient than the enumeration tree strategy.

4.3. HANP-Miner

In this section, we present HANP-Miner and analyze its time and space complexities.

HANP-Miner involves the following steps.

Step 1: Scan *SDB*, calculate the average utility of each item, and store HANPs with length 1 into *HAU* and HABPs with length 1 into HU_1.

Step 2: Generate the candidate pattern set C_{m+1} from HU_m using the pattern join strategy.

Step 3: For each pattern $\mathbf{p}$ in set C_{m+1}, calculate its average utility and the upper bound on the average utility. If pattern $\mathbf{p}$ is an HANP, then store it in sets *HAU* and HU_{m+1}. If pattern $\mathbf{p}$ is an HABP, then store it in the high average utility upper bound pattern set HU_{m+1}.

Step 4: Repeat Steps 2 and 3 until C_{m+2} or HU_{m+1} is empty. The HANPs are stored in the set *HAU*.

HANP-Miner is shown in Algorithm 3. Since Theorem 3 shows that the upper bound on the average utility satisfies the Apriori property, we use the PatternJoin algorithm to generate candidate patterns, as shown in Algorithm 4. To efficiently find the first position to start, PatternJoin adopts the binary search principle which was used in our previous work [34].

Algorithm 3 HANP-Miner

Input: Sequence database *SDB*, *minpua*, *gap* = [*a*, *b*], and the utilities array.
Output: The set of HANPs *HAU*

```
Scan the sequence database SDB, calculate the average utility of each character, and store the HANPs with length 1 into HAU_1 and HABPs with length 1 into HU_1;
C_2 ← PatternJoin (HU_1);
m ← 2;
while C_m ≠ NULL do
    for each pattern p in C_m do
        sup(p,SDB) ← DFOM (SDB, p, gap);
        PAU(p,SDB) ← Σ_{i=1}^{m} array.get(p[i]) × sup(p,SDB)/m;
        if PAU(p, SDB) ≥ minpau then
            HAU_m ← HAU_m ⋃ p
            HU_m ← HU_m ⋃ p;
        else
            SPU(p,SDB) ← array.max × sup(p,SDB);
            if SPU(p, SDB) ≥ minpau then
                HU_m ← HU_m ⋃ p;
            end if
        end if
    end for
    C_{m+1} ← PatternJoin (HU_m);
    m ← m + 1;
end while
return HAU;
```

Algorithm 4 PatternJoin

Input: HABPs with length of *m*, i.e. HU_m
Output: Candidate patterns with length *m*+1, i.e. C_{m+1}

```
C_{m+1} ← {};
for i=1 to HU_m.size do
    p_suffix ← suffix(HU_m[i]);
    start ← Use binary search to find the first pattern q in HU_m whose prefix pattern is the same as p_suffix;
    p_prefix ← prefix(HU_m[start]);
    while p_suffix == p_prefix do
        r = HU_m[i] ⨁ HU_m[start];
        C_{m+1} ← C_{m+1} ⋃ r;
        start ← start+1;
        p_prefix ← prefix(HU_m[start]);
    end while
end for
return C_{m+1};
```

Theorem 5. *The time complexity of HANP-Miner is $O(M \times L \times C)$, where M, L, and C are the maximum pattern length, the total length of SDB, and the number of candidate patterns, respectively.*

Table 6
Number of candidate patterns with different lengths.

Pattern length	Number of candidate patterns by enumeration tree	Number of candidate patterns by pattern join	Number of HANPs
Length = 1	4	4	2
Length = 2	16	9	6
Length = 3	36	14	8
Length = 4	56	12	7
Length = 5	48	6	5
Length = 6	24	3	1
Length = 7	12	0	0
Total	196	48	29

Proof. According to Theorem 1, the time complexity of DFOM is $O(m \times L)$, where m is the length of the pattern. Thus, the time complexity of DFOM in HANP-Miner is $O(M \times L)$. The utilities are stored in a sorted array. Thus, the time complexity of getting an item with binary search is $O(\log(r))$, where r is the size of Σ. Therefore, the time complexity of calculating $\sum_{j=1}^{m} U(p_j)$ is $O(M \times \log(r))$. Thus, the time complexity of calculating $PU(\mathbf{p}, SDB)$ of a candidate pattern is $O(M \times (L + \log(r)))$.

Now, we analyze the time complexity of PatternJoin. Suppose the size of HU_m is h_m. Thus, the time complexity of Line 4 is $O(log(h_m))$. The time complexity of Lines 6 to 11 is $O(r)$. Therefore, the time complexity of PatternJoin is $O(h_m \times (log(h_m) + r))$. Since there are C candidate patterns and $C \geq \sum_{m=1}^{M} h_m$, the time complexity of generating candidate patterns is $O(C \times (log(h_m) + r))$. Hence, the time complexity of HANP-Miner is $O((M \times (L + \log(r)) + log(h_m) + r) \times C) = O(M \times L \times C)$ based on the fact that r and h_m are far less than L, i.e., $r \ll L$.

Theorem 6. *The space complexity of HANP-Miner is $O(M \times n_1 + M \times C)$, where n_1 is the maximum sequence length in SDB.*

Proof. The space complexity of HANP-Miner is made up of four parts: the space complexity of DFOM, the utilities of the items, HABPs, and HANPs. According to Theorem 1, the space complexity of DFOM is $O(M \times n_1)$. The utilities of the items are stored in array whose space complexity is $O(r)$. Since there are C candidate patterns, the space complexity of HABPs is $O(M \times C)$. From Theorem 4, we know that the space complexity of HANPs is less than that of HABPs. Thus, the space complexity of HANPs is also $O(M \times C)$. Since r is far less than n_1, i.e., $r \ll n_1$, the space complexity of HANP-Miner is $O(M \times n_1 + r + M \times C + M \times C) = O(M \times n_1 + M \times C)$.

5. Experimental analysis

Section 5.1 describes the experimental datasets used and the algorithms used for comparison. In Section 5.2, we analyze the mining efficiency of different candidate pattern generation and support calculation strategies, and in Section 5.3, we compare and analyze the mining ability of the proposed algorithm and competitive algorithms. Section 5.4 presents a comparison of the mining results of NOSEP, GSgrow, and HANP-Miner, and an analysis of the superiority of adding the high average utility to traditional SPM. In Section 5.5, we compare the results of high average utility and high utility sequential pattern mining. In Section 5.6, we analyze the mining performance of HANP-Miner with different parameters. Section 5.7 explores the application of HANP-Miner in biological and sales sequences. All experiments were conducted on a computer with an Intel(R)Core(TM)i5-4210U processor, 8 GB memory, the Windows 8.1 operating system, and VC++6.0 as the experimental environment.

5.1. Datasets and baseline methods

Table 7 summarizes the experimental datasets. To verify the performance of our algorithm, we randomly set the utility for each item, since all datasets are not quantitative datasets.

To evaluate the efficiency of HANP-Miner, six experiments are designed. In the first, we evaluate the running efficiency by reporting the number of HANPs, running time, the number of candidate patterns, and memory usage. The second experiment is to explore the mining ability of HANP-Miner on different lengths DNA and virus sequences. The third experiment aims to analyze the advantage of adding the concept of high average utility to traditional SPM. In the fourth experiment, the proposed algorithm is compared with the high utility and average utility sequential pattern mining algorithms. The fifth experiment shows the impact of parameters on the extraction process. In the last experiment, the application of HANP-Miner in real-world scenarios is explored on the virus and sales sequences. Brief descriptions of these algorithms used in the above experiments are given as follows.

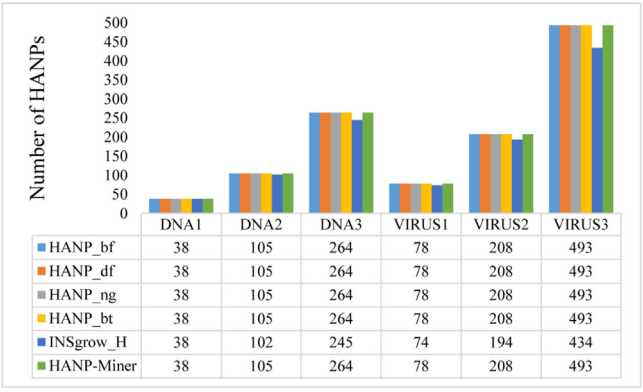
1. HANP_bf and HANP_df: To verify the efficiency of the pattern join strategy, we propose the HANP_bf and HANP_df algorithms, which apply breadth-first and depth-first strategies to generate candidate patterns, respectively.
2. NETGAP_H, BackTr_H, and INSgrow_H: To evaluate the superiority of HANP-Miner using the depth-first online matching algorithm to calculate the support, we design the NETGAP_H, BackTr_H, and INSgrow_H algorithms, which employ the NETGAP [71], the BackTr [47], and the INSgrow [25] algorithms in the support calculation step, respectively.
3. HANP_nogap: To analyze the effect of the gap constraint on the mining results of HANP-Miner, we propose the HANP_nogap algorithm to mine high average utility patterns with no gap constraint.
4. INSupport_H and Sail_H: To demonstrate the impact of nonoverlapping conditions, we select the INSupport_H [28] and Sail_H [72] algorithms to mine high average utility patterns with gap constraint under no condition and one-off condition, respectively.
5. NOSEP and GSgrow: To report the difference between HANP-Miner and classic pattern mining algorithms, we select NOSEP algorithm [34] and GSgrow algorithm [25] as competitor algorithms.
6. USpan and USpan_A: To show the difference between high average utility sequential pattern mining and high utility sequential pattern mining, we select USpan algorithm [73] and USpan_A algorithm which use USpan method to mine high average utility pattern as competitor algorithms.

The algorithms and datasets can be downloaded from https://github.com/ wuc567/Pattern-Mining/tree/master/HANP-Miner.

Table 7
Datasets.

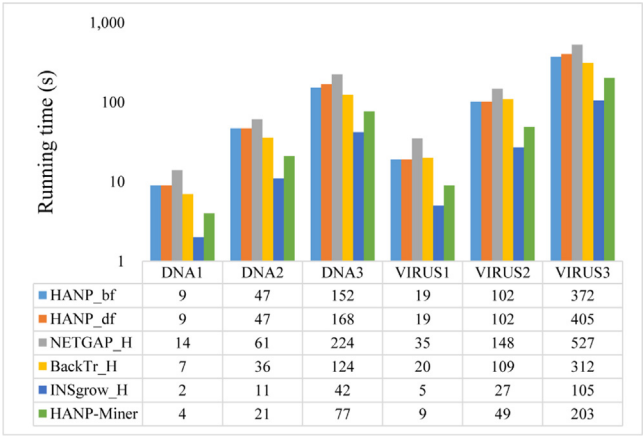
Dataset	Type	From	$\|\Sigma\|$	Length
DNA1[a]	DNA	Homo sapiens AL158070	4	6000
DNA2	DNA	Homo sapiens AL158070	4	8000
DNA3	DNA	Homo sapiens AL158070	4	10,000
VIRUS1[b]	Virus	Severe acute respiratory syndrome coronavirus 2	4	7000
VIRUS2	Virus	Severe acute respiratory syndrome coronavirus 2	4	9000
VIRUS3	Virus	Severe acute respiratory syndrome coronavirus 2	4	11,000
SARS-COV-2	Virus	Severe acute respiratory syndrome coronavirus 2	4	29,903
SARS[c]	Virus	Severe acute respiratory syndrome	4	29,751
BABYSALE[d]	Sales	Sales of baby products	6	29,971
GAMESALE [e]	Sales	Video game sales with ratings	14	100,302
PROTEIN[f]	Protein	ASTRAL95_1_161	20	130,684

[a]DNA1-3 sequence datasets were used in [34], and can be downloaded from http://www.ncbi.nlm.nih.gov/nuccore/AL158070.11.
[b]VIRUS1-3 and SARS-COV-2 are gene sequences of the virus causing COVID-19, and can be downloaded from https://www.ncbi.nlm.nih.gov/nuccore/MN908947.
[c]SARS is the gene sequence of the virus that caused severe acute respiratory syndrome in 2003, and can be downloaded from https://www.ncbi.nlm.nih.gov/nuccore/30271926.
[d]BABYSALE is a sales dataset for infant products, which can be downloaded from https://tianchi.aliyun.com/dataset/dataDetail?dataId=45.
[e]GAMESALE is a video game sales with rating, which can be downloaded from http://dataju.cn/Dataju/publishing/payContentPublishingDescription/688.
[f]PROTEIN sequence datasets were used in [34], and can be downloaded from http://www.ncbi.nlm.nih.gov/COG.



	DNA1	DNA2	DNA3	VIRUS1	VIRUS2	VIRUS3
HANP_bf	38	105	264	78	208	493
HANP_df	38	105	264	78	208	493
HANP_ng	38	105	264	78	208	493
HANP_bt	38	105	264	78	208	493
INSgrow_H	38	102	245	74	194	434
HANP-Miner	38	105	264	78	208	493

Fig. 4. Comparison of number of HANPs.



	DNA1	DNA2	DNA3	VIRUS1	VIRUS2	VIRUS3
HANP_bf	9	47	152	19	102	372
HANP_df	9	47	168	19	102	405
NETGAP_H	14	61	224	35	148	527
BackTr_H	7	36	124	20	109	312
INSgrow_H	2	11	42	5	27	105
HANP-Miner	4	21	77	9	49	203

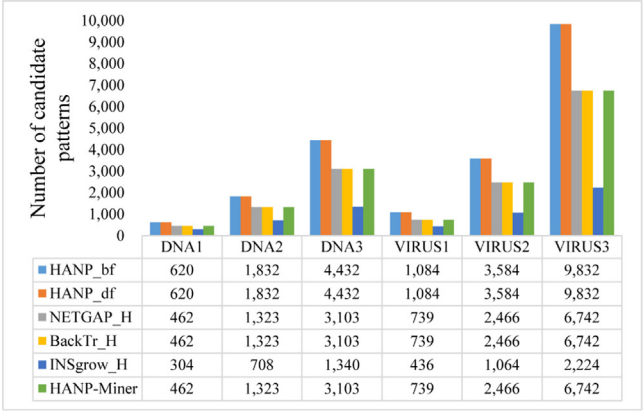
Fig. 5. Comparison of running time.

5.2. Mining efficiency

In this section, we compare the mining performance of HANP-Miner with different candidate pattern generation and support calculation strategies. We use six datasets: DNA1, DNA2, DNA3, VIRUS1, VIRUS2, and VIRUS3. The parameters are *gap* = [0,3], *min-pau* = 2500, U(A) = 3, U(G) = 6, U(C) = 6, and U(T) = 4. HANP_bf, HANP_df, NETGAP_H, BackTr_H, INSgrow_H, and HANP-Miner are selected. Comparisons of the number of HANPs, running time, the number of candidate patterns, and memory usage are shown in Figs. 4 to 7, respectively.

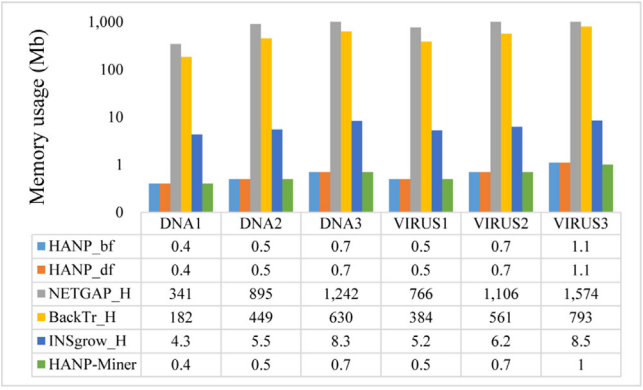
The results indicate the following observations.

1. HANP-Miner outperforms both HANP_bf and HANP_df. Fig. 4 shows that HANP_bf, HANP_df and HANP-Miner mine the same number of HANPs, while Fig. 5 shows that HANP-Miner runs faster than both HANP_bf and HANP_df. The difference between these three algorithms lies in the fact that they use different strategies to generate candidate patterns. HANP_bf, HANP_df, and HANP-Miner use breadth-first, depth-first and pattern join strategies to generate candidate patterns, respectively. As the analysis in Section 4.2, the pattern join strategy outperforms the breadth-first and depth-first strategies. Thus, HANP-Miner calculates fewer candidate patterns than HANP_bf and HANP_df. For example, according to Fig. 6, on VIRUS3, HANP-Miner calculates 6742 candidate patterns, while HANP_bf and HANP_df calculate 9832. Hence, HANP-Miner achieves better performance than HANP_bf and HANP_df.
2. HANP-Miner outperforms both NETGAP_H and BackTr_H. Figs. 4 and 6 show that NETGAP_H, BackTr_H and HANP-Miner find the same number of HANPs with the same



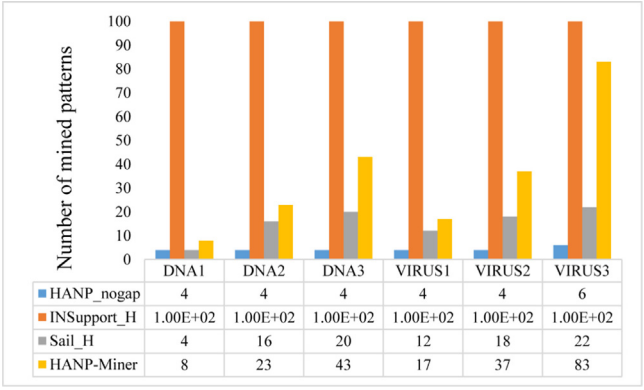
	DNA1	DNA2	DNA3	VIRUS1	VIRUS2	VIRUS3
HANP_bf	620	1,832	4,432	1,084	3,584	9,832
HANP_df	620	1,832	4,432	1,084	3,584	9,832
NETGAP_H	462	1,323	3,103	739	2,466	6,742
BackTr_H	462	1,323	3,103	739	2,466	6,742
INSgrow_H	304	708	1,340	436	1,064	2,224
HANP-Miner	462	1,323	3,103	739	2,466	6,742

Fig. 6. Comparison of number of candidate patterns.



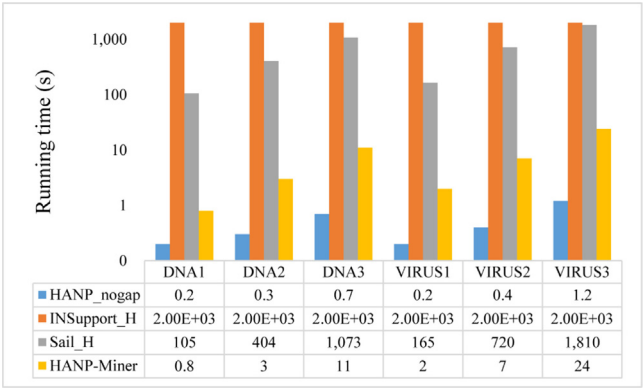
	DNA1	DNA2	DNA3	VIRUS1	VIRUS2	VIRUS3
HANP_bf	0.4	0.5	0.7	0.5	0.7	1.1
HANP_df	0.4	0.5	0.7	0.5	0.7	1.1
NETGAP_H	341	895	1,242	766	1,106	1,574
BackTr_H	182	449	630	384	561	793
INSgrow_H	4.3	5.5	8.3	5.2	6.2	8.5
HANP-Miner	0.4	0.5	0.7	0.5	0.7	1

Fig. 7. Comparison of memory usage.



	DNA1	DNA2	DNA3	VIRUS1	VIRUS2	VIRUS3
HANP_nogap	4	4	4	4	4	6
INSupport_H	1.00E+02	1.00E+02	1.00E+02	1.00E+02	1.00E+02	1.00E+02
Sail_H	4	16	20	12	18	22
HANP-Miner	8	23	43	17	37	83

Fig. 8. Comparison of number of mined patterns.



	DNA1	DNA2	DNA3	VIRUS1	VIRUS2	VIRUS3
HANP_nogap	0.2	0.3	0.7	0.2	0.4	1.2
INSupport_H	2.00E+03	2.00E+03	2.00E+03	2.00E+03	2.00E+03	2.00E+03
Sail_H	105	404	1,073	165	720	1,810
HANP-Miner	0.8	3	11	2	7	24

Fig. 9. Comparison of running time.

number of candidate patterns, but it can be seen from Figs. 5 and 7 that HANP-Miner runs faster than NETGAP_H and BackTr_H and uses less memory. For example, on the DNA2 dataset, NETGAP_H, BackTr_H and HANP-Miner discover 105 HANPs (as shown in Fig. 4), and the number of candidate patterns is 1323 (as shown in Fig. 6). However, from Figs. 5 and 7, the running time of the three algorithms is 61, 36, and 21 s, respectively, and the memory usage of the three algorithms is 895, 449, and 0.5 Mb, respectively. The reasons for this phenomenon are as follows. (i) HANP-Miner, NETGAP_H, and BackTr_H apply the same candidate pattern generation strategy, and therefore generate the same number of candidate patterns. (ii) HANP-Miner adopts DFOM to calculate the support which employs depth-first and backtracking strategies, and does not need to create a whole Nettree and prune invalid nodes. Therefore, DFOM is more efficient than NETGAP and BackTr. HANP-Miner hence runs faster than NETGAP_H and BackTr_H. (iii) According to Theorem 1, the space complexity of DFOM is $O(m \times n)$. DFOM thus consumes less memory than NETGAP and BackTr. Hence, HANP-Miner uses less memory than NETGAP_H and BackTr_H. These experiments verify HANP-Miner achieves better performance than HANP_ng and HANP_bt.

3. Although INSgrow_H runs faster than HANP-Miner, HANP-Miner outperforms INSgrow_H. From Fig. 5, we see that INSgrow_H runs faster than HANP-Miner, but Fig. 4 shows that INSgrow_H mines fewer HANPs. For example, on the VIRUS2 dataset, the running time of INSgrow_H and HANP-Miner is 27 and 49 s, respectively. However, INSgrow_H and HANP-Miner discover 194 and 208 HANPs, respectively. The reasons for this phenomenon are as follows. (i) INSgrow_H is an approximate mining algorithm, while HANP-Miner is a complete mining algorithm. INSgrow_H employs INSgrow to calculate the support, which adopts the depth-first search strategy to find the occurrences. This method will lead to the loss of some feasible occurrences. Hence, INSgrow is an approximate algorithm [25]. To overcome this drawback, HANP-Miner employs DFOM to calculate the support, which adopts depth-first and backtracking strategies to find the occurrences. DFOM can find all occurrences, and is a complete algorithm. Thus, DFOM is more complex than INSgrow due to its avoiding the loss of feasible occurrences, which causes DFOM to be slower than INSgrow. (ii) INSgrow_H checks fewer candidate patterns than HANP-Miner. From Fig. 6, the number of candidate patterns of INSgrow_H and HANP-Miner is 1,064 and 2,466, respectively. Since INSgrow_H is an approximate algorithm, it loses many feasible candidate patterns and HANPs. We know that with the increase of candidate patterns, the running time increases. Therefore, INSgrow_H runs faster than HANP-Miner, but finds fewer HANPs than HANP-Miner. In summary, HANP-Miner has better mining performance than INSgrow_H.

5.3. Mining ability

In this section, three competitive algorithms, HANP_nogap, INSupport_H and Sail_H, are examined in order to compare the mining ability of HANP-Miner for sequences with different lengths. Six datasets are selected: DNA1, DNA2, DNA3, VIRUS1, VIRUS2, and VIRUS3. The parameters are *gap* = [0,3], *minpau* = 4000, U(A) = 3, U(G) = 6, U(C) = 6, and U(T) = 4. The number of patterns and running time are shown in Figs. 8 and 9, respectively.

The results indicate the following observations.

1. The introduction of gap constraint effectively improves the mining ability of the algorithm. Although HANP_nogap runs faster compared with HANP-Miner, fewer HANPs can be found. For example, from Fig. 8, we can see that for the VIRUS3 dataset, HANP_nogap and HANP-Miner mine 6 and 83 patterns, respectively. The reason for this phenomenon is as follows. HANP_nogap does not consider the gap constraint, and hence the support is lower than that under the nonoverlapping condition. The number of candidate patterns and mined patterns of HANP_nogap are therefore lower than those of HANP-Miner, and HANP_nogap runs faster than HANP-Miner.
2. The mining ability of our proposed algorithm under the nonoverlapping condition is better than those under no

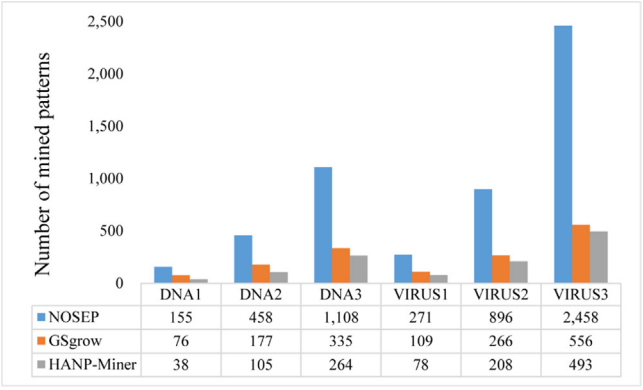


Fig. 10. Comparison of number of mined patterns.

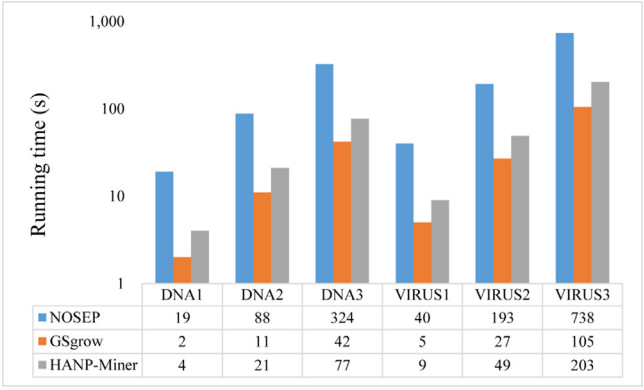


Fig. 11. Comparison of running time.

condition and the one-off condition. From Fig. 8, we can see that HANP-Miner mines fewer patterns than INSupport_H and more patterns than Sail_H. However, Fig. 9 shows that HANP-Miner has a shorter running time than INSupport_H and Sail_H. For example, it takes 11 s for HANP-Miner to mine 43 HANPs, 2.00E+03s for INSupport_H to mine 1.00E+02 patterns and 1073 s for Sail_H to mine 20 patterns. HANP-Miner is faster than both INSupport_H and Sail_H since it employs the depth-first strategy to calculate the pattern support, which effectively improves the efficiency of the algorithm and optimizes the running time. More importantly, HANP-Miner neither generates too many valueless patterns nor overlooks meaningful patterns, since the nonoverlapping condition is stricter than no condition and looser than the one-off condition. Hence, HANP-Miner can mine a moderate number of HANPs, and is faster than INSupport_H and Sail_H.

5.4. High average utility pattern mining VS frequent pattern mining

In this section, we compare the mining results of NOSEP, GSgrow, and HANP-Miner, and analyze the advantage of adding high average utility to traditional SPM. We select six datasets: DNA1, DNA2, DNA3, VIRUS1, VIRUS2, and, VIRUS3. The parameters are *gap* = [0,3], *minpau* = 2500, *minsup* = 416, $U(A) = 3$, $U(G) = 6$, $U(C) = 6$, and $U(T) = 4$. Since the maximum utility is 6 and $SPU(\mathbf{p},\mathbf{s}) = sup(\mathbf{p},\mathbf{s}) \times U_{max}$, we set *minpau* = 2500 in HANP-Miner and *minsup* = 2500/6 = 416 in NOSEP and GSgrow to make a fair comparison. The number of mined patterns, running time, and average utilities of patterns are shown in Figs. 10 to 12, respectively.

The results indicate the following observations.

For all datasets, the mining ability of HANP-Miner is superior to those of NOSEP and GSgrow. From Fig. 10, we see that HANP-Miner mines less patterns than NOSEP and GSgrow, while Fig. 11 shows that HANP-Miner runs faster than NOSEP, and slower than GSgrow. For example, on VIRUS2, NOSEP, GSgrow, and HANP-Miner mine 896, 266, and 208 patterns, respectively, and the running time of the three algorithms is 193, 27, and 49 s, respectively. The reasons for this phenomenon are as follows: (i) HANP-Miner not only adopts support to evaluate the importance of the patterns, but also uses utility to put forward more stringent requirements on the patterns. HANP-Miner therefore mines fewer patterns than NOSEP and GSgrow. (ii) HANP-Miner uses the DFOM algorithm to calculate the pattern support, which effectively reduces the calculation time. HANP-Miner therefore runs faster than NOSEP. Since GSgrow is an incomplete algorithm,

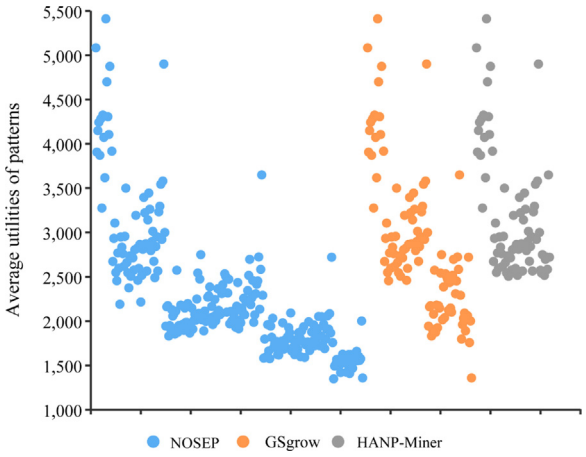


Fig. 12. Comparison of average utility of patterns on VIRUS1.

and fewer candidate patterns are calculated, GSgrow runs faster than HANP-Miner.

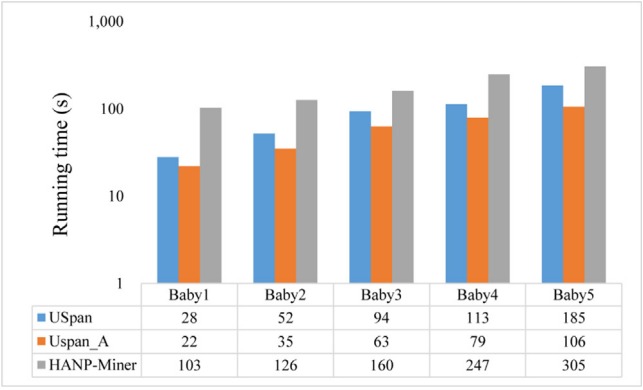
More importantly, from Fig. 12, we see that most discovered patterns by NOSEP and GSgrow are not HANPs since their average utilities are less than 2500. However, all mined patterns by HANP-Miner are HANPs. Hence, the mining ability of HANP-Miner is superior to those of NOSEP and GSgrow.

5.5. High average utility pattern mining VS frequent pattern mining

In this section, we compare the running time and memory usage of USpan, USpan_A, and HANP-Miner. We select BABYSALE as the dataset, and intercept five different length sequences. USpan is a high utility sequential pattern mining algorithm, while USpan_A and HANP-Miner are high average utility sequential pattern mining algorithms. Hence, we set greater minutil for USpan to make the number of patterns mined by these three algorithms similar. Since USpan_A does not consider the repetition, while our problem does, the threshold of USpan_A should be less than that of HANP-Miner. Thus, we set different thresholds for these three algorithms according to different requirements. Since the number of mined patterns is almost the same, we only report running time and memory usage as shown in Figs. 13 and 14, respectively.

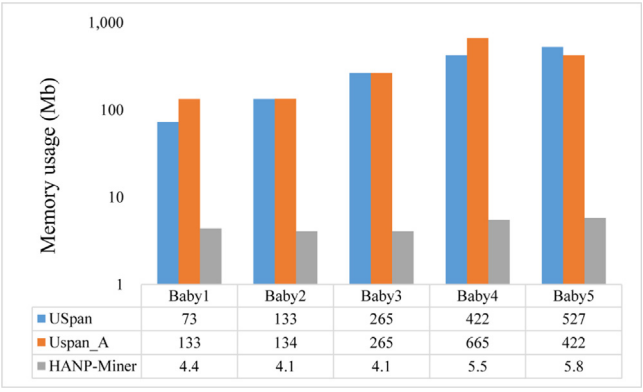
The results indicate the following observations.

For all datasets, USpan and USpan_A run faster than HANP-Miner, while the memory usage of USpan and USpan_A are higher than that of HANP-Miner. For example, from Figs. 13 and 14, on Baby2 dataset, the running time of USpan, USpan_A, and HANP-Miner is 52, 35, and 126 s, respectively, and the memory usage



	Baby1	Baby2	Baby3	Baby4	Baby5
USpan	28	52	94	113	185
Uspan_A	22	35	63	79	106
HANP-Miner	103	126	160	247	305

Fig. 13. Comparison of running time.



	Baby1	Baby2	Baby3	Baby4	Baby5
USpan	73	133	265	422	527
Uspan_A	133	134	265	665	422
HANP-Miner	4.4	4.1	4.1	5.5	5.8

Fig. 14. Comparison of memory usage.

of the three algorithms is 133, 134, and 4.1 Mb, respectively. The reasons for this phenomenon are as follows. (i) HANP-Miner algorithm needs to calculate the exact number of occurrences of the pattern in a sequence, while USpan and USpan_A algorithms only need to detect whether the pattern exists in a sequence. Hence, USpan and USpan_A run faster than HANP-Miner, but ignore the repetition of patterns, which will lead to the loss of many important patterns. (ii) HANP-Miner algorithm uses a simplified Nettree structure to calculate the support, and releases memory after each calculation, while USpan and USpan_A do not. Therefore, HANP-Miner uses less memory than USpan and USpan_A.

5.6. Mining performance with different parameters

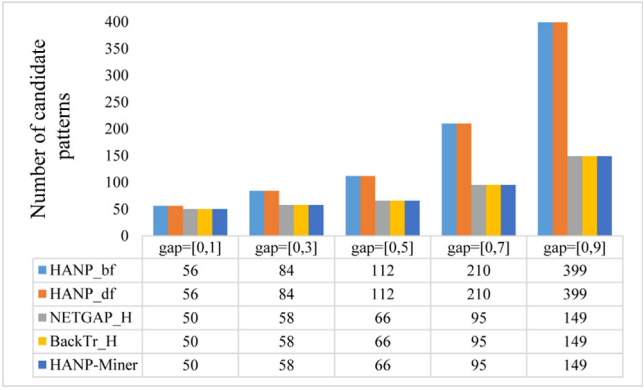
To show the impacts of different gap constraints, minpau and length of dataset on mining performance, we select HANP_bf, HANP_df, NETGAP_H, BackTr_H, and HANP-Miner as competitive algorithms.

(a) Gap constraint

We use GAMESALES as the dataset, set the *minpau* to 2500, and conduct experiments with different gap constraints [0,1], [0,3], [0,5], [0,7], and [0,9]. The number of candidate patterns, running time, and memory usage are shown in Figs. 15 to 17, respectively.

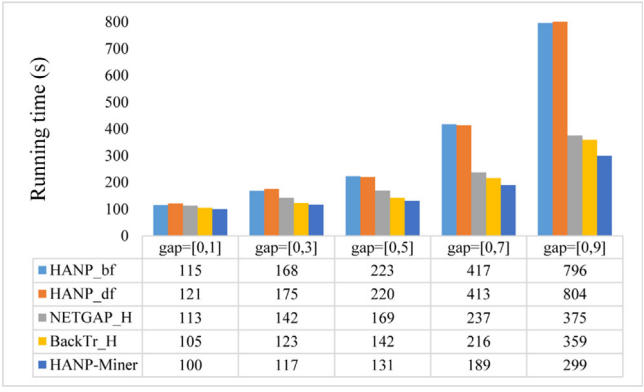
The results indicate the following observations.

With the increase of gap constraint, the number of candidate patterns, running time, and memory usage also increase. For example, when gap constraint is [0,1], there are 50 candidate patterns, and it costs 100 s and uses 0.8 Mb memory. However, when gap constraint is [0,5], there are 66 candidate patterns, and



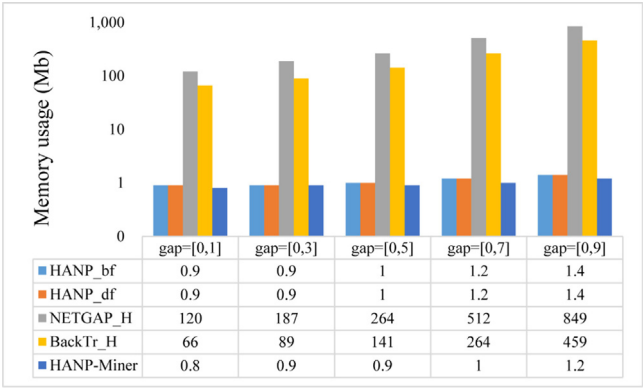
	gap=[0,1]	gap=[0,3]	gap=[0,5]	gap=[0,7]	gap=[0,9]
HANP_bf	56	84	112	210	399
HANP_df	56	84	112	210	399
NETGAP_H	50	58	66	95	149
BackTr_H	50	58	66	95	149
HANP-Miner	50	58	66	95	149

Fig. 15. Comparison of number of candidate patterns with different gap constraints.



	gap=[0,1]	gap=[0,3]	gap=[0,5]	gap=[0,7]	gap=[0,9]
HANP_bf	115	168	223	417	796
HANP_df	121	175	220	413	804
NETGAP_H	113	142	169	237	375
BackTr_H	105	123	142	216	359
HANP-Miner	100	117	131	189	299

Fig. 16. Comparison of running time with different gap constraints.



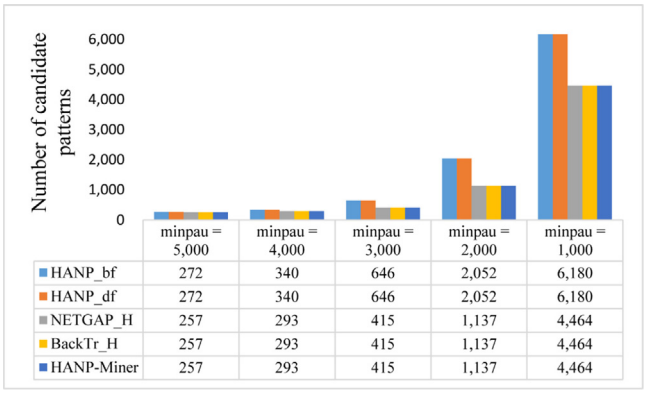
	gap=[0,1]	gap=[0,3]	gap=[0,5]	gap=[0,7]	gap=[0,9]
HANP_bf	0.9	0.9	1	1.2	1.4
HANP_df	0.9	0.9	1	1.2	1.4
NETGAP_H	120	187	264	512	849
BackTr_H	66	89	141	264	459
HANP-Miner	0.8	0.9	0.9	1	1.2

Fig. 17. Comparison of memory usage with different gap constraints.

it costs 131 s and uses 0.9 Mb memory. The reasons are as follows. With the increase of gap constraint, according to Definitions 2 and 3, the support also increases. Thus, more patterns can be HANPs and candidate patterns. We know that it costs HANP-Miner about 100 s to check 50 candidate patterns, which means that for one candidate pattern it costs about 100/50 = 2 s. Therefore, for 66 candidate patterns, it costs 131 s. According to Theorem 6, the space complexity of HANP-Miner is $O(M \times n_1 + M \times L)$. Hence, with the increase of the number of candidate patterns, the memory usage also increases.

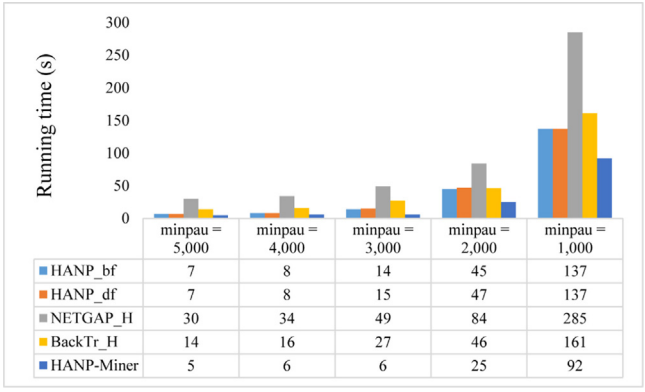
(b) *minpau*

We use PROTEIN as the dataset, set the gap constraint to [0,3], and conduct experiments with different *minpau* 5000, 4000, 3000,



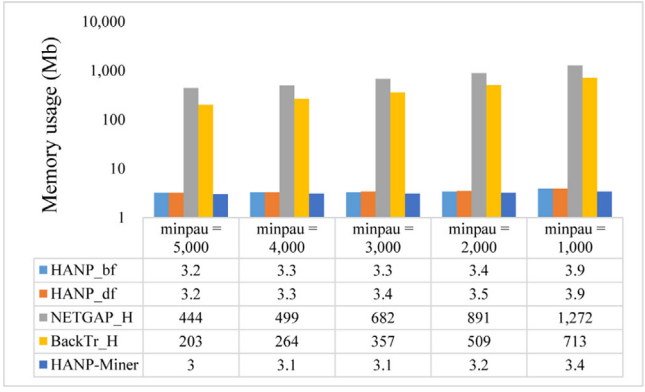
	minpau = 5,000	minpau = 4,000	minpau = 3,000	minpau = 2,000	minpau = 1,000
HANP_bf	272	340	646	2,052	6,180
HANP_df	272	340	646	2,052	6,180
NETGAP_H	257	293	415	1,137	4,464
BackTr_H	257	293	415	1,137	4,464
HANP-Miner	257	293	415	1,137	4,464

Fig. 18. Comparison of number of candidate patterns with different *minpau*.



	minpau = 5,000	minpau = 4,000	minpau = 3,000	minpau = 2,000	minpau = 1,000
HANP_bf	7	8	14	45	137
HANP_df	7	8	15	47	137
NETGAP_H	30	34	49	84	285
BackTr_H	14	16	27	46	161
HANP-Miner	5	6	6	25	92

Fig. 19. Comparison of running time with different *minpau*.



	minpau = 5,000	minpau = 4,000	minpau = 3,000	minpau = 2,000	minpau = 1,000
HANP_bf	3.2	3.3	3.3	3.4	3.9
HANP_df	3.2	3.3	3.4	3.5	3.9
NETGAP_H	444	499	682	891	1,272
BackTr_H	203	264	357	509	713
HANP-Miner	3	3.1	3.1	3.2	3.4

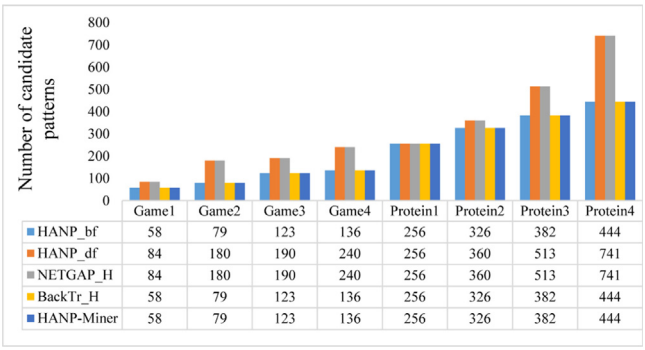
Fig. 20. Comparison of memory usage with different *minpau*.

2000, and 1000. The number of candidate patterns, running time, and memory usage are shown in Figs. 18 to 20, respectively.

The results indicate the following observations.

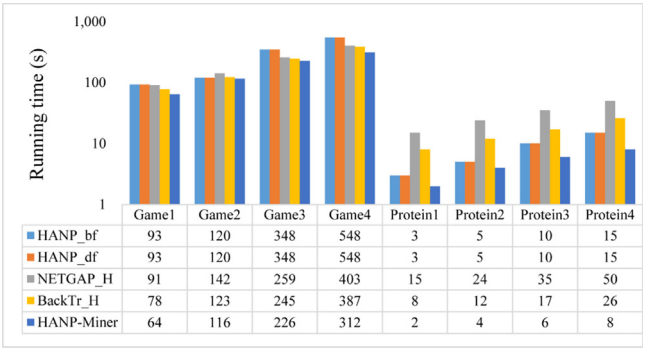
With the decrease of *minpau*, the number of candidate patterns, running time and memory usage increase. For example, when *minpau* is 5000, there are 257 candidate patterns, and it costs 5 s and uses 3 Mb memory. However, when *minpau* is 3000, there are 415 candidate patterns, and it costs 6 s and uses 3.1 Mb memory. The reason is as follows. With the decrease of *minpau*, according to Definition 4, more patterns can be HANPs and candidate patterns. Hence, the running time and memory usage increase.

(c) Length of dataset



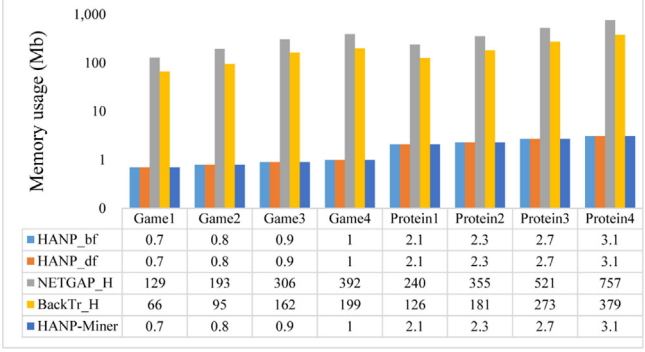
	Game1	Game2	Game3	Game4	Protein1	Protein2	Protein3	Protein4
HANP_bf	58	79	123	136	256	326	382	444
HANP_df	84	180	190	240	256	360	513	741
NETGAP_H	84	180	190	240	256	360	513	741
BackTr_H	58	79	123	136	256	326	382	444
HANP-Miner	58	79	123	136	256	326	382	444

Fig. 21. Comparison of number of candidate patterns with different length of datasets.



	Game1	Game2	Game3	Game4	Protein1	Protein2	Protein3	Protein4
HANP_bf	93	120	348	548	3	5	10	15
HANP_df	93	120	348	548	3	5	10	15
NETGAP_H	91	142	259	403	15	24	35	50
BackTr_H	78	123	245	387	8	12	17	26
HANP-Miner	64	116	226	312	2	4	6	8

Fig. 22. Comparison of running time with different length of datasets.



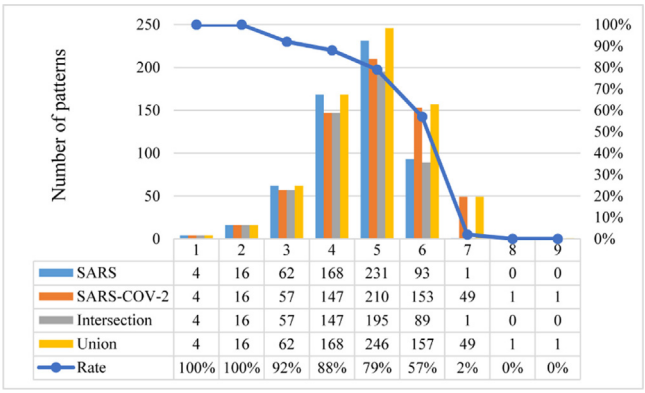
	Game1	Game2	Game3	Game4	Protein1	Protein2	Protein3	Protein4
HANP_bf	0.7	0.8	0.9	1	2.1	2.3	2.7	3.1
HANP_df	0.7	0.8	0.9	1	2.1	2.3	2.7	3.1
NETGAP_H	129	193	306	392	240	355	521	757
BackTr_H	66	95	162	199	126	181	273	379
HANP-Miner	0.7	0.8	0.9	1	2.1	2.3	2.7	3.1

Fig. 23. Comparison of memory usage with different length of datasets.

To verify the effect of different length of datasets on mining performance, we select two datasets in Table 7: GAMESALE and PROTEIN. To make the number of HANPs almost the same, we randomly set the utility for each item, *gap* = [0,3] and *minpau* = 3000 for GAMESALE, and *gap* = [0,3] and *minpau* = 1500 for PROTEIN. For the fairness of comparison, in this section, we select 4 different length for each dataset, i.e., Game1 with length = 70,000, Game2 with length = 80,000, Game3 with length = 90,000, Game4 with length = 100,302, Protein1 with length = 70,000, Protein2 with length = 90,000, Protein3 with length = 110,000, and Protein4 with length = 130,684. The number of candidate patterns, running time, and memory usage are shown in Figs. 21 to 23, respectively.

The results indicate the following observations.

With the increase of dataset length, the number of candidate patterns, running time, and memory usage also increase. For example, in PROTEIN dataset, when dataset length is 70,000, there



	1	2	3	4	5	6	7	8	9
SARS	4	16	62	168	231	93	1	0	0
SARS-COV-2	4	16	57	147	210	153	49	1	1
Intersection	4	16	57	147	195	89	1	0	0
Union	4	16	62	168	246	157	49	1	1
Rate	100%	100%	92%	88%	79%	57%	2%	0%	0%

Fig. 24. Mining results for different virus sequences.

are 256 candidate patterns, and it costs 2 s and uses 2.1 Mb memory. However, when dataset length is 130,684, there are 444 candidate patterns, and it costs 8 s and uses 3.1 Mb memory. The reasons are as follows. With the increase of dataset length, according to Definitions 3 and 4, more patterns can be HANPs and candidate patterns. According to Theorems 5 and 6, the time and space complexities of HANP-Miner are $O(M \times N \times L)$ and $O(M \times n_1 + M \times L)$, respectively. Hence, with the increase of the number of candidate patterns, the running time and memory usage also increase.

5.7. Case study

In this section, we explore the application of HANP-Miner in biological and sales sequences.

(a) Application in biological sequences

In this section, we explore the application of HANP-Miner in biological sequences. We conduct experiments on the SARS and SARS-COV-2 virus sequences to explore the similarities and differences between the two viruses. In this experiment, we set parameters *gap* = [0,3], *minpau* = 4000, U(A) = 2, U(G) = 3, U(C) = 2, U(T) = 3. The experimental results are shown in Fig. 24.

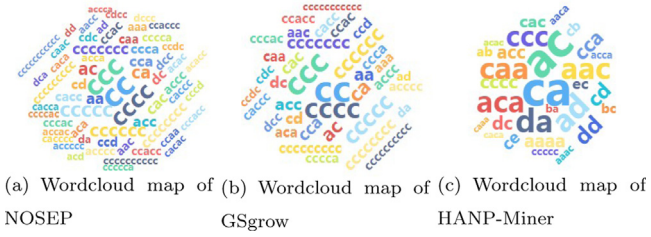
The experimental results can be summarized as follows.

When the pattern length is less than five, the number of HANPs discovered from the SARS and SARS-COV-2 datasets are almost the same, but when the pattern length is greater than four, there are differences between the number of HANPs. For example, it can be seen from Fig. 24 that when the pattern length is three, the number of HANPs discovered in the SARS and SARS-COV-2 datasets are 62 and 57, respectively, and the rate between the intersection and the union is 92%. When the pattern length is six, the number of HANPs are 93 and 153, respectively, and the rate between the intersection and the union is only 57%. This is because although the genetic patterns of SRAS and SRAR-COV-2 are basically the same, there is a combination of diversity in the longer patterns.

(b) Comparison of mining results in e-commerce sequence

Section 5.4 shows that HANP mining has more actual significance than frequent SPM. Here, we use a real case to verify this statement.

Retail merchants carry many kinds of products, each of which generates a different profit. Since it is impossible to recommend all products, the retailer tends to recommend products with higher profits. We therefore conduct experiments on the BABYSALE dataset to explore how to establish a reasonable marketing strategy to obtain the maximum profits. The BABYSALE dataset is a large-scale sales dataset that contains historical transaction information about the sales of baby products by Taobao



(a) Wordcloud map of NOSEP (b) Wordcloud map of GSgrow (c) Wordcloud map of HANP-Miner

Fig. 25. Wordcloud maps of mining results. In (a) and (b), the larger the font, the greater the support of the pattern. In(c), the larger the font, the greater the average utility of the pattern.

members. Each transaction can be reduced to the root category of the product. To simplify, we use symbols to represent the root category, and the daily transaction information will be converted into a sequence. In this dataset, there are six categories of products, listed as a, b, c, d, e, and f. In the experiment, we set the utility for each item as follows: U(a) = 1, U(b) = 2, U(c) = 0.3, U(d) = 1, U(e) = 5, and U(f) = 2, and set the following parameters: *gap*=[0,4], *minpau*=1200 for HANP-Miner, and *gap*=[0,4], *minsup*=1200/5=600 for NOSEP and GSgrow. Wordcloud maps [74] of NOSEP, GSgrow, and HANP-Miner are shown in Fig. 25.

The experimental results indicate the following observations.

1. The mining results of HANP-miner are more practical than NOSEP and GSgrow. According to Figs. 25 (a) and (b), the most frequent pattern is "cc" mined by NOSEP and GSgrow, since it has the highest frequency. However, actually considering with the utilities of various products, this product combination cannot generate higher profit. Many patterns composed of "c" are mined by NOSEP and GSgrow, such as patterns "cccc", "cccac", "ccccc", "cccccc". Although these long patterns have high supports, they are meaningless patterns. Hence, HANP mining has more actual significance than frequent SPM.
2. Adding c-type products to a product combination can generate higher profits. For example, b-type and e-type products have high unit profits, but when they are sold separately, their product sales are low, resulting in low profits. Only by bundling c-type products can a balance between sales and profits be achieved that make the retailer obtain a higher profit. Besides, combining a-type with c-type products can make the retailer to achieve the maximum profit. As shown in Fig. 25(c), patterns such as "ac", "ca", "caa", and "aac" have a larger font, which indicates that these goods create large profits for the retailer. After purchasing a-type and c-type products, users are most inclined to purchase a-type products, followed by c-type products. From Fig. 25(c), the retailer will recommend an a-type product, followed by a c-type product.

6. Conclusion and future work

To discover high average utility patterns in a *SDB*, we focus on HANP mining, which takes into account not only the frequency of the pattern and the utility value of each item, but also the influence of the pattern length of the sequence. To mine HANPs, we propose HANP-Miner, which is based on two main steps: pattern support calculation and candidate pattern reduction. To calculate the pattern support, we propose the DFOM algorithm, which applies a depth-first search online matching strategy. The space and time complexities of DFOM for a sequence are both $O(m \times n)$, which are lower than those of the existing methods.

Since HANP mining does not satisfy the antimonotonic property, we propose a method based on an upper bound on the average utility and a pattern join strategy that effectively reduces the number of candidate patterns. Extensive experiments are conducted on DNA, virus, sales sequence and protein datasets, and the experimental results show that HANP-Miner is superior to traditional SPM algorithms, i.e., it not only significantly improves the running time, but can also mine patterns that occur less frequently but are valuable. More importantly, this method can be used in product recommendations, which can help retailers achieve the maximum profits.

However, there are still some problems to be solved. First, HANP-Miner has to scan *SDB* repeatedly, which leads to the inefficiency of the algorithm. How to improve the efficiency of HANP-Miner is worth being investigated. Second, we focus on mining patterns with high average utility. However, other types of patterns also make sense, such as high number of occurrences but low utility patterns. In the future work, we will use more quality measures to mine other useful patterns. Third, we focus on mining HANPs from sequence database. It is worth noticing that the characteristic of this kind of sequence database is that there is only one item in each itemset. DNA sequences, protein sequences, and time series use this type of sequence database. However, the more general type of sequence database is that each itemset has more than one item. This type of sequence database can be used to describe the purchase behaviors of users. Apparently, the sequence database studied in this paper is a special case of general type of sequence database. It is more difficult and more valuable to mine HANPs on it. At last, we address HANP mining and propose HANP-Miner algorithm. The algorithm requires the users to input gap constraint and minpau with prior knowledge. In the future, self-adaptive gap constraint and Top-*k* HANP mining will be investigated to deal with the situation that users do not have prior knowledge.

CRediT authorship contribution statement

Youxi Wu: Conceptualization, Writing – original draft, Supervision, Funding acquisition. **Meng Geng:** Software, Writing – original draft, Validation, Investigation, Data curation. **Yan Li:** Conceptualization, Methodology, Formal analysis. **Lei Guo:** Validation, Resources. **Zhao Li:** Validation, Resources. **Philippe Fournier-Viger:** Writing – review & editing. **Xingquan Zhu:** Investigation, Writing – review & editing. **Xindong Wu:** Supervision, Funding acquisition.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work was partly supported by National Natural Science Foundation of China (61976240, 52077056, 917446209), National Key Research and Development Program of China (2016YFB1000 901), and Natural Science Foundation of Hebei Province, China (Nos. F2020202013, E2020202033).

References

[1] P. Fournier-Viger, A. Gomariz, T. Gueniche, A. Soltani, V.S. Tseng, SPMF: A java open-source pattern mining library, J. Mach. Learn. Res. 15 (1) (2015) 3389–3393.

[2] Y. Wu, Y. Wang, J. Liu, M. Yu, J. Liu, Y. Li, Mining distinguishing subsequence patterns with nonoverlapping condition, Cluster Comput. 22 (2019) 5905–5917, http://dx.doi.org/10.1007/s10586-017-1671-0.

[3] J. Yeo, S.W. Hwang, S. Kim, E. Koh, N. Lipka, Conversion prediction from clickstream: Modeling market prediction and customer predictability, IEEE Trans. Knowl. Data Eng. 32 (2) (2020) 246–259, http://dx.doi.org/10.1109/TKDE.2018.2884467.

[4] J. Ge, Y. Xia, J. Wang, C.H. Nadungodage, S.K. Prabhakar, Sequential pattern mining in databases with temporal uncertainty, Knowl. Inf. Syst. 51 (3) (2016) 821–850, http://dx.doi.org/10.1007/s10115-016-0977-1.

[5] Y. Wu, J. Fan, Y. Li, L. Guo, X. Wu, NetDAP: (delta, gamma) approximate pattern matching with length constraints, Appl. Intell. 50 (2020) 4094–4116, http://dx.doi.org/10.1007/s10489-020-01778-1.

[6] H. Jiang, X. Chen, T. He, Z. Chen, X. Li, Fuzzy clustering of crowdsourced test reports for apps, ACM Trans. Internet Technol. 18 (2) (2018) 1–28, http://dx.doi.org/10.1145/3106164.

[7] H. Jiang, X. Li, Z. Ren, J. Xuan, Z. Jin, Toward better summarizing bug reports with crowdsourcing elicited attributes, IEEE Trans. Reliab. 68 (1) (2019) 2–22, http://dx.doi.org/10.1109/TR.2018.2873427.

[8] G. Srivastava, J.C.W. Lin, M. Pirouz, Y. Li, U. Yun, A pre-large weighted-fusion system of sensed high-utility patterns, IEEE Sens. J. 1 (1) (2020) 99, http://dx.doi.org/10.1109/JSEN.2020.2991045.

[9] W. Gan, J.C.W. Lin, P. Fournier-Viger, H.C. Chao, P.S. Yu, A survey of utility-oriented pattern mining, IEEE Trans. Knowl. Data Eng. 33 (4) (2021) 1306–1327, http://dx.doi.org/10.1109/TKDE.2019.2942594.

[10] J.C.W. Lin, M. Pirouz, Y. Djenouri, C.F. Cheng, U. Ahmed, Incrementally updating the high average-utility patterns with pre-large concept, Appl. Intell. 50 (2020) 3788–3807, http://dx.doi.org/10.1007/s10489-020-01743-y.

[11] J.C.W. Lin, Y. Djenouri, G. Srivastava, U. Yun, P. Fournier-Vigerg, A predictive GA-based model for closed high-utility itemset mining, Appl. Soft Comput. 108 (2021) 107422, http://dx.doi.org/10.1016/j.asoc.2021.107422.

[12] T. Truong, D. Hai, B. Le, P. Fournier-Viger, H. Fujita, Efficient algorithms for mining frequent high utility sequences with constraints, Inform. Sci. 568 (5) (2021) http://dx.doi.org/10.1016/j.ins.2021.01.060.

[13] W. Gan, J.C.W. Lin, P. Fournier-Viger, H.C. Chao, P.S. Yu, HUOPM: High-utility occupancy pattern mining, IEEE Trans. Cybern. 50 (3) (2020) 1195–1208, http://dx.doi.org/10.1109/TCYB.2019.2896267.

[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, Inform. Sci. 513 (2019) 222–240, http://dx.doi.org/10.1016/j.ins.2019.10.033.

[15] J.C.W. Lin, T. Li, M. Pirouz, J. Zhang, P. Fournier-Viger, High average-utility sequential pattern mining based on uncertain databases, Knowl. Inf. Syst. 62 (2020) 1199–1228, http://dx.doi.org/10.1007/s10115-019-01385-8.

[16] G. Lee, U. Yun, Performance and characteristic analysis of maximal frequent pattern mining methods using additional factors, Soft Comput. 22 (2017) 4267–4273, http://dx.doi.org/10.1007/s00500-017-2820-3.

[17] S.J. Lin, Y.C. Chen, D.L. Yang, J. Wu, Discovering long maximal frequent pattern, in: Eighth International Conference on Advanced Computational Intelligence, 2016, pp. 136–142, http://dx.doi.org/10.1109/ICACI.2016.7449817.

[18] F. Min, Z. Zhang, W. Zhai, R. Shen, Frequent pattern discovery with tri-partition alphabets, Inform. Sci. 507 (2018) 715–732, http://dx.doi.org/10.1016/j.ins.2018.04.013.

[19] X. Dong, P. Qiu, J. Lü, L. Cao, T. Xu, Mining top-*k* useful negative sequential patterns via learning, IEEE Trans. Neural Netw. Learn. Syst. 30 (9) (2019) 2764–2778, http://dx.doi.org/10.1109/TNNLS.2018.2886199.

[20] T. Guyet, R. Quiniou, NegPSpan: Efficient extraction of negative sequential patterns with embedding constraints, Data Min. Knowl. Discov. 34 (2020) 563–609, http://dx.doi.org/10.1007/s10618-019-00672-w.

[21] L. Wang, X. Bao, L. Zhou, Redundancy reduction for prevalent co-location patterns, IEEE Trans. Knowl. Data Eng. 30 (1) (2018) 142–155, http://dx.doi.org/10.1109/TKDE.2017.2759110.

[22] T. Wang, L. Duan, G. Dong, Z. Bao, Efficient mining of outlying sequence patterns for analyzing outlierness of sequence data, ACM Trans. Knowl. Discov. Data 14 (5) (2020) 62, http://dx.doi.org/10.1145/3399671.

[23] J.C.W. Lin, Y. Djenouri, G. Srivastava, Efficient closed high-utility fusion patter model in large-scale databases, Inf. Fusion 76 (2021) 122–132, http://dx.doi.org/10.1016/j.inffus.2021.05.011.

[24] B. Le, H. Duong, T. Truong, P. Fournier-Viger, M. FcloS, FGenSM: two efficient algorithms for mining frequent closed and generator sequences using the local pruning strategy, Knowl. Inf. Syst. 52 (2017) 71–107, http://dx.doi.org/10.1007/s10115-017-1032-6.

[25] B. Ding, D. Lo, J. Han, S. Khoo, Efficient mining of closed repetitive gapped subsequences from a sequence database, in: IEEE 25th International Conference on Data Engineering, 2009, pp. 1024–1035, doi:10.1109/ICDE.2009.104.

[26] N. Mordvanyuk, B. López, A. Bifet, Verttirp: Robust and efficient vertical frequent time interval-related pattern mining, Expert Syst. Appl. 168 (2021) 114276, http://dx.doi.org/10.1016/j.eswa.2020.114276.

[27] Y. Wu, X. Liu, W. Yan, L. Guo, X. Wu, Efficient solving algorithm for strict pattern matching under nonoverlapping condition, J. Softw. (2021) http://dx.doi.org/10.13328/j.cnki.jos.006054.
[28] Y. Wu, L. Wang, J. Ren, W. Ding, X. Wu, Mining sequential patterns with periodic wildcard gaps, Appl. Intell. 41 (1) (2014) 99–116, http://dx.doi.org/10.1007/s10489-013-0499-4.
[29] H. Liu, L. Wang, Z. Liu, P. Zhao, X. Wu, Efficient pattern matching with periodical wildcards in uncertain sequences, Intell. Data Anal. 22 (4) (2018) 829–842, http://dx.doi.org/10.3233/IDA-173435.
[30] F. Xie, X. Wu, X. Zhu, Efficient sequential pattern mining with wildcards for keyphrase extraction, Knowl.-Based Syst. 115 (2017) 27–39, http://dx.doi.org/10.1016/j.knosys.2016.10.011.
[31] Y. Wu, X. Wang, Y. Li, L. Guo, Z. Li, J. Zhang, X. Wu, OWSP-Miner: Self-adaptive one-off weak-gap strong pattern mining, ACM Trans. Manag. Inf. Syst. (2021) http://dx.doi.org/10.1145/3476247.
[32] Y. Wu, C. Zhu, Y. Li, L. Guo, X. Wu, NetNCSP: Nonoverlapping closed sequential pattern mining, Knowl.-Based Syst. 196 (2020) 105812, http://dx.doi.org/10.1016/j.knosys.2020.105812.
[33] Q. Shi, J. Shan, W. Yan, Y. Wu, X. Wu, NetNPG: Nonoverlapping pattern matching with general gap constraints, Appl. Intell. 50 (2020) 1832–1845, http://dx.doi.org/10.1007/s10489-019-01616-z.
[34] Y. Wu, Y. Tong, X. Zhu, X. Wu, NOSEP: Nonoverlapping sequence pattern mining with gap constraints, IEEE Trans. Cybern. 48 (10) (2018) 2809–2822, http://dx.doi.org/10.1109/TCYB.2017.2750691.
[35] W. Song, Y. Liu, J. Li, Mining high utility itemsets by dynamically pruning the tree structure, Appl. Intell. 40 (2014) 29–43, http://dx.doi.org/10.1007/s10489-013-0443-7.
[36] T. Truong, H. Duong, B. Le, P. Fournier-Viger, U. Yun, H. Fujita, Efficient algorithms for mining frequent high utility sequences with constrsints, Inform. Sci. 568 (2021) 239–264, http://dx.doi.org/10.1016/j.ins.2021.01.060.
[37] U. Yun, D. Kim, E. Yoon, H. Fujita, Damped window based high average utility pattern mining over data streams, Knowl.-Based Syst. 144 (2018) 188–205, http://dx.doi.org/10.1016/j.knosys.2017.12.029.
[38] Y. Wu, Z. Tang, H. Jiang, X. Wu, Approximate pattern matching with gap constraints, J. Inf. Sci. 42 (5) (2016) 639–658, http://dx.doi.org/10.1177/0165551515603286.
[39] B.C. Kachhadiya, B. Patel, A survey on sequential pattern mining algorithm for web log pattern data, in: 2018 2nd International Conference on Trends in Electronics and Informatics (ICOEI), 2018, pp. 1269-1273. doi:10.1109/ICOEI.2018.8553691.
[40] U. Yun, H. Nam, J. Kim, H. Kim, W. Pedrycz, Efficient transaction deleting approach of pre-large based high utility pattern mining in dynamic databases, Future Gener. Comput. Syst. 103 (2020) 58–78, http://dx.doi.org/10.1016/j.future.2019.09.024.
[41] T. Van, B. Vo, B. Le, Mining sequential patterns with itemset constraints, Knowl. Inf. Syst. 57 (2018) 311–330, http://dx.doi.org/10.1007/s10115-018-1161-6.
[42] T. Van, B. Le, Mining sequential rules with itemset constraints, Appl. Intell. (2021) http://dx.doi.org/10.1007/s10489-020-02153-w.
[43] T. Van, A. Yoshitaka, B. Le, Mining web access patterns with super-pattern constraint, Appl. Intell. 48 (2018) 3902–3914, http://dx.doi.org/10.1007/s10489-018-1182-6.
[44] H.M. Huynh, L.T.T. Nauyen, B. Vo, U. Yun, Z.K. Oplatková, T. Hong, Efficient algorithms for mining clickstream patterns using pseudo-idlists, Future Gener. Comput. Syst. 107 (2020) 18–30, http://dx.doi.org/10.1016/j.future.2020.01.034.
[45] H.M. Huynh, L.T.T. Nauyen, B. Vo, A. Nauyen, V.S. Tseng, Efficient methods for mining weighted clickstream patterns, Expert Syst. Appl. 145 (2020) 112993, http://dx.doi.org/10.1016/j.eswa.2019.112993.
[46] A. Rahman, Y. Xu, K. Radke, E. Foo, Finding anomalies in scada logs using rare sequential pattern mining, in: International Conference on Network and System Security, 2016, pp. 499–506.
[47] F. Fumarola, P.F. Lanotte, M. Ceci, D. Malerba, CloFAST: Closed sequential pattern mining using sparse and vertical id-lists, Knowl. Inf. Syst. 48 (2) (2016) 429–463, http://dx.doi.org/10.1007/s10115-015-0884-x.
[48] M. Tran, B. Le, B. Vo, Combination of dynamic bit vectors and transaction information for mining frequent closed sequences efficiently, Eng. Appl. Artif. Intell. 38 (2015) 183–189, http://dx.doi.org/10.1016/j.engappai.2014.10.021.
[49] Z. Zhang, F. Min, G. Chen, S. Shen, Z. Wen, X. Zhou, Tri-partition state alphabet-based sequential pattern for multivariate time series, Cogn. Comput. (2021) http://dx.doi.org/10.1007/s12559-021-09871-4.
[50] Y. Wu, L. Luo, Y. Li, L. Guo, P. Fournier-Viger, X. Zhu, X. Wu, NTP-Miner: Nonoverlapping three-way sequential pattern mining, ACM Trans. Knowl. Discov. Data (2021) http://dx.doi.org/10.1145/3480245.
[51] S. Cheng, Y. Wu, Y. Li, F. Yao, F. Min, TWD-SFNN: Three-way decisions with a single hidden layer feedforward neural network, Inform. Sci. (2021) http://dx.doi.org/10.1016/j.ins.2021.07.091.
[52] P. Qiu, Y. Gong, Y. Zhao, L. Cao, C. Zhang, X. Dong, An efficient method for modeling nonoccurring behaviors by negative sequential patterns with loose constraint, IEEE Trans. Neural Netw. Learn. Syst. (2021) http://dx.doi.org/10.1109/TNNLS.2021.3063162.
[53] T. Truong, H. Duong, B. Le, P. Fournier-Viger, U. Yun, Efficient high average-utility itemset mining using novel vertical weak upper-bounds, Knowl.-Based Syst. 183 (2019) 104847, http://dx.doi.org/10.1016/j.knosys.2019.07.018.
[54] T. Truong, H. Duong, B. Le, P. Fournier-Viger, EHAUSM: An efficient algorithm for high average utility sequence mining, Inform. Sci. 515 (2020) 302–323, http://dx.doi.org/10.1016/j.ins.2019.11.018.
[55] X. Dong, Y. Gong, L. Cao, E-RNSP: An efficient method for mining repetition negative sequential patterns, IEEE Trans. Cybern. 50 (5) (2020) 2084–2096, http://dx.doi.org/10.1109/TCYB.2018.2869907.
[56] U. Yun, G. Lee, E. Yoon, Advanced approach of sliding window based erasable pattern mining with list structure of industrial fields, Inform. Sci. 494 (2019) 37–59, http://dx.doi.org/10.1016/j.ins.2019.04.050.
[57] W. Song, B. Jiang, Y. Qiao, Mining multi-relational high utility itemsets from star schemas, Intell. Data Anal. 22 (1) (2018) 143–165, http://dx.doi.org/10.3233/IDA-163231.
[58] C. Rjeily, G. Badr, A. Hassani, E. Andres, Medical data mining for heart diseases and the future of sequential mining in medical field, in: Machine Learning Paradigms, 2019, pp. 71–99. doi:10.3233/IDA-163231.
[59] P. Fournier-Viger, J. Li, J.C.W. Lin, T.T. Chi, R.U. Kiran, Mining cost-effective patterns in event logs, Knowl.-Based Syst. 191 (2020) 105241, http://dx.doi.org/10.1016/j.knosys.2019.105241.
[60] X. Chen, Y. Xie, H. Wang, Y. Zhao, J. Yin, Sentiment classification using negative and intensive sentiment supplement information, Data Sci. Eng. 4 (2) (2019) 109–118, http://dx.doi.org/10.3233/IDA-163231.
[61] H. Nam, U. Yun, E. Yoon, J.C.W. Lin, Efficient approach of recent high utility stream pattern mining with indexed list structure and pruning strategy considering arrival times of transactions-sciencedirect, Inform. Sci. 529 (2020) 1–27, http://dx.doi.org/10.1016/j.ins.2020.03.030.
[62] J. Kim, U. Yun, E. Yoon, J.C.W. Lin, P. Fournier-Viger, One scan based high average-utility pattern mining in static and dynamic databases, Future Gener. Comput. Syst. 111 (2020) 143–158, http://dx.doi.org/10.1016/j.future.2020.04.027.
[63] H.T. Lam, F. Moerchen, D. Fradkin, T. Calders, Mining compressing sequential patterns, Stat. Anal. Data Min. 71 (1) (2014) 34–52, http://dx.doi.org/10.1002/sam.11192.
[64] Y. Wu, Y. Wang, Y. Li, X. Zhu, X. Wu, Top-k self-adaptive contrast sequential pattern mining, IEEE Trans. Cybern. (2021) http://dx.doi.org/10.1109/TCYB.2021.3082114.
[65] X. Wu, X. Zhu, Y. He, A.N. Arslan, PMBC: Pattern mining from biological sequences with wildcard constraints, Comput. Biol. Med. 43 (2013) 481–492, http://dx.doi.org/10.1016/j.compbiomed.2013.02.006.
[66] H. Liu, Z. Liu, H. Huang, X. Wu, Sequential pattern matching with general gap and one-off condition, J. Softw. 29 (2018) 363–382.
[67] H. Li, Q. Yang, J. Wang, M. Li, Efficient mining of gap-constrained subsequences and its various applications, ACM Trans. Knowl. Discov. Data 6 (1) (2012) 1–39, http://dx.doi.org/10.1145/2133360.2133362.
[68] G. Srivastava, J.C.W. Lin, X. Zhang, Y. Li, Large-scale high-utility sequential pattern analytics in internet of things, IEEE Internet Things J. 1 (1) (2020) 99, http://dx.doi.org/10.1109/JIOT.2020.3026826.
[69] H. Kim, U. Yun, Y. Baek, J. Kim, B. Vo, E. Yoon, H. Fujita, Efficient list based mining of high average utility patterns with maximum average pruning strategies, Inform. Sci. 543 (2020) 85–105, http://dx.doi.org/10.1016/j.ins.2020.07.043.
[70] Y. Wu, R. Lei, Y. Li, L. Guo, X. Wu, HAOP-Miner:Self-adaptive high-average utility one-off sequential pattern mining, Expert Syst. Appl. (2021) http://dx.doi.org/10.1016/j.eswa.2021.115449.
[71] Y. Wu, C. Shen, H. Jiang, X. Wu, Strict pattern matching under non-overlapping condition, Sci. China Inf. Sci. 60 (1) (2017) 012101, http://dx.doi.org/10.1007/s11432-015-0935-3.
[72] G. Chen, X. Wu, X. Zhu, A.N. Arslan, Y. He, Efficient string matching with wildcards and length constraints, Knowl. Inf. Syst. 10 (4) (2006) 399–419, http://dx.doi.org/10.1007/s10115-006-0016-8.
[73] J. Yin, Z. Zheng, L. Cao, USpan: An efficient algorithm for mining high utility sequential patterns, in: 18th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD '12), 2012, pp. 660–668. doi:10.1145/2339530.2339636.
[74] F. Heimerl, S. Lohmann, S. Lange, T. Ertl, Word cloud explorer: Text analytics based on word clouds, in: 2014 47th Hawaii International Conference on System Sciences, 2014, pp. 1833–1842. doi:10.1109/HICSS.2014.231.