OWSP-Miner: Self-adaptive One-off Weak-gap Strong Pattern Mining

YOUXI WU, School of Artificial Intelligence, Hebei University of Technology, China and Hebei Key Laboratory of Big Data Computing, China
XIAOHUI WANG, School of Artificial Intelligence, Hebei University of Technology, China
YAN LI, School of Economics and Management, Hebei University of Technology, China
LEI GUO, State Key Laboratory of Reliability and Intelligence of Electrical Equipment, Hebei University of Technology, China
ZHAO LI, Alibaba Group, China
JI ZHANG, School of Sciences, The University of Southern Queensland, Australia
XINDONG WU, Key Laboratory of Knowledge Engineering with Big Data (Hefei University of Technology), Ministry of Education, China and Mininglamp Technology, China

Gap constraint sequential pattern mining (SPM), as a kind of repetitive SPM, can avoid mining too many useless patterns. However, this method is difficult for users to set a suitable gap without prior knowledge and each character is considered to have the same effects. To tackle these issues, this article addresses a self-adaptive One-off Weak-gap Strong Pattern (OWSP) mining, which has three characteristics. First, it determines the gap constraint adaptively according to the sequence. Second, all characters are divided into two groups: strong and weak characters, and the pattern is composed of strong characters, while weak characters are allowed in the gaps. Third, each character can be used at most once in the process of support (the frequency of pattern) calculation. To handle this problem, this article presents OWSP-Miner, which equips with two key steps: support calculation and candidate pattern generation. A reverse-order filling strategy is employed to calculate the support of a candidate pattern, which reduces the time complexity. OWSP-Miner generates candidate patterns using pattern join strategy, which effectively reduces the candidate patterns. For clarification, time series is employed in the experiments and the results show that OWSP-Miner is not only more efficient but also is easier to mine valuable patterns. In the experiment of stock application, we also employ OWSP-Miner

This work was partly supported by National Natural Science Foundation of China (Grants No. 61976240, No. 52077056, and No. 917446209), National Key Research and Development Program of China (Grant No. 2016YFB1000901), and Natural Science Foundation of Hebei Province, China (Grants No. F2020202013 and No. E2020202033).
Authors' addresses: Y. Wu, School of Artificial Intelligence, Hebei University of Technology, Tianjin, China, 300401; Hebei Key Laboratory of Big Data Computing, Tianjin, China, 300401; email: wuc567@163.com; X. Wang, School of Artificial Intelligence, Hebei University of Technology, Tianjin, China, 300401; email: 1692540457@qq.com; Y. Li, School of Economics and Management, Hebei University of Technology, Tianjin, China, 300401; email: lywuc@163.com; L. Guo, State Key Laboratory of Reliability and Intelligence of Electrical Equipment, Hebei University of Technology, Tianjin, China, 300401; email: guoshengrui@163.com; Z. Li (corresponding author), Alibaba-ZJU Joint Research Institute of Frontier Technologies, Zhejiang University, Zhejiang, China, 310000; email: zhao_li@zju.edu.cn; J. Zhang (corresponding author), School of Sciences, The University of Southern Queensland, Toowoomba, Australia, 4350; email: Ji.Zhang@usq.edu.au; X. Wu (corresponding author), Key Laboratory of Knowledge Engineering with Big Data (Hefei University of Technology), Ministry of Education, Hefei, China, 230009, Mininglamp Technology, Beijing, China, 100084; email: xwu@hfut.edu.cn.

2158-656X/2022/02-ART25 $15.00
https://doi.org/10.1145/3476247

to mine OWSPs and the results show that OWSPs mining is more meaningful in real life. The algorithms and data can be downloaded at https://github.com/wuc567/Pattern-Mining/tree/master/OWSP-Miner.

CCS Concepts: • **Information systems → Data mining;**

Additional Key Words and Phrases: Sequential pattern mining, self-adaptive, time series, gap constraint, weak gap

ACM Reference format:
Youxi Wu, Xiaohui Wang, Yan Li, Lei Guo, Zhao Li, Ji Zhang, and Xindong Wu. 2022. OWSP-Miner: Self-adaptive One-off Weak-gap Strong Pattern Mining. *ACM Trans. Manage. Inf. Syst.* 13, 3, Article 25 (February 2022), 23 pages.
https://doi.org/10.1145/3476247

1 INTRODUCTION

Sequential pattern mining (SPM) has been widely applied in many fields [1, 3, 4], such as to feature extraction [5], event logs [6–8], analyze bioinformation information [9], big data [10], inspection reports [11, 12], and time series [13, 14]. To handle some specific issues, many kinds of SPM methods have been proposed to mine special patterns, such as negative patterns [15–17], co-location patterns [18], periodic patterns [19], maximal patterns [20, 21], closed patterns [22], and high utility patterns [23–25]. However, traditional SPM neglects the number of occurrences of a pattern in a sequence. To tackle this issue, gap constraint (or wildcard gaps) SPM [26] was proposed, which aims to discover frequent patterns that satisfy the support threshold and gap constraints. The pattern with gap constraints can be described as $P = p_1[min_1, max_1] \ldots p_j[min_j, max_j] \ldots [min_{m-1}, max_{m-1}]p_m$, where min_j and max_j represent the minimum and maximum gap constraints between p_j and p_{j+1} $(0 < j < m)$, respectively [27, 28].

However, traditional gap constraint SPM neglects the effect of each character (for example, in DNA sequence, the characters are "A," "C," "G," and "T," and in users purchase sequence, the characters can be seen as items, such as clothes and diamonds [29–31]), resulting in patterns that might not be of interest to the users [32–35]. To handle this drawback, SPM with weak-gap constraints [36, 37] was proposed, which divides characters into strong and weak characters according to effect extent. The strong characters can exist only in patterns, while the weak ones exist in gaps. For clarification, a time series is employed to illustrate the issue. For instance, in the stock market, if we use traditional methods to mine frequent patterns, then users are not interested in most of the results, since the daily closing price usually fluctuates little. Generally, users pay more attention to large fluctuations. To effectively mine these large fluctuations, the fluctuation of time series is converted into a sequence, and large fluctuations correspond to strong characters, while small fluctuations correspond to weak ones. For example, the time series in Figure 1(a), which is transformed into sequence S_1 = OOAab, is a small fluctuation, while the time series transformed into S_2 = CdEcf in Figure 1(b) is a large one. The time series in Figures 1(c) and 1(d) is transformed into S_3 = CedcEFcDf and S_4 = CAdaEbcOf, respectively. Both S_3 and S_4 can be matched with S_2 with gap constraints [0,1]. The trend of Figure 1(c) is dissimilar to that of Figure 1(b), since characters "e," "c," "F," and "D" in S_3 are strong characters. However, the trend of Figure 1(d) is similar to that of Figure 1(b), since "A," "a," "b," and "O" in S_4 are weak characters.

One-off SPM [38, 39] is a kind of SPM with gap constraints, which means that each character in sequence can be matched at most once. Introducing the one-off condition not only reduces the support (or the number of occurrences) but also finds more valuable patterns [40, 41]. Example 1 illustrates support, which is a key issue in the one-off SPM.

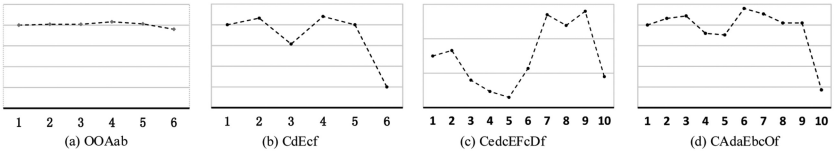


Fig. 1. Time series and their corresponding symbolic sequences. Panel (a) is composed of weak characters, which the users are not interested in, while (b) consists of strong characters. The trends of panels (b) and (c) are dissimilar, since "e," "c," "F," and "D" are strong characters in sequence S_3 = CedcEFcDf. However, the trends in panels (b) and (d) are similar, since "A," "a," "b," and "O" are weak characters in sequence S_4 = CAdaEbcOf.

		1	2	3	4	5	6	7	8	
S	=	B	C	B	B	A	C	B	C	
		B	C	B						First occurrence
		B	C		B					Second occurrence
				B			C	B		Third occurrence
					B		C	B		Fourth occurrence

Fig. 2. All occurrences of pattern P = B[0, 2]C[0, 2]B in sequence S = BCBBACBC.

Example 1. Suppose we have sequence $S = s_1s_2s_3s_4s_5s_6s_7s_8$=BCBBACBC and pattern $P = p_1[0,2]p_2[0,2]p_3$ = B[0, 2]C[0, 2]B. All occurrences are as shown in Figure 2.

To make it easier to describe, an occurrence is described in the form of a sequence landmark. For example, the first occurrence of P in S is $s_1s_2s_3$, which can be written as <1, 2, 3>. Similarly, the other occurrences are <1, 2, 4>, <3, 6, 7>, and <4, 6, 7>. The results are shown in Figure 2. Among them, <1, 2, 3> and <1, 2, 4> are not two one-off occurrences, since s_1 and s_2 are matched twice in the two occurrences, while $< 1, 2, 3 >$ and <4, 6, 7> are two one-off occurrences, since each character is matched at most once. Thus, the one-off occurrences are <1, 2, 3> and <4, 6, 7>. Hence, the one-off support of pattern P in sequence S is 2.

More importantly, most of the previous work required users to set the gap in advance [42, 43]. However, it is difficult for users to set a reasonable gap without prior knowledge. If an inappropriate gap is set, then some valuable information will not be mined. In Example 1, if the gap is [4, 5], then there is no occurrence.

To effectively mine the patterns that users are interested in, all characters are divided into strong and weak characters, where strong characters correspond to strong interests, while weak ones correspond to weak interests. The pattern is composed of strong characters, while only weak characters are allowed in the gaps. Moreover, each character can be used at most once in the process of support calculation. More importantly, to avoid the loss of some interesting patterns due to improper gap constraints, self-adaptive gaps are adopted. We therefore address self-adaptive **one-off weak-gap strong pattern (OWSP)** mining. The main contributions of this article are as follows.

- This article addresses self-adaptive OWSP mining and proposes an effective algorithm named OWSP-Miner.
- OWSP-Miner contains two key steps: support calculation and candidate pattern generation. To calculate the support, we propose a reverse-order filling strategy, which can efficiently calculate the support, since it avoids creating the useless nodes and does not need to prune the redundant nodes after finding an occurrence. Besides, a pattern join strategy is proposed to prune the candidate patterns effectively.

Table 1. Pattern Occurrences Under Different Conditions

Conditions	Support	Occurrences
No-condition	4	<1, 2, 4>, <1, 2, 5>, <1, 4, 5>, and <3, 4, 5>
Nonoverlapping condition	2	<1, 2, 4> and <3, 4, 5>
One-off condition	1	<1, 2, 4>

- Experiments on time series verify the effectiveness of OWSP-Miner and demonstrate that OWSP-Miner has better performance than other competitive algorithms. More importantly, OWSP-Miner is easier to mine valuable patterns.

The rest of this article is organized as follows. Section 2 introduces the related work. Section 3 defines the problem. Section 4 proposes the OWSP-Miner algorithm and presents the complexity analysis. Section 5 reports the algorithm performance and comparisons. Section 6 concludes this article.

2 RELATED WORK

SPM is an important research area and plays a significant role in practical applications [44–48]. For example, Sumalatha and Subramanyam [49] explored high utility time interval sequential pattern mining for big data. Wu et al. [50] proposed a new approach to predict stock trends based on SPM. The above methods ignore the similarity between sub-sequences and patterns in trends, which leads to identifying patterns that are not all interesting to users. To handle this problem, many methods have been proposed. From the perspective of distance, Guo et al. [51] focused on an efficient algorithm for mining approximate frequent patterns in DNA and protein sequences, which employed the Hamming distance to measure the approximation. Zhang and Atallah [52] controlled the difference between the two characters within a given threshold δ and further considered the approximate degree. From the perspective of characters, Tan et al. [37] studied to mine frequent patterns with weak wildcards and made the patterns and sub-sequence have similar trends.

Gap constraint SPM [53–55] has attracted much attention due to its flexibility. On this basis, many achievements have been made, such as contrast SPM [56–58] and closed SPM [59–61]. These studies are based on three types of conditions, no-condition [62–64], the nonoverlapping condition [65, 66] and the one-off condition [67, 68]. A comparison of the occurrences of pattern P = D*C*C in sequence S = DCDCC under different conditions is shown in Table 1.

As can be seen from Table 1, the no-condition means that the sequence letters can be matched multiple times by patterns. Therefore, there are four occurrences under no-condition, i.e., {<1, 2, 4>, <1, 2, 5>, <1, 4, 5>, <3, 4, 5>}. The nonoverlapping condition means that the sequence letters can be matched by the same pattern letter at most once [69]. Thus, there are two nonoverlapping occurrences, i.e., {<1, 2, 4>, <3, 4, 5>}. The one-off condition requires that the characters of the sequence can be used at most once. According to this condition, there is only one occurrence <1, 2, 4>. Obviously, the characters in the sequence will be reused under nonoverlapping and no condition, causing lots of redundant information. In addition, it is more reasonable to use the one-off condition in practical applications. For example, when customers purchase a mobile phone and its accessories, it is more meaningful to consider them as a one-off occurrence. Hence, the one-off SPM can find more valuable information. Table 2 shows a comparison of related works.

From Table 2, Wu et al. [9] and Tan et al. [37] are the closest to this article. Wu et al. [9] used the one-off SPM but it did not consider the problem of weak-gaps, and the efficiency of the support calculating algorithm is not well enough. However, this article mines self-adaptive OWSPs under weak-gap and proposes the ROF algorithm, which employs the reverse-order filling strategy to

Table 2. Related Works

Research	Research type	Gap type	Type of condition	Pruning strategy
Ji et al. [53]	Mining	Traditional gap	No-condition	Apriori-like
Li et al. [62]	Mining	Traditional gap	No-condition	Apriori-like
Zhu and Wu [64]	Mining	Traditional gap	No-condition	Apriori-like
Guo et al. [67]	Matching	Traditional gap	One-off	—
Wu et al. [9]	Mining	Self-adaptive and traditional gap	One-off	Apriori
Tan et al. [37]	Mining	Weak-gap	No-condition	Apriori
This article	Mining	Self-adaptive and weak-gap	One-off	Apriori

calculate the pattern support to reduce the time complexity. Tan et al. [37] adopted no-condition to calculate support and set the gap constraints to mine the no-condition weak-gap patterns. Whereas, this article employs the one-off condition to calculate support and does not need to set the gap constraints to find self-adaptive OWSPs. Hence, this article focuses on self-adaptive OWSP mining.

3 PROBLEM DEFINITION

In this section, we show the definitions of the addressed problem and several examples are given to illustrate the definitions clearly.

Definition 1. A sequence with length n can be expressed as $S = s_1 \ldots s_i \ldots s_n$, where $s_i (1 \leq i \leq n) \in \Sigma$, where Σ represents a set of different characters and $|\Sigma|$ indicates the size. A sequence dataset with N sequences can be expressed as $SDB = \{S_1, S_2, \ldots, S_N\}$.

Definition 2. All characters (Σ) can be divided into strong (Γ) and weak (Ω) characters, i.e., $\Sigma = \Gamma \cup \Omega$ and $\Gamma \cap \Omega = \emptyset$.

Example 2. Suppose we have a sequence $S = s_1 s_2 s_3 s_4 s_5 s_6 s_7 s_8$ = caddccdd. We know that Σ = {a, c, d} and $|\Sigma| = 3$. If Ω = {a}, then Γ = {c, d}.

Definition 3. The self-adaptive strong pattern of length m can be expressed as $P = p_1[0, +\infty) \ldots p_j[0, +\infty) \ldots p_m$ or $P = p_1 * \ldots p_j * \ldots p_m (p_j \in \Gamma, 1 \leq j \leq m)$.

Definition 4. Suppose we have a sequence $S = s_1 s_2 \cdots s_n$ and a pattern $P = p_1 * \ldots p_j * \ldots p_m$. Suppose $L = <l_1, l_2 \ldots l_j \ldots l_m>$ ($s_{l_j} = p_j$, $1 \leq j \leq m$ and $1 \leq l_j \leq n$) is an occurrence of pattern P in sequence S. *occ* is a weak-gap occurrence if and only if for all $s_y \in \Omega$ ($l_{j-1} < y < l_j$ and $2 \leq j \leq m$). Suppose $occ' = <l'_1, l'_2, \ldots, l'_j, \ldots, l'_m>$ is another weak-gap occurrence. *occ* and *occ′* are two one-off weak-gap occurrences if and only if $l_i \neq l'_j$ $(1 \leq i, j \leq m)$.

Example 3. In Example 2, given strong pattern P = c*d*d and *minsup* = 2, <5, 7, 8> is not a weak-gap occurrence, since the character between s_5 and s_7 is s_6 = c, which is a strong character, i.e., c $\notin \Omega$. However, <1, 3, 4> is a weak-gap occurrence of P, since the character between s_1 and s_3 is s_2 = a $\in \Omega$ and there is no character between s_3 and s_4. Similarly, <6, 7, 8> is a weak-gap occurrence of P. Thus, <1, 3, 4> and <6, 7, 8> are two one-off weak-gap occurrences.

Definition 5. The one-off support of a strong pattern P in a sequence S is the number of one-off weak-gap occurrences, denoted by $sup(P, S)$. The one-off support of a strong pattern P in a sequence dataset SDB, denoted by $sup(P, SDB)$, is the sum of its support in each sequence, i.e., $sup(P, SDB) = \sum_{k=1}^{N} sup(P, S_k)$. If $sup(P, S)$ or $sup(P, SDB)$ is no less than the support threshold, then P is called an OWSP. Our problem is to find all OWSPs when the sequence (sequence dataset), Ω and the support threshold are given.

Example 4. In Example 3, suppose the support threshold is *minsup*=2. We know pattern P is an OWSP, since there are two one-off weak-gap occurrences: <1, 3, 4> and <6, 7, 8>, i.e., $sup(P, S) = 2 \geq minsup$. When Ω = {a}, all OWSPs are {c, d, c*d, d*d, c*d*d}.

4 OWSP-MINER

To deal with self-adaptive OWSP mining, there are two main factors affecting mining performance, support calculation and candidate pattern generation. In Section 4.1, we propose a simple algorithm called the **obverse order filling (OOF)** algorithm, which uses the obverse order filling strategy to calculate the pattern support. However, OOF will create some redundant nodes. To avoid creating these redundant nodes, Section 4.2 presents the **reverse-order filling (ROF)** algorithm, which adopts the reverse-order filling strategy. ROF therefore improves the efficiency of support calculations. Section 4.3 employs the pattern join strategy to generate candidate patterns. The OWSP-Miner algorithm is proposed in Section 4.4.

4.1 OOF

In this section, we propose the OOF algorithm, which adopts the obverse order filling strategy to calculate the support of pattern. The steps of the OOF algorithm are shown as follows. Step 1: Create m-level queues, where m is the length of the pattern. Step 2: Read the sequence character s_i from left to right and determine whether $s_i = p_j$ from the first level to the last. If $s_i = p_j$, then there are three cases. Case 1: If $j = 1$, then node n_1^i is created in the first level. Case 2: If $j > 1$, $num_j < num_{j-1}$, $i > p_{j-1}$.front, and the gap characters between n_j^i and n_{j-1}^k belong to Ω, then node n_j^i is created in the jth level, where num_j represents the number of nodes in the jth level, n_{j-1}^k is the last node in the $(j-1)$th level, and p_{j-1}.front represents the first node in the $(j-1)$th level. Case 3: If $j > 1$, $num_j < num_{j-1}$, $i > p_{j-1}$.front, and the gap characters between n_j^i and n_{j-1}^k belong to Γ, then it means that node n_j^i and n_{j-1}^k cannot reach the leaf; then, n_j^i and n_{j-1}^k and its corresponding sub-occurrence nodes are pruned. Step 3: Once a node is created in the mth level, it means finding an occurrence. The nodes of the occurrence and their related nodes are pruned. Step 4: Iterate Steps 2 and 3 until all sequence characters are processed.

Example 5 illustrates the principle of the OOF algorithm.

Example 5. Suppose we have sequence $S = s_1s_2s_3s_4s_5s_6s_7s_8s_9s_{10}s_{11}$ = DACDDCCDCBD, Ω = {A, B}, and pattern $P = p_1p_2p_3$ = D*C*D.

Since the length of pattern P = D*C*D is 3, three-level queues are created.

Read s_1 = D and determine it with p_1, p_2 and p_3 in turn. Since $s_1 = p_1$, according to Case 1, node 1 is created in the first level, i.e., n_1^1. Since $s_1 \neq p_2$, no node needs to be created. Although $s_1 = p_3$, no node needs to be created in the third level, since there is no node in the second level. Read s_2 = A, which cannot match any p_j, i.e., $s_2 \neq p_j$. Thus, no node is created. Let us consider s_3 = C. We can know that $num_2 < num_1$, since there is one node in the first level and no node in the second level. The first node in the first level is 1, i.e., p_{2-1}.front = 1. Therefore, n_2^3 is created according to Case 2, since $s_3 = p_2$, $num_2 < num_1$, 3> p_{2-1}.front = 1 and the gap character between s_3 and s_1 is s_2 = A, which belongs to Ω. Dealing with s_4 = D, since $s_4 = p_1$, n_1^4 is created. Similarly, n_3^4 is also created, as shown in Figure 3. Once n_3^4 is created, it means finding an occurrence, i.e., <1, 3, 4>, whose corresponding nodes are n_1^1, n_2^3 and n_3^4. Then, these nodes are pruned. Meanwhile, node n_1^4 is a related node with n_3^4 under the one-off condition and is also pruned.

Similarly, nodes n_1^5 and n_2^6 are created. Since s_7 = C, which is a strong character, node n_1^5 and n_2^6 cannot reach a leaf according to Case 3. Thus, nodes n_2^6 and its corresponding sub-occurrence node n_1^5 are pruned, as shown in Figure 4.

Fig. 3. The blue nodes are the first occurrence <1, 3, 4> of pattern P in sequence S. After getting the first occurrence, nodes n_1^1, n_2^3, and n_3^4 are pruned. Meanwhile, node n_1^4 is also pruned, since it is a related node with n_3^4 under the one-off condition.

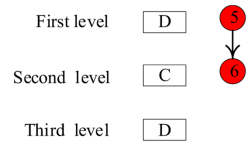


Fig. 4. Although nodes n_1^5 and n_2^6 are created, s_7 = C is a strong character. Therefore, nodes n_1^5 and n_2^6 cannot reach the third level and are pruned.

Fig. 5. The blue nodes are the second occurrence <8, 9, 11> of pattern P in sequence S.

Similarly, we create nodes n_1^8, n_2^9, and n_1^{11}. According to Case 2, n_3^{11} is also created, as shown in Figure 5. Once n_3^{11} is created, it means finding another occurrence, i.e., <8, 9, 11>. According to Step 3 of OOF, nodes n_1^8, n_2^9, n_1^{11}, and n_3^{11} are pruned.

The OOF algorithm is shown in Algorithm 1.

4.2 ROF

The OOF algorithm, which is easy to understand, can calculate the support. However, OOF creates some redundant nodes, such as nodes n_1^4 and n_1^{11} in Figures 3 and 5. To avoid creating the redundant nodes, in this section, we present the ROF algorithm, which employs the reverse-order filling strategy to efficiently calculate the support. The steps of the ROF algorithm are shown as follows.

Step 1: Create m-level queues, where m is the length of the pattern;

Step 2: Read the sequence character s_i from left to right and determine whether $s_i = p_j$ from the last level to the first. If $s_i = p_j$, then there are three cases.

Case 4: If $j > 1$, $num_j < num_{j-1}$, and the gap characters between n_j^i and n_{j-1}^k belong to Ω, then node n_j^i is created in the jth level, where num_j represents the number of nodes in the jth level and n_{j-1}^k is the last node in the $(j-1)$th level.

Case 5: If $j > 1$, $num_j < num_{j-1}$, and the gap characters between n_j^i and n_{j-1}^k belong to Γ, then it means that node n_j^i and n_{j-1}^k cannot reach the leaf; then, n_j^i and n_{j-1}^k and its corresponding sub-occurrence nodes are pruned.

Case 6: If $j = 1$, then node n_1^i is created in the first level.

Step 3: Once a node is created in the mth level, it means finding an occurrence.

Step 4: Iterate Steps 2 and 3 until all sequence characters are processed.

We also take Example 5 to illustrate the principles of the ROF algorithm

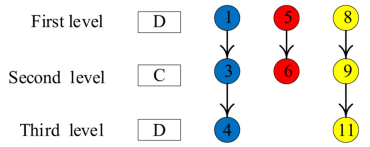


Fig. 6. All nodes of P in S according to the ROF algorithm (the number of nodes in the third level is the support of pattern P in sequence S). Two occurrence are <1, 3, 4> and <8, 9, 11>. Although nodes n_1^5 and n_2^6 are created, they should be pruned, since s_7 = C is a strong character, thus nodes n_1^5 and n_2^6 cannot reach the third level.

ALGORITHM 1: OOF: Calculate the support of pattern P in sequence S

Input: Sequence S, pattern P, weak characters sets Ω
Output: $sup(P, S)$

```
1: Create m-level queues according to the length of pattern P;
2: for i=1 to the length of S step 1 do
3:     for j=1 to m step 1 do
4:         if Case 1 is satisfied then
5:             Create node n_1^i;
6:         end if
7:         if Case 2 is satisfied then
8:             Create node n_j^i;
9:             if j=m then
10:                sup(P, S)++;
11:                Prune the occurrence of nodes and their related nodes;
12:            end if
13:        else if Case 3 is satisfied then
14:            Prune node n_{j-1}^k and its corresponding sub-occurrence nodes;
15:        end if
16:    end for
17: end for
18: return sup(P, S)
```

Since the length of pattern P = D*C*D is 3, three-level queues are created.

Read s_1 = D and determine it with p_3, p_2, and p_1 in turn. Although $s_1 = p_3$, no node needs to be created in the third level, since there is no node in the second level. Since $s_1 = p_1$, according to Case 6, node 1 is created in the first level, i.e., n_1^1. Read s_2 = A, which cannot match any p_j, i.e., $s_2 \neq p_j$. Thus, no node is created. Read $s_3 = C = p_2$, n_2^3 is created according to Case 4, since $num_2 < num_1$ and the gap character between s_3 and s_1 is s_2 = A, which belongs to Ω. Similarly, n_3^4 is also created. Once n_3^4 is created, it means finding an occurrence, i.e., <1, 3, 4 >, whose corresponding nodes are n_1^1, n_2^3, and n_3^4. Similarly, nodes n_1^5 and n_2^6 are created. Since s_7 = C, which is a strong character, node n_1^5 and n_2^6 cannot reach a leaf according to Case 5. Thus, nodes n_1^5 and n_2^6 are pruned. According to the above method, the ROF finds another occurrence, i.e., <8, 9, 11>. The ROF algorithm ends when all sequence characters are processed, as shown in Figure 6.

As we can see, the two algorithms find the same number of occurrences. However, compared with OOF, the ROF algorithm has two advantages: (1) The ROF algorithm does not create useless nodes. (2) The ROF algorithm does not need to remove the nodes after finding an occurrence. Hence, the ROF algorithm greatly improves time efficiency. The ROF algorithm is shown in Algorithm 2.

ALGORITHM 2: ROF: Calculate the support of pattern P in sequence S

Input: Sequence S, pattern P, weak characters sets Ω
Output: $sup(P, S)$

```
Create m-level queues according to the length of pattern P;
for i=1 to the length of S step 1 do
    for j=m to 1 step 1 do
        if Case 4 is satisfied then
            Create node n_j^i;
            if j=m then
                sup(P,S)++;
            end if
        else if Case 5 is satisfied then
            Prune node n_{j-1}^k and its corresponding sub-occurrence nodes;
        else if Case 6 is satisfied then
            Create node n_1^i;
        end if
    end for
end for
return sup(P,S)
```

Theorem 1. *The time complexity of ROF is $O(m \times n)$, where m and n are the pattern length and sequence length, respectively.*

Proof. Obviously, s_i is matched with pattern P, and the time complexity is $O(m)$. Since the sequence length is n, the time complexity of ROF is $O(m \times n)$. □

Theorem 2. *The space complexity of ROF is $O(n)$, where n is the sequence length.*

Proof. According to the ROF algorithm, s_i only appears in the queue at most once, and the overall length will not exceed $O(n)$. Hence, the space complexity of ROF is $O(n)$. □

4.3 Pattern Join

Since the one-off SPM satisfies the Apriori property, we adopt the pattern join strategy to generate candidate patterns, which can effectively prune the candidate patterns and improve mining efficiency.

Definition 6. Suppose we have a pattern $P = p_1 p_2 \cdots p_m$ and event items c and d, if $Q = Pc$, then P is called the prefix sub-pattern of Q, denoted by $prefix(Q)$=P. Similarly, if R=dP, then P is called the suffix sub-pattern of R, denoted by $suffix(R)$=P. Since $prefix(Q)$=$suffix(R)$=P, we can get a super-pattern $T = prefix(\mathrm{Q}) \oplus suffix(R)$=d$P$c.

Example 6. Let pattern P=ATTC. The prefix sub-pattern and the sufix sub-pattern of P are ATT and TTC, respectively. If Q=TTCG, then super-pattern $T = prefix(\mathrm{Q}) \oplus suffix(P)$=ATTCG.

Although both breadth-first and depth-first strategies can be employed to generate the candidate patterns, the pattern join strategy outperforms the above two strategies. An illustrative example is as follows.

Example 7. Sequence $S = s_1 s_2 s_3 s_4 s_5 s_6 s_7 s_8 s_9 s_{10}$ = DABCDBACDA, Ω = {a} and $minsup$ = 2. There are three OWSPs with length 2, {"B*C", "C*D", "D*B"}. Σ = {A,B,C,D}, Γ = {B,C,D}. With either breadth-first or depth-first strategies, $3 \times 3 = 9$ candidate patterns with length 3 will be generated, since each OWSP with length 2 will generate three candidate patterns. For example, B*C and

ALGORITHM 3: OWSP-Miner: Mine all OWSPs

Input: Sequence database *SDB*, *minsup*, Ω.
Output: OWSP set *F*

```
1: Scan sequence database SDB, calculate the support of each item, and store the OWSPs with length 1 into
   F_1;
2: m ← 1;
3: C ← PatternJoin (F_m);
4: while C ≠ null do
5:     for each P in C do
6:         support ← 0;
7:         for each S in SDB do
8:             support ← support+ ROF(S, P, Ω);
9:         end for
10:        if support ≥ minsup then
11:            F_{m+1} ← F_{m+1} ⋃ P;
12:        end if
13:    end for
14:    C ← PatternJoin ( F_{m+1}) ;
15:    m ← m + 1;
16: end while
17: return F;
```

Γ = {B,C,D} will generate B*C*B, B*C*C and B*C*D. In addition, since "C*B" is not an OWSP, super-pattern "C*B*C" is not an OWSP according to the Apriori property and can be pruned. Therefore, there are 3 candidate patterns with length 3, {"B*C*D", "C*D*B", "D*B*C"}, by pattern join strategy. Hence, the pattern join strategy is more efficient than breadth-first and depth-first strategies.

4.4 OWSP-Miner

In this section, we propose the OWSP-Miner algorithm and analyse its time and space complexities.

The steps of the OWSP-Miner algorithm are shown as follows.

Step 1: Get OWSPs of length 1 from the *SDB*.

Step 2: Generate candidate pattern set C with length m+1 using OWSP set F_m.

Step 3: Calculate the support of pattern P in set C.

Step 4: If $sup(P) > minsup$, then store pattern P in F_{m+1}.

Step 5: Repeat Steps 3 and 4 until all patterns in set C have been calculated.

Step 6: Repeat the above steps until set C is empty.

The OWSP-Miner algorithm is given in Algorithm 3.

Theorem 3. *The time complexity of OWSP-Miner is $O(m \times n \times L)$, where m, n and L are the pattern length, the sequence length and the number of the candidate patterns, respectively.*

Proof. According to Theorem 1, the time complexity of ROF is $O(m \times n)$. For all candidate patterns, the time complexity of ROF is $O(m \times n \times L)$. Since the pattern join method is used to generate candidate patterns, the time complexity of generating all candidate patterns is $O(L \times log(L))$. Thus, the time complexity of the OWSP-Miner algorithm is $O(m \times n \times L + L \times log(L)) = O(m \times n \times L)$ □

Theorem 4. *The space complexity of the OWSP-Miner algorithm is $O(m \times L+n)$, where m, n, and L are the pattern length, the sequence length, and the number of the candidate patterns, respectively.*

Proof. The space of the OWSP-Miner algorithm consists of two parts, the space of ROF and the candidate patterns. According to Theorem 2, the space complexity of ROF is $O(n)$. Meanwhile, the space complexity of candidate patterns is $O(m \times L)$. Hence, the space complexity of the OWSP-Miner algorithm is $O(m \times L+n)$. □

5 EXPERIMENT ANALYSIS

To clarify the performance of OWSP-Miner, some numerical time series and protein datasets are employed. Section 5.1 proposes the data pre-processing that transforms the numerical time series into a character sequence, since OWSP-Miner discovers OWSPs in a character sequence. Section 5.2 explains the benchmark datasets and introduces the competitive algorithms. Section 5.3 shows the efficiency of OWSP-Miner. Section 5.4 presents the mining effects of OWSP-Miner. Section 5.5 verifies the mining capability of OWSP-Miner and competitive algorithms. Section 5.6 shows the self-adaptive capability of OWSP-Miner. Section 5.7 further reports the stock application.

All experiments are conducted on a computer with an Intel Core i5, 1.6 GHz CPU, 12 GB RAM, and the Windows 10 operating system. The algorithm and competitive algorithms are developed in VC++6.0, which runs as the experimental environment.

5.1 Data Pre-processing

Data pre-processing is a significant process that transforms the numerical time series into a character sequence. According to previous studies [36, 37], data fluctuation is more valuable than data itself. Thus, we use the data fluctuation to convert the time series into a sequence.

Before introducing the pre-processing process, some relevant definitions are given.

Definition 7. Suppose we have a time series $T = \{t_i \mid i = 1, 2, \ldots, n+1\}$, the fluctuation from time i to $i+1$ is $f_i = (t_{i+1} - t_i)/t_i$, where $1 \le i \le n$.

Definition 8. The weak character rate is the ratio of the number of weak characters to total characters, denoted as ρ.

The data pre-processing process is as follows.

Step 1: Calculate the fluctuations according to Definition 7 and sort the fluctuations in ascending order.

Step 2: Generate the code table. According to the fluctuation with the smallest absolute value, a group of fluctuations satisfying ρ is found around it. This group of fluctuations is equally distributed to Ω, and the remaining fluctuations are assigned to Γ.

Step 3: Time series T is converted into S according to the code table.

Example 8 illustrates the principles of data pre-processing.

Example 8. Suppose we have time series T = {5, 4.65, 4.4, 4.2, 4.074, 4.074, 4.1, 4.223, 4.4, 4.65, 4.9755}, ρ = 40%, Ω = {A, B, C, a, b, c}, and Γ = {D, E, F, G, d, e, f, g}.

Step 1: T is converted into 10 fluctuations, which are then sorted in ascending order, i.e., −7%, −5.3%, −4.5%, −3%, 0%, 0.6%, 3%, 4.2%, 5.6%, 7%.

Step 2: Generate the code table shown in Table 3. Find the fluctuation with the smallest absolute value, i.e., 0%, and take the $10 \times 40\% = 4$ fluctuation around it, i.e., −3%, 0%, 0.6%, 3%. Allocate them to the Ω = {A, B, C, a, b, c}, and the rest to the Γ = {D, E, F, G, d, e, f, g}.

Step 3: T is converted into S = GFECaacefg.

5.2 Benchmark Datasets and Baseeline Methods

Table 4 explains the benchmark datasets used in the following experiments.

Table 3. Code Table

f_i	Code	f_i	Code
[−1%, 0%)	A	(0%, 1%]	a
[−2%, −1%)	B	(1%, 2%]	b
[−3%, −2%)	C	(2%, 3%]	c
[−4%, −3%)	D	(3%, 4%]	d
[−5%, −4%)	E	(4%, 5%]	e
[−6%, −5%)	F	(5%, 6%]	f
[−7%, −6%)	G	(6%, 7%]	g

Table 4. Description of Benchmark Datasets

Dataset	Type	From	$\|\Sigma\|$	Number of sequences	Average sequence length	Length
SDB1[1]	Weather	Nongzhanguan	14	1	3,024	3,024
SDB2	Weather	Changping	11	1	4,006	4,006
SDB3	Weather	Aotizhongxin	13	1	5,040	5,040
SDB4	Weather	Tiantan	14	1	6,024	6,024
SDB5	Weather	Shunyi	14	1	6,998	6,998
SDB6[2]	Stock	Nasdaq	14	1	2,409	2,409
SDB7	Stock	Dow	14	1	2,666	2,666
SDB8	Stock	S&P500	14	1	5,148	5,148
SDB9[3]	Greenhouse gas	CO_2	14	1	18,500	185,000
SDB10[4]	Protein	ASTRAL95_1_161	20	338	186	62,985
SDB11	Protein	ASTRAL95_1_171	20	590	185	109,424
SDB12[5]	Sales	Video game sales	14	1	100,303	100,303
SDB13[6]	Clickstream	Ferenc Bodon	14,892	25,518	8	204,801

SDB10 and SDB11 are protein data. Proteins contain 20 amino acids, among which there are the essential amino acids [36]. This article therefore selects the essential amino acids as strong characters Γ = {H, I, L, K, M, F, T, W, V} and others as weak ones.

We verify the performance of OWSP-Miner from five experiments, the efficiency mining effect, mining capability, self-adaptive capability, and case study. In the first experiment, we verify the efficiency of OWSP-Miner, and four competitive algorithms including Ows-OWSP, OWSP-Oof, OWSP-Bf, and OWSP-Df are selected. Ows-OWSP and OWSP-Oof are used to report the efficiency of reverse-order filling strategy, and OWSP-Bf and OWSP-Df are used to verify the efficiency of pattern join strategy. In the second experiment, we report the performances of frequent patterns under the one-off condition and OWSPs, and PMBC is selected to discover the former ones. In the third experiment, we compare the mining capability, and three competitive algorithms including OWSP-Nogap, PWM and PMBC are selected. OWSP-Nogap is used to prove the advantage of gap constraint, PWM is used to demonstrate the advantage of the one-off SPM, and PMBC is used to show that OWSP mining is more valuable. In the fourth experiment, we propose Sail-OWSP to

[1]SDB1-5 databases are the temperature data of various stations in Beijing, which comes from https://archive.ics.uci.edu/ml/datasets/Beijing+Multi-Site+Air-Quality+Data.

[2]SDB6-8 databases are commonly used in time series analysis and can be downloaded from https://finance.yahoo.com.

[3]SDB9 is taken from https://www.esrl.noaa.gov/gmd/ccgg/trends.

[4]SDB10-11 can be downloaded from https://scop.berkeley.edu/astral/.

[5]SDB12 is the U.S. video game sales dataset, which can be downloaded from http://dataju.cn/Dataju/publishing/payContentPublishingDescription/688.

[6]SDB13 can be downloaded from http://fimi.uantwerpen.be/data/.

Table 5. Baseline Methods

Algorithms	Support calculation	Gap type	Type of condition	Candidate pattern generation	Mining types
Ows-OWSP	One-way scan [9]	Self-adaptive and weak-gap	One-off	Pattern join	OWSPs
OWSP-Oof	OOF	Self-adaptive and weak-gap	One-off	Pattern join	OWSPs
OWSP-Bf	ROF	Self-adaptive and weak-gap	One-off	Breadth-first	OWSPs
OWSP-Df	ROF	Self-adaptive and weak-gap	One-off	Depth-first	OWSPs
OWSP-Nogap	ROF	Self-adaptive	One-off	Pattern join	OWSPs
PWM	PWM [37]	Weak-gap	No-condition	Pattern join	Frequent patterns
PMBC	PMBC [9]	Self-adaptive and traditional gap	One-off	Apriori	Frequent patterns
Sail-OWSP	Sail [68]	Traditional gap	One-off	Pattern join	OWSPs
OWSP-Miner	ROF	Self-adaptive and weak-gap	One-off	Pattern join	OWSPs

demonstrate the ability of self-adaptive mining. The final experiment shows the significance of OWSP-Miner in real life. The specific competitive algorithms are as follows.

1. Ows-OWSP and OWSP-Oof: Ows-OWSP and OWSP-Oof are proposed to verify the performance of ROF in calculating pattern support. The Ows-OWSP algorithm employs the one-way scan strategy proposed in Reference [9] to calculate the pattern support. The OWSP-Oof algorithm applies the OOF algorithm proposed in Section 4.1 to calculate the support. The above two algorithms also generate candidate patterns using the pattern join strategy.

2. OWSP-Bf and OWSP-Df: To verify the efficiency of the pattern join strategy, OWSP-Bf and OWSP-Df are proposed to generate candidate patterns by the breadth-first and depth-first strategies, respectively. The support calculation methods of these two algorithms are the same as that of this article.

3. OWSP-Nogap: To analyze the effect of gap constraint, OWSP-Nogap is proposed to mine continuous OWSPs without a gap. The support calculation and candidate pattern generation strategies are the same as that of this article.

4. PWM: To analyze the mining effect of the one-off condition, we employ PWM [37] as a competitive algorithm to mine self-adaptive OWSPs under no conditions.

5. PMBC: To analyze the difference between OWSP-Miner and traditional SPM algorithms, we employ PMBC [9] as a competitive algorithm to mine self-adaptive one-off frequent patterns.

6. Sail-OWSP: To report the effect of self-adaptive SPM, Sail-OWSP is proposed to mine the one-off OWSPs under gap constraint. The OWSP-Sail algorithm employs SAIL strategy proposed in Reference [68] to calculate the pattern support and generate candidate patterns using the pattern join strategy.

For clarification, we show the characteristics of the competitive algorithms in Table 5.

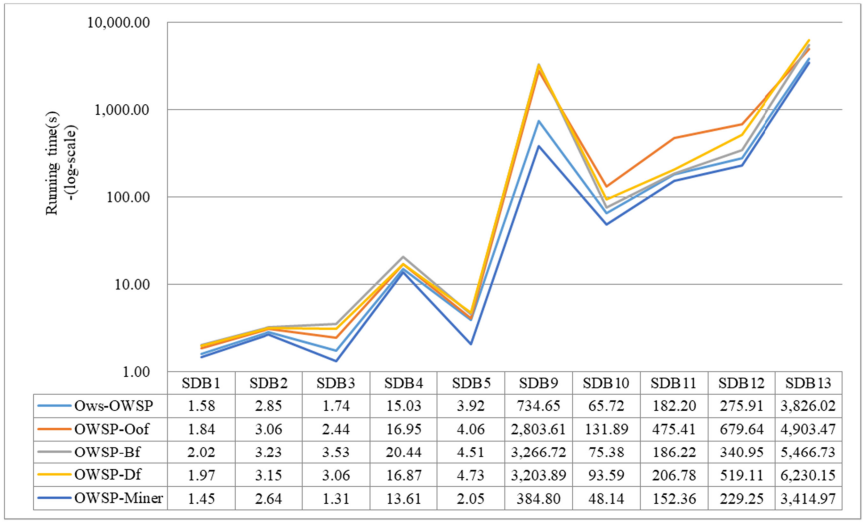
5.3 Efficiency

In this section, we will verify the efficiency of OWSP-Miner with different strategies. Four competitive algorithms are selected, which are Ows-OWSP, OWSP-Oof, OWSP-Bf, and OWSP-Df. We use ten databases, SDB1, SDB2, SDB3, SDB4, SDB5, SDB9, SDB10, SDB11, SDB12, and SDB13 to carry out the experiment. The parameters are Ω = {A, B, C, a, b, c}, ρ = 70%, and *minsup* = 60. The comparisons of running time, number of candidate patterns and number of OWSPs are shown in Figures 7–9, respectively.

The results indicate the following observations:

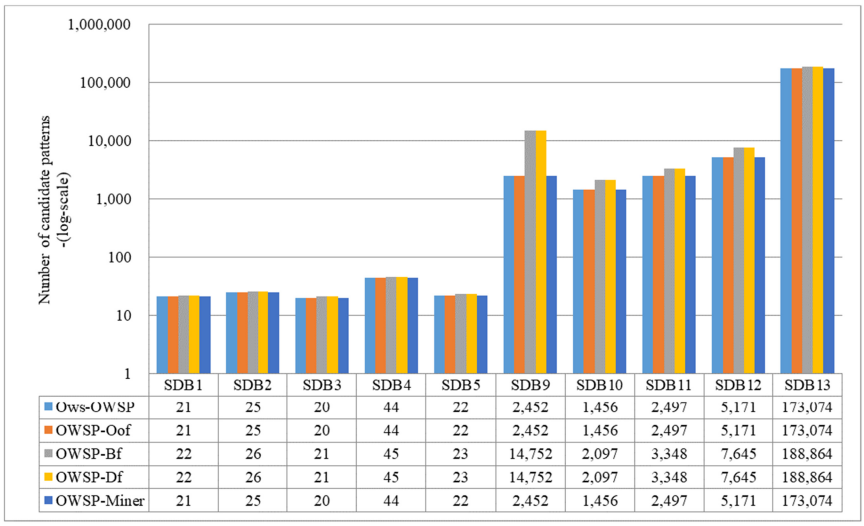
1. The ROF strategy is significantly efficient.

According to Figures 7–9, Ows-OWSP, OWSP-Oof, and OWSP-Miner yield the same number of patterns and candidate patterns, but the running time of OWSP-Miner is less than that of Ows-OWSP and OWSP-Oof. For example, for SDB10 in Figure 7, the running time of Ows-OWSP, OWSP-Oof, and OWSP-Miner is 65.72, 131.89, and 48.14 s, respectively. The reason is that



	SDB1	SDB2	SDB3	SDB4	SDB5	SDB9	SDB10	SDB11	SDB12	SDB13
Ows-OWSP	1.58	2.85	1.74	15.03	3.92	734.65	65.72	182.20	275.91	3,826.02
OWSP-Oof	1.84	3.06	2.44	16.95	4.06	2,803.61	131.89	475.41	679.64	4,903.47
OWSP-Bf	2.02	3.23	3.53	20.44	4.51	3,266.72	75.38	186.22	340.95	5,466.73
OWSP-Df	1.97	3.15	3.06	16.87	4.73	3,203.89	93.59	206.78	519.11	6,230.15
OWSP-Miner	1.45	2.64	1.31	13.61	2.05	384.80	48.14	152.36	229.25	3,414.97

Fig. 7. Comparison of running times.



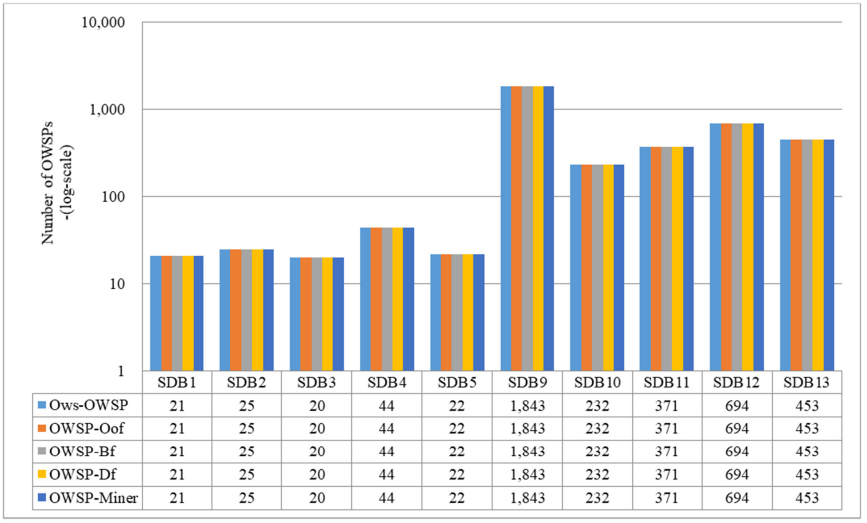
	SDB1	SDB2	SDB3	SDB4	SDB5	SDB9	SDB10	SDB11	SDB12	SDB13
Ows-OWSP	21	25	20	44	22	2,452	1,456	2,497	5,171	173,074
OWSP-Oof	21	25	20	44	22	2,452	1,456	2,497	5,171	173,074
OWSP-Bf	22	26	21	45	23	14,752	2,097	3,348	7,645	188,864
OWSP-Df	22	26	21	45	23	14,752	2,097	3,348	7,645	188,864
OWSP-Miner	21	25	20	44	22	2,452	1,456	2,497	5,171	173,074

Fig. 8. Comparison of number of candidate patterns.

OWSP-Miner employs the ROF strategy and reduces the running time by avoiding creating invalid nodes. Thus, OWSP-Miner has better performance than Ows-OWSP and OWSP-Oof.

2. The pattern join strategy is more efficient.

OWSP-Bf, OWSP-Df, and OWSP-Miner mine the same pattern number of OSWPs, but OWSP-Miner is faster than OWSP-Bf and OWSP-Df. For example, for SDB10 in Figure 7, OWSP-Bf and OWSP-Df run 75.38 and 93.59 s, respectively, while the running time of OWSP-Miner is 48.14 s. The reason is that OWSP-Bf and OWSP-Df calculate 2,097 candidate patterns according to Figure 8, while OWSP-Miner calculates 1,456, because OWSP-Miner employs the pattern join strategy to generate the candidate patterns, while OWSP-Bf and OWSP-Df employ the breadth-first and



	SDB1	SDB2	SDB3	SDB4	SDB5	SDB9	SDB10	SDB11	SDB12	SDB13
Ows-OWSP	21	25	20	44	22	1,843	232	371	694	453
OWSP-Oof	21	25	20	44	22	1,843	232	371	694	453
OWSP-Bf	21	25	20	44	22	1,843	232	371	694	453
OWSP-Df	21	25	20	44	22	1,843	232	371	694	453
OWSP-Miner	21	25	20	44	22	1,843	232	371	694	453

Fig. 9. Comparison of number of OWSPs.

depth-first strategies, respectively. Thus, the pattern join strategy is more efficient than the breadth-first and depth-first strategies, which is consistent with the analysis in Example 7. Therefore, OWSP-Miner outperforms OWSP-Bf and OWSP-Df.

Hence, OWSP-Miner has better running performance than all competitive algorithms.

5.4 Mining Effect

To report the difference between frequent patterns under one-off condition and OWSPs, in this section, PMBC, as a state-of-the-art algorithm to mine the previous defined patterns, is selected as a competitive algorithm. Dataset SDB6 is selected, which is the closing price of Nasdaq index from Jan 1, 2011, to Jul 1, 2020. For OWSP-Miner, the parameters are Ω = {A, B, C, a, b, c}, ρ = 70% and *minsup* = 15. Wordcloud map [70] is employed to show mined patterns in Figure 10.

The results indicate the following observations:

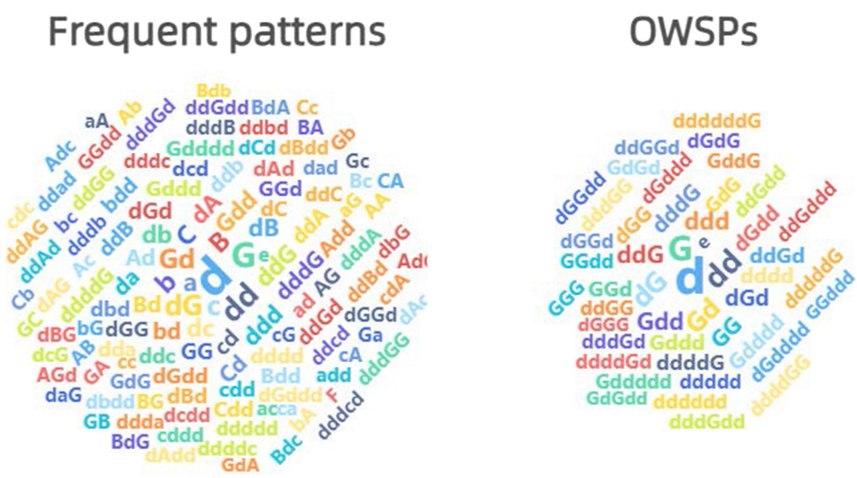
OWSP-Miner has better performance than PMBC, since the former is not only faster but also finds more important patterns than the latter. OWSP-Miner only takes 0.50 s to mine 24 OWSPs, while PMBC takes 3.69 s to discover 73 frequent patterns. More importantly, in the stock market, users pay more attention to large fluctuations. According to Figure 10(a), the patterns discovered by PMBC contain many weak characters, which are less significant than strong ones. However, OWSP-Miner mines OWSPs in Figure 10(b), which are composed of strong characters and are more valuable. Therefore, OWSP-Miner is more meaningful than PMBC.

5.5 Mining Capability

To compare the mining capability of the algorithms on different datasets, we carry out the experiment on ten datasets, SDB4, SDB5, SDB6, SDB7, SDB8, SDB9, SDB10, SDB11, SDB12, and SDB13. The parameters were Ω = {A, B, C, a, b, c}, ρ = 50%, and *minsup* = 50. The results are shown in Figures 11–13.

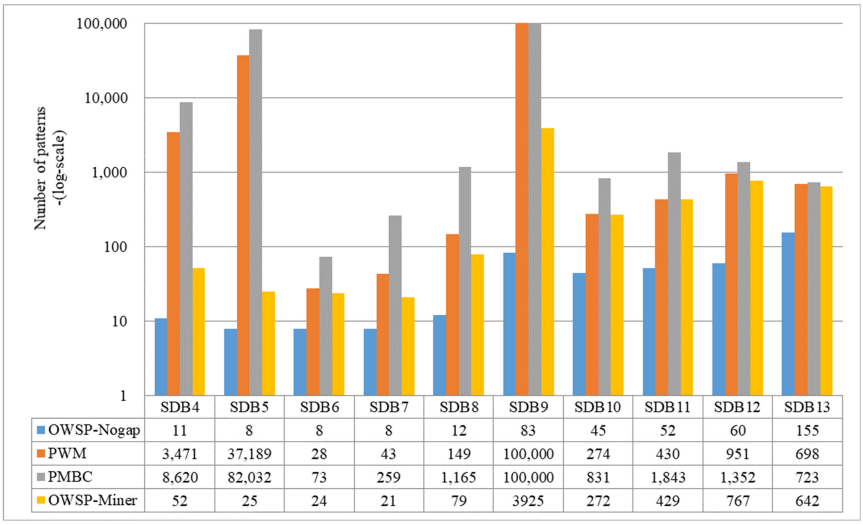
The results indicate the following observations:

1. The introduction of gap constraints improves the mining ability of the algorithm. For example, for SDB7 in Figure 11, OWSP-Miner obtains 21 patterns, while OWSP-Nogap only mines 8 patterns.



(a) Wordcloud map from PMBC (b) Wordcloud map from OWSP-Miner

Fig. 10. Comparison of Wordcloud maps from PMBC and OWSP-Miner. In panel (a), the patterns are mainly composed of weak characters, such as "a," "b," and "c," while those in panel (b) only contains strong patterns, such as "E," "F," and "G.")

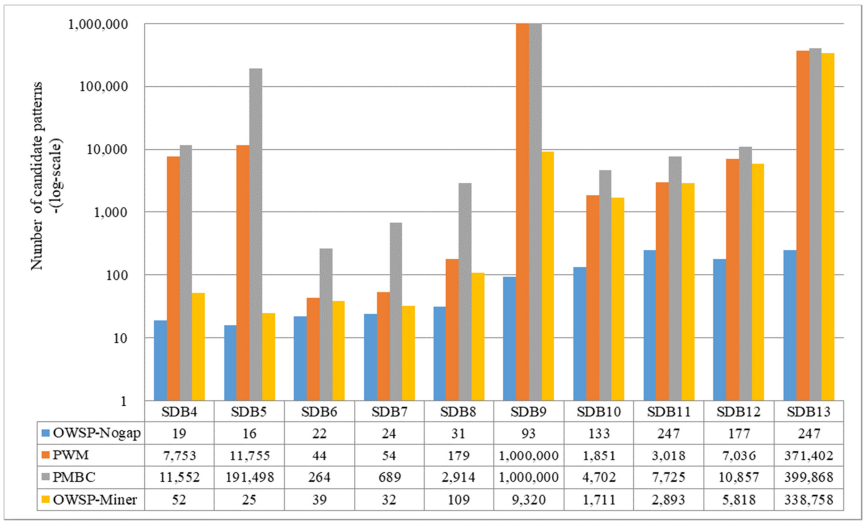


	SDB4	SDB5	SDB6	SDB7	SDB8	SDB9	SDB10	SDB11	SDB12	SDB13
OWSP-Nogap	11	8	8	8	12	83	45	52	60	155
PWM	3,471	37,189	28	43	149	100,000	274	430	951	698
PMBC	8,620	82,032	73	259	1,165	100,000	831	1,843	1,352	723
OWSP-Miner	52	25	24	21	79	3925	272	429	767	642

Fig. 11. Comparison of number of patterns (OWSP-Nogap mines continuous OWSPs without gap, PWM mines self-adaptive OWSPs under no conditions, PMBC mines self-adaptive one-off frequent patterns, and OWSP-Miner mines self-adaptive OWSPs).

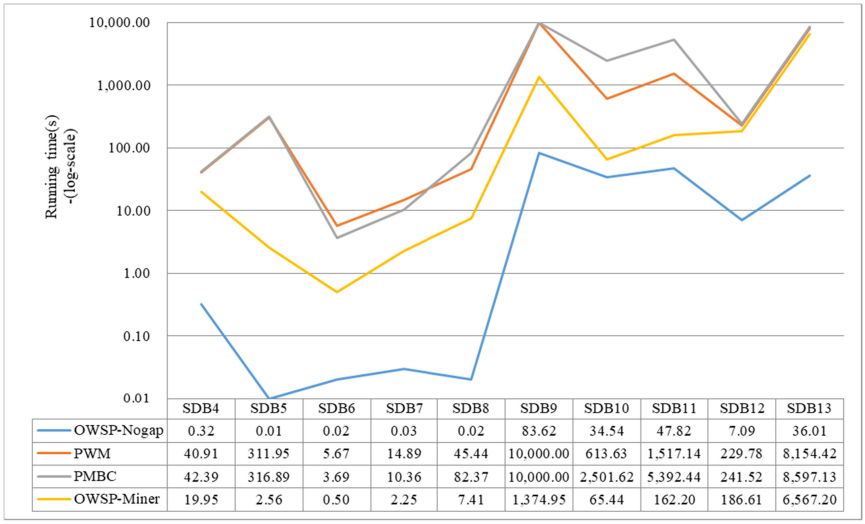
The reason is that OWSP-Miner introduces the gap constraint, which makes more patterns be mined, while OWSP-Nogap only mines continuous patterns without gaps.

2. PWM cannot mine patterns efficiently. For example, for SDB7 in Figures 11 and 13, it takes PWM 14.89 s to obtain 43 patterns while it takes OWSP-Miner 2.25 s to find 21. The reason for this phenomenon is that the no-condition SPM does not satisfy the Apriori property. The support of the super-pattern may be more than the sub-pattern, resulting in much redundant informa-



	SDB4	SDB5	SDB6	SDB7	SDB8	SDB9	SDB10	SDB11	SDB12	SDB13
OWSP-Nogap	19	16	22	24	31	93	133	247	177	247
PWM	7,753	11,755	44	54	179	1,000,000	1,851	3,018	7,036	371,402
PMBC	11,552	191,498	264	689	2,914	1,000,000	4,702	7,725	10,857	399,868
OWSP-Miner	52	25	39	32	109	9,320	1,711	2,893	5,818	338,758

Fig. 12. Comparison of number of candidate patterns.

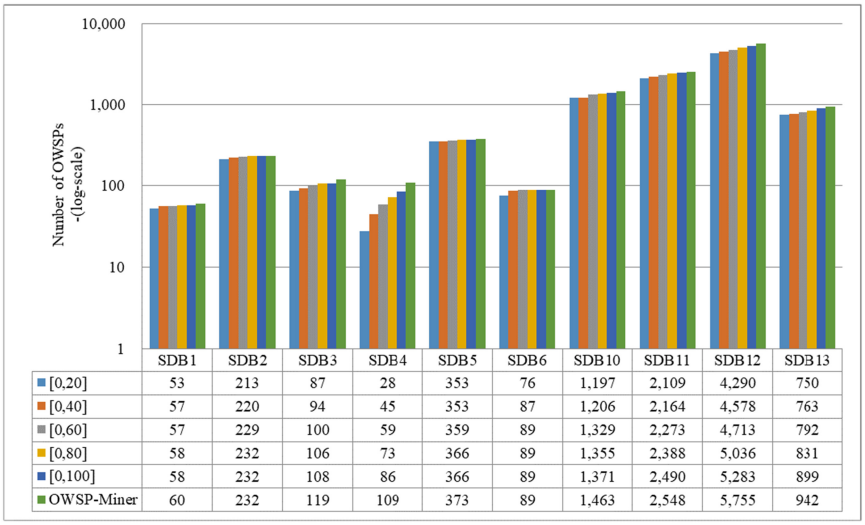


	SDB4	SDB5	SDB6	SDB7	SDB8	SDB9	SDB10	SDB11	SDB12	SDB13
OWSP-Nogap	0.32	0.01	0.02	0.03	0.02	83.62	34.54	47.82	7.09	36.01
PWM	40.91	311.95	5.67	14.89	45.44	10,000.00	613.63	1,517.14	229.78	8,154.42
PMBC	42.39	316.89	3.69	10.36	82.37	10,000.00	2,501.62	5,392.44	241.52	8,597.13
OWSP-Miner	19.95	2.56	0.50	2.25	7.41	1,374.95	65.44	162.20	186.61	6,567.20

Fig. 13. Comparison of running times.

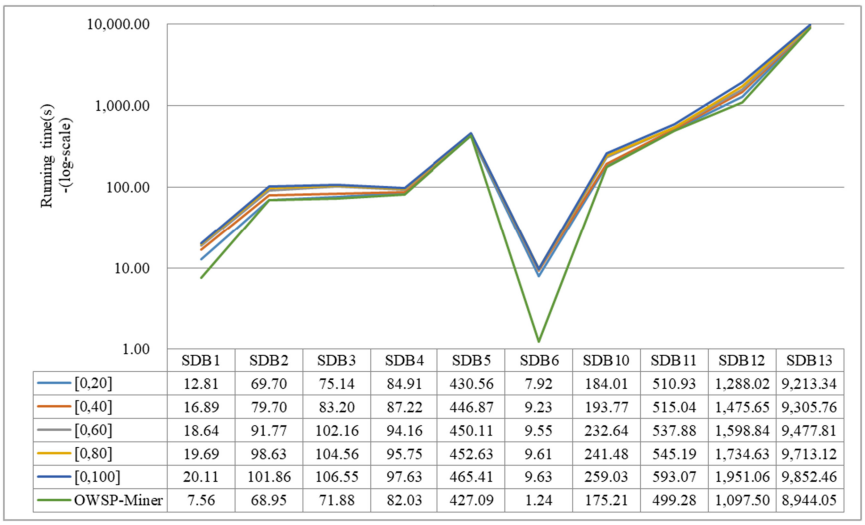
tion. However, the one-off SPM is stricter than the no-condition SPM and mines more valuable information.

3. OWSP-Miner can denoise the pattern effectively. For example, in Figure 11, SDB7, PMBC, and OWSP-Miner find 259 and 21 patterns, respectively. We employ the denoising rate (denoising rate = (number of patterns mined by PMBC − number of OWSPs mined by OWSP-Miner)/number of patterns mined by PMBC) to show the difference between PMBC and OWSP-Miner. Thus, the denoising rate is (259 − 21)/259 = 91.89%. As analyzed in Section 5.4, PMBC discovers frequent patterns that contain many weak characters, but these patterns are not concerned by users. The mined OWSPs by OWSP-Miner are composed of strong characters, which are the focus of users.



	SDB1	SDB2	SDB3	SDB4	SDB5	SDB6	SDB10	SDB11	SDB12	SDB13
[0,20]	53	213	87	28	353	76	1,197	2,109	4,290	750
[0,40]	57	220	94	45	353	87	1,206	2,164	4,578	763
[0,60]	57	229	100	59	359	89	1,329	2,273	4,713	792
[0,80]	58	232	106	73	366	89	1,355	2,388	5,036	831
[0,100]	58	232	108	86	366	89	1,371	2,490	5,283	899
OWSP-Miner	60	232	119	109	373	89	1,463	2,548	5,755	942

Fig. 14. Comparison of number of OWSPs.



	SDB1	SDB2	SDB3	SDB4	SDB5	SDB6	SDB10	SDB11	SDB12	SDB13
[0,20]	12.81	69.70	75.14	84.91	430.56	7.92	184.01	510.93	1,288.02	9,213.34
[0,40]	16.89	79.70	83.20	87.22	446.87	9.23	193.77	515.04	1,475.65	9,305.76
[0,60]	18.64	91.77	102.16	94.16	450.11	9.55	232.64	537.88	1,598.84	9,477.81
[0,80]	19.69	98.63	104.56	95.75	452.63	9.61	241.48	545.19	1,734.63	9,713.12
[0,100]	20.11	101.86	106.55	97.63	465.41	9.63	259.03	593.07	1,951.06	9,852.46
OWSP-Miner	7.56	68.95	71.88	82.03	427.09	1.24	175.21	499.28	1,097.50	8,944.05

Fig. 15. Comparison of running times.

In conclusion, OWSP-Miner can denoise the pattern effectively and is more meaningful than competitive algorithms.

5.6 Self-adaptive Capability

To verify the self-adaptive capability of OWSP-Miner, we selected Sail-OWSP as a comparison algorithm. We carry out the experiment on ten databases, SDB1, SDB2, SDB3, SDB4, SDB5, SDB6, SDB10, SDB11, SDB12, and SDB13. The parameters were $\Omega = \{A, B, C, a, b, c\}$, $\rho = 70\%$, and $minsup = 10$. The results are shown in Figures 14 and 15.

Fig. 16. Comparison of Wordcloud maps from PMBC and OWSP-Miner (In panel (a), the patterns are mainly composed of weak characters, such as "a," "b," and "c," while those in panel (b) only contains strong patterns, such as "E," "F," and "G.")

The results indicate the following observations:

OWSP-Miner has better mining ability than Sail-OWSP, since OWSP-Miner not only finds more patterns but also has a shorter mining time. For SDB1 in Figure 14, Sail-OWSP mines 53 patterns when the gap is [0, 20], while OWSP-Miner mines 60 patterns using a self-adaptive gap. In Figure 15 SDB1, the running time of Sail-OWSP is 12.81 s, while it takes OWSP-Miner 7.56 s to mine patterns. Moreover, if users without prior knowledge set unreasonable gap, some important information will be missed. However, OWSP-Miner determines the gap adaptively according to the sequence. Hence, OWSP-Miner outperforms Sail-OWSP.

5.7 Case Study

In this section, we show the significance of OWSP-Miner in real life and mine OWSPs from SDB8 datasets, which is the closing price of S&P500 index from January 1, 2000, to July 1, 2020. Due to the financial crisis in 2008 and 2020, we select S&P500 index for these two years and the other four years: 2009, 2010, 2018, and 2019. The parameters are Ω = {A, B, C, a, b, c}, ρ = 50%, and *minsup* = 5. The Wordcloud maps of OWSPs are shown in Figure 16.

The results indicate the following observations:

Figure 16 shows all OWSPs in 2008, 2009, 2010, 2018, 2019, and 2020. Intuitively, in 2009, 2010, 2018, and 2019, most of the patterns are composed of characters with less influence, such as "D," "d," and "e." Moreover, the most OWSPs in 2019 are composed by "d," such as OWSPs "ddDd," "ddddd," and "dddddd." This result indicates that the S&P500 index grew steadily and rapidly in 2019. However, in 2008 and 2020, most of the patterns contain characters with greater influence such as "F" and "G." The reason for these phenomena is that the United States' stock development was relatively stable in 2009, 2010, 2018, and 2019. Especially, the economy of the United States was rapid developed in 2019. However, the United States experienced a financial crisis in 2008, resulting in greater fluctuation in the stock market. In addition, the novel coronavirus pneumonia in 2020 has had much impact on the United States. This case study shows that the OWSPs can clearly show the fluctuations in the stock market. Hence, OWSP mining is more meaningful in real life.

Based on the above experimental results, we draw the following conclusions.

- OWSP-Miner has better running performance than all competitive algorithms, since OWSP-Miner equips with two efficient steps: the ROF strategy to calculate the support and the pattern join strategy to prune the candidate patterns.
- OWSP-Miner achieves better mining performance than PMBC, since the former only focuses on strong characters, which are users more interested in, while the latter does on all characters. PMBC therefore discovers many useless patterns. The experimental results verify that OWSPs outperform traditional frequent patterns.
- OWSP-Miner can mine more meaningful patterns than competitive algorithms.

- Compared with gap constraint methods, OWSP-Miner can not only find more patterns but can also run faster than competitive algorithm.
- The OWSPs can clearly show the fluctuations in the stock market.

6 CONCLUSION

This article studies self-adaptive OWSP mining, which mainly solves the following two problems. First, users need to set a suitable gap in advance. Second, each character is considered to have the same effects. To solve the problem, this article proposed the OWSP-Miner algorithm, which has two major steps, support calculation and candidate pattern generation. To calculate the support, we proposed the ROF algorithm, which adopts the reverse-order filling strategy to calculate the one-off support of patterns. Since OWSP mining satisfies the Apriori property, OWSP-Miner adopts the pattern join strategy to generate the candidate patterns. The experimental results show that OWSP-Miner is not only more efficient but can also denoise pattern effectively. In the experimental stock application, we also employed OWSP-Miner to mine OWSPs and the results show that OWSPs mining is more meaningful in real life.

This article focuses on mining frequent OWSPs. The users have to set appropriate frequency threshold. However, if the frequency threshold is too big, there will be no OWSPs, which results in mining failure. If it is too small, then there will be many OWSPs, which makes the mining results difficult to be used. Therefore, Top-k OWSPs is worth investigating. Suppose users have some knowledge to set the frequency threshold. To reduce the number of frequent OWSPs, the maximal or closed OWSPs can be explored in the future. More importantly, as mentioned in Related work section, there are three types of conditions: no-condition, the nonoverlapping condition, and the one-off condition. The strong pattern with weak-gaps under the nonoverlapping condition has not been investigated, which can be studied in the future.

REFERENCES

[1] Philippe Fournier-Viger, Antonio Gomariz, Ted Gueniche, Azadeh Soltani, Chengwei Wu, and Vincent S. Tseng. 2014. SPMF: A java open-source pattern mining library. *J. Mach. Learn. Res.* 15, 1 (2014), 3569–3573.

[2] Youcef Djenouri, Jerry Chun-Wei Lin, Kjetil Norvag, Heri Ramampiaro, and Philip S. Yu. 2021. Exploring decomposition for solving pattern mining problems. *ACM Trans. Manage. Info. Syst.* 12, 2 (2021), 15.

[3] Lei Duan, Guanting Tang, Jian Pei, James Bailey, Akiko Campbell, and Changjie Tang. 2015. Mining outlying aspects on numeric data. *Data Min. Knowl. Discov.* 29, 5 (2015), 1116–1151.

[4] Tingting Wang, Lei Duan, Guozhu Dong, and Zhifeng Bao. 2020. Efficient mining of outlying sequence patterns for analyzing outlierness of sequence data. *ACM Trans. Knowl. Discov. Data* 14, 5 (2020), 62.

[5] Youxi Wu, Yuehua Wang, Yan Li, Xingquan Zhu, and Xindong Wu. 2021. Top-k self-adaptive contrast sequential pattern mining. *IEEE Trans. Cybernet.* DOI:10.1109/TCYB.2021.3082114

[6] Philippe Fournier-Viger, Jiaxuan Li, Jerry Chun-Wei Lin, Tin Truong Chi, and R. Uday Kiran. 2020. Mining cost-effective patterns in event logs. *Knowl.-Based Syst.* 191 (2020), 105241.

[7] Anastasiia Pika, Michael Leyer, Moe T. Wynn, Colin J. Fidge, Arthur H. M. ter Hofstede, and Wil M. P. vander Aalst. 2017. Mining resource profiles from event logs. *ACM Trans. Manage. Info. Syst.* 8, 1 (2017) 1–30.

[8] Youcong Ni, Rui Wu, Xin Du, Peng Ye, Wangbiao Li, and Ruliang Xiao. 2019. Evolutionary algorithm for optimization of energy consumption at GCC compile time based on frequent pattern mining. *J. Softw.* 30, 5 (2019), 1269–1287.

[9] Xindong Wu, Xingquan Zhu, Yu He, and Abdullah N. Arslan. 2013. PMBC: Pattern mining from biological sequences with wildcard constraints. *Comput. Biol. Med.* 43, 5 (2013), 481–492.

[10] Xindong Wu, Xingquan Zhu, Gongqing Wu, and Wei Ding. 2014. Data mining with big data. *IEEE Trans. Knowl. Data Eng.* 26, 1 (2014), 97–107.

[11] He Jiang, Xiaochen Li, Zhilei Ren, Jifeng Xuan, and Zhi Jin. 2019. Toward better summarizing bug reports with crowdsourcing EliciteWd attribute. *IEEE Trans. Reliabil.* 68, 1 (2019), 2–22.

[12] He Jiang, Xin Chen, Tieke He, Zhenyu Chen, and Xiaochen Li. 2018. Fuzzy clustering of crowdsourced test reports for apps. *ACM Trans. Internet Technol.* 18, 2 (2018), 1–28.

[13] Hoonseok Park and Jae-Yoon Jung. 2020. SAX-ARM: Deviant event pattern discovery from multivariate time series using symbolic aggregate approximation and association rule mining. *Expert Syst. Appl.* 141 (2020), 112950.

[14] Mohammad Shokoohi-Yekta, Yanping Chen, Bilson Campana, Bing Hu, Jesin Zakaria, and Eamonn Keogh. 2015. Discovery of meaningful rules in time series. In *Proceedings of the 21st ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 1085–1094.
[15] Xiangjun Dong, Yongshun Gong, and Longbing Cao. 2020. e-RNSP: An efficient method for mining repetition negative sequential patterns. *IEEE Trans. Cybernet.* 50, 5 (2020), 2084–2096.
[16] Xiangjun Dong, Ping Qiu, Jinhu Lü, Longbing Cao, and Tiantian Xu. 2019. Mining top-k useful negative sequential patterns via learning. *IEEE Trans. Neural Netw. Learn. Syst.* 30, 9 (2019), 2764–2778.
[17] Xingming Chen, Yanghui Rao, Haoran Xie, Fu Lee Wang, Yingchao Zhao, and Jian Yin. 2019. Sentiment classification using negative and intensive sentiment supplement information. *Data Sci. Eng.* 4, 2 (2019), 109–118.
[18] Lizhen Wang, Xuguang Bao, and Lihua Zhou. 2021. Redundancy reduction for prevalent co-location patterns. *IEEE Trans. Knowl. Data Eng.* 30(1) (2018), 142–155.
[19] Philippe Fournier-Viger, Peng Yang, Rage Uday Kiran, Sebastián Ventura, and José María Luna. 2021. Mining local periodic patterns in a discrete sequence. *Info. Sci.* 544 (2021), 519–548.
[20] Tin C. Truong, Hai Duong, Bac Le, and Philippe Fournier-Viger. 2019. FMaxCloHUSM: An efficient algorithm for mining frequent closed and maximal high utility sequences. *Eng. Appl. Artific. Intell.* 85 (2019), 1–20.
[21] Bav Vo, Sang Pham, Tuong Le, and Zhi-Hong Deng. 2017. A novel approach for mining maximal frequent patterns. *Expert Syst. Appl.* 73 (2017), 178–186.
[22] Yacine Abboud, Anne Boyer, and Armelle Brun. 2017. CCPM: A scalable and noise-resistant closed contiguous sequential patterns mining algorithm. In *Proceedings of the International Conference on Machine Learning and Data Mining in Pattern Recognition*. Springer, 147–162.
[23] Unil Yun, Donggyu Kim, Eunchul Yoon, and Hamido Fujita. 2018. Damped window based high average utility pattern mining over data streams. *Knowl.-Based Syst.* 144 (2018), 188–205.
[24] Jerry Chun-Wei Lin, Matin Pirouz, Youcef Djenouri, Chien-Fu Cheng, and Usman Ahmed. 2020. Incrementally updating the high average-utility patterns with pre-large concept. *Appl. Intell.* 50, 11 (2020), 3788–3807.
[25] Jerry Chun-Wei Lin, Ting Li, Matin Pirouz, Ji Zhang, and Philippe Fournier-Viger. 2020. High average-utility sequential pattern mining based on uncertain databases. *Knowl. Info. Syst.* 62, 3 (2020), 1199–1228.
[26] Youxi Wu, Yao Tong, Xingquan Zhu, and Xindong Wu. 2018. NOSEP: Nonoverlapping sequence pattern mining with gap constraints. *IEEE Trans. Cybernet.* 48, 10 (2018), 2809–2822.
[27] Youxi Wu, Jinquan Fan, Yan Li, Lei Guo, and Xindong Wu. 2020. NetDAP: (delta, gamma) - Approximate pattern matching with length constraints. *Appl. Intell.* 50, 11 (2020), 4094–4116.
[28] Youxi Wu, Zhiqiang Tang, He Jiang, and Xindong Wu. 2016. Approximate pattern matching with gap constraints. *J. Info. Sci.* 42, (5) (2016), 639–658.
[29] Tin Truong, Hai Duong, Bac Le, Philippe Fournier-Viger, and Unil Yun. 2019. Efficient high average-utility itemset mining using novel vertical weak upper-bounds. *Knowl.-Based Syst.* 183 (2019), 104847.
[30] Jen-Wei Huang, Bijay Prasad Jaysawal, Kuan-Ying Chen, and Yong-Bin Wu. 2019. Mining frequent and top-K high utility time interval-based events with duration patterns. *Knowl. Info. Syst.* 61, 3 (2019), 1331–1359.
[31] Martin Husák, Tomáš Bajtoš, Jaroslav Kašpar, Elias Bou-Harb, and Pavel Čeleda. 2020. Predictive cyber situational awareness and personalized blacklisting: A sequential rule mining approach. *ACM Trans. Manage. Info. Syst.* 11, 4, Article 19 (Sep. 2020), 16 pages.
[32] Wei Song, Yu Liu, and Jinhong Li. 2014. Mining high utility itemsets by dynamically pruning the tree structure. *Appl. Intell.* 40, 1 (2014), 29–43.
[33] Wei Song, Beisi Jiang, and Yangyang Qiao. 2018. Mining multi-relational high utility itemsets from star schemas. *Intell. Data Anal.* 22, 1 (2018), 143–165.
[34] Loan T. T. Nguyen, Phuc Nguyen, Trinh D. D. Nguyen, Bay Vo, Philippe Fournier-Viger, and Vincent S. Tseng. 2019. Mining high-utility itemsets in dynamic profit databases. *Knowl.-Based Syst.* 175 (2019), 130–144.
[35] Jerry Chun-Wei Lin, Wensheng Gan, Philippe Fournier-Viger, Tzung-Pei Hong, and Justin Zhan. 2016. Efficient mining of high-utility itemsets using multiple minimum utility thresholds. *Knowl.-Based Syst.* 113 (2016), 100–115.
[36] Fan Min, Zhi-Heng Zhang, Wen-Jie Zhai, and Rong-Ping Shen. 2020. Frequent pattern discovery with tri-partition alphabets. *Info. Sci.* 507 (2020), 715–732.
[37] Chaodong Tan, Fan Min, Min Wang, Hengru Zhang, and Zhiheng Zhang. 2016. Discovering patterns with weak-wildcard gaps. *IEEE Access* 4 (2016), 4922–4932.
[38] Youxi Wu, Rong Lei, Yan Li, Lei Guo, and Xindong Wu. 2021. HAOP-Miner: Self-adaptive high-average utility one-off sequential pattern mining. *Expert Syst. Appl.* 184 (2021), 115449.
[39] Huiting Liu, Lili Wang, Zhizhong Liu, Peng Zhao, and Xindong Wu. 2018. Efficient pattern matching with periodical wildcards in uncertain sequences. *Intell. Data Anal.* 22, 4 (2018), 829–842.
[40] Fei Xie, Xindong Wu, and Xingquan Zhu. 2017. Efficient sequential pattern mining with wildcards for keyphrase extraction. *Knowl.-Based Syst.* 115 (2017), 27–39.

[41] Thanh Lam Hoang, Fabian Mörchen, Dmitriy Fradkin, and Toon Calders. 2014. Mining compressing sequential patterns. *Stat. Anal. Data Min.* 7, 1 (2014), 34–52.
[42] Youxi Wu, Shuai Fu, He Jiang, and Xindong Wu. 2015. Strict approximate pattern matching with general gaps. *Appl. Intell.* 42, 3 (2015), 566–580.
[43] Youxi Wu, Cong Shen, He Jiang, and Xindong Wu. 2017. Strict pattern matching under non-overlapping condition. *Sci. China Info. Sci.* 60, 1 (2017), 012101.
[44] Yifan Wang. 2003. On-demand forecasting of stock prices using a real-time predictor. *IEEE Trans. Knowl. Data Eng.* 15, 4 (2003), 1033–1037.
[45] Wei Wang, Thomas Guyet, Rene Quiniou, Marie-Odile Cordier, Florent Masseglia, and Xiangliang Zhang. 2014. Autonomic intrusion detection: Adaptively detecting anomalies over unlabeled audit data streams in computer networks. *Knowl.-Based Syst.* 70 (2014), 103–117.
[46] Mengchi Liu and Jun-Feng Qu. 2012. Mining high utility itemsets without candidate generation. In *Proceedings of the 21st ACM International Conference on Information and Knowledge Management.* ACM, 55–64.
[47] Wensheng Gan, Jerry Chun-Wei Lin, Philippe Fournier-Viger, Han-Chieh Chao, and Philip S. Yu. 2020. HUOPM: High-utility occupancy pattern mining. *IEEE Trans. Cybernet.* 50(3) (2020), 1195–1208.
[48] Wensheng Gan, Jerry Chun-Wei Lin, Jiexiong Zhang, Han-Chieh Chao, Hamido Fujita, and Philip S. Yu. 2019. ProUM: Projection-based utility mining on sequence data. *Info. Sci.* 513 (2019), 222–240.
[49] Saleti Sumalatha and R. B. V. Subramanyam. 2020. Distributed mining of high utility time interval sequential patterns using mapreduce approach. *Expert Syst. Appl.* 141 (2020), 112967.
[50] Kuoping Wu, Yungpiao Wu, and Hahn-Ming Lee. 2014. Stock Trend prediction by using K-Means and AprioriAll algorithm for sequential chart pattern mining. *J. Info. Sci. Eng.* 30, 3 (2014), 669–686.
[51] Dan Guo, Ermao Yuan, and Xuegang Hu. 2016. Frequent pattern mining based on approximate edit distance matrix. In *Proceedings of the IEEE 1st International Conference on Data Science in Cyberspace (DSC'16).* 179–188.
[52] Peng Zhang and Mikhail J. Atallah. 2017. On approximate pattern matching with thresholds. *Info. Process. Lett.* 123, 7 (2017), 21–26.
[53] Xiaonan Ji, James Bailey, and Guozhu Dong. 2007. Mining minimal distinguishing subsequence patterns with gap constraints. *Knowl. Info. Syst.* 11, 3 (2007), 259–286.
[54] Unil Yun. 2007. Efficient mining of weighted interesting patterns with a strong weight and/or support affinity. *Info. Sci.* 177, 17 (2007), 3477–3499.
[55] Bac Le, Minh-Thai Tran, and Bay Vo. 2015. Mining frequent closed inter-sequence patterns efficiently using dynamic bit vectors. *Appl. Intell.* 43, 1 (2015), 74–84.
[56] Chao Gao, Lei Duan, Guozhu Dong, Haiqing Zhang, Hao Yang, and Changjie Tang. 2016. Mining top-k distinguishing sequential patterns with flexible gap constraints. In *Proceedings of the International Conference on Web-Age Information Management.* Springer, Cham, 82–94.
[57] Youxi Wu, Yuehua Wang, Jingyu Liu, Ming Yu, Jing Liu, and Yan Li. 2019. Mining distinguishing subsequence patterns with nonoverlapping condition. *Cluster Comput.* 22, 3 (2019), 5905–5917.
[58] Lei Duan, Li Yan, Guozhu Dong, Jyrki Nummenmaa, and Hao Yang. 2017. Mining top-k distinguishing temporal sequential patterns from event sequences. In *Proceedings of the International Conference on Database System for Advanced Applications.* Springer, 235–250.
[59] Sayma Akther, M. R. Rezaul Karim, Md. Samiullah, and Chowdhury Farhan Ahmed. 2018. Mining non-redundant closed flexible periodic patterns. *Eng. Appl. Artific. Intell.* 69 (2018), 1–23.
[60] Huei-Wen Wu and Anthony J. T. Lee. 2010. Mining closed flexible patterns in time-series databases. *Expert Syst. Appl.* 37, 3 (2010), 2098–2107.
[61] Youxi Wu, Changrui Zhu, Yan Li, Lei Guo, and Xindong Wu. 2020. NetNCSP: Nonoverlapping closed sequential pattern mining. *Knowl.-Based Syst.* 196 (2020), 105812.
[62] Chun Li, Qingyan Yang, Jianyong Wang, and Ming Li. 2012. Efficient mining of gap-constrained subsequences and its various applications. *ACM Trans. Knowl. Discov. Data* 6, 1 (2012), 1–39.
[63] Youxi Wu, Lingling Wang, Jiadong Ren, Wei Ding, and Xindong Wu. 2014. Mining sequential patterns with periodic wildcard gaps. *Appl. Intell.* 41, 1 (2014), 99–116.
[64] Xingquan Zhu and Xindong Wu. 2007. Mining complex patterns across sequences with gap requirements. 2007. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence.* Morgan Kaufmann, 2934–2940.
[65] Qiaoshuo Shi, Jinsong Shan, Wenjie Yan, Youxi Wu, and Xindong Wu. 2020. NetNPG: Nonoverlapping pattern matching with general gap constraints. *Appl. Intell.* 50, 6 (2020), 1832–1845.
[66] Bolin Ding, David Lo, Jiawei Han, and Siau-Cheng Khoo. 2009. Efficient Mining of Closed Repetitive Gapped Subsequences from a Sequence Database. In *Proceedings of the 25th International Conference on Data Engineering.* ICDE, 1024–1035.

[67] Dan Guo, Xuegang Hu, Fei Xie, and Xindong Wu. 2013. Pattern matching with wildcards and gap-length constraints based on a centrality-degree graph. *Appl. Intell.* 39, 1 (2013), 57–74.
[68] Gong Chen, Xindong Wu, Xingquan Zhu, Abdullah N. Arslan, and Yu He. 2006. Efficient string matching with wildcards and length constraints. *Knowl. Info. Syst.* 10, 4 (2006), 399–419.
[69] Youxi Wu, Xi Liu, Wenjie Yan, Lei Guo, and Xindong Wu. 2021. Efficient algorithm for solving strict pattern matching under nonoverlapping condition. *J. Softw.* 32, 11 (2021), 3331–3350.
[70] Florian Heimerl, Steffen Lohmann, Simon Lange, and Thomas Ertl. 2014. Word cloud explorer: Text analytics based on word clouds. In *Proceedings of the International Conference on System Science*. IEEE, 1833–1842.

Received November 2020; revised May 2021; accepted July 2021