NTP-Miner: Nonoverlapping Three-Way Sequential Pattern Mining

YOUXI WU, School of Artificial Intelligence, Hebei University of Technology and Hebei Key Laboratory of Big Data Computing
LANFANG LUO, School of Artificial Intelligence, Hebei University of Technology
YAN LI, School of Economics and Management, Hebei University of Technology
LEI GUO, State Key Laboratory of Reliability and Intelligence of Electrical Equipment, Hebei University of Technology
PHILIPPE FOURNIER-VIGER, School of Humanities and Social Sciences, Harbin Institute of Technology (Shenzhen)
XINGQUAN ZHU, Department of Computer & Electrical Engineering and Computer Science, Florida Atlantic University
XINDONG WU, Key Laboratory of Knowledge Engineering with Big Data (Hefei University of Technology), Ministry of Education and Mininglamp Academy of Sciences, Mininglamp Technology

Nonoverlapping sequential pattern mining is an important type of sequential pattern mining (SPM) with gap constraints, which not only can reveal interesting patterns to users but also can effectively reduce the search space using the Apriori (anti-monotonicity) property. However, the existing algorithms do not focus on attributes of interest to users, meaning that existing methods may discover many frequent patterns that are redundant. To solve this problem, this article proposes a task called nonoverlapping three-way sequential pattern (NTP) mining, where attributes are categorized according to three levels of interest: strong, medium, and weak interest. NTP mining can effectively avoid mining redundant patterns since the NTPs are composed of strong and medium interest items. Moreover, NTPs can avoid serious deviations (the occurrence is significantly different from its pattern) since gap constraints cannot match with strong interest patterns. To

This work was partly supported by National Natural Science Foundation of China (61976240, 52077056, 917446209), National Key Research and Development Program of China (2016YFB1000901), National Science Foundation under grant Nos. IIS-1763452 & CNS-1828181, Natural Science Foundation of Hebei Province, China (Nos. F2020202013, E2020202033), and Graduate Student Innovation Program of Hebei Province (CXZZSS2020030).
Authors' addresses: Y. Wu, School of Artificial Intelligence, Hebei University of Technology, Tianjin 300401, China and Hebei Key Laboratory of Big Data Computing, Tianjin 300401, China; email: wuc567@163.com; L. Luo, School of Artificial Intelligence, Hebei University of Technology, Tianjin, 300401, China; email: luofangluo68@163.com; Y. Li (corresponding author), School of Economics and Management, Hebei University of Technology, Tianjin 300401, China; email: lywuc@163.com; L. Guo, State Key Laboratory of Reliability and Intelligence of Electrical Equipment, Hebei University of Technology, Tianjin 300401, China; email: guoshengrui@163.com; P. Fournier-Viger, College of Computer Science and Software Engineering, Shenzhen University, Shenzhen, 518060, China; email: philfv@szu.edu.cn; X. Zhu, Department of Computer & Electrical Engineering and Computer Science, Florida Atlantic University, FL 33431; email: xzhu3@fau.edu; X. Wu, Key Laboratory of Knowledge Engineering with Big Data (Hefei University of Technology), Ministry of Education, Hefei 230009, China, and Mininglamp Academy of Sciences, Mininglamp Technology, Beijing 100084, China; email: xwu@hfut.edu.cn.

1556-4681/2021/10-ART51 $15.00
https://doi.org/10.1145/3480245

mine NTPs, an effective algorithm is put forward, called NTP-Miner, which applies two main steps: support (frequency occurrence) calculation and candidate pattern generation. To calculate the support of an NTP, depth-first and backtracking strategies are adopted, which do not require creating a whole Nettree structure, meaning that many redundant nodes and parent–child relationships do not need to be created. Hence, time and space efficiency is improved. To generate candidate patterns while reducing their number, NTP-Miner employs a pattern join strategy and only mines patterns of strong and medium interest. Experimental results on stock market and protein datasets show that NTP-Miner not only is more efficient than other competitive approaches but can also help users find more valuable patterns. More importantly, NTP mining has achieved better performance than other competitive methods in clustering tasks. Algorithms and data are available at: https://github.com/wuc567/Pattern-Mining/tree/master/NTP-Miner.

CCS Concepts: • **Theory of computation → Design and analysis of algorithms**; • **Computing methodologies → Knowledge representation and reasoning;**

Additional Key Words and Phrases: Sequential pattern mining, frequent pattern, three-way decisions, gap constraint, Apriori property

ACM Reference format:
Youxi Wu, Lanfang Luo, Yan Li, Lei Guo, Philippe Fournier-Viger, Xingquan Zhu, and Xindong Wu. 2021. NTP-Miner: Nonoverlapping Three-Way Sequential Pattern Mining. *ACM Trans. Knowl. Discov. Data.* 16, 3, Article 51 (October 2021), 21 pages.
https://doi.org/10.1145/3480245

1 INTRODUCTION

Data mining aims at extracting meaningful information from large amounts of data [1]. **Sequential pattern mining** (**SPM**) is a key process in data mining [2, 3]. The goal is to find subsequences (or patterns) that appear in at least *minsup* sequences of characters (symbols), where *minsup* is a user-defined parameter [4, 5]. SPM has been applied in various fields such as to analyze biological information [6, 7] and event logs [8], for knowledge point recommendation [9], behavioural informatics [10, 11], and feature selection [12]. To give more flexibility to the user in terms of specifying constraints on patterns to be found, two types of wildcards have been proposed. The "?" wildcard [13] indicates a match with a single character, while "*" denotes a match with multiple characters. For example, "a?b" can match with "acb", "adb" and "aeb", while "a*c" can match "ac", "adbfc" and "afbdbc". In recent years, a more flexible wildcard called gap constraint [7] has been developed. A pattern with gap constraints can be represented as $\mathbf{p} = p_1[a_1, b_1]p_2 \ldots p_{m-1}[a_{m-1}, b_{m-1}]p_m$ [14], and [15], where a_j and b_j are the minimum and maximum number of wildcard between characters p_j and p_{j+1}, respectively. For example, the pattern a[0, 2]c is of the form $\mathbf{p} = p_1[a_1, b_1]p_2$ and represents three wildcard patterns, (ac),(a?c), and (a??c). A pattern $\mathbf{p} = p_1[a_1, b_1]p_2 \ldots p_{m-1}[a_{m-1}, b_{m-1}]p_m$ is called a pattern with periodic gap constraints in the case where $a_1 = a_2 = \cdots = a_{m-1} = \mathrm{x}$ and $b_1 = b_2 = \cdots = b_{m-1} = \mathrm{y}$, and can be abbreviated as $\mathbf{p} = p_1p_2 \ldots p_m$ with $gap = [\mathrm{x}, \mathrm{y}]$ [16–18]. This problem formulation is clearer for the user, and has been widely used in many applications, such as for user purchasing behaviour analysis [11], biological sequence analysis [19], feature extraction [20], and text keyword extraction [21].

However, the current form of SPM with gap constraints does not focus on attributes of interest to users, which gives rise to two problems: (i) Gap constraints can cause serious deviations or even distortions of the original pattern. For example, in stock data, users are mostly interested in volatile patterns. Therefore, volatile characters cannot be matched by gap constraints. (ii) Since all characters can form patterns, this not only leads to a large number of patterns being mined, but also takes a long time. More importantly, users are not interested in certain patterns. Inspired by three-way decisions [22–24] and the concept of tri-pattern mining [6], this article explores

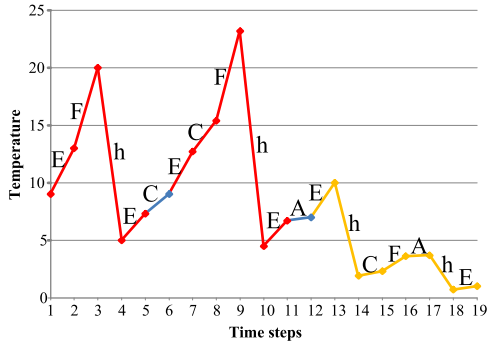


Fig. 1. The red and yellow lines indicate three nonoverlapping occurrences of $\mathbf{p} = p_1p_2p_3p_4$ = EFhE, which can be written as $<1, 2, 3, 4>$, $<6, 8, 9, 10>$ and $<12, 15, 17, 18>$. There is a serious deviation between the yellow line and the two red ones.

nonoverlapping three-way sequential pattern (**NTP**) mining. Two illustrative examples are presented below.

Example 1. Consider the time series shown in Figure 1 where a behavioural attribute "temperature" is measured over 19 time steps. After transforming the time series into characters representing trends, the character sequence $\mathbf{s}_1 = s_1s_2s_3s_4s_5s_6s_7s_8s_9s_{10}s_{11}s_{12}s_{13}s_{14}s_{15}s_{16}s_{17}s_{18}$ = EFhECECFhEAEhCFAhE is obtained. Thus, the set of all characters Σ is {A, C, E, F, h}. Suppose we have a pattern $\mathbf{p} = p_1p_2p_3p_4$ = EFhE with $gap = [a, b] = [0, 2]$, the set of strong character is Γ = {h}, the set of medium characters is Λ = {E, F}, and that of weak characters is Ω = {A, C}.

In Figure 1, the first red line represents the occurrence of $\mathbf{p}$ at time steps <1, 2, 3, 4>, for which the trend is the same as that of $\mathbf{p}$, and there is no gap in the occurrence. The second red line indicates the occurrence <6, 8, 9, 10>, for which the trend is close to that of $\mathbf{p}$, and there is a weak character "C" between positions 6 and 8. The yellow line indicates the occurrence <12, 15, 17, 18>, for which the trend is different from that of $\mathbf{p}$, since there is a strong character "h" between positions 12 and 15, which leads to a different overall trend.

Example 2. Extending Example 1, a second sequence = $s_1s_2s_3s_4s_5s_6s_7s_8s_9s_{10}$ = AAFEAhAAEA is added to have a sequence database $SDB = \{\mathbf{s}_1, \mathbf{s}_2\}$. If the minimum support threshold is set to $minsup$ = 4, there are 12 frequent patterns {A, E, F, h, AA, AE, Eh, FE, Fh, hE, AAA, FhE} appearing in at least 4 sequences of SDB, but only 7 frequent NTPs {E, F, h, Eh, Fh, hE, FhE}. Since users are primarily interested in patterns with larger fluctuations, some patterns that are composed of weak characters are ignored, as these have smaller fluctuations, such as "AA" and "AAA".

To effectively mine patterns that users are interested in, all characters are divided into three levels of interest: strong, medium, and weak. The NTP mining can be applied in many cases.

(i) In some cases, the characters of the datasets are divided into three categories by domain experts. For example, biologists divide the amino acids into essential, conditional essential, and nonconditional essential amino acids. Thus, the essential amino acids are strong characters, conditional essential amino acids are medium characters, and nonconditional essential amino acids are weak characters [6].
(ii) This mining method can be applied in some specific applications, such as rare pattern mining [25]. In rare pattern mining, the character distributions are different. Some characters are frequent, while others are not. Obviously, rare patterns are not made of the frequent

characters, which therefore can be seen as weak characters. The infrequent characters can be seen as strong characters, and the other characters are medium characters.

(iii) In some applications, the users have some priori knowledge. Thus, the interesting characters are strong characters, while the disinteresting characters are weak ones, and the rest are medium ones. For example, in the stock market, the larger fluctuations correspond to strong characters, while the smaller fluctuations correspond to weak ones, and the rest correspond to medium characters.

In the above applications, the NTP mining task can reduce the number of redundant patterns since the NTPs are composed of strong and medium interest characters. More importantly, the NTPs can avoid serious deviations since the gap constraints cannot match with strong interest characters.

The main contributions of this study are as follows:

(1) To avoid mining redundant and noise patterns, the article addresses the NTP mining to discover NTPs, in which all characters are partitioned into three interest levels: strong, medium, and weak. The patterns are composed of strong and medium characters. The gaps are matched with medium and weak characters.

(2) An effective algorithm called NTP-Miner is proposed, which has two key steps: calculating the support (occurrence frequency) of patterns and generating candidate patterns. NTP-Miner employs the depth-first and backtracking strategies to find all nonoverlapping occurrences without creating a whole Nettree and applies a pattern join strategy to reduce the number of candidate patterns. Hence, the time and space complexities are effectively reduced.

(3) An experimental evaluation verifies that NTP-Miner not only outperforms the competitive algorithms, but can also reveal meaningful patterns to users.

The rest of this manuscript is organized as follows. Section 2 introduces related work, and Section 3 defines the problem. Section 4 proposes the NTP-Miner algorithm, which employs depth-first and backtracking strategies to calculate NTP support, and applies a pattern join strategy to generate candidate patterns. Section 5 evaluates the efficiency and performance of NTP-Miner, and Section 6 concludes this article.

2 RELATED WORK

SPM has been applied in many fields such as industry [26], and the original task has been extended in many ways such as for occupancy mining [27], negative SPM [28], closed SPM [29], top-k SPM [30], progressive mining [31], co-location pattern mining [32], SPM with gap constraints [19] and high-utility SPM [33–36]. High-utility SPM takes certain aspects into account such as the profit [37–39] or weight of items (characters) [40–42] to evaluate the interests of users and meet their needs.

In recent years, several other studies have been done to find patterns that are interesting to users. For example, Shen et al. [43] employed a utility occupancy function to identify meaningful patterns, but the efficiency of this approach is low. Gan et al. [1] later proposed a more efficient algorithm. But a drawback of these methods is that a utility value must be assigned to each item, which is difficult to do accurately without prior knowledge. An unsuitable value for the utility will lead to serious deviations and even to miss important patterns. To solve this problem, Tan et al. [44] proposed SPM with weak-wildcard gaps. In this method, the set of all characters was partitioned into strong and weak characters, and weak-wildcard gaps can only match with weak characters. The drawback of this approach is that weak characters can be used to form patterns, even though weak characters are usually uninteresting to users. Inspired by three-way decisions [22, 45], tri-partition alphabets SPM was proposed [6], which partitions the set of characters into

three sets: strong, medium, and weak characters. In this approach, a pattern can only contain strong or medium characters, and gap constraints can only be medium or weak characters.

Both weak-wildcard gap SPM [44] and tri-partition alphabets SPM [6] are special cases of SPM with gap constraints, which can be further divided into three cases: no-condition [16], the one-off condition [46–48] and the nonoverlapping condition [49, 50]. In the case of no-condition, the same sequence characters can be reused in any positions. Although it is easy to calculate the support in this case, the support (or the support ratio) does not satisfy the Apriori (anti-monotonicity) property. Thus, it is necessary to expand the search space to find all patterns [44]. Under the one-off condition, the same sequence characters cannot be reused in any positions. SPM with gap constraints under the one-off condition has been used to extract keywords [21]. However, calculating the support of a pattern is an NP-hard problem, meaning that the support cannot be calculated exactly. Therefore, algorithms for the one-off SPM task are approximate [21]. Under the nonoverlapping condition, the same sequence characters can be reused in different positions, but cannot be reused in the same positions. Although the nonoverlapping condition is more complicated than the other two cases in terms of calculating the support [50], it can be calculated in polynomial time, and nonoverlapping SPM satisfies the Apriori property [7]. More importantly, it is easier to find meaningful patterns in nonoverlapping SPM than the case of no-condition or the one-off condition.

Inspired by previous work [6, 7], this article explores NTP mining. Although both tri-partition alphabets SPM [6] and NTP mining aim at discovering patterns with tri-partition alphabets, there are several differences between these approaches. The former calculates the support of a pattern under no-condition and adds unmatchable characters at the end of a sequence to satisfy the Apriori property [6]. Therefore, it is an approximate mining method. Differently, the scheme proposed in this article calculates the support under the nonoverlapping condition and does not need to add characters to discover all feasible patterns. On the basis of nonoverlapping SPM [7], this article divides the characters into three types: strong, medium, and weak. Only strong and medium characters can form patterns, to ensure that they are valuable to users. More importantly, this method reduces the number of candidate patterns and increases the mining speed. It is worth noting that when the strong and weak character sets are empty, the problem is transformed into nonoverlapping SPM [7], and hence nonoverlapping SPM can be seen as a special case of NTP mining. To improve the mining speed, the proposed approach employs depth-first and backtracking strategies to calculate the support, and does not need to create a whole Nettree structure unlike prior work [7].

3 PROBLEM DEFINITION

The problem studied in this article is based on the following definitions.

Definition 1 (Sequence). A sequence with length n can be described as $\mathbf{s} = s_1 s_2 \dots s_i \dots s_n$, where $s_i \in \Sigma (1 \le i \le n)$. The set of all characters Σ is partitioned into three subsets: the set of strong characters Γ, the set of medium characters Λ, and that of weak characters Ω. The sets Γ, Λ, and Ω are called tri-character sets.

Definition 2 (Three-way Pattern with Periodic tri-wildcard Gap Constraints). A three-way pattern (tri-pattern) with periodic tri-wildcard gap constraints can be expressed as $\mathbf{p} = p_1[a,b]p_2 \dots p_j[a,b] \dots [a,b]p_m$ ($1 < j \le m\text{-}1$, $0 \le a \le b$), and can also be abbreviated as $\mathbf{p} = p_1 p_2 \dots p_m$ with tri-wildcard gap constraints $gap = [a, b]$, where $p_j \in (\Gamma \cup \Lambda)$ and a and b represent the minimum and maximum tri-wildcards, respectively.

Definition 3 (Occurrence). Given a sequence $\mathbf{s} = s_1 s_2 \dots s_i \dots s_n$, and a tri-pattern $\mathbf{p} = p_1 p_2 \dots p_m$ with tri-wildcard $gap = [a, b]$, a list of positions $L = < l_1, l_2, \dots l_j, \dots, l_m >$ is an occurrence of pattern $\mathbf{p}$ in sequence $\mathbf{s}$, if and only if $0 \le l_1 < l_2 < \dots l_j < \dots < l_m \le n$, $a \le l_{j+1} - l_j - 1 \le b$, where $s_{l_j} = p_j (1 \le j \le m\text{-}1)$, $s_{l_j} < s_x < s_{l_{j+1}} (1 \le j \le m\text{-}1)$, and $s_x \in (\Lambda \cup \Omega)$.

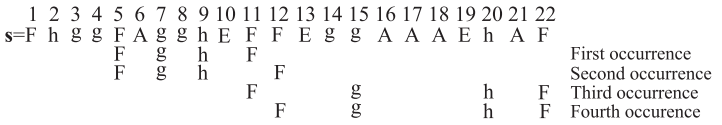


Fig. 2. All occurrences of tri-pattern **p** = FghF with tri-wildcard *gap* = [1, 4] in sequence **s**.

Example 3. Consider a sequence **s** = $s_1s_2s_3s_4s_5s_6s_7s_8s_9s_{10}s_{11}s_{12}s_{13}s_{14}s_{15}s_{16}s_{17}s_{18}s_{19}s_{20}s_{21}s_{22}$ = FhggFAgghEFFEggAAAEhAF, |**s**| = 22, Σ = {F, h, g, A, E}, Γ = {h}, Λ = {E, F, g}, Ω = {A} and a tri-pattern **p** = $p_1p_2p_3p_4$= FghF with a tri-wildcard gap constraints *gap* = [1, 4]. The four matching occurrences of pattern **p** in sequence **s** are shown in Figure 2.

From Figure 2, there are four occurrences of tri-pattern **p** in sequence **s**: <5, 7, 9, 11>, <5, 7, 9, 12>, <11, 15, 20, 22>, and <12, 15, 20, 22>. Although <1, 4, 9, 11> satisfies the gap constraints, it is not an occurrence of **p**, since there is a strong character "h" between positions 1 and 4.

Definition 4 (Nonoverlapping Occurrence Set and Support). Given two occurrences $L = \{l_1, l_2, \ldots, l_j, \ldots, l_m\}$ and $L' = \{l'_1, l'_2, \ldots l'_j, \ldots, l'_m\}$, L and L' are two nonoverlapping occurrences, if and only if for any $j(1 \leq j \leq m)$, $l_j \neq l'_j$. If any two occurrences in a set are nonoverlapping occurrences, then the set is a nonoverlapping occurrence set. The nonoverlapping support of **p** in **s** is the size of the maximum nonoverlapping occurrence set, represented by *sup*(**p**, **s**).

In Example 3, <5, 7, 9, 11> and <5, 7, 9, 12> are two overlapping occurrences of **p**, since characters s_5, s_7, and s_9 are reused by p_1, p_2, and p_3, respectively. Although s_{11} is used twice in occurrences <5, 7, 9, 11 > and <11, 15, 20, 22>, these are two nonoverlapping occurrences, since s_{11} is used by p_4 in the former, and by p_1 in the latter. Thus, the nonoverlapping occurrence set of **p** is <5, 7, 9, 11>, <11, 15, 20, 22>, and its support is *sup*(**p**, **s**) = 2.

Definition 5 (Support for Sequence Database). A sequence database with N sequences can be described as $SDB = \{s_1, s_2, \ldots, s_N\}$. The support of a tri-pattern **p** in *SDB* (*sup*(**p**, *SDB*)) is the sum of the support of **p** in each sequence, that is $sup(\mathbf{p}, SDB) = \sum_{k=1}^{N} sup(\mathbf{p}, \mathbf{s}_k)$, where $1 \leq k \leq N$.

Definition 6 (NTP and NTP Mining). If the support of a tri-pattern **p** in *SDB* is no less than a user-specified minimum support threshold *minsup*, **p** is frequent NTP. The purpose of the proposed method is to discover all frequent NTPs in an *SDB*, given the tri-wildcard gap constraints, Γ, Λ, Ω, and *minsup*.

When Γ and Ω are empty and Λ = Σ, NTP mining becomes nonoverlapping SPM [7]. Hence, nonoverlapping SPM can be seen as a special case of NTP mining.

Example 4. Suppose we have the tri-character sets Γ = {h}, Λ = {E, F, g}, and Ω = {A}, the tri-wildcard *gap* = [1, 4], *minsup* = 3, and *SDB* = {$\mathbf{s}_1$ = $s_1s_2s_3s_4s_5s_6s_7s_8s_9s_{10}s_{11}s_{12}s_{13}s_{14}s_{15}s_{16}s_{17}s_{18}s_{19}s_{20}s_{21}s_{22}$ = FhggFAgghEFFEggAAAEhAF, $\mathbf{s}_2$ = $s_1s_2s_3s_4s_5s_6s_7s_8$ = FAgFghAF}. The support of tri-pattern **p** = $p_1p_2p_3p_4$ = FghF in sequence $\mathbf{s}_1$ is *sup*(**p**, $\mathbf{s}_1$) = 2. Similarly, *sup*(**p**, $\mathbf{s}_2$) = 1. Therefore, *sup*(**p**, *SDB*) = *sup*(**p**, $\mathbf{s}_1$) + *sup*(**p**, $\mathbf{s}_2$) = 2 + 1 = 3. Hence, **p** is an NTP in *SDB*, since *sup*(**p**, *SDB*) ≥ *minsup* = 3. Although *sup*("A", *SDB*) = 7 is greater than *minsup* = 3, the character "A" is not an NTP, since it is a weak character. The set of all NTPs in *SDB* are {F, h, g, E, Fg, hF, gh, gg, Fgh, gghF, FghF}.

4 PROPOSED ALGORITHM

There are two major steps in NTP mining: calculating the support of patterns and generating candidate patterns. Section 4.1 proposes an effective algorithm for calculating the support, called

simplified Nettree for tri-pattern with tri-wildcard gAP constraints (Sim-NAP). Section 4.2 presents a candidate pattern generation strategy based on pattern join. Section 4.3 proposes the NTP-Miner algorithm for NTP mining, and analyzes the space and time complexities of NTP-Miner from a theoretical perspective.

4.1 Sim-NAP Algorithm

The Nettree structure is an extended tree with multiple roots and multiple parents [51]. Some nodes may have the same node ID on different levels. The notation n_j^i refers to node i at the jth level. A path from a root to a leaf is called a full path. Although all occurrences of a pattern **p** in a sequence **s** can be represented as a Nettree, it is difficult to determine which character in sequence **s** can be reused in the nonoverlapping condition. However, it is easy to solve this problem using a Nettree, since any node can be used at most once in this structure. Although the NETGAP algorithm can find all nonoverlapping occurrences by creating a whole Nettree, it fails to handle tri-patterns with tri-wildcard gap constraints. More importantly, NETGAP is inefficient, since it needs to create a whole Nettree, and find and prune invalid nodes only after obtaining an occurrence.

To efficiently calculate the support of an NTP, we propose the Sim-NAP algorithm, which adopts the depth-first search and backtracking strategies to find all nonoverlapping occurrences without creating the whole Nettree. The main steps are as follows:

Step 1: For each s_i ($1 \le i \le n$), check whether or not s_i is the same as p_1. If $s_i = p_1$, then root n_1^i is created at the first level and Sim-NAP selects root n_1^i as the current node.

Step 2: Suppose that the current node is n_j^t ($1 \le j \le m$-1). Sim-NAP finds the child node of n_j^t by the depth-first and backtracking strategies. For each s_k (t+a+1 $\le k \le$ t+b+1), Sim-NAP determines whether s_k is the same as p_{j+1}. If $s_k = p_{j+1}$, there are two cases:

Case 1: There is a strong character between s_t and s_k. In this case, Sim-NAP backtracks to the parent node of n_j^t as the current node.

Case 2: There is no strong character between s_t and s_k. If the (j+1)-th level does not have node n_{j+1}^k, Sim-NAP creates node n_{j+1}^k, establishes the parent–child relationship between nodes n_j^t and n_{j+1}^k, and selects node n_{j+1}^k as the current node. Otherwise, it will find another child node of n_j^t.

Step 3: Sim-NAP repeats Step 2 until $j = m$ or $j = 0$. If $j = m$, then the path from n_1^i to n_j^t is a nonoverlapping occurrence, and Sim-NAP returns to Step 1 to find another root. If $j = 0$, then Sim-NAP returns to Step 1 to find another root.

Step 4: Sim-NAP repeats Step 1 until all characters in the sequence have been processed.

Example 5 illustrates the principle of Sim-NAP.

Example 5. Suppose we have a sequence **s** = $s_1s_2s_3s_4s_5s_6s_7s_8s_9s_{10}s_{11}s_{12}s_{13}s_{14}s_{15}s_{16}s_{17}s_{18}s_{19}s_{20}s_{21}s_{22}$ = FhggFAgghEFFEggAAAEhAF and the tri-patterns **p** = FghF with tri-wildcard gap constraints, and the tri-character sets are Γ = {h}, Λ = {E,F,g}, and Ω = {A}. The simplified Nettree of the tri-pattern **p** in sequence **s** is shown in Figure 3.

Since $s_1 = p_1$, Sim-NAP creates n_1^1, and then finds the child of n_1^1. Although $s_3 = p_2$ = "g", there is s_2 = "h", meaning that there is a strong character between s_1 and s_3. Thus, Sim-NAP backtracks to the parent node of n_1^1. In this case, we know that $j = 0$, and Sim-NAP finds another root. Since $s_5 = p_1$, Sim-NAP creates n_1^5, and then finds the child of n_1^5. Since $s_7 = p_2$ = "g" and there is no strong character between s_5 and s_7, Sim-NAP creates n_2^7. In the same way, Sim-NAP creates n_3^9 and n_4^{11}. After creating n_4^{11}, Sim-NAP finds a full path $<n_1^5, n_2^7, n_3^9, n_4^{11}>$ whose corresponding occurrence is <5, 7, 9, 11>. After Step 3, Sim-NAP returns to Step 1 and finds another root n_1^{11}, and then finds the

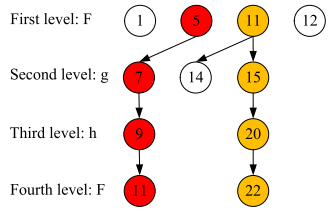


Fig. 3. Simplified Nettree of the tri-patterns **p** = FghF with tri-wildcard gap constraints *gap* = [1, 4] in sequence **s**. Nodes n_1^{11} and n_4^{11} have the same ID 11 at the first and fourth levels, respectively. <5, 7, 9, 11> and <11, 15, 20, 22> are two nonoverlapping occurrences, since the corresponding full paths <$n_1^5, n_2^7, n_3^9, n_4^{11}$> and <$n_1^{11}, n_2^{15}, n_3^{20}, n_4^{22}$> do not share a common node.

ALGORITHM 1: Sim-NAP

Input: sequence **s**, tri-pattern **p** with tri-wildcard *gap* = [*a*, *b*], tri-character sets Γ, Λ and Ω
Output: *sup*(**p**, **s**)

```
1: for i = 1 to n do
2:     if s_i = p_1 then
3:         Create root n_1^i;
4:         j←1 and t←i;
5:         while 0 < j < m do
6:             Iteratively find the child node of n_j^t according to the tri-wildcard gap = [a, b], tri-character sets, and use
               depth-first and backtracking strategies, i.e. node n_{j+1}^k and j←j+1;
7:         end while
8:         if j = m then
9:             sup(p, s)++;
10:        end if
11:    end if
12: end for
13: return sup(p, s);
```

child of n_1^{11}, which is n_2^{14}. With tri-wildcard gap constraint *gap* = [1, 4], n_2^{14} does not have a child. Thus, Sim-NAP backtracks to the parent of n_2^{14}, which is n_1^{11}. Sim-NAP then finds another child of n_1^{11}, which is n_2^{15}. Finally, Sim-NAP finds another full path <$n_1^{11}, n_2^{15}, n_3^{20}, n_4^{22}$> whose corresponding occurrence is <11, 15, 20, 22>. The simplified Nettree produced by Sim-NAP is shown in Figure 3.

The Nettree in Figure 3 can be seen as a forest, since each node has only one parent. Compared with NETGAP, Sim-NAP is more efficient. The reasons are two aspects. First, NETGAP has to create a whole Nettree, in which a node may have many parents. Sim-NAP creates a forest, in which a node has only one parent. Thus, Sim-NAP does not need to create redundant parent–child relationships. Second, NETGAP has to find and prune invalid nodes, while Sim-NAP does not. Hence, Sim-NAP effectively reduces the space and time complexities.

The Sim-NAP algorithm is shown in Algorithm 1.

The space and time complexities of Sim-NAP are both $O(m \times n)$. The reasons are as follows. A Nettree has m levels, the number of nodes in each level does not exceed n, and each node has only one parent. The space complexity of Sim-NAP is $O(m \times n)$, since each node can be visited only once. Hence, the time complexity of Sim-NAP is also $O(m \times n)$.

4.2 Generation of Candidate Patterns

In the proposed NTP-Miner algorithm, pattern generation is done as follows.

Definition 7 (Prefix and Suffix Patterns). For a pattern $\mathbf{p} = p_1 p_2 \dots p_{m-1} p_m$, the prefix of **p** is prefix(**p**) = $p_1 p_2 \dots p_{m-1}$, while the suffix of **p** is suffix(**p**) = $p_2 \dots p_{m-1} p_m$.

Definition 8 (Pattern Join). For a pattern $\mathbf{r}$ and some characters α and β ($\alpha, \beta \in (\Gamma \cup \Lambda)$) patterns $\mathbf{p} = \alpha\mathbf{r}$ and $\mathbf{q} = \mathbf{r}\beta$ are super-patterns of $\mathbf{r}$. A new super-pattern $\mathbf{t}$ can be generated by pattern join, i.e., $\mathbf{t} = \mathbf{p} \oplus \mathbf{q} = \alpha\mathbf{r}\beta$, since suffix($\mathbf{p}$) = prefix($\mathbf{q}$) = $\mathbf{r}$.

Theorem 1. *NTP satisfies the Apriori property. In other words, if a pattern is not an NTP, its super-patterns are also not NTPs.*

Proof. The prefix and suffix patterns of pattern $\mathbf{p}$ are $\mathbf{q}$ and $\mathbf{r}$, respectively. If the nonoverlapping occurrence set of $\mathbf{p}$ in sequence $\mathbf{s}$ is L, the nonoverlapping occurrence sets of $\mathbf{q}$ and $\mathbf{r}$ in sequence $\mathbf{s}$ are L_1 and L_2, respectively, where $L \subseteq L_1$ and $L \subseteq L_2$. Hence, if $\mathbf{q}$ or $\mathbf{r}$ is not a frequent NTP, then $\mathbf{p}$ is not a frequent NTP. Obviously, the above cases are still valid in a sequence database. Therefore, NTP satisfies the Apriori property.

In this article, we employ a pattern join strategy to generate candidate patterns, an approach that can effectively reduce the number of candidate patterns. Example 6 shows that the pattern join strategy outperforms the depth-first and breadth-first strategies.

Example 6. Consider that the sequence $\mathbf{s}_2 = s_1s_2s_3s_4s_5s_6s_7s_8$ = FAgFghAF is added to Example 5 to obtain a sequence database $SDB = \{\mathbf{s}_1, \mathbf{s}_2\}$, and that we aim at finding all frequent NTPs for a minimum support threshold $minsup = 3$.

The depth-first and breadth-first strategies are used to generate candidate patterns, respectively. There are four frequent NTPs of length 2 {Fg, gg, gh, hF}. Since $|\Gamma| + |\Lambda| = 4$, $4 \times 4 = 16$ candidate patterns are generated. But if the pattern join strategy is used, only six candidate patterns are generated: {Fgg, Fgh, ggg, ggh, ghF, hFg}. This example shows that the pattern join strategy can generate much less candidate patterns than the depth-first and breadth-first strategies.

4.3 NTP-Miner Algorithm

This subsection describes the proposed NTP-Miner algorithm and presents an analysis of its space and time complexities. The NTP-Miner algorithm is applied in the following six steps:

Step 1: Generate a candidate pattern set *cand* containing patterns of length m+1, using the frequent pattern set *fre*[m].
Step 2: Calculate the support of pattern $\mathbf{p}$ in *cand*.
Step 3: If pattern $\mathbf{p}$ is frequent, then store it in the frequent pattern set *fre*[m+1].
Step 4: Repeat Steps 2 and 3 until all patterns in *cand* have been checked.
Step 5: All patterns remaining in *fre*[m+1] are NTPs, and are stored in *fre*.
Step 6: Repeat the above steps until the candidate pattern set *cand* is empty.

The NTP-Miner algorithm is shown in Algorithm 2.

Theorem 2. *The space complexity of the NTP-Miner algorithm is $O(M \times (L + n))$, where M, n, and L are the length of the longest pattern, the length of the longest sequence in the database, and the number of candidate patterns, respectively.*

Proof. The memory usage of NTP-Miner algorithm consists of two parts: the space for candidate patterns and the space for calculating the support of frequent patterns. It is easy to see that the space complexity of the first part is $O(M \times L)$, and the space complexity of the Sim-NAP algorithm is $O(M \times n)$. Hence, the space complexity of NTP-Miner is $O(M \times (L + n))$.

Theorem 3. *The time complexity of NTP-Miner is $O(L \times M \times n)$.*

ALGORITHM 2: NTP-Miner: Mine all NTPs

Input: sequence database *SDB*, *minsup*, tri-wildcard *gap* = [*a*, *b*], tri-character sets Γ, Λ and Ω
Output: NTP set.

```
1: Scan the sequence database SDB once, calculate the support of each event item, and store frequent NTPs with length
   1 in fre[1];
2: len←1;
3: cand←PatternJoin(fre[1]);
4: while cand≠null do
5:     for each p in cand do
6:         support←Sim-NAP(SDB, p);
7:         if support≥minsup then
8:             fre[len+1]←fre[len+1]∪p;
9:         end if
10:    end for
11:    cand←PatternJoin(fre[len+1]);
12:    len←len+1;
13: end while
14: return fre;
```

Proof. The time complexity of generating all frequent patterns is $O(L \times logL)$. The time complexity of the Sim-NAP algorithm used to calculate a pattern's support is $O(M \times n)$. Hence, the time complexity of the NTP-Miner algorithm is $O((M \times n + logL) \times L) = O(L \times M \times n)$.

5 EXPERIMENTAL RESULTS AND ANALYSIS

This section presents the experimental evaluation of the proposed NTP-Miner algorithm. Sections 5.1 and 5.2 first describes the benchmark datasets and the data preprocessing approach. Section 5.3 presents the baseline methods. Section 5.4 compares the mining performance of NTP-Miner with other algorithms. Section 5.5 reports the mining capability of tri-wildcard gap constraints. Section 5.6 compares and analyzes the performance of NTPs. Section 5.7 compares the mining performance under the no-condition and the nonoverlapping condition. Section 5.8 shows the case study.

All experiments are conducted on a computer with AMD A10-7300 Radeon R6, 10 Compute Cores 4C+6G 1.90 GHz processor, 4 GB memory, and the Windows operating system. All the algorithms are developed using the Visual Studio C++ 6.0 environment and can be downloaded from https://github.com/wuc567/Pattern-Mining/tree/master/NTP-Miner.

5.1 Benchmark Datasets

Table 1 summarizes characteristics of the benchmark datasets used in this article. Two types of datasets are selected: for character sequences, we use protein sequence datasets, while for time series, we use daily closing prices from the stock market.

5.2 Data Preprocessing

Since protein data is used as character sequence datasets and daily closing stock prices as time series datasets, two different data preprocessing methods are used:

(1) Protein data

Amino acids are the basic units of proteins. Biologists divide amino acids into essential, conditional essential and nonconditional essential amino acids in the context of human biology. In this article, the essential amino acids are strong characters, conditional essential amino acids are medium characters, and nonconditional essential amino acids are weak characters [6]. Hence, Γ = {H, I, L, K, M, F, T, W, V}, Λ = {R, C, Q, G, P, S, Y, N}, and Ω = {A, D, E, U, O, X}.

Table 1. Benchmark Datasets

Dataset	Type	From	Number of sequences	Total length
SDB1[1]	protein	ASTRAL95_1_161	507	91,875
SDB2	protein	ASTRAL95_1_161	338	62,985
SDB3	protein	ASTRAL95_1_161	169	32,503
SDB4	protein	ASTRAL95_1_171	590	109,424
SDB5	protein	ASTRAL95_1_171	400	73,425
SDB6	protein	ASTRAL95_1_171	200	37,327
SP[2]	stock	S&P 500	1	2,516
DJI	stock	Dow30	1	2,516
IXIC	stock	Nasdaq	1	2,516
HSI	stock	Hang Seng Index	1	2,516
SSEC[3]	stock	SZSE composite index	1	2,431
SZI	stock	Shanghai composite index	1	2,431
CAR[4]	sensor	Car	8	4,608
WORMS	eigenworm	Worms	10	8,990

(2) Stock data

We first convert the time series $\mathbf{t} = \{t_i \mid i = 1, \ldots, k+1\}$ into the character sequence $\mathbf{s}$. The main steps are as follows:

Step 1: The fluctuation between time i and i+1 is calculated using $g_i = (t_{i+1} - t_i)/t_i$, where $1 \leq i \leq k$+1.

Step 2: g_i is standardized.

Step 3: We regard 0%–40% of g_i as the weak interval, and convert it to the set of weak characters Ω = {A, B, C, D, a, b, c, d}. The ranges −10% to 0% and 0% −10% of g_i are converted to the characters "a" and "A", respectively, while the ranges 10%–20%, 20%–30%, and 30%–40% of g_i are converted to the characters "B", "C", and "D", respectively. Similarly, the medium interval is defined as 40%–70%, and is converted to the set of medium characters Λ = {E, F, G, e, f, g}. The strong interval is defined as 70%–100%, and is converted to the set of strong character Γ = {H, I, J, h, i, j}.

5.3 Baseline Methods

To verify the performance of the proposed NTP-Miner algorithm, three state-of-the-art algorithms are selected: TPM [6], NOSEP [7], and GSgrow [49]. Several versions of these approaches are also compared to evaluate the efficiency of the proposed algorithm: NTP-ntg, NTP-ntp, NTP-c, NTP-bf, NTP-df, NOSEP-a, and GSgrow-a. A brief introduction to these algorithms is given below.

(1) NOSEP [7]: It mines frequent patterns under the nonoverlapping condition.

(2) TPM [6]: It mines tri-patterns for the case of no-condition.

(3) NTP-ntp: It mines frequent patterns with the tri-wildcard gap constraints.

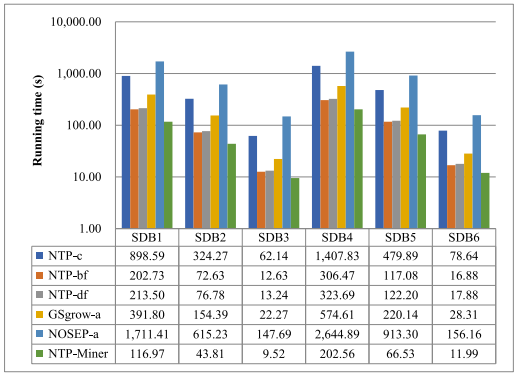
(4) NTP-ntg: To analyze the influence of the gap constraints and the tri-wildcard gap constraints, the NTP-ntg algorithm mines tri-patterns with traditional gap constraints.

[1]SDB1-6 is taken from reference [7].

[2]The SP500, Dow30, Nasdaq and HSI stock datasets can be downloaded from https://finance.yahoo.com/.

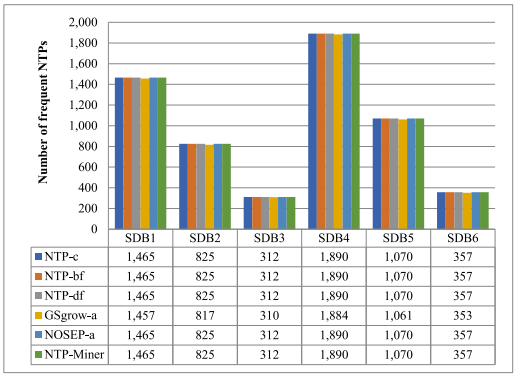
[3]The SSEC and SZI stock datasets can be downloaded from https://money.163.com/.

[4]The CAR and WORMS datasets can be downloaded from http://www.timeseriesclassification.com/index.php.



	SDB1	SDB2	SDB3	SDB4	SDB5	SDB6
NTP-c	898.59	324.27	62.14	1,407.83	479.89	78.64
NTP-bf	202.73	72.63	12.63	306.47	117.08	16.88
NTP-df	213.50	76.78	13.24	323.69	122.20	17.88
GSgrow-a	391.80	154.39	22.27	574.61	220.14	28.31
NOSEP-a	1,711.41	615.23	147.69	2,644.89	913.30	156.16
NTP-Miner	116.97	43.81	9.52	202.56	66.53	11.99

Fig. 4. Comparison of running time.



	SDB1	SDB2	SDB3	SDB4	SDB5	SDB6
NTP-c	1,465	825	312	1,890	1,070	357
NTP-bf	1,465	825	312	1,890	1,070	357
NTP-df	1,465	825	312	1,890	1,070	357
GSgrow-a	1,457	817	310	1,884	1,061	353
NOSEP-a	1,465	825	312	1,890	1,070	357
NTP-Miner	1,465	825	312	1,890	1,070	357

Fig. 5. Comparison of number of frequent NTPs.

(5) NTP-c: To verify the performance of Sim-NAP, the NTP-c algorithm uses the NAP algorithm to calculate the support. The NAP creates the whole Nettree of a tri-pattern and employs depth-first and backtracking strategies to find nonoverlapping occurrences within the Nettree.

(6) NTP-bf and NTP-df: To evaluate the efficiency of the pattern join strategy, NTP-bf and NTP-df use breadth-first and depth-first strategies to generate candidate patterns, respectively.

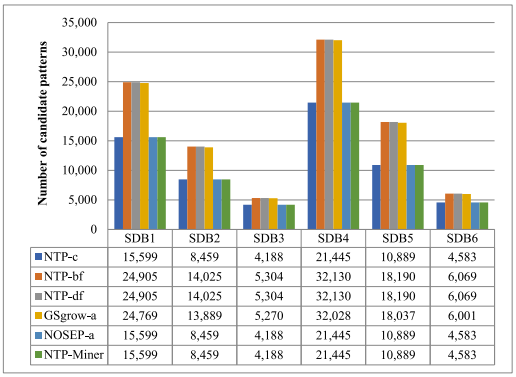
(7) NOSEP-a and GSgrow-a: To validate the performance of NTP-Miner, NOSEP-a and GSgrow-a consider the tri-pattern and the tri-wildcard gap constraints. NOSEP-a employs the NETGAP [7] algorithm to calculate the support of the pattern, while GSgrow-a uses the INSgrow [49] algorithm.

5.4 Efficiency

To verify the mining performance, we conduct experiments on six protein databases, SDB1 to SDB6. The parameter metrics are *gap* = [0, 3] and *minsup* = 50, and five competitive algorithms are selected: NTP-c, NTP-bf, NTP-df, GSgrow-a, and NOSEP-a. The evaluation metrics include running time, number of frequent patterns and number of candidate patterns, which are shown in Figures 4–6, respectively. The more the mined frequent patterns are and the shorter the running time is, the better the efficiency of the algorithm is. The less the candidate patterns are, the fewer the support calculations are and the shorter the running time is.

The following observations can be made from the following results:

(1) NTP-Miner outperforms NTP-c and NOSEP-a. Figures 5 and 6 show that NTP-Miner, NTP-c and NOSEP-a find the same number of frequent NTPs with the same number of candidate



	SDB1	SDB2	SDB3	SDB4	SDB5	SDB6
NTP-c	15,599	8,459	4,188	21,445	10,889	4,583
NTP-bf	24,905	14,025	5,304	32,130	18,190	6,069
NTP-df	24,905	14,025	5,304	32,130	18,190	6,069
GSgrow-a	24,769	13,889	5,270	32,028	18,037	6,001
NOSEP-a	15,599	8,459	4,188	21,445	10,889	4,583
NTP-Miner	15,599	8,459	4,188	21,445	10,889	4,583

Fig. 6. Comparison of number of candidate patterns.

patterns, while NTP-Miner runs faster than both NTP-c and NOSEP-a. For example, for SDB5, NTP-Miner, NTP-c, and NOSEP-a mine 1,070 frequent NTPs according to Figure 5, and the candidate patterns are 10,889 in Figure 6. However, the time cost for NTP-Miner is 66.53 s, while for NTP-c and NOSEP-a it is 479.89 and 913.30 in Figure 4, respectively. The same results are found for the other datasets. The reasons for this are as follows: (i) Since Sim-NAP can find the same number of occurrences as NAP and NETGAP. NTP-Miner, NTP-c and NOSEP-a employ Sim-NAP, NAP and NETGAP, repectively, to calculate the support. Therefore, the three algorithms mine the same number of frequent NTPs. (ii) NTP-Miner, NTP-c and NOSEP-a apply the same candidate pattern reduction strategy, and therefore obtain the same number of candidate patterns. (iii) Since NETGAP needs to create a whole Nettree, and finds and prunes invalid nodes only after finding an occurrence. Meanwhile, the time complexity of NETGAP is $O(m \times n \times W \times W)$, where m, n, and W are the pattern length, sequence length, and width of gap b–a+1, respectively [7]. The NAP creates a whole Nettree and employs depth-first and backtracking strategies to find nonoverlapping occurrences within the Nettree. Since the Nettree has only m levels, each level has at most n nodes, and each node has at most W parents, it is easy to show that the space and time complexities of NAP are both $O(m \times n \times W)$. However, the space and time complexities of Sim-NAP is $O(m \times n)$. Thus, Sim-NAP is more effective than NAP, and NAP is more effective than NETGAP. Therefore, Sim-NAP is more effective than both NAP and NETGAP. Hence, NTP-Miner outperforms both NTP-c and NOSEP-a.

(2) NTP-Miner outperforms NTP-bf and NTP-df. Figure 5 shows that the NTP-bf, NTP-df and NTP-Miner algorithms mine the same number of frequent NTPs, while NTP-Miner is faster than NTP-bf and NTP-df in Figure 4. The explanation for this is as follows. Firstly, all three algorithms use the Sim-NAP algorithm to calculate the support, but they use different strategies to generate candidate patterns. As shown in the analysis in Section 4.2, the pattern join strategy outperforms the breadth-first and depth-first strategies. NTP-Miner, NTP-bf, and NTP-df use pattern join, breadth-first and depth-first strategies, respectively, to generate candidate patterns, meaning that NTP-Miner calculates fewer candidate patterns than NTP-bf and NTP-df. For example, for SDB1, NTP-Miner calculates 15,599 candidate patterns, while NTP-bf and NTP-df calculate 24,905. Hence, NTP-Miner outperforms both NTP-bf and NTP-df.

(3) The performance of NTP-Miner is better than that of GSgrow-a. From Figure 5, we can see that NTP-Miner mines more frequent NTPs than GSgrow-a, while Figure 4 shows that

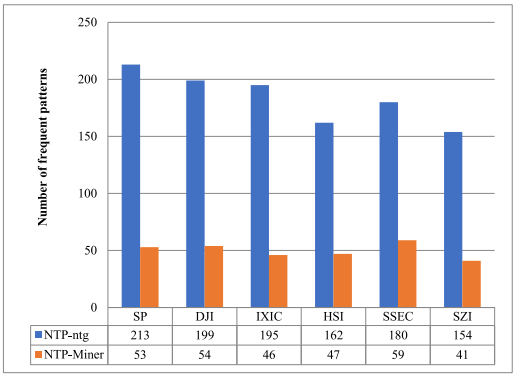


Fig. 7. Comparison of the number of patterns found using traditional and tri-wildcard gap constraints.

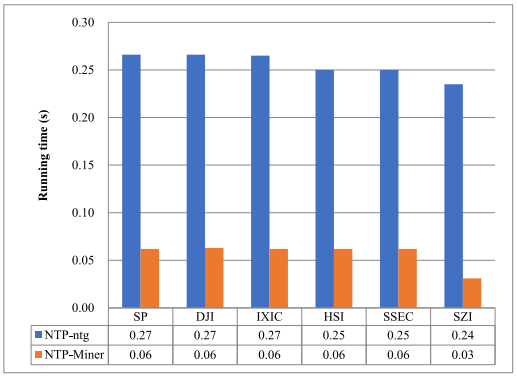


Fig. 8. Comparison of running time for traditional and tri-wildcard gap constraints.

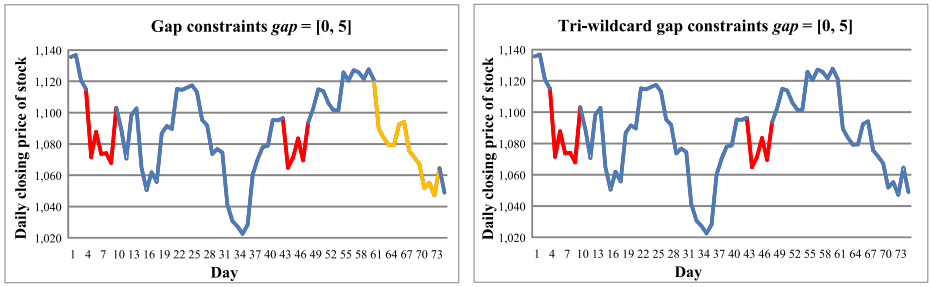
NTP-Miner has shorter running time than GSgrow-a. For example, it takes 202.56 s for NTP-Miner to mine 1,890 frequent NTPs from SDB4, whereas it takes 574.61 s for GSgrow-a to mine 1,884. Therefore, NTP-Miner outperforms GSgrow-a.

In summary, NTP-Miner has better performance than all comparable algorithms.

5.5 Performance of Tri-Wildcard Gap Constraints

To illustrate the performance of the tri-wildcard gap constraints, NTP-ntg is applied to mine the tri-patterns with traditional gap constraints. We carry out the experiments on six stock datasets (SP, DJI, IXIC, HSI, SSEC, and SZI). The parameter metrics are gap = [0, 5] and $minsup$ = 20. The evaluation metrics include the number of frequent patterns and running time, which are shown in Figures 7 and 8, respectively. To further illustrate the difference, we also select partial time series segments of SP dataset. The comparison of the occurrences of $\mathbf{p}$ = jIiJ with traditional and tri-wildcard gap constraints is shown in Figure 9. The closer the volatility trends of all occurrence are, the more accurate the mining patterns are.

NTP-Miner outperforms NTP-ntg, since it finds fewer patterns and takes less time. For example, Figures 7 and 8 show that it takes NTP-Miner 0.06 s to mine 53 patterns from SP, while it takes NTP-ntg 0.27 s to mine 213. The reason for this is that the tri-wildcard gap constraints match only medium or weak characters, while gap constraints match all characters. As demonstrated in the analysis in Example 1, some occurrences are illegal under the tri-wildcard gap constraints, but legal



(a) Occurrences with traditional gap constraints (b) Occurrences with tri-wildcard gap constraints

Fig. 9. Comparison of occurrences of **p** = jIiJ in SP.

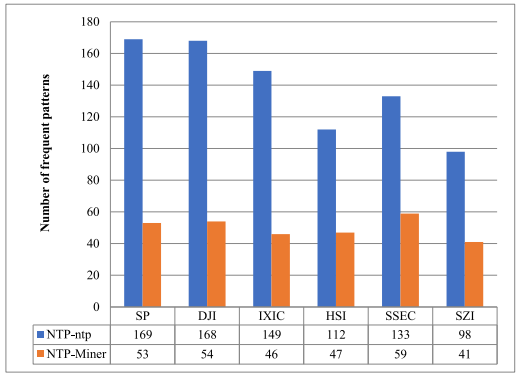


Fig. 10. Comparison of number of patterns on time series.

under the traditional gap constraints. Consequently, fewer occurrences are found for a tri-pattern with tri-wildcard gap constraints than with traditional gap constraints. Figure 9 also illustrates this phenomenon, and we can see that the fluctuation in the yellow section is inconsistent with those in the red sections, producing pattern occurrences that are inconsistent with the fluctuation trends of the pattern. Hence, NTP-Miner can find more accurate patterns than NTP-ntg.

5.6 NTP Performance

To demonstrate the performance of the tri-pattern, NTP-ntp is used to mine frequent patterns with tri-wildcard gap constraints. We conduct experiments on six stock datasets (SP, DJI, IXIC, HSI, SSEC, and SZI). The parameter metrics are *gap* = [0, 5] and *minsup* = 20. We use the number of patterns and the wordcloud map [52] indicators to measure the performance of the tri-pattern. Figures 10 and 11 show the number of patterns and the wordcloud map. The larger the proportion of strong and medium characters in the wordcloud map, the better the algorithm meets the user's needs.

NTP-Miner outperforms NTP-ntp, since it finds fewer patterns. For example, Figure 10 shows that NTP-ntp mines 169 patterns from SP, while NTP-Miner mines 53. This is because NTP-Miner focuses on finding patterns that are composed of strong or medium characters, while NTP-ntp finds patterns composed of all characters. From Figure 11(a) , it can be seen that many of the patterns are composed of weak characters, such as "A", "B", "CC", "BC", and "bC", which are valueless patterns, since "A", "a", "B", "b", "C", "c", "D", and "d" are weak characters that represent weak fluctuations. Figure 11(b) shows that all of the patterns found are composed of strong or medium characters,

(a) Wordcloud map of NTP-ntp

(b) Wordcloud map of NTP-Miner

Fig. 11. Comparison of wordcloud maps on SP dataset.

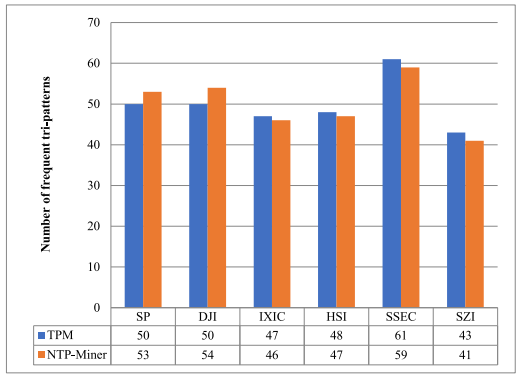


Fig. 12. Comparison of number of tri-patterns on time series.

and are of value to the user, since in practical applications, users are more interested in larger fluctuations. Hence, NTP-Miner can meet the user's needs more easily than NTP-ntp.

5.7 Performance for Nonoverlapping Condition

To compare the mining performance between no-condition and the nonoverlapping condition, we select six time series, SP, DJI, IXIC, HSI, SSEC, and SZI. The parameter metrics are $gap = [0, 5]$ and $minsup = 20$ under nonoverlapping condition. Under no-condition, we adjust the minimum support threshold minsup so that the number of frequent tri-patterns mined under the two conditions are very close. We use the number of frequent patterns and running time to evaluate the mining performance between no-condition and the nonoverlapping condition. Figures 12 and 13 show the number of frequent patterns and running time, respectively. The more the mined frequent patterns are and the shorter the running time is, the better the efficiency of the algorithm is. Table 2 illustrates the different tri-patterns found under no-condition and the nonoverlapping condition. The larger the proportion of strong characters in the patterns is, the better the algorithm meets the user's needs.

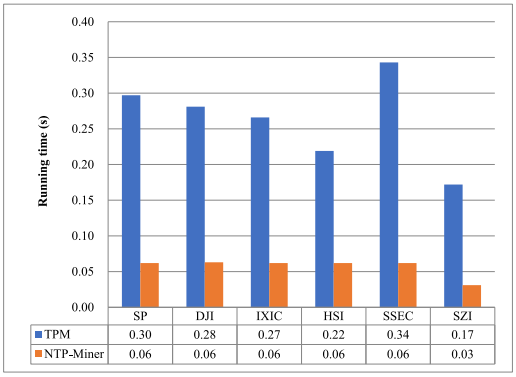


Fig. 13. Comparison of running time on time series.

Table 2. Comparison of Different Tri-Patterns Found under No-Condition and Nonoverlapping Condition

Dataset	Algorithm	Frequent tri-patterns
SP	TPM	FG Ff Fg Gf eF gG
	NTP-Miner	**HI IH Ii Jj** ee g**I hH iJ ij**
DJI	TPM	EF FF **J**E fG
	NTP-Miner	**H**F **Hj IJ Ie Ii Ij JI ij**
IXIC	TPM	GE Ge Gf **H**G gF
	NTP-Miner	**Hh II** If **Ij**
HSI	TPM	Ef Fe GE Gg ef **j**F
	NTP-Miner	**HH II Jj hI iI jI**
SSEC	TPM	Eg E**i I**e eE ef g**I** gg
	NTP-Miner	**IH hi ih ii** FFF
SZI	TPM	E**I** Ef FE Fg GF **I**g ef fF gg
	NTP-Miner	**Hi I**F **IH Ii JI Ji iH**

NTP-Miner outperforms TPM, since the number of frequent tri-patterns mined under the two conditions are very similar in Figure 12, and NTP-Miner runs faster than TPM in Figure 13. For example, it can be seen from Figure 12 that for HSI, NTP-Miner, and TPM mine 47 and 48 tri-patterns, respectively. However, the time cost for NTP-Miner is 0.06 s, as compared with 0.22 s for TPM in Figure 13. Similar results can be found on the other datasets. This is because NTP-Miner calculates the support under the nonoverlapping condition, and does not need to add characters to discover all feasible patterns, while TPM calculates the support of a pattern under no-condition, and adds unmatchable characters to the end of the sequence to meet the Apriori property [7]. In addition, we can see from Table 2 that the bold characters are strong characters, and the remainder are medium characters. Table 2 shows that under the nonoverlapping condition, the tri-patterns are mostly composed of strong characters with a large fluctuation trend, while under no-condition, the tri-patterns are mostly composed of medium characters. Users are more interested in tri-patterns with a large fluctuation trend, meaning that the mining performance under the nonoverlapping condition is better than no-condition. Hence, NTP-Miner not only runs faster than TPM, but can also mine patterns that are more valuable.

Table 3. Comparison of the Clustering Performance

	Algorithm	NMI	h
Car	MNOSEP	0.54	0.45
	MTPM	0.65	0.52
	MNTP-Miner	**0.74**	**0.66**
Worms	MNOSEP	0.61	0.50
	MTPM	0.65	0.55
	MNTP-Miner	**0.80**	**0.71**

5.8 Case Study

To further demonstrate the utility of the proposed NTP-Miner algorithm, we carry out a clustering experiment on Car and Worms datasets. For each dataset, we process it as follows:

(1) MNOSEP is used to mine the maximal frequent patterns under the nonoverlapping condition.
(2) MTPM is adopted to mine the maximal tri-patterns under no-conditon.
(3) MNTP-Miner is employed to mine the maximal NTPs.
(4) The maximal frequent patterns, tri-patterns, under no-condition NTPs and their supports are recorded, respectively.
(5) The k-means [53] method is applied to cluster the mining data, respectively.
(6) To evaluate the clustering performance, we select two criteria: **Normalized Mutual Information** (***NMI***) [54] and Homogeneity (h) [55], which can be calculated by Equations (1) and (2), respectively. *NMI* and h reflect the similarity between the clustering results and the actual values. The greater the *NMI* and h are, the more similar the clustering results are to the actual values.

$$NMI(X,Y) = \frac{\sum_{i=1}^{|X|}\sum_{j=1}^{|Y|} P(i,j)log(\frac{P(i,j)}{P(i)P(j)})}{\sqrt{\sum_{i=1}^{|X|} P(i)logP(i)\sum_{j=1}^{|Y|} P(j)logP(j)}}, \tag{1}$$

$$h(X,Y) = 1 - \frac{-\sum_{i=1}^{|X|}\sum_{j=1}^{|Y|} P(i,j)logP(i|j)}{-\sum_{i=1}^{|X|} P(i)logP(i)}. \tag{2}$$

The experiments are conducted on $gap = [0, 3]$. The results are shown in Table 3.

As shown in Table 3, NTPs can improve the clustering performance. For example, *NMI* and h of the NOSEP clustering result are 0.54 and 0.45 on Car, respectively, those of TPM are 0.65 and 0.52, respectively, while those of NTP-Miner are 0.74 and 0.66, respectively. Both evaluation metrics show that NTP-Miner can improve the clustering performance. The reasons are as follows. NOSEP does not consider the attributes that users are interested in, which gives rise to a large number of redundant patterns being mined. Since TPM considers the attributes that users are interested in, the clustering performance of TPM is better than that of NOSEP. As found in the analysis of Section 5.7, the patterns mined by NTP-Miner are more valuable than those mined by TPM. Hence, the clustering performance of NTP-Miner is better than TPM. Hence, NTP mining provides better feature extraction capability for clustering.

6 CONCLUSION

To mine patterns that users are interested in more accurately, the NTP mining is studied, which partitions all characters into three interest levels: strong, medium, and weak. The patterns are

composed of strong and medium characters. The gaps are matched with medium and weak characters. Therefore, NTP mining can avoid mining redundant patterns and the NTPs can avoid serious deviations. To effectively mine NTPs, the NTP-Miner algorithm is proposed, which involves two major steps: calculating the support of each pattern, and generating candidate patterns. In the first stage, we propose the Sim-NAP algorithm, which uses the depth-first search and backtracking strategies to find all nonoverlapping occurrences without creating a whole Nettree. Since NTP mining satisfies the Apriori property, a pattern join strategy is applied to generate candidate patterns. NTP-Miner has lower space and time complexities than the state-of-the-art algorithms. Experimental results on stock market and protein datasets validate that NTP-Miner not only is more efficient than other competitive approaches, but can also help users find more valuable patterns. More importantly, NTP mining has achieved better performance in clustering tasks.

REFERENCES

[1] Wensheng Gan, J. C.-W. Lin, P. Fournier-Viger, H.-C. Chao, and P. S. Yu. 2020. Huopm: High-utility occupancy pattern mining. *IEEE Transaction on Cybernetics* 50, 3 (2020), 1195–1208.

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

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

[4] Philippe Fournier-Viger, J. C-W. Lin, T. Truong-Chi, and R. Nkambou. 2019. A survey of high utility itemset mining. *High-Utility Pattern Mining. Studies in Big Data* 51, 1 (2019), 1–45.

[5] Philippe Fournier-Viger, P. Yang, R. U. Kiran, S. Ventura, and J. M. Luna. 2021. Mining local periodic patterns in a discrete sequence. *Information Sciences* 544 (2021), 519–548.

[6] Fan Min, Z.-H. Zhang, W.-J. Zhai, and R.-P. Shen. 2020. Frequent pattern discovery with tri-partition alphabets. *Information Sciences* 507 (2020), 715–732.

[7] Youxi Wu, Y. Tong, X. Zhu, and X. Wu. 2018. Nosep: Nonoverlapping sequence pattern mining with gap constraints. *IEEE Trans. Cybern.* 48, 10 (2018), 2809–2822.

[8] Philippe Fournier-Viger, J. Li, J. C.-W. Lin, T. T. Chi, and R. U. Kiran. 2020. Mining cost-effective patterns in event logs. *Knowledge-Based Systems* 191 (2020), 105241.

[9] Zhaoyu Shou, Y. Wang, Y. Wen, and H. Zhang. 2020. Knowledge point recommendation algorithm based on enhanced correction factor and weighted sequential pattern mining. *International Journal of Performability Engineering* 16, 4 (2020), 549–559.

[10] Xiangjun Dong, P. Qiu, J. Lu, L. Cao, and T. Xu. 2019. Mining top-k useful negative sequential patterns via learning. *IEEE Transactions on Neural Networks and Learning Systems* 30, 9 (2019), 2764–2778.

[11] Johannes D. Smedt, G. Deeva, and J. D. Weerdt. 2020. Mining behavioral sequence constraints for classification. *IEEE Transactions on Knowledge and Data Engineering* 32, 6 (2020), 1130–1142.

[12] Kui Yu, W. Ding, D. A. Simovici, H. Wang, J. Pei, and X. Wu. 2015. Classification with streaming features: An emerging-pattern mining approach. *ACM Transaction on Knowledge Discovery from Data* 9, 4 (2015), 1–31.

[13] Udi Manber and R. Baeza-Yates. 1991. An algorithm for string matching with a sequence of don't cares. *Information Processing Letters* 37, 3 (1919), 133–136.

[14] Chao Gao, L. Duan, G. Dong, H. Zhang, H. Yang, and C. Tang. 2016. Mining top-k distinguishing sequential patterns with flexible gap constraints. In *Proceedings of the International Conference on Web-Age Information Management*, Springer International Publishing. 82–94.

[15] Xindong Wu, J. Qiang, and F. Xie. 2014. Pattern matching with flexible wildcards. *Journal of Computer Science and Technology* 29, 5 (2014), 740–750.

[16] Youxi Wu, L. Wang, J. Ren, W. Ding, and X. Wu. 2014. Mining sequential patterns with periodic wildcard gaps. *Applied Intelligence* 41, 1 (2014), 99–116.

[17] Youxi Wu, M. Geng, Y. Li, L. Guo, Z. Li, P. Fournier-Viger, X. Zhu, and X. Wu. 2021. HANP-Miner: High average utility nonoverlapping sequential pattern mining. *Knowledge-Based Systems.* 229 (2021), 107361.

[18] Youxi Wu, X. Liu, W. Yan, L. Guo, and X. Wu. 2021. Efficient solving algorithm for strict pattern matching under nonoverlapping condition. *Journal of Software.* DOI: 10.13328/j.cnki.jos.006054

[19] Youxi Wu, C. Zhu, Y. Li, L. Guo, and X. Wu. 2020. Netncsp: Nonoverlapping closed sequential pattern mining. *Knowledge-Based Systems* 196 (2020), 105812.

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

[21] Fei Xie, X. Wu, and X. Zhu. 2017. *Efficient sequential pattern mining with wildcards for keyphrase extraction. Knowledge-Based Systems* 115 (2017), 27–39.
[22] Yiyu Yao. 2010. Three-way decisions with probabilistic rough sets. *Information Sciences* 180, 3 (2010), 341–353.
[23] Jianming Zhan, J. Ye, W. Ding, and P. Liu. 2021. A novel three-way decision model based on utility theory in incomplete fuzzy decision systems. *IEEE Transactions on Fuzzy Systems*.
[24] Shuhui Cheng, Y. Wu, Y. Li, F. Yao, and F. Min. 2021. TWD-SFNN: Three-way decisions with a single hidden layer feedforward neural network. *Information Sciences* 579 (2021), 15–32.
[25] Zhongyu Zhou and D. Pi. 2019. Mining method of minimal rare pattern oriented to satellite telemetry data stream. *Chinese Journal of Computers Journal of Software*, 1351–1366. DOI: 10.13328/j.cnki.jos.006054
[26] Unil Yun, G. Lee, and E. Yoon. 2019. Advanced approach of sliding window based erasable pattern mining with list structure of industrial fields. *Information Sciences* 494 (2019), 37–59.
[27] Lei Zhang, P. Luo, L. Tang, E. Chen, Q. Liu, M. Wang, and H. Xiong. 2015. Occupancy-based frequent pattern mining*. *ACM Transaction on Knowledge Discovery from Data* 10, 2 (2015), 1–33.
[28] Xiangjun Dong, Y. Gong, and L. Cao. 2020. E-rnsp: An efficient method for mining repetition negative sequential patterns. *IEEE Transactions on Cybernetics* 50, 5 (2020), 2084–2096.
[29] Tin Truong, H. Duong, B. Le, and P. Fournier-Viger. 2019. Fmaxclohusm: An efficient algorithm for mining frequent closed and maximal high utility sequences. *Engineering Applications of Artificial Intelligence* 85 (2019), 1–20.
[30] Youxi Wu, Y. Wang, Y. Li, X. Zhu, and X. Wu. 2021. Top-k self-adaptive contrast sequential pattern mining. *IEEE Transactions on Cybernetics*. 1–15. DOI: 10.1109/TCYB.2021.3082114
[31] Jaysawal B. Prasad and J.-W. Huang. 2018. Psp-ams: Progressive mining of sequential patterns across multiple streams. *ACM Transaction on Knowledge Discovery from Data* 13, 1 (2018), 1–23.
[32] Lizhen Wang, X. Bao, and L. Zhou. 2018. Redundancy reduction for prevalent co-location patterns. *IEEE Transactions on Knowledge and Data Engineering* 30, 1 (2018), 142–155.
[33] Philippe Fournier-Viger, Y. Zhang, J. C.-W. Lin, D.-T. Dinh, and H. B. Le. 2020. Mining correlated high-utility itemsets using various measures. *Logic Journal of the IGPL* 28, 1 (2020), 19–32.
[34] Wei Song, Y. Liu, and J. Li. 2013. Mining high utility itemsets by dynamically pruning the tree structure. *Applied Intelligence* 40, 1 (2013), 29–43.
[35] Youxi Wu, L. Wang, J. Ren, W. Ding, and X. Wu. 2014 . Mining sequential patterns with periodic wildcard gaps. *Applied Intelligence* 41, 1 (2014), 99–116.
[36] Wei Song, B. Jiang, and Y. Qiao. 2018. Mining multi-relational high utility itemsets from star schemas. *Intelligent Data Analysis* 22, 1 (2018), 143–165.
[37] Jerry C.-W. Lin, T. Li, M. Pirouz, J. Zhang, and P. Fournier-Viger. 2019. High average-utility sequential pattern mining based on uncertain databases. *Knowledge and Information Systems* 62, 3 (2019), 1199–1228.
[38] S. Vincent, Tseng, Bai-En, C. W. Shie, P. S. Wu, and Yu. 2013. Efficient algorithms for mining high utility itemsets from transactional databases. *IEEE Transactions on Knowledge and Data Engineering* 25, 8 (2013), 1772–1786.
[39] Tin Truong, H. Duong, B. Le, and P. Fournier-Viger. 2019. Efficient vertical mining of high average-utility itemsets based on novel upper-bounds. *IEEE Transactions on Knowledge and Data Engineering* 31, 2 (2019), 301–314.
[40] C. F. M. M. Rahman, K. S. Ahmed, and Leung. 2018. Mining weighted frequent sequences in uncertain databases. *Information Sciences* 479 (2018), 76–100.
[41] Hyoju Nam, U. Yun, E. Yoon, and J. C-W. Lin. 2020. Efficient approach for incremental weighted erasable pattern mining with list structure. *Expert Systems with Applications* 143, 4 (2020), 113087.
[42] Unil Yun and K. H. Ryu. 2013. Efficient mining of maximal correlated weight frequent patterns. *Intelligent Data Analysis* 17, 5 (2013), 917–939.
[43] Bilong Shen, Z. Wen, Y. Zhao, D. Zhou, and W. Zheng. 2016. OCEAN: Fast discovery of high utility occupancy itemsets. In *Proceedings of the Pacific-Asia Conference on Knowledge Discovery and Data Mining*. 354–365
[44] Chao-Dong Tan, F. Min, M. Wang, H. R. Zhang, and Z. H. Zhang. 2016. Discovering patterns with weak-wildcard gaps. *IEEE Access* 4 (2016), 4922–4932.
[45] Yu Fang, C. Gao, and Y. Yao. 2020. Granularity-driven sequential three-way decisions: A cost-sensitive approach to classification. *Information Sciences* 507, 1 (2020), 644–664.
[46] Huiting Liu, Z. Liu, H. Huang, and X. Wu. 2018. Sequential pattern matching with general gaps and one-off condition. *Journal of Software* 2 (2018), 363–382.
[47] Youxi Wu, R. Lei, Y. Li, L. Guo, and X. Wu. 2021. Haop-miner: Self-adaptive high-average utility one-off sequential pattern mining. *Expert Systems with Applications* 184 (2021), 115449.
[48] Youxi Wu, X. Wang, Y. Li, L. Guo, Z. Li, J. Zhang, and X. Wu. 2021. OWSP-Miner: Self-adaptive one-off weak-gap strong pattern mining. *ACM Transactions on Management Information Systems*. DOI: 10.1145/3476247
[49] Bolin Ding, D. Lo, J. Han, and S. C. Khoo. 2009. Efficient mining of closed repetitive gapped subsequences from a sequence database. In *Proceedings of the International Conference on Data Engineering*. 1024–1035.

[50] Qiaoshuo Shi, J. Shan, W. Yan, Y. Wu, and X. Wu. 2020. Netnpg: Nonoverlapping pattern matching with general gap constraints. *Appl Intell* 50, 6 (2020), 1832–1845.
[51] Youxi Wu, C. Shen, H. Jiang, and X. Wu. 2016. Strict pattern matching under non-overlapping condition. *Science China Information Sciences* 60, 1 (2016), 5–20.
[52] Florian Heimerl, S. Lohmann, S. Lange, and T. Ertl. 2014. Word cloud explorer: Text analytics based on word clouds. In *Proceedings of the Hawaii International Conference on System Sciences*. 1833–1842.
[53] Marco Capo, A. Perez, and J. A. Antonio. Lozano. 2020. An efficient split-merge re-start for the *k*-means algorithm. *IEEE Transactions on Knowledge and Data Engineering*. DOI: 10.1109/TKDE.2020.3002926
[54] Leon Danon, A. Díaz-Guilera, J. Duch, and A. Arenas. 2005. Comparing community structure identification. *Journal of Statistical Mechanics Theory and Experimen* 2005, 09 (2005), P09008–P09008.
[55] Andrew Rosenberg and J. Hirschberg. 2007. V-measure: A. Conditional entropy-based external cluster evaluation measure. In *Proceedings of the Empirical Methods in Natural Language Processing and Computational Natural Language Learning*. 410–420.

Received Februray 2021; revised June 2021; accepted August 2021