Expert Systems With Applications 184 (2021) 115449

Contents lists available at ScienceDirect

Expert Systems With Applications

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

HAOP-Miner: Self-adaptive high-average utility one-off sequential pattern mining

Youxi Wu [a,d,*], Rong Lei [a], Yan Li [c], Lei Guo [b], Xindong Wu [e,f]

[a] School of Artificial Intelligence, Hebei University of Technology, Tianjin 300401, China
[b] State Key Laboratory of Reliability and Intelligence of Electrical Equipment, Hebei University of Technology, Tianjin 300401, China
[c] School of Economics and Management, Hebei University of Technology, Tianjin 300401, China
[d] Hebei Key Laboratory of Big Data Computing, Tianjin 300401, China
[e] Key Laboratory of Knowledge Engineering with Big Data (Hefei University of Technology), Ministry of Education, Hefei 230009, China
[f] Mininglamp Academy of Sciences, Mininglamp Technology, Beijing 100084, China

ARTICLE INFO

Keywords:
Sequential pattern mining
Self-adaptive
High average utility
Apriori property
One-off condition

ABSTRACT

One-off sequential pattern mining (SPM) (or SPM under the one-off condition) is a kind of repetitive SPM with gap constraints, and has been widely applied in many fields. However, current research on one-off SPM ignores the utility (can be price or profit) of items, resulting in some low-frequency but extremely important patterns being ignored. To solve this issue, this paper addresses self-adaptive High-Average utility One-off sequential Pattern (HAOP) mining which has following three characteristics. Any two occurrences cannot share any letter in the sequence. The support (number of occurrences), utility and length of the pattern are considered simultaneously. The HAOP mining discovers patterns with a self-adaptive gap which means that users do not need to set the gap constraints. We propose an effective algorithm called HAOP-Miner that involves two key steps: support calculation and candidate pattern generation. For the support calculation, we propose a heuristic algorithm named the Reverse filling (Rf) algorithm that can effectively calculate the support by avoiding creating redundant nodes and pruning the redundant and useless nodes after finding an occurrence. Since HAOP mining does not satisfy the Apriori property, a support lower bound method combined with the pattern growth strategy is adopted to generate the candidate patterns. The experimental results first validate the effectiveness of HAOP-Miner, and then demonstrate that HAOP-Miner has better performance than other state-of-the-art algorithms. More importantly, HAOP-Miner is easier to mine valuable patterns. The algorithms and datasets are available at https://github.com/wuc567/Pattern-Mining/tree/master/HAOP-Miner.

1. Introduction

Sequential pattern mining (SPM) (He, Zhang, & Wu, 2019; Wu, Zhu, Li, Guo, & Wu, 2020) is a type of data mining method in which subsequences (also known as patterns) are discovered from sequences (Fournier-Viger, Li, Lin, Kiran, & Fujita, 2019). This approach has been widely applied in many fields, such as big data mining (Wu, Zhu, Wu, & Ding, 2013), big data intelligence (Liu et al., 2019; Wu & Wu, 2019), inspection reports (Jiang, Chen, He, Chen, & Li, 2018; Jiang, Li, Ren, Xuan, & Jin, 2019), e-commerce shopping analysis (Le, Tran, & Vo, 2015; Nam, Yun, Yoon, & Lin, 2020), and biological sequence analysis (Wang, Hou, & Wang, 2018). Many SPM methods have been proposed to meet a range of requirements, such as constrast SPM (Wu, Wang, Li, Zhu, & Wu, 2021; Ghosh, Li, Cao, & Ramamohanarao, 2017) negative SPM (Dong, Gong, & Cao, 2020; Dong, Qiu, L, & Xu, 2019; Chen et al., 2019), tri-partition pattern mining (Min, Zhang, Zhai, & Shen, 2020), high utility pattern mining (Choi & Park, 2019; Gan, Lin, Fournier-Viger, Chao, & Yu, 2020; Kim et al., 2021) and gap constraint SPM (Wu, Tong, Zhu, & Wu, 2018; Zhang, Kao, Cheung, & Yip, 2007). One of the disadvantages of traditional SPM is that it neglects the repetition in the sequence. For example, pattern "AC" occurs in sequence "ABC". Thus, the support (number of occurrences) of pattern "AC" in sequence "ABC" is 1 according to traditional SPM. However, pattern "AC" occurs more than once in sequence "AACCACC". If the support of pattern "AC" in sequence "AACCAC" is also 1, the repetition is neglected. To solve this issue and avoid mining some meaningless patterns, gap constraint SPM was proposed. The gap constraint pattern can be expressed as $\mathbf{p} = p_1[a, b]p_2 \cdots p_{m-1}[a, b]p_m$, where a and $b(0 \leqslant a \leqslant b)$ represent the minimum and

* Corresponding author at: School of Artificial Intelligence, Hebei University of Technology, Tianjin 300401, China.
E-mail address: wuc@scse.hebut.edu.cn (Y. Wu).

https://doi.org/10.1016/j.eswa.2021.115449
Received 25 October 2020; Received in revised form 22 December 2020; Accepted 12 June 2021
Available online 27 June 2021

maximum wildcards, respectively. For example, pattern "C[1,2]G" means that there are one or two wildcards between "C" and "G". Thus, pattern "C[1,2]G" occurs in sequence "CAG", while does not occur in sequence "CG". Since patterns can be flexibly matched, gap constraint SPM has been used in many applications such as time series (Huang, Jaysawal, Chen, & Wu, 2019; Miao, Vespier, Cachucho, Meeng, & Knobbe, 2016; Sumalatha & Subramanyam, 2020), biological sequence retrieval (Ghosh et al., 2017) and feature selection (Wei, Xing, Shi, Ji, & Zou, 2017).

One-off SPM (Huang et al., 2009; Wu, Xie, Huang, Hu, & Gao, 2013) is a branch of gap constraint SPM that mines frequent patterns from sequences under the one-off condition, which refers that each letter in the sequence can be used at most once. An illustrative example is given below.

Example 1. Suppose we have sequence $\mathbf{s} = s_1s_2s_3s_4s_5s_6s_7s_8 =$ ACGAGACG, pattern $\mathbf{p_1} = p_1p_2 =$ A[0,3]G.

From Fig. 1, it can be seen that there are five occurrences for pattern $\mathbf{p_1}$ in sequence **s**. The sub-sequence s_1s_3 is an occurrence of pattern $\mathbf{p_1}$ that can be written as $\langle 1,3\rangle$. Similarly, the other four occurrences are $\langle 1,5\rangle$, $\langle 4,5\rangle$, $\langle 4,8\rangle$ and $\langle 6,8\rangle$. Occurrences $\langle 1,3\rangle$ and $\langle 1,5\rangle$ do not satisfy the one-off condition since s_1 is used twice. However, occurrences $\langle 1,3\rangle$ and $\langle 4,5\rangle$ satisfy the one-off condition since there is no common used character in sequence. There are three occurrences $\langle 1,3\rangle$, $\langle 4,5\rangle$ and $\langle 6,8\rangle$ under the one-off condition. Hence, the support of pattern $\mathbf{p_1}$ in sequence **s** is 3.

Current research on one-off SPM ignores the utility (can be price or profit) of items. For example, although a gene may not appear frequently, its behaviour may be extremely remarkable. If we only consider the number of patterns, these highly expressed genes will not be mined (Morteza, Heidar, & Aijun, 2017). We also employ Example 1 to illustrate that high utility pattern mining is more meaningful. The utility of each item is shown in Table 1. The utility of pattern $\mathbf{p_1}$ = A[0,3]G in **s** is $(1+2)\times 3 = 9$ since utilities of "A" and "G" are 1 and 2, respectively. Similarly, the one-off support of pattern $\mathbf{p_2}$ = A[0,3]C[0,3]G in sequence **s** is 2. The utility of pattern is $(1+4+2)\times 2 = 14$. This example shows that the frequency of pattern $\mathbf{p_1}$ is greater than that of $\mathbf{p_2}$, while the utility of pattern $\mathbf{p_1}$ is less than that of $\mathbf{p_2}$. Therefore, high utility one-off SPM is worthy to be investigated, and the following two issues should be considered.

(1) Gap setting. It is very difficult to set suitable gap constraints without prior knowledge, which makes it challenging to discover valuable patterns (Wang, Duan et al., 2016). In Example 1, there is no occurrence for pattern $\mathbf{p_3}$ = A[4,5]G, which means that improper gap constraints will lead to mining failure.
(2) Length of the pattern. As the length of the pattern increases, its utility also increases (Lin, Li, Pirouz, Zhang, & Fournier-Viger, 2020). For instance, in Example 1, although $\mathbf{p_3}$ = C[0,1]G[0,1]A[0,1]G[0,1]A[0,1]C[0,1]

G appears only once in sequence **s**, $\mathbf{p_3}$ is also a high utility pattern since $PU(\mathbf{p_3},\mathbf{s}) = (4+2+1+2+1+4+2)\times 1 = 16$. Obviously, it is not reasonable.

To solve these issues, this paper proposes self-adaptive High-Average utility One-off sequential Pattern (HAOP) mining with the following

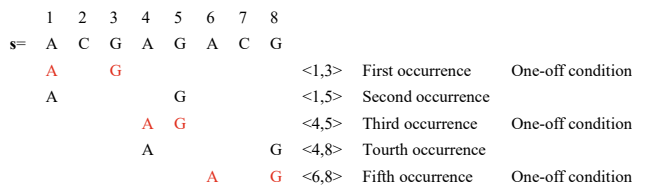


Fig. 1. All occurrences of pattern $\mathbf{p_1}$ = A[0,3]G in sequence **s** = ACGAGACG.

Table 1
Utility of each item.

Item	A	C	G
Utility	1	4	2

characteristics: (1) any two occurrences cannot share any letter in the sequence; (2) the support, utility and length of the pattern are considered simultaneously; (3) this method discovers patterns with a self-adaptive gap which means that users do not need to set the gap constraints. The main contributions of this paper are as follows.

(1) This paper addresses self-adaptive HAOP mining to discover HAOPs and proposes an effective algorithm called HAOP-Miner that contains two key steps: support calculation and candidate pattern generation.
(2) For the support calculation, we propose a Reverse filling (Rf) strategy which can effectively calculate the support since it avoids creating redundant nodes and does not need to prune the redundant and useless nodes after finding an occurrence.
(3) HAOP mining does not satisfy the Apriori property, and thus a support lower bound method combined with a pattern growth strategy is proposed to prune the candidate patterns effectively.
(4) The experimental results validate the effectiveness of HAOP-Miner, and demonstrate that HAOP-Miner has better performance than other state-of-the-art algorithms. More importantly, HAOP-Miner is easier to mine valuable patterns.

The remainder of this paper is organised as follows. Section 2 introduces related work. Section 3 defines the problem considered here. Section 4 designs the Rf strategy to calculate the support, and describes the high lower bound pattern, which satisfies the Apriori property and is used to prune the candidate patterns effectively. Finally, the HAOP-Miner algorithm is proposed. Section 5 reports the results of experiments on biological sequences and sales dataset. Finally, Section 6 presents the conclusion of this paper.

2. Related work

SPM has been widely applied in various fields, such as event log (Dalmas, Fournier-Viger, & Norre, 2017; Fournier-Viger, Li, Lin, Truong, & Rage, 2020), data streams (Chen, Xiao, Xin, Lin, & Lin, 2018), transaction databases (Karim, Cochez, Beyan, Ahmed, & Decker, 2018) and biological sequences (Wu, Zhu, He, & Arslan, 2013). Frequent pattern mining considers the support (number of occurrences) of patterns, but ignores the effect of utility (can be price or profit) on patterns (Yun, Kim, Yoon, & Fujita, 2018; Gan, Lin, Zhang, & Yu, 2020), meaning that some low-frequency but extremely important patterns are ignored, while high utility pattern mining (Qu, Liu, & Fournier-Viger, 2019; Song, Liu, & Li, 2014; Yun, Ryang, Lee, & Fujita, 2017) considers both the support and utility. Although Ahmed, Tanbeer, and Jeong (2010) have proposed two effective algorithms to find high utility patterns, high utility pattern mining unfortunately does not satisfy the Apriori property. A series of research methods involving an upper bound on utility have been proposed to solve this problem, such as sequence-weighted utilisation (Yin, Zheng, & Cao, 2012), prefix extension utility, reduced sequence utility (Wang, Huang, & Chen, 2016) and maximal extension utility strategies (Lin, Zhang, Fournier-Viger, & P, 2017). However, one of the main limitations of high utility pattern mining is that the length of the pattern is not considered, and thus short patterns tend to be overlooked. To solve this problem, high average utility pattern mining (Irfan & Mete, 2019; Lan, Hong, & Tseng, 2012; Lu, Vo, Nguyen, & Hong, 2014) was proposed. Since high average utility pattern mining does not satisfy the Apriori property, an upper bound on the average utility (Lin, Ren, Fournier-Viger, Hong, & Tzung, 2017; Lin, Li, Fournier-Viger, Zhang, & Guo, 2019) was proposed. However, the researches of the

above high average utility pattern mining ignore the repetition of pattern in a sequence, which means that a great deal of important information is lost. Gap constraint SPM was proposed in order to take into account the repetition (the number of occurrences) of a pattern in a sequence, and can be divided into three types, no condition (Li, Yang, Wang, & Li, 2012; Wu, Wang, Ren, Ding, & Wu, 2014; Wu, Fan, Li, Guo, & Wu, 2020), the non-overlapping condition (Ding, Lo, Han, & Khoo, 2009; Shi, Shan, Yan, Wu, & Wu, 2020; Wu, Shen, Jiang, & Wu, 2017) and the one-off condition (Liu, Wang, Liu, Zhao, & Wu, 2018; Xie, Wu, & Zhu, 2017). To illustrate the difference between these three constraints, the occurrences of pattern $\mathbf{p}$ = A*G*A in sequence $\mathbf{s} = s_1s_2s_3s_4s_5$ = AGAGA are shown in Table 2.

As shown in Table 2, there are 4 occurrences under no condition, i.e. $\{\langle 1,2,3\rangle, \langle 1,2,5\rangle, \langle 1,4,5\rangle, \langle 3,4,5\rangle\}$, as this imposes no restrictions on the use of a letter in a sequence. There are 2 occurrences under the non-overlapping condition, i.e. $\{\langle 1,2,3\rangle, \langle 3,4,5\rangle\}$ since in this case, two occurrences cannot use the same sequence letter in a given position. It is obvious that the letters in sequence **s** will be reused under no condition and the non-overlapping condition, meaning that there will be redundant information in the matching process. However, as mentioned in the Introduction section, there is only one occurrence under the one-off condition, i.e. $\langle 1, 2, 3\rangle$. Thus, the matching results under the one-off condition are more concise and practical. For example, in biological sequences, the gene forms specific RNA during transcription, and the position in the DNA sequence can only be matched by a pathogenic gene at most once (Liu, Liu, Huang, & Wu, 2018). Table 3 shows a comparison of related studies.

Although some studies focused on gap constraint SPM, they are quite different from our research. For example, Wu, Zhu et al. (2020) explored mining closed patterns under nonoverlapping condition. Our research focuses on mining high average utility patterns under one-off condition. From Table 3, the work in Wu, Zhu, He et al. (2013) is most similar to this paper. The differences are three aspects.

(1) The targets of the two researches are different. Wu, Zhu, He et al. (2013) focused on mining all frequent patterns, but the utility is ignored, which means that each item has the same weight. However, this paper mines HAOPs and takes utility into account, which means that each item has a different weight.
(2) The pruning candidate strategies are different. Frequent pattern mining under the one-off condition satisfies the Apriori property (Wu, Zhu, He et al., 2013). Unfortunately, HAOP mining does not satisfy the Apriori property. To tackle this issue, we propose a support lower bound method to prune the candidate patterns which is an Apriori-like method.
(3) The support calculation methods are different. Wu, Zhu, He et al. (2013) scanned the letters in the sequence multiple times to calculate the support. However, we adopt a more efficient Reverse filling strategy to calculate the support.

3. Problem definition

In this section, we formally define HAOP mining.

Definition 1. A sequence **s** is described by $s_1 \ldots s_i \ldots s_n$, where $s_i (1 \leqslant i \leqslant n) \in \sum$, $\sum$ represents the set of items in sequence **s**, and the size of $\sum$ can be expressed as $|\sum|$. Since this paper is a self-adaptive gap, we define pattern **p** as p_1*…*p_j*…*p_m,where * is the traditional wildcard, meaning that any letter can appear between the letters in the pattern.

Table 2
Occurrences under different constraints for pattern **p** = A*G*A in sequence **s** = AGAGA.

Constraints	Support	Occurrences
No condition	4	⟨1, 2, 3⟩, ⟨1, 2, 5⟩, ⟨1, 4, 5⟩, ⟨3, 4, 5⟩
Non-overlapping condition	2	⟨1, 2, 3⟩,⟨3, 4, 5⟩
One-off condition	1	⟨1, 2, 3⟩

Table 3
Comparison of related studies.

Literature	Type of pattern	Pruning strategy	Type of condition	Gap constraint	Repetitions
Yun et al. (2017)	High utility	Other	Ignored	–	Ignored
Qu et al. (2019)	High utility	Other	Ignored	–	Ignored
Irfan and Mete (2019)	High average utility	Other	Ignored	–	Ignored
Lin et al. (2019)	High average utility	Other	Ignored	–	Ignored
Zhang et al. (2007)	Frequent	Apriori-like	No condition	Given	Captured
Wu et al. (2018)	Frequent	Apriori	Non-overlapping condition	Given	Captured
Wu, Zhu et al. (2020)	Closed	Apriori	on-overlapping condition	Given	Captured
Wu, Zhu, He et al. (2013)	Frequent	Apriori	One-off condition	Self-adaptive	Captured
This paper	High average utility	Apriori-like	One-off condition	Self-adaptive	Captured

Definition 2. $L = \langle l_1, l_2, \ldots, l_m\rangle$ is an occurrence of pattern **p** in sequence **s**, if and only if $1 \leqslant l_1 < \ldots < l_j < \ldots < l_m \leqslant n$, where $s_{l_j} = p_j (1 \leqslant j \leqslant m \text{ and } 1 \leqslant l_j \leqslant n)$. Suppose $L' = \langle l'_1, l'_2, \ldots, l'_m\rangle$ is another occurrence. L and L' are two one-off occurrences if and only if for any k and q, $\forall 1 \leqslant k, q \leqslant m$, and l_k is not equal to l'_q. The one-off support of pattern **p** in sequence **s** is represented by $sup\ (\mathbf{p}, \mathbf{s})$.

Definition 3. The utility of pattern **p** of length m in sequence **s** is denoted as $PU(\mathbf{p}, \mathbf{s})$

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

where $U(p_j)$ is the utility of each item p_j.

Definition 4. The average utility of pattern **p** in sequence **s** is denoted as $PAU(\mathbf{p}, \mathbf{s})$:

$$PAU(\mathbf{p}, \mathbf{s}) = \frac{PU(\mathbf{p}, \mathbf{s})}{m} \tag{2}$$

Definition 5. If a pattern's average utility is no less than the minimum average utility threshold *minau*, the pattern is called an HAOP; otherwise, it is not.

Given minau and the utilities of items, our problem is to mine all HAOPs in the sequence.

Example 2. In Example 1, Σ = A,C,G and $|\Sigma| = 3$. The one-off occurrences of $\mathbf{p_1}$ = A*G in **s** are ⟨1,3⟩,⟨4,5⟩ and ⟨6,8⟩. Thus, $sup(\mathbf{p_1}, \mathbf{s}) = 3$. Suppose $minau = 4$. $\mathbf{p_1}$ is an HAOP since $PAU(\mathbf{p_1}, \mathbf{s}) = (1 + 2) \times 3/2 = 9/2 = 4.5 \geqslant 4$. Similarly, $\mathbf{p_3}$ = C*G*A*G*A*C*G is not an HAOP since $PAU(\mathbf{p_3}, \mathbf{s}) = (4 + 2 + 1 + 2 + 1 + 4 + 2) \times 1/7 = 16/7 < 4$. In this example, all HAOPs are $\{C, G, A*C, A*G, C*C, C*G, A*C*G\}$.

The main symbols used in this paper are listed in Table 4.

Table 4
Summary of symbols and their explanations.

Symbol	Definition
$sup(\mathbf{p},\mathbf{s})$	The support (number of occurrences) of pattern **p** in sequence **s**
$U(C)$	The utility of item C
$PU(\mathbf{p},\mathbf{s})$	The utility of pattern **p** in sequence **s**
$PAU(\mathbf{p},\mathbf{s})$	The average utility of pattern **p** in sequence **s**
lsup	The support lower bound
minau	A predefined minimum average utility threshold
minsup	A predefined minimum support threshold
prefix(**q**)	The prefix pattern of pattern **q**
suffix(**q**)	The suffix pattern of pattern **q**

4. Proposed algorithm

Fig. 2 shows the overall workflow of HAOP-Miner. HAOP-Miner has two key steps: support calculation and candidate pattern generation. Section 4.1 shows that the support calculation is an NP-hard problem. To calculate the pattern support, Section 4.2 explores a heuristic algorithm named the Positive filling (Pf) algorithm, which is easier to understand at first. To overcome the shortages of Pf, Section 4.3 roposes a more effective heuristic method named the Rf algorithm. Section 4.4 employs a support lower bound method combined with the pattern growth strategy to prune the candidate patterns. Finally, the HAOP-Miner algorithm is proposed in Section 4.5.

4.1. Theoretical analysis

Calculating $sup(\mathbf{p},\mathbf{s})$ is an NP-hard problem since the determination version is $sup(\mathbf{p},\mathbf{s}) = k$ which is an NP-complete problem. This problem can be proved by the reduction of the iterated shuffle problem which is proved to be NP-complete in Warmuth and Haussler (1984). The proof is shown as follows.

Suppose we have two sequence $\mathbf{v} = v_1 s_2 \ldots v_n$and $\mathbf{t} = t_1 t_2 \ldots t_n$ with length n. The shuffle of **s** and **t** is defined as $\mathbf{v} \odot \mathbf{t} = \{v_1 t_1 v_2 t_2 \ldots v_n t_n\}$. The iterated shuffle of **v** is $\varepsilon \cup \{\mathbf{v}\} \cup (\mathbf{v} \odot \mathbf{v}) \cup (\mathbf{v} \odot \mathbf{v} \odot \mathbf{v}) \ldots$. For example, $\mathbf{v} =$ ABAABB is in the iterated shuffle of $\mathbf{w} =$ AB since $\mathbf{v} \in (\mathbf{w} \odot \mathbf{w} \odot \mathbf{w})$. However, $\mathbf{v} =$ BAAB is not in the iterated shuffle of $\mathbf{w} =$ AB. The iterated shuffle problem is to determine whether or not **v** is in the iterate shuffle of **w**, where the sequence length $|\mathbf{v}| = k|\mathbf{w}|$. Apparently, $sup(\mathbf{p},\mathbf{s}) = k$ can be seen as the problem whether **s** is in the iterate shuffle of **p**.

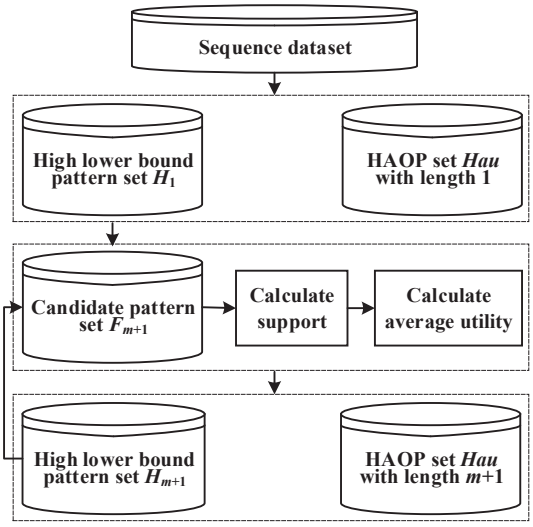


Fig. 2. Workflow of HAOP-Miner.

4.2. Pf

In this subsection, we propose a heuristic algorithm, named Pf, to calculate $sup(\mathbf{p},\mathbf{s})$ whose principle is shown as follows.

The Pf algorithm can be regarded as a method of brute force matching. It creates m queues at first. Each s_i will match with$p_1, p_2, \ldots, p_m$ one by one. If certain condition is satisfied, a node with ID i will be created in the j-th level. Thus, nodes with ID i may be created many times. Once a node is created in the m-th level, it means that an occurrence is found. After that, Pf will find and prune the redundant or useless nodes. The main steps of Pf are as follows.

Step 1. Create m queues, where m is the length of pattern **p**.

Step 2. For each s_i in **s**, we determine whether or not $s_i = p_j$ from p_1 to p_m. If $s_i = p_j$, then Rules 1 and 2 are employed to create node i in the j-th queue, denoted as n_j^i.

Rule 1. If $j = 1$, then node n_1^i will be created.

Rule 2. If $j > 1$ and $i > t$, then node n_j^i will be created, where t is the first node in the $(j-1)$-th level.

Step 3. When node n_m^i in the m-th queue is created, it means that an occurrence has been found. Then, we delete three kinds of nodes: the corresponding nodes of the occurrence, the related nodes under the one-off condition (redundant nodes), and the nodes that cannot reach the first queue (useless nodes).

Step 4. Iterate Steps 2 and 3, until all letters in the sequence are processed.

An illustrative example is shown as follows.

Example 3. Suppose we have sequence $\mathbf{s} = s_1 s_2 s_3 s_4 s_5 s_6 s_7 s_8 =$ AAACGACG, and a pattern $\mathbf{p} = p_1 * p_2 * p_3 =$ A*A*G.

Step 1. Three queues are created since the length of pattern **p** is three.

Step 2. s_1 is read and compared with p_1, p_2 and p_3 in turn. Since $s_1 = p_1 =$ A and $j = 1$, node n_1^1 is created according to Rule 1. The first node in the first queue is n_1^1 since there is no node in the second level. Although $s_1 = p_2 =$ A, Rule 2 is not satisfied. Thus, no node can be created in the second queue, and no node can be created in the third queue since $s_1 =$ A $\neq p_3$. s_2 is read and compared with p_1, p_2 and p_3 in turn. Since $s_2 = p_1$ and $j = 1$, node n_1^2 is created. Similarly, the first node in the first level is n_1^1. Since $2 > 1$ and $s_2 = p_2 =$ A, node n_2^2 is created according to Rule 2. Similarly, n_1^3 and n_2^3 are created.

Step 3. s_5 is read, and Rule 2 is satisfied since $s_5 = p_3$ and $5 > 2$. Thus, node n_3^5 is created in the third queue. When we create node n_3^5, we get an occurrence, i.e. $\langle 1,2,5 \rangle$. Then we remove the corresponding nodes of the occurrence, i.e. n_1^1, n_2^2 and n_3^5. In addition, node n_1^2 is pruned under the one-off condition. More importantly, we need to find and prune those nodes that cannot reach the first queue. After pruning node n_1^2, node n_2^3 is also pruned since it cannot reach the first queue.

Step 4. Steps 2 and 3 are repeated, and we obtain the occurrence $\langle 3,6,8 \rangle$. All nodes are shown in Fig. 3.

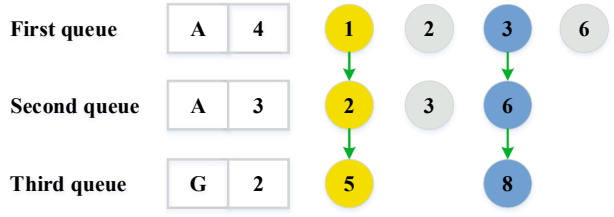


Fig. 3. Matching process for pattern $\mathbf{p} =$ A*A*G in sequence $\mathbf{s} =$ AAACGACG according to Pf algorithm. There are three queues since the length of pattern **p** is three. The yellow nodes indicate the first occurrence of pattern **p** in sequence **s**, i.e. $\langle 1,2,5 \rangle$. After nodes n_1^1, n_2^2 and n_3^5 have been removed, node n_1^2 is deleted under the one-off condition. Furthermore, node n_2^3 is pruned since it cannot reach the first queue. Similarly, we find the second occurrence, $\langle 3,6,8 \rangle$, which is represented by the blue nodes.

Pf algorithm is given in Algorithm 1.

Algorithm 1. Pf

Input: Sequence **s**, Candiate pattern **p** (where the length of **p** is m)
Output: $sup(\mathbf{p},\mathbf{s})$

```
Create m queues according to the length of pattern p;
for i = 1 to length of s step 1 do
  for j = 1 to m step 1 do
    if Rule 1 is satisfied then
      Create node n_1^i;
    end if
    if Rule 2 is satisfied then
      Create node n_j^i;
      if j == m then
        sup(p,s) + +;
        Delete the corresponding nodes, redundant nodes and useless nodes;
      end if
    end if
  end for
end for
return sup(p,s)
```

4.3. Rf

Although the Pf algorithm can be used to calculate the pattern support, Pf has two disadvantages. 1. Some redundant nodes are created. For example, in Fig. 3, nodes n_1^2, n_2^3 and n_1^6 are redundant. 2. After finding an occurrence, some redundant and useless nodes should be found and pruned. For example, when $\langle 1,2,5\rangle$ is found, nodes n_1^2 and n_2^3 which are redundant and useless nodes should be found and pruned. To avoid redundant creating and pruning nodes, we propose the Rf algorithm, whose principle is shown as follows.

Different from Pf, Rf matches s_i with p_m ,..., p_2, p_1 one by one. If certain condition is satisfied, a node with ID i will be created in the j-th level. Thus, node with ID i is created at most once. Therefore, Rf does not create redundant nodes. More importantly, Rf does not need to find and prune the redundant and useless nodes when an occurrence is found. Hence, Rf is more effective than Pf. The main steps of Rf are as follows.

Step 1. Create m queues.

Step 2. For each s_i in **s**, we determine whether or not $s_i = p_j$ from p_m to p_1. If $s_i = p_j$, then Rules 3 and 4 are proposed to create node i in the j-th queue, denoted as n_j^i.

Rule 3. If $j > 1$ and $num_j < num_{j-1}$, then node n_j^i will be created, where num_j represents the number of nodes in the j-th queue.

Rule 4. If $j = 1$, then node n_1^i will be created.

Step 3. When node n_m^i in the m-th queue is created, it means that an occurrence has been found. Step 2 is iterated until all letters in the sequence have been processed.

An illustrative example is shown as follows.

Example 4. We employ the same pattern and sequence as in Example 3.

Step 1. Three queues are created.

Step 2. s_1 is read and compared with p_3, p_2 and p_1 in turn. There is no node in the first queue, meaning that although $s_1 = p_2 =$ A, Rule 3 is not satisfied, and thus node n_2^1 cannot be created. Since $s_1 = p_1 =$ A and $j = 1$, node n_1^1 is created. s_2 is read, and node n_2^2 is created according to Rule 3 since there is one node in the first level and no node in the second level. s_2 needs no more judgment with the other letters in pattern **p**. We can create node n_1^3 in a similar way.

Step 3. s_5 is read, and node n_3^5 is created according to Rule 3 since $s_5 = p_3$ and there is one node in the second level and no node in the third level. When node n_3^5 is created, we obtain an occurrence, i.e. $\langle$1,2,5$\rangle$. Step 2 is iterated, and we obtain the occurrence $\langle$3,6,8$\rangle$. All nodes and occurrences are shown in Fig. 4.

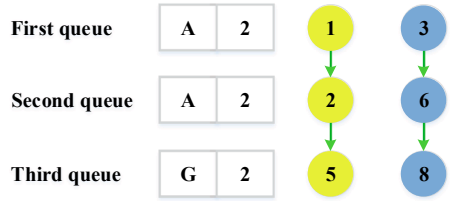


Fig. 4. Matching process for pattern **p** = A*A*G in sequence **s** = AAACGACG according to Rf algorithm.

The Rf algorithm is given in Algorithm 2.

Algorithm 2. Rf

Input: Sequence **s**, Candiate pattern **p** (where the length of **p** is m)
Output: $sup(\mathbf{p},\mathbf{s})$

```
Create m queues according to the length of pattern p;
for i = 1 to length of s step 1 do
  for j = m to 1 step 1 do
    if Rule 3 is satisfied then
      Create node n_j^i;
      if j == m then
        sup(p,s) + +;
      end if
    else if Rule 4 is satisfied then
      Create node n_1^i;
    end if
  end for
end for
sup(p,s)
```

The Rf algorithm has an advantage that it avoids creating useless nodes, which means that nodes with the same node ID will be created at most once. For example, in Fig. 3, there are two nodes with ID 2 according to Pf. However, in Fig. 4, there is only one node with ID 2 according to Rf. Therefore, Rf is more effective than Pf.

Theorem 1. *The time complexity of the Rf algorithm is* $O(m \times n)$*, where* m *and* n *are the length of* **p** *and* **s***, respectively.*

Proof. Each s_i is matched with p_j. Thus, in the worst case, s_i judges m times. Since the sequence length is n, the time complexity is $O(m \times n)$. □

Theorem 2. *The space complexity of the Rf algorithm is* $O(n)$.

Proof. According to the Rf algorithm, each s_i appears in the queues at most once. Therefore, no more than $O(n)$ nodes will be created. Hence, the space complexity is $O(n)$. □

4.4. Generating candidate patterns

In this subsection, we will show that HAOP mining does not satisfy the Apriori property at first. How to prune candidate patterns becomes a challenging problem. We propose a support lower bound approach to tackle this problem.

Example 5. Suppose we have sequence **s** = AAACGACG, and $minau$ = 4.5 and the utility of each item is shown in Table 1. Under the one-off condition, the related patterns are shown in Fig. 5. Suppose we have patterns $\mathbf{p_1}$ = A and $\mathbf{p_2}$ = A*C. we know that $sup(\mathbf{p_1},\mathbf{s}) = 4$ and $sup(\mathbf{p_2},\mathbf{s}) = 2$. According to Eq. (2), we know that $PAU(\mathbf{p_1},\mathbf{s}) = 1 \times 4/1 = 4$ and $PAU(\mathbf{p_2},\mathbf{s}) = (1+4) \times 2/2 = 5$. $\mathbf{p_1}$ is not an HAOP since $PAU(\mathbf{p_1},\mathbf{s}) = 4 < 4.5$. However, $\mathbf{p_2}$ is an HAOP since $PAU(\mathbf{p_2},\mathbf{s}) = 5 \geqslant 4.5$. Meanwhile, $\mathbf{p_2}$ is a super pattern of $\mathbf{p_1}$. This example illustrates that HAOP mining does not satisfy the Apriori property.

Now, we will introduce the principle of support lower bound approach.

Definition 6. The support lower bound is denoted as *lsup*,

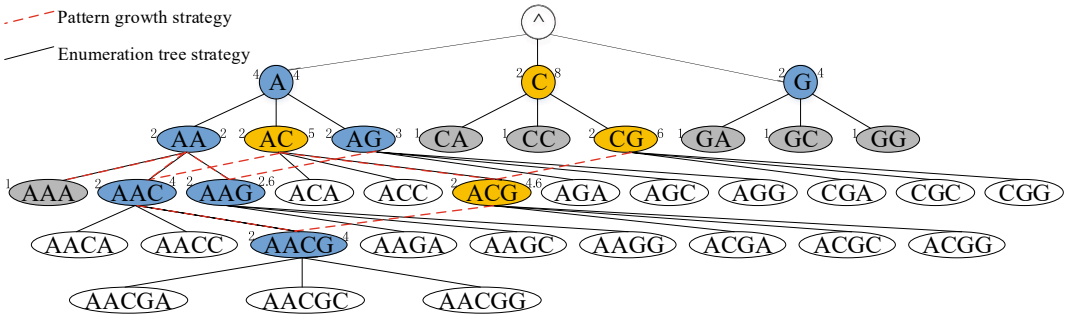


Fig. 5. The related patterns for sequence $\mathbf{s}$ = AAACGACG and *minau* = 4.5. The enumeration tree strategy generates 36 candidate patterns, i.e. all nodes in the figure. However, the pattern growth strategy generates 17 patterns, i.e. color marked nodes. The blue and yellow nodes can be used to generate new candidate patterns, and the grey nodes cannot. The yellow nodes are HAOPs {C, AC, CG, ACG}. The number in the left-top and right-top represent its support and average utility, respectively.

$$lsup = \left\lceil \frac{minau}{U_{max}} \right\rceil \tag{3}$$

where U_{max} represents the maximum utility of each item, and operator $\lceil . \rceil$ refers to round up.

Definition 7. If the support of a pattern is not less than *lsup*, the pattern is a high lower bound pattern.

Theorem 3. *The support of an HAOP is no less than lsup.*

Proof. The average utility of $\mathbf{p}$ with length m is $PAU(\mathbf{p},\mathbf{s}) = \frac{\sum_{j=1}^{m} U(p_j)}{m} \times sup(\mathbf{p},\mathbf{s}) \leqslant \frac{m \times U_{max}}{m} \times sup(\mathbf{p},\mathbf{s}) = U_{max} \times sup(\mathbf{p},\mathbf{s})$. If pattern $\mathbf{p}$ is an HAOP, then $PAU(\mathbf{p},\mathbf{s}) \geqslant minau$, i.e. $minau \leqslant PAU(\mathbf{p},\mathbf{s}) \leqslant U_{max} \times sup(\mathbf{p},\mathbf{s})$. Thus, $sup(\mathbf{p},\mathbf{s}) \geqslant minau/U_{max} = lsup$. Hence, the support of an HAOP is no less than *lsup*. □

According to Theorem 3, if the support of a pattern is less than *lsup*, then the pattern is not an HAOP.

Theorem 4. *If the support of pattern* $\mathbf{p}$ *is less than lsup, then pattern* $\mathbf{p}$ *and its super patterns are not HAOPs.*

Proof. According to Theorem 3, pattern $\mathbf{p}$ is not an HAOP since $sup(\mathbf{p},\mathbf{s}) < lsup$. We will now show that the super patterns of pattern $\mathbf{p}$ are also not HAOPs. Suppose $\mathbf{q}$ is a super pattern of p. We know that $sup(\mathbf{q},\mathbf{s}) \leqslant sup(\mathbf{p},\mathbf{s})$. Furthermore, $sup(\mathbf{q},\mathbf{s}) < lsup$. According to Theorem 3, pattern $\mathbf{q}$ is also not an HAOP.

Example 6. In Example 5, $minau = 4.5$ and $U_{max} = 4$. According to Eq. 3, $lsup = \lceil 4.5/4 \rceil = 2$. We know that the support of $\mathbf{p_3}$ = C*G*A is 1, i.e. $sup(\mathbf{p_3},\mathbf{s}) = 1$. According to Theorem 3, $\mathbf{p_3}$ is not an HAOP since $sup(\mathbf{p_3},\mathbf{s}) = 1 < lsup = 2$.

Definition 8. Suppose we have pattern $\mathbf{p} = p_1{*}p_2{*}\ldots{*}p_r$, and items r and l. If $\mathbf{q} = \mathbf{p}{*}\mathrm{r} = p_1{*}p_2{*}\ldots{*}p_m{*}r, \mathbf{p}$ is called the prefix pattern of $\mathbf{q}$, denoted as $prefix(\mathbf{q}) = \mathbf{p}$. Similarly, if $\mathbf{r} = l{*}\mathbf{p} = \mathrm{l}{*}p_1{*}p_2{*}\ldots{*}p_m, \mathbf{p}$ is called the suffix pattern of $\mathbf{r}$, denoted as $suffix(\mathbf{r}) = \mathbf{p}$. Since $prefix(\mathbf{q}) = suffix(\mathbf{r}) = \mathbf{p}$, operator $\oplus$ is used to connect $\mathbf{q}$ and $\mathbf{p}$ to generate a super-pattern $\mathbf{t}$ with length $m + 2$, i.e. $\mathbf{t} = \mathbf{r} \oplus \mathbf{q} = l{*}\mathbf{p}{*}r$. This process is called pattern growth.

Example 7. Suppose we have pattern $\mathbf{p}$ = A*T, $\mathbf{r}$ = G*A*T and $\mathbf{q}$ = A*T*C. We know that $prefix(\mathbf{q}) = suffix(\mathbf{r}) = \mathbf{p}$. Hence, we can obtain a super pattern $\mathbf{t}$ of length four by pattern growth, i.e. $\mathbf{t} = \mathbf{r} \oplus \mathbf{q}$ = G*A*T*C.

Although the enumeration tree strategy can be applied to generate candidate patterns, the following example shows that the pattern growth strategy outperforms the enumeration tree strategy.

Example 8. Fig. 5 shows all candidate patterns generated by the enumeration tree strategy. In Example 5, the high lower bound patterns with length two are {A*A,A*C,A*G,C*G}. Hence, the enumeration tree strategy generates $4 \times 3 = 12$ candidate patterns. The reason is that it adds each letter in Σ at the end of each pattern to generate |Σ| candidate patterns. For example, based on pattern A*A, it generates three candidate patterns, A*A*A, A*A*C and A*A*G since Σ = {A,C,G}. However, the pattern growth strategy generates four candidate patterns. i.e. {A*A*A, A*A*C, A*A*G, A*C*G}. Hence, this example illustrates that the pattern growth strategy outperforms the enumeration tree strategy.

4.5. HAOP-Miner Algorithm

In this subsection, we propose the HAOP-Miner algorithm and analyse its time and space complexities. The steps of HAOP-Miner are as follows.

Step 1: Scan the sequence, and obtain the high lower bound pattern set H_1 and HAOP set *Hau* with length 1.

Step 2: Generate the candidate pattern set F_{m+1} by set H_m with length m.

Step 3: Calculate the support of pattern $\mathbf{p}$ in set F_{m+1}.

Step 4: If pattern $\mathbf{p}$ is a high lower bound pattern, store it in high lower bound pattern set H_m If pattern $\mathbf{p}$ is an HAOP, store it in the HAOP set *Hau*.

Step 5: Repeat Steps 2 to 4 until the candidate pattern set F_{m+1} or high lower bound pattern set H_m is empty. The patterns in the HAOP set *Hau* are HAOPs.

Example 9. We use Example 5 to illustrate the principle of HAOP-Miner.

According to Fig. 5, we know that pattern "A" is a high lower bound pattern, not an HAOP, due to $sup(\text{"A"},\mathbf{s}) = 4 \geqslant 2$ and $PAU(\mathrm{A},\mathbf{s}) = 1 \times 4/1 = 4 < 4.5$. Similarly, patterns "C" and "G" are both high lower bound patterns, and pattern "C" is an HAOP, while pattern "G" is not an HAOP. Hence, the set of H_1 is {A, C, G}, and the set of *Hau* with length one is "C". We generate the candidate pattern set F_2 using the high lower bound pattern set H_1, which is {"A*A", "A*C", "A*G", "C*A", "C*C", "C*G", "G*G", "G*A", "G*C", "G*G"}. Similarly, H_2 is {"A*A", "A*C", "A*G", "C*G"}, and HAOPs with length two is {"A*C", "C*G"}. Iterate the above process, and we generate F_3 using H_2, which is {"A*A*A", "A*A*C", "A*A*G", "A*C*G"}. H_3 is {"A*A*C", "A*A*G", "A*C*G"}, and HAOP with length three is {"A*C*G"}. H_4 is {"A*A*C*G"}. Since H_4 cannot generate the candidate pattern set F_5, HAOP mining is finished.

HAOP-Miner is given in Algorithm 3.

Algorithm 3. HAOP-Miner: Mine all HAOPs

Input: Sequence $\mathbf{s}$, *minau* and the utilities.
Output: *Hau*
1: Scan sequence s, calculate the support of each event item, store the high lower bound patterns with length 1 into F_1, and store HAOPs into *Hau*;

(*continued on next page*)

(*continued*)

Algorithm 3. HAOP-Miner: Mine all HAOPs

```
2: m←1;
3: F_{m+1}←PatternGrowth (H_m) ;
4: While F_{m+1} ≠ null do
5:   for each p in F_{m+1} do
6:     support←Rf (p,s);
7:     if support≥lsup then
8:       F_{m+1}←F_{m+1} ∪ p
9:       if PAU(p,s)≥minau then
10:          Hau←Hau ∪ p
11:      end if
12:    end if
13:  end for
14:  F_{m+2}←PatternGrowth (H_{m+1});
15:  m←m + 1;
16: end while
17: return Hau;
```

Theorem 5. *The time complexity of the HAOP-Miner algorithm is* $O(n \times m \times L)$*, where* n, m *and* L *are the length of the sequence, the maximum length of an HAOP and the number of the candidate patterns, respectively.*

Proof. According to Theorem 1, the time complexity of the Rf algorithm is $O(n \times m)$. Thus, for all candidate patterns, the time complexity of HAOP-Miner is $O(n \times m \times L)$. Since a binary search is used in pattern growth to generate candidate patterns, the time complexity of the generation of all candidate patterns is $O(L \times \log(L))$. Hence, the time complexity of the HAOP-Miner algorithm is $O(n \times m \times L + L \times \log(L)) = O(n \times m \times L)$. □

Theorem 6. *The space complexity of the HAOP-Miner algorithm is* $O(m \times L + n)$.

Proof. The space of the HAOP-Miner algorithm is mainly composed of four elements: the candidate patterns, high lower bound patterns, HAOPs and the space of Rf. It is easy to see that the space complexities of the candidate patterns, high lower bound patterns and HAOPs are $O(m \times L)$. Meanwhile, according to Theorem 2, the space complexity of the Rf algorithm is $O(n)$. Hence, the space complexity of the HAOP-Miner algorithm is $O(m \times L + n)$. □

5. Experimental results and analysis

We introduce the benchmark datasets in Section 5.1. To evaluate the performance of our approach, we also propose some competitive algorithms whose principles are introduced in Section 5.2. Section 5.3 shows the efficiency of HAOP-Miner. Section 5.4 validate the running performance of HAOP-Miner. Mining performance is evaluated in Section 5.5. Section 5.6 furtherly reports the application in the BABYSALE dataset.

5.1. Benchmark datasets

The experimental running environment was an Intel Core (TM) i5-5200M, 2.50 GHZ CPU, 8 GB RAM, Windows 7, and a 64-bit operating system computer. The program development environment was VC++6.0. To verify the performance of the HAOP-Miner algorithm, we used DNA and virus sequences as experimental data. These datasets are summarised in Table 5.

Table 5
Summary of benchmark datasets.

Dataset	Type	Source	Length
AX829174[a]	DNA	Homo sapiens (human)	10,011
DNA1[b]	DNA	Homo sapiens AL158070	6000
DNA2	DNA	Homo sapiens AL158070	8000
DNA3	DNA	Homo sapiens AL158070	10,000
Norwalk[c]	Virus	GenBankAF093797.1	7601
Potato_virus[d]	Virus	Potato virus Y Wilga MV99	9699
BABYSALE[e]	Sales	Baby trade sales	29,971

[a] AX829174 was used in Wu, Xie et al. (2013), and can be downloaded from https://www.ncbi.nlm.nih.gov/nuccore/AX829174.
[b] DNA1-3 databases were analysed in Reference Zhang et al. (2007), and can be downloaded from https://www.ncbi.nlm.nih.gov/nuccore/AL158070.11.
[c] Norwalk database was downloaded from https://www.ncbi.nlm.nih.gov/nuccore/AF093797.1.
[d] Potato_virus database was downloaded from https://www.ebi.ac.uk/ena/browser/view/Taxon:1107954.
[e] BABYSALE database is downloaded from https://tianchi.aliyun.com/dataset/dataDetail?dataId=45.

5.2. Baseline methods

We will assess the efficiency of HAOP-Miner in Section 5.3. HAOP-Miner has two key parts: support calculation and candidate pattern generation. We verify the efficiency of these two parts. Therefore, we propose the following competitive algorithms whose principles are introduced as follows.

(1) Sow-H and HAOP-Pf: To verify the efficiency of the Rf algorithm, the Scan_oneway algorithm (Wu, Zhu, He et al., 2013) and the proposed Pf algorithm are selected as competitive strategies. Sow-H and HAOP-Pf use Scan_oneway and Pf to calculate the support, respectively, and employe pattern growth strategy to generate candidate patterns.
(2) HAOP-Bf and HAOP-Df: To validate the efficiency of the pattern growth strategy, HAOP-Bf and HAOP-Df are proposed which generate candidate patterns using breadth first and depth first strategies, respectively. The support calculation methods in these two algorithms are both Rf.

To further validate the running performance, Section 5.4 selects three state-of-the-art algorithms: i-OFMI (Wu, Xie et al., 2013), SPMW (Xie et al., 2017) and PMBC (Wu, Zhu, He et al., 2013).

(3) i-OFMI(Wu, Xie et al., 2013) and SPMW(Xie et al., 2017): i-OFMI and SPMW were proposed to mine frequent patterns with gap constraints under the one-off condition.
(4) PMBC (Wu, Zhu, He et al., 2013): PMBC is a state-of-the-art algorithm to mine frequent patterns with a self-adaptive gap under the one-off condition.

We will evaluate the mining performance of HAOP-Miner in Section 5.5. To analyse the effect of the gap constraint, and show the difference between frequent patterns under the one-off condition, frequent patterns under the nonoverlapping condition and HAOPs, we propose HAOP-Nogap and select two state-of-the-art algorithms: PMBC (Wu, Zhu, He et al., 2013) and NOSEP (Wu et al., 2018) as competitive algorithms. The principles of HAOP-Nogap and NOSEP are introduced as follows.

(5) HAOP-Nogap: HAOP-Nogap is proposed to mine HAOPs without a gap. Other characteristics of HAOP-Nogap are the same as those of HAOP-Miner.
(6) NOSEP (Wu et al., 2018): NOSEP was designed to mine frequent patterns with gap constraints under the nonoverlapping condition.

5.3. Efficiency

We know that HAOP-Miner has two key steps: support calculation and candidate pattern generation. The Rf algorithm and pattern growth strategy are used to calculate support and generate candidate patterns, respectively. In this subsection, we verify the efficiency of the Rf algorithm and pattern growth strategy. Four competitive algorithms are selected: Sow-H, HAOP-Pf, HAOP-Bf and HAOP-Df. We use six datasets to carry out the experiments: DNA1, DNA2, DNA3, Norwalk, Potato_-

virus and AX829174. The mining parameters are $U(A) = 2, U(C) = 2, U(G) = 3, U(T) = 3$ and *minau* = 3600. The mining results are shown from Figs. 6–8.

It can be seen from Figs. 6–8 that HAOP-Miner achieves better running performance than the other competitive algorithms. The five alternative algorithms find the same number of HAOPs shown in Fig. 6, while HAOP-Miner is faster than the other algorithms shown in Fig. 7. For example, all five algorithms discover 208 HAOPs on AX829174, and the running time of HAOP-Miner is 8420 ms, which is faster than the other four algorithms. The reasons for this are as follows.

1. The Rf algorithm is an efficient method of calculating the support. As can be seen from Fig. 8, although Sow-H and HAOP-Pf give the same number of candidate patterns as HAOP-Miner, HAOP-Miner is faster than Sow-H and HAOP-Pf. For example, the running time for Sow-H and HAOP-Pf is 89210 ms and 306450 ms, respectively. The other experiments give the same results, thus verifying that the Rf algorithm is more efficient than the Scan_oneway (Wu, Zhu, He et al., 2013) and Pf algorithms.
2. Pattern growth strategy is an efficient method of reducing the number of candidate patterns. From Fig. 7, it can be seen that HAOP-Miner runs faster than HAOP-Bf and HAOP-Df. For example, HAOP-Bf and HAOP-Df take 19490 ms and 18110 ms, respectively. As can be seen from Fig. 8, the number of candidate patterns produce by HAOP-Miner is 974, while HAOP-Bf and HAOP-Df both produce 2092. HAOP-Miner adopts a pattern growth strategy to generate the candidate patterns, while HAOP-Bf and HAOP-Df adopt depth-first and breadth-first strategies, respectively. Therefore, the pattern growth strategy is better than the other two strategies, which is consistent with the analysis of Example 8. Hence, HAOP-Miner outperforms HAOP-Bf and HAOP-Df.

In short, HAOP-Miner employs the Rf algorithm and pattern growth strategy to calculate support and generate candidate patterns, respectively, which are more efficient than other competitive methods.

5.4. Running performance

To further validate the running performance, i-OFMI, SPMW and PMBC are selected as the competitive algorithms to mine frequent patterns. We know that when $U(A) = 1, U(C) = 1, U(G) = 1$ and $U(T) = 1$, the HAOPs mined by HAOP-Miner are frequent patterns. Gap constraints of [0, 5], [0, 15] and [0, 25] are selected for i-OFMI and SPMW. We also selecte the six datasets used in Section 5.3 to carry out the experiments: DNA1, DNA2, DNA3, Norwalk, Potato_virus and AX829174, and *minsup* = *minau* = 1300. The comparisons of number of mined patterns and average running time for per pattern are shown in Tables 6 and 7, respectively.

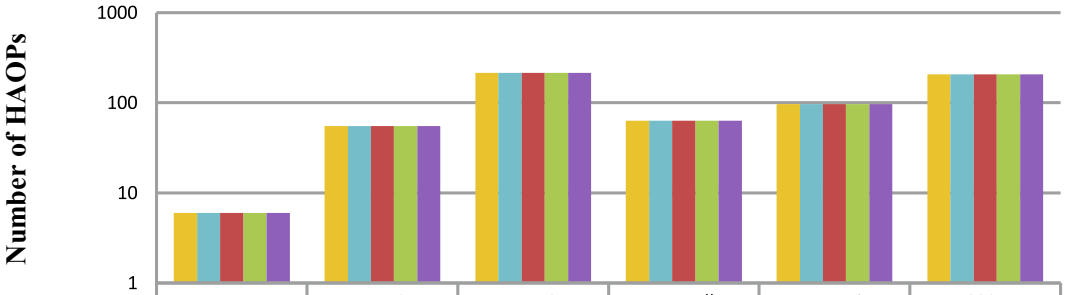
The results indicate the following observations.

1. HAOP-Miner not only runs faster than i-OFMI and SPMW, but also mines more patterns than i-OFMI and SPMW. For example, on DNA3, i-OFMI and SPMW discover 175 and 163 patterns with [0,25], while HAOP-Miner discovers 522 patterns which is more than i-OFMI and SPMW. More importantly, i-OFMI and SPMW cost 3505 ms and 94 ms, while HAOP-Miner only costs 9 ms for per pattern. All of the other experiments give the similar phenomena, showing that if the gap setting is unreasonable, some patterns will be missed and HAOP-Miner runs faster than the gapped method.
2. HAOP-Miner discovers the same number of patterns as PMBC, but runs faster than PMBC. For example on DNA2, HAOP-Miner and PMBC discover 64 patterns, while HAOP-Miner cost only 6 ms, which is faster than PMBC. All of the other experiments have the same phenomena. The reason is as follows. Both HAOP-Miner and PMBC discover patterns with a self-adaptive gap. Therefore, the two algorithms mine the same number of patterns. HAOP-Miner employs Rf to calculate the support which is more efficient than that of PMBC. Hence, HAOP-Miner runs faster than PMBC.

In short, HAOP-Miner outperforms all state-of-the-art algorithms.

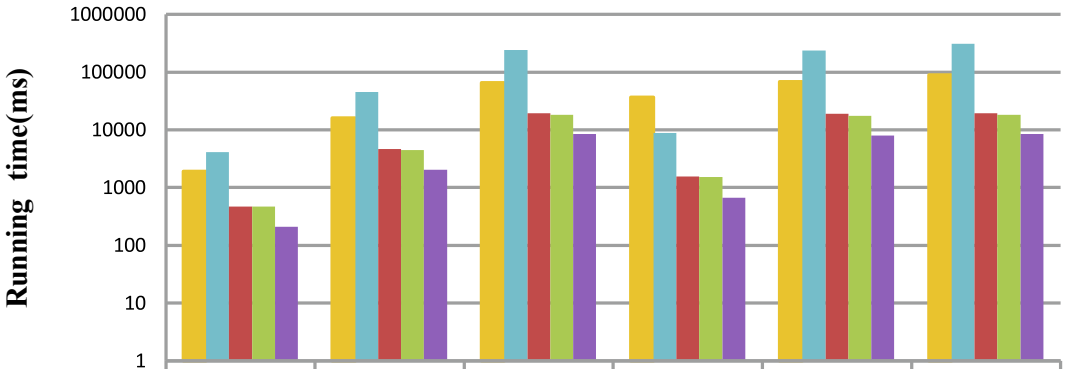
5.5. Mining performance

In this subsection, we report the mining performance of HAOP-Miner. Section 5.4 shows that i-OFMI and SPMW discovers less patterns than HAOP-Miner. To further validate the mining performance of HAOP-Miner, we select HAOP-Nogap and PMBC as competitive algorithms. HAOP-Nogap mines HAOPs without a gap. PMBC mines frequent patterns under the one-off condition. Six sequences are chosen: DNA1, DNA2, DNA3, Norwalk, Potato_virus and AX829174. Parameters $U(A) = 2, U(C) = 2, U(G) = 3, U(T) = 3$ and *minau* = 3900 are selected for HAOP-Nogap and HAOP-Miner to mine HAOPs. For fairness concerning, parameter *minsup* = 1300 (3900/3) is selected for PMBC to mine frequent patterns since the maximum utility is 3, and [0,6] is selected for NOSEP which can mine approximately similar number of patterns. The comparisons of the number of the mined patterns and candidate patterns are shown in Fig. 9 and Fig. 10, respectively. For clarification, Fig. 11 shows the comparison of average utility patterns on DNA2.



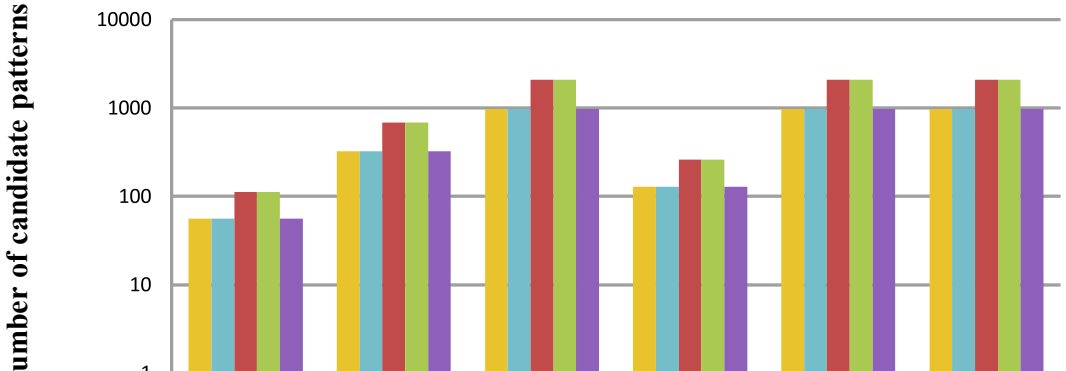
	DNA1	DNA2	DNA3	Norwalk	Potato_virus	AX829174
Sow-H	6	55	214	63	97	208
HAOP-Pf	6	55	214	63	97	208
HAOP-Bf	6	55	214	63	97	208
HAOP-Df	6	55	214	63	97	208
HAOP-Miner	6	55	214	63	97	208

Fig. 6. Comparison of number of HAOPs.



	DNA1	DNA2	DNA3	Norwalk	Potato_virus	AX829174
Sow-H	1910	15930	65550	36810	67640	89210
HAOP-Pf	4120	45220	240690	8890	235950	306450
HAOP-Bf	470	4670	19470	1570	18920	19490
HAOP-Df	470	4430	18290	1510	17490	18110
HAOP-Miner	210	2020	8510	670	8010	8420

Fig. 7. Comparison of running time (ms).



	DNA1	DNA2	DNA3	Norwalk	Potato_virus	AX829174
Sow-H	56	324	974	128	974	974
HAOP-Pf	56	324	974	128	974	974
HAOP-Bf	112	684	2092	260	2092	2092
HAOP-Df	112	684	2092	260	2092	2092
HAOP-Miner	56	324	974	128	974	974

Fig. 8. Comparison of number of candidate patterns.

Table 6
Comparison of number of mined patterns.

	Gap	DNA1	DNA2	DNA3	Norwalk	Potato_virus	AX829174
i-OFMI	[0,5]	2	6	15	4	18	16
	[0,15]	4	25	92	34	101	77
	[0,25]	4	37	175	64	170	126
SPMW	[0,5]	2	6	15	4	18	16
	[0,15]	4	25	92	34	92	75
	[0,25]	4	37	163	64	163	126
PMBC		4	64	522	64	170	522
HAOP-Miner		**4**	**64**	**522**	**64**	**170**	**522**

Table 7
Comparison of average running time for per pattern (ms)

	Gap	DNA1	DNA2	DNA3	Norwalk	Potato_virus	AX829174
i-OFMI	[0,5]	23	39	59	51	52	61
	[0,15]	28	95	278	100	290	213
	[0,25]	47	269	3505	660	4289	1999
SPMW	[0,5]	27	43	82	47	76	86
	[0,15]	23	49	88	51	75	77
	[0,25]	31	65	94	70	80	106
PMBC		24	97	133	58	140	178
HAOP-Miner		**11**	**6**	**9**	**6**	**8**	**9**

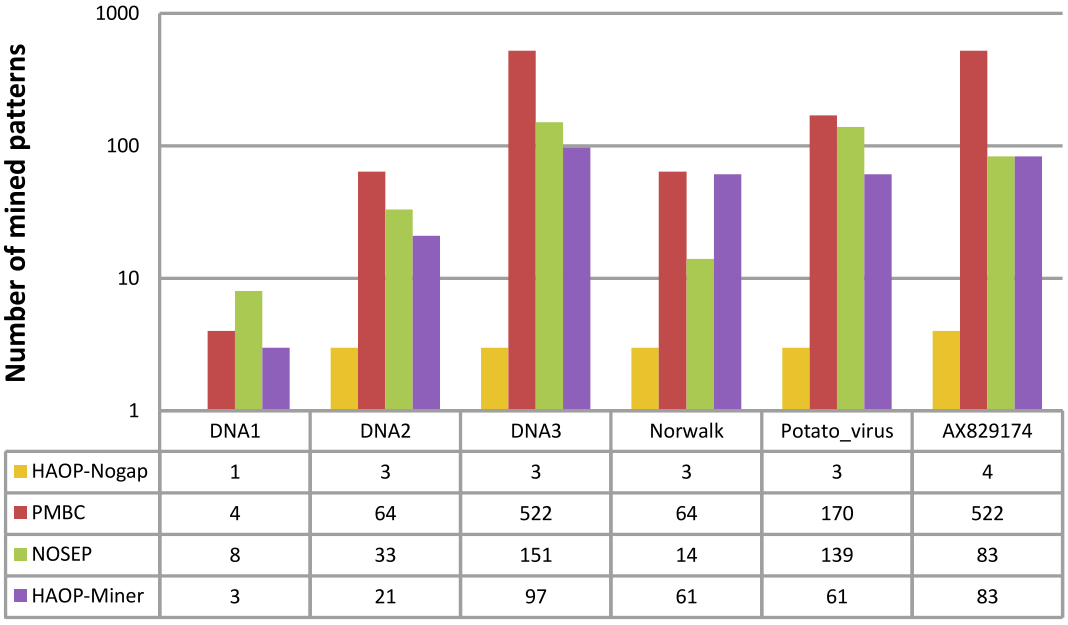


Fig. 9. Comparison of number of patterns (HAOP-Nogap and HAOP-Miner mine HAOPs, while PMBC and NOSEP mine frequent patterns).

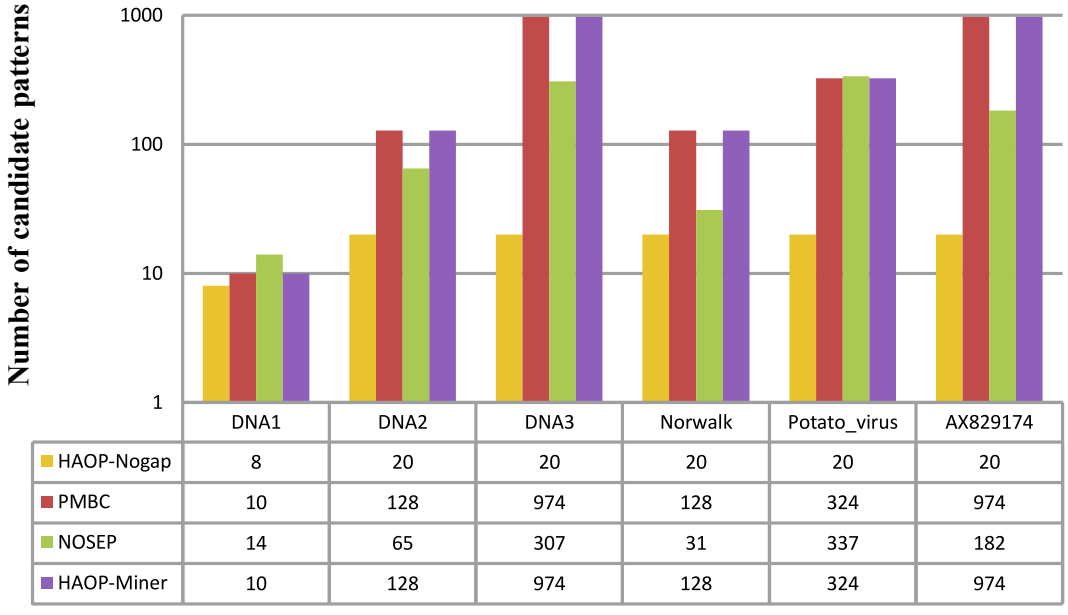


Fig. 10. Comparison of number of candidate patterns.

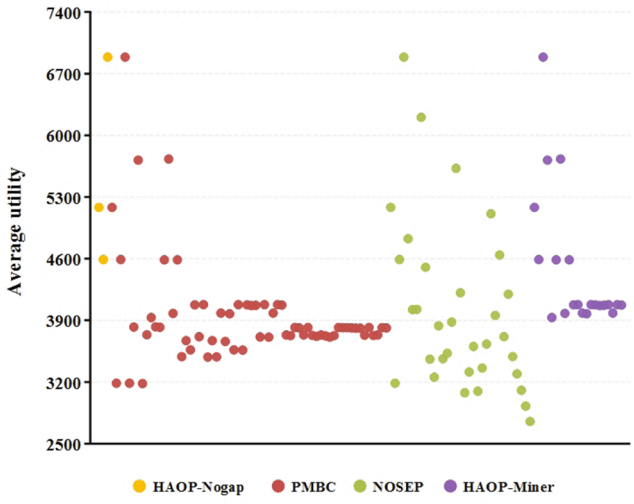


Fig. 11. Comparison of average utility of patterns on DNA2.

Fig. 12. Wordcloud map of mining results.

The results report the following observations.

1. The gap constraint can effectively improve the mining ability of the algorithm. From Fig. 9, HAOP-Nogap and HAOP-Miner discover 3 and 21 HAOPs on DNA2, respectively. From Fig. 11, although the three discovered patterns by HAOP-Nogap are HAOPs, HAOP-Nogap mines less HAOPs than HAOP-Miner. This phenomenon can be seen on all datasets. The reason for this is that HAOP-Miner introduces gap constraints to make mining more flexible. Hence, HAOP-Miner discovers more HAOPs than HAOP-Nogap.
2. The mining ability of HAOP-Miner is superior to those of PMBC and NOSEP. For example, From Fig. 10, we know that the number of candidate patterns of PMBC and HAOP-Miner are both 128 on DNA2, while from Fig. 9, PMBC, NOSEP and HAOP-Miner discover 64, 33 and 21 patterns, respectively. According to Fig. 11, most discovered patterns by PMBC and NOSEP are not high-average utility patterns since their average utilities are less than 3900. However, all mined patterns by HAOP-Miner are HAOPs. Hence, HAOP-Miner is easier to mine valuable patterns than PMBC and NOSEP.

In summary, HAOP-Miner has better mining ability than other competitive algorithms.

5.6. Case study

We conduct experiments on the BABYSALE dataset to explore how to set a reasonable marketing strategy to obtain maximum profits. The BABYSALE dataset is a sales dataset for infant products. The dataset is a large-scale dataset since it contains historical trade information of Taobao members. Each trade belongs to a root category. To simplify, we symbolize the root category and convert the daily trade information into a sequence. In this dataset, there are six kinds of products according to its root category. It is impossible for the merchant to recommend all products to customers. To obtain higher profits, the merchant could recommend the products according to the mining results. Suppose the profit of each product is $U(\mathrm{A}) = 1.5, U(\mathrm{B}) = 2, U(\mathrm{C}) = 2$, $U(\mathrm{D}) = 3$, $U(\mathrm{E}) = 5, U(\mathrm{F}) = 4$ and $minau = 3000$. The Wordcloud map (Heimerl, Lohmann, Lange, & Ertl, 2014) is employed to show the mining results which are shown in Fig. 12.

In Fig. 12, the larger the fontsize of the pattern is, the higher the profit brings to the merchant. According to Fig. 12, we have the following observations. Although products "E" and "F" have higher single profit, the results show that products "C" and "D" can achieve higher profits. The reason is that products "E" and "F" are not hot sellers, while products "C" and "D" are. Suppose customers purchase products "CA". To obtain higher profits, the merchant can recommend the products "C" or "D" according to Fig. 12 since patterns "CAC" and "CAD" are larger font than other patterns.

6. Conclusion

In order to consider the utility of items and to solve the problem of setting gaps without prior knowledge, this paper explores self-adaptive HAOP mining which can discover some extremely important but low-frequency patterns. Self-adaptive HAOP mining as a kind of repetitive SPM (or sequence pattern mining), considers the support (number of occurrences), utility and the pattern length simultaneously. This paper proposes an efficient algorithm, named HAOP-Miner which has two key steps: support calculation and candidate pattern generation. At the support calculation stage, this paper explores a Reverse filling strategy to calculate the support. The Reverse filling strategy can efficiently improve the efficiency since it avoids creating redundant nodes and does not need to prune the redundant and useless nodes after finding an occurrence. At the candidate pattern generation stage, since HAOP mining does not satisfy the Apriori property, HAOP-Miner employs a support lower bound method combined with pattern growth strategy that can prune the candidate patterns efficiently. Experimental results validate the efficiency of HAOP-Miner and show that HAOP-Miner has better running and mining performances than other competitive algorithms.

There are some limitations for HAOP-Miner. The HAOP mining does not satisfy the Apriori property. To tackle this issue, this paper adopts a support lower bound method combined with a pattern growth strategy to reduce the candidate patterns. However, if the utility of an item is very large, the lower bound support will be very small. Thus, many patterns could be high lower bound patterns, which means that many patterns can be candidate patterns. Hence, a more efficient pruning strategy should be explored in this case. Moreover, HAOP-Miner can only be applied to character datasets. If the sequence is other type, such as time series, it must be converted to character sequences first. How to mine HAOPs on time series is a problem worthy for further study.

CRediT authorship contribution statement

Youxi Wu: Conceptualization, Methodology, Formal analysis, Supervision, Funding acquisition. **Rong Lei:** Software, Writing - original draft, Validation, Investigation, Data curation. **Yan Li:** Investigation,

Writing - review & editing. **Lei Guo:** Validation, Resources. **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.

Acknowledgements

This work was partly supported by National Natural Science Foundation of China (61976240, 52077056, 917446209), National Key Research and Development Program of China (2016YFB 1000901), and Natural Science Foundation of Hebei Province, China (Nos. F2020202013, E2020202033).

References

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

Chen, X., Xiao, R., Xin, D., Lin, X. & Lin, L. (2018). Constructing a novel spark-based distributed maximum frequent sequence pattern mining for IoT log. In The 8th International Conference on Communication and work Security (pp. 112–116). ACM.

Chen, X., Rao, Y., Xie, H., Wang, F. L., Zhao, Y., & Yin, J. (2019). Sentiment classification using negative and intensive sentiment supplement information. *Data Science and Engineering, 4*, 109–118.

Choi, H.-J., & Park, C. (2019). Emerging topic detection in twitter stream based on high utility pattern mining. *Expert Systems with Applications, 115*, 27–36.

Dalmas, B., Fournier-Viger, P., & Norre, S. (2017). TWINCLE: A constrained sequential rule mining algorithm for event logs. *Procedia Computer Science, 112*, 205–214.

Ding, B., Lo, D., Han, J., & Khoo, S. (2009). Efficient mining of closed repetitive gapped subsequences from a sequence database. In *IEEE 25th International Conference on Data Engineering* (pp. 1024–1035). IEEE.

Dong, X., Qiu, P., L, J. Cao, L. & Xu, T. (2019). Mining top-k useful negative sequential patterns via learning. IEEE Transactions on Neural Networks and Learning Systems, 30, 2764–2778.

Dong, X., Gong, Y., & Cao, L. (2020). e-RNSP: An efficient method for mining repetition negative sequential patterns. *IEEE Transactions on Cybernetics, 50*, 2084–2096.

Fournier-Viger, P., Li, Z., Lin, J. C.-W., Kiran, R., & Fujita, H. (2019). Efficient algorithms to identify periodic patterns in multiple sequences. *Information Sciences, 489*, 205–226.

Fournier-Viger, P., Li, J., Lin, J. C.-W., Truong, T., & Rage, U. K. (2020). Mining cost-effective patterns in event logs. *Knowledge-Based Systems, 191*.

Gan, W., Lin, J. C.-W., Fournier-Viger, P., Chao, H.-C., & Yu, P. S. (2020). HUOPM: High-utility occupancy pattern mining. *IEEE Transactions on Cybernetics, 50*, 1195–1208.

Gan, W., Lin, J. C.-W., Zhang, J., & Yu, P. S. (2020). Utility mining across multi-sequences with individualized thresholds. ACM/IMS Transactions on Data. *Science, 1, 2*, 18-es.

Ghosh, S., Li, J., Cao, L., & Ramamohanarao, K. (2017). Septic shock prediction for ICU patients via coupled HMM walking on sequential contrast patterns. *Journal of Biomedical Informatics, 66*, 19–31.

Heimerl, F., Lohmann, S., Lange, S., & Ertl, T. (2014). Word cloud explorer: Text analytics based on word clouds. In *Hawaii International Conference on System Sciences* (pp. 1833–1842). IEEE.

He, Z., Zhang, S., & Wu, J. (2019). Significance-based discriminative sequential pattern mining. *Expert Systems with Applications, 122*, 54–64.

Huang, J., Jaysawal, B., Chen, K., & Wu, Y. (2019). Mining frequent and top-K high utility time interval-based events with duration patterns. *Knowledge and Information Systems, 61*, 1331–1359.

Huang, Y., Wu, X., Hu, X., Xie, F., Gao, J., & Wu, G. (2009). Mining frequent patterns with gaps and one-off condition. In *International Conference on Computational Science and Engineering* (pp. 180–186). IEEE.

Irfan, Y., & Mete, C. (2019). An efficient tree-based algorithm for mining high average-utility itemset. *IEEE Access, 7*, 144245–144263.

Jiang, H., Chen, X., He, T., Chen, Z., & Li, X. (2018). Fuzzy clustering of crowdsourced test reports for apps. *ACM Transactions on Internet Technology (TOIT), 18*, 1–28.

Jiang, H., Li, X., Ren, Z., Xuan, J., & Jin, Z. (2019). Toward better summarizing bug reports with crowdsourcing eliciteWd attribute. *IEEE Transactions on Reliability, 68*, 2–22.

Karim, M. R., Cochez, M., Beyan, O. D., Ahmed, C. F., & Decker, S. (2018). Mining maximal frequent patterns in transactional databases and dynamic data streams: a spark-based approach. *Information Sciences, 432*, 278–300.

Kim, H., Yun, U., Baek, Y., Kim, J., Vo, B., Yoon, E., & Fujita, H. (2021). Efficient list based mining of high average utility patterns with maximum average pruning strategies. *Information Sciences, 543*, 85–105.

Lan, G., Hong, T., & Tseng, V. S. (2012). Efficiently mining high average-utility item sets with an improved upper-bound strategy. *Journal of Information Technology and Decision Making, 11*, 1009–1030.

Le, B., Tran, M., & Vo, B. (2015). Mining frequent closed inter-sequence patterns efficiently using dynamic bit vectors. *Expert Systems with Applications, 43*, 74–84.

Lin, C., Ren, S., Fournier-Viger, P. & Hong, Tzung, P. (2017). EHAUPM: Efficient high average-utility pattern mining with tighter upper-bound model. IEEE Access, 5, 12927–12940.

Lin, J., Zhang, J. & Fournier-Viger, P (2017). High-utility sequential pattern mining with multiple minimum utility thresholds. In Asia-Pacific Web (APWeb) and Web-Age Information Management (WAIM) Joint Conference on Web and Big Data.

Lin, J.C.- W., Li, T., Pirouz, M., Zhang, J. & Fournier-Viger, P. (2020). High average-utility sequential pattern mining based on uncertain databases. Knowledge and Information Systems, 62, 1199–1228.

Lin, C., Li, T., Fournier-Viger, P., Zhang, J., & Guo, X. (2019). Mining of high average-utility patterns with item-level thresholds. *Journal of Inter Technology, 20*, 187–194.

Liu, J., Chang, Z., Leung, C., Wong, R., Xu, Y., & Zhao, R. (2019). Efficient mining of extraordinary patterns by pruning and predicting. *Expert Systems with Applications, 125*, 55–68.

Liu, H., Liu, Z., Huang, H., & Wu, X. (2018). Sequential pattern matching with general gaps and one-off condition. *Journal of Software, 2*, 363–382.

Liu, H., Wang, L., Liu, Z., Zhao, P., & Wu, X. (2018). Efficient pattern matching with periodical wildcards in uncertain sequences. *Intelligent Data Analysis, 22*, 829–842.

Li, C., Yang, Q., Wang, J., & Li, M. (2012). Efficient mining of gap-constrained subsequences and its various applications. *ACM Transactions on Knowledge Discovery from Data, 6*, 1–39.

Lu, T., Vo, B., Nguyen, H. T., & Hong, T. P. (2014). A new method for mining high average utility itemsets. In *IFIP TC8 Computer Information Systems and Industrial Management* (pp. 33–42). Springer.

Miao, S., Vespier, U., Cachucho, R., Meeng, M., & Knobbe, A. (2016). Predefined pattern detection in large time series. *Information Sciences, 329*, 950–964.

Min, F., Zhang, Z., Zhai, W., & Shen, R. (2020). Frequent pattern discovery with tri-partition alphabets. *Information Sciences, 507*, 715–732.

Morteza, Z., Heidar, D., & Aijun, A. (2017). Mining significant high utility generegulation sequential patterns. *Bmc Systems Biology, 11*, 109.

Nam, H., Yun, U., Yoon, E., & Lin, J. C.-W. (2020). Efficient approach for incremental weighted erasable pattern mining with list structure. *Expert Systems with Applications, 143*.

Qu, J. F., Liu, M., & Fournier-Viger, P. (2019). Efficient algorithms for high utility itemset mining without candidate generation. *Algorithms and Applications*, 131–160.

Shi, Q., Shan, J., Yan, W., Wu, Y., & Wu, X. (2020). Nonoverlapping pattern matching with general gap constraints. *Applied Intelligence, 50*, 1832–1845.

Song, W., Liu, Y., & Li, J. (2014). Mining high utility itemsets by dynamically pruning the tree structure. *Applied Intelligence, 40*, 29–43.

Sumalatha, S., & Subramanyam, R. (2020). Distributed mining of high utility time interval sequential patterns using mapreduce approach. *Expert Systems with Applications, 141*, Article 112967 .

Wang, H., Duan, L., Zuo, J., Wang, W., Li, Z., & Tang, C. (2016). Efficient mining of distinguishing sequential patterns without a predefined gap constraint. *Chinese Journal of Computer, 39*, 1979–1991.

Wang, Y., Hou, W., & Wang, F. (2018). Mining co-occurrence and sequence patterns from cancer diagnoses in New York State. *PLoS ONE, 13*.

Wang, J., Huang, J., & Chen, Y. (2016). On efficiently mining high utility sequential patterns. *Knowledge and Information Systems, 49*, 597–627.

Warmuth, M. K., & Haussler, D. (1984). On the complexity of iterated shuffle. *Journal of Computer and System Sciences, 28*, 345–358.

Wei, L., Xing, P., Shi, G., Ji, Z., & Zou, Q. (2017). Fast prediction of protein methylation sites using a sequence-based feature selection technique. *IEEE/ACM Transactions on Computational Biology and Bioinformatics, 16*, 1264–1273.

Wu, Y., Fan, J., Li, Y., Guo, L., & Wu, X. (2020). Approximate pattern matching with length constraints. *Applied Intelligence, 50*, 4094–4116.

Wu, Y., Shen, C., Jiang, H., & Wu, X. (2017). Strict pattern matching under non-overlapping condition. *Science China Information Sciences, 60*, 1–16.

Wu, Y., Tong, Y., Zhu, X., & Wu, X. (2018). NOSEP: Nonoverlapping sequence pattern mining with gap constraints. *IEEE Transactions on Cybernetics, 48*, 2809–2822.

Wu, Y., Wang, Y., Li, Y., Zhu, X., & Wu, X. (2021). Top-k self-adaptive contrast sequential pattern mining. *IEEE Transactions on Cybernetics*. https://doi.org/10.1109/TCYB.2021.3082114

Wu, Y., Wang, L., Ren, J., Ding, W., & Wu, X. (2014). Mining sequential patterns with periodic wildcard gaps. *Applied Intelligence, 41*, 99–116.

Wu, M., & Wu, X. (2019). On big wisdom. *Knowledge and Information Systems, 58*, 1–8.

Wu, X., Xie, F., Huang, Y., Hu, X., & Gao, J. (2013). Mining sequential patterns with wildcards and the One-Off condition. *Journal of Software, 24*, 1804–1815.

Wu, X., Zhu, X., He, Y., & Arslan, A. N. (2013). PMBC: Pattern mining from biological sequences with wildcard constraints. *Computers in Biology and Medicine, 43*, 481–492.

Wu, Y., Zhu, C., Li, Y., Guo, L., & Wu, X. (2020). NetNCSP: Nonoverlapping closed sequential pattern mining. *Knowledge-Based Systems*. https://doi.org/10.1016/j.knosys.2020.105812

Wu, X., Zhu, X., Wu, G.-Q., & Ding, W. (2013). Data mining with big data. *IEEE Transactions on Knowledge and Data Engineering, 26*, 97–107.

Xie, F., Wu, X., & Zhu, X. (2017). Efficient sequential pattern mining with wildcards for keyphrase extraction. *Knowledge-Based Systems, 115*, 27–39.

Yin, J., Zheng, Z. & Cao, L. (2012). USpan: An efficient algorithm for mining high utility sequential patterns. In ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (pp. 660–668). ACM.

Yun, U., Kim, D., Yoon, E., & Fujita, H. (2018). Damped window based high average utility pattern mining over data streams. *Knowledge-Based Systems, 144*, 188–205.
Yun, U., Ryang, H., Lee, G., & Fujita, H. (2017). An efficient algorithm for mining high utility patterns from incremental databases with one database scan. *Knowledge-Based Systems, 124*, 188–206.
Zhang, M., Kao, B., Cheung, D., & Yip, K. Y. (2007). Mining periodic patterns with gap requirement from sequences. *ACM Transactions on Knowledge Discovery from Data (TKDD), 1*, 7-es.