Applied Intelligence (2022) 52:6646–6661
https://doi.org/10.1007/s10489-021-02763-y

Self-adaptive nonoverlapping sequential pattern mining

Yuehua Wang[1] · Youxi Wu[2,3,4] · Yan Li[1] · Fang Yao[3] · Philippe Fournier-Viger[5] · Xindong Wu[6,7]

Accepted: 9 August 2021 / Published online: 14 September 2021

Abstract
Repetitive sequential pattern mining (SPM) with gap constraints is a data analysis task that consists of identifying patterns (subsequences) appearing many times in a discrete sequence of symbols or events. By using gap constraints, the user can filter many meaningless patterns, and focus on those that are the most interesting for his needs. However, it is difficult to set appropriate gap constraints without prior knowledge. Hence, users generally find suitable constraints by trial and error, which is time-consuming. Besides, current algorithms are inefficient as they repeatedly check whether the gap constraints are satisfied. To address these problems, this paper presents a complete algorithm called SNP-Miner that has two key phases: candidate pattern generation and support (number of occurrences or occurrence frequency) calculation. To reduce the number of candidate patterns, SNP-Miner employs a pattern join strategy. Moreover, to efficiently calculate the support, SNP-Miner uses an incomplete Nettree structure stored in an array, and scans the structure once to avoid redundant calculations and reduce the time complexity. Experimental results show that SNP-Miner not only outperforms competitive algorithms, but can also discover more valuable patterns without user-predefined gap constraints. Algorithms and data can be downloaded from https://github.com/wuc567/Pattern-Mining/tree/master/SNP-Miner.

Keywords Sequential pattern mining · Frequent pattern · Self-adaptive · Gap constraint · Nonoverlapping condition

1 Introduction

Sequential pattern mining (SPM) [1–3] aims to find subsequences that appear frequently in discrete sequences of symbols or events. A subsequence is frequent if its number of occurrences, called support, is no less than a user-defined minimum support threshold. SPM plays an important role in sequence analysis [4], information extraction [5, 6], dynamic stream data analysis [7, 8], natural language processing [9], transportation systems integration [10] and so on. However, most SPM methods [11, 12] only focus on whether a pattern appears in each sequence and ignore repetitive occurrences [13] of the pattern within a sequence. For instance, suppose we have two sequences $\mathbf{s}_1$ and $\mathbf{s}_2$, which are "abc" and "abcabcabc", respectively. Pattern $\mathbf{p}$=abc appears once in $\mathbf{s}_1$ and three times in $\mathbf{s}_2$. If these repetitions are ignored, pattern $\mathbf{p}$ occurs in sequences $\mathbf{s}_1$ and $\mathbf{s}_2$ with a support (number of occurrences or occurrence frequency) of one. In practice, these repetitive occurrences [14] are not only able to capture the repetition of a pattern in different sequences but also to record repetitions within a single sequence, which can be useful to identify valuable information in sequences such as virus, DNA, and protein sequences.

To provide more flexibility to users in selecting patterns, the concept of gap constraint was introduced into repetitive SPM [15, 16]. Using gap constraints makes it possible to filter out patterns that have large gaps and are often uninteresting, thus finding a smaller but more relevant set of patterns. SPM with gap constraints has been widely used in medical insurance fraud detection [17, 18], pattern analysis [19], market basket analysis [20, 21], and disease diagnosis [22, 23]. However, since there are no restrictions on the number of characters in an input sequence, the number of occurrences of a pattern can increase exponentially with its length [24]. To address this problem, Ding et al. [25] introduced a nonoverlapping condition that allows a given sequence character to be rematched by characters at different positions in a pattern. Wu et al. [19, 26] then showed that an efficient and complete algorithm can be designed for nonoverlapping SPM, since the Apriori (anti-monotonicity) property is satisfied. Example 1 is used to illustrate the idea of nonoverlapping support.

✉ Youxi Wu
wuc567@163.com

Extended author information available on the last page of the article.

Example 1 Consider a sequence **s**=TATAA and a pattern **p**= T[0,2]A[0,2]A. The pattern can be interpreted as that there are 0 to 2 characters between "T" and "A", then followed by another "A" after 0 to 2 characters. All occurrences of this pattern are shown in Fig. 1.

In Fig. 1, each occurrence is represented as a list of sequence positions. For example, the first occurrence of **p** in **s** is denoted as $< 1, 2, 4 >$, which indicates that the first, second and fourth characters of **s** are matched by the first, second and third characters of **p**, respectively. The other occurrences of pattern **p** in **s** are $< 1, 2, 5 >$, $< 1, 4, 5 >$ and $< 3, 4, 5 >$. Let p_1, p_2, and p_3 denote the three characters of pattern **p**, and s_1, s_2, s_3, s_4, and s_5 denote the characters of sequence **s**. Although occurrences $< 1, 2, 4 >$ and $< 3, 4, 5 >$ contain the position s_4, these occurrences are nonoverlapping, since they are matched by p_3 and p_2, respectively. On the other hand, the occurrences $< 1, 2, 4 >$ and $< 1, 2, 5 >$ are said to be overlapping since p_1 and p_2 match with the same positions in **s** (s_1 and s_2, respectively). Hence, the nonoverlapping support (number of occurrences or occurrence frequency) of **p** is two.

In recent years, many researchers have found that nonoverlapping SPM makes it easier to find interesting patterns than traditional SPM methods [27, 28]. But, unfortunately, most nonoverlapping SPM algorithms require the users to set gap constraints in advance [29, 30], which is difficult for users without sufficient knowledge about a dataset. Hence, users often repeatedly run an algorithm with different constraints to finally find a suitable set of patterns, which is time-consuming. Moreover, if the gap constraints are not appropriate, valuable information may be missed. For example, in Example 1, if the gap constraint is [2,3], the pattern **p**=T[2,3]A[2,3]A has no occurrence.

To tackle this problem of finding a suitable gap constraint and to improve the performance of pattern discovery, this paper proposes an efficient algorithm which employs a self-adaptive gap to obtain frequent patterns. The main contributions of the paper are fourfold:

1. The problem of self-adaptive nonoverlapping sequential pattern (SNP) mining is defined, and a complete and efficient algorithm called SNP-Miner is presented.
2. Two compact data structures called a single-branch Nettree and an incomplete Nettree are designed to calculate the supports of patterns for SNP mining. More importantly, the positions of all of the characters in the sequences are stored in an array to avoid repeatedly scanning the database.
3. A pattern join strategy is adopted to reduce the search space. Based on this, SNP-Miner can discover SNPs using a self-adaptive gap via a single database scan.
4. Extensive experiments on both biological sequence and time series data show that SNP-Miner can discover more frequent patterns than gap constraint SPM methods, and that SNP-Miner is more efficient than the compared state-of-the-art algorithms.

The rest of this paper is organized as follows. Section 2 reviews related work, and Section 3 introduces relevant definitions. Section 4 presents the SNP-Miner algorithm and several optimization strategies. Section 5 reports results from a performance evaluation of the algorithm and a comparison with alternative approaches. Finally, Section 6 draws conclusions.

2 Related work

SPM [31–34] is a very active research topic, and plays an important role in real life applications, such as biological sequence analysis [35], big data analysis [36], event data analysis [37], information retrieval [38, 39], credit card fraud detection [40] and travel route recommendation [41]. Recently, a series of novel SPM have emerged to discover patterns with weight or high profit [42, 43], such as three-way SPM [44, 45], high-utility SPM [46, 47] and high average utility SPM [48, 49], and contrast SPM [50, 51]. Research on SPM can be classified from the perspectives of gap constraints, the repetition of subsequences, support calculating methods, pattern types, pattern mining methods, search space pruning strategies and so on.

From the perspective of gap constraints, SPM includes traditional SPM [52, 53], gap constraint SPM [38, 54], and self-adaptive gap SPM [50, 55, 56]. Although traditional SPM has made great progress in many fields, it does not consider repetitive occurrences [57, 58], and may therefore overlook some interesting patterns, especially in longer sequences. Gap constraint SPM [26] considers repetitive occurrences but requires that the user sets gap constraints

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

	1	2	3	4	5		
s =	T	A	T	A	A		Sequence **s**
	T	A		A		<1,2,4>	First occurrence of **p**
	T	A			A	<1,2,5>	Second occurrence of **p**
	T			A	A	<1,4,5>	Third occurrence of **p**
			T	A	A	<3,4,5>	Fourth occurrence of **p**

before mining. Although this method is flexible and can find a relatively large amount of information, it is difficult for the user to set reasonable gap constraints without prior knowledge about the data. If a gap constraint is too tight, valuable patterns may be missed. More importantly, in this mining method, constant checks must be made to determine if the gap constraints are satisfied. The running time is positively correlated with the gap size, and if the gap is too large, users may have to wait a very long time to obtain the results. To address these issues, a self-adaptive gap [35] was proposed, which can be represented using the "$*$" symbol. This self-adaptive gap is generated based on the characteristics of a sequence. For instance, in Example 1, the occurrence $< 1, 2, 5 >$ does not match the gap constraint pattern "T[0,1]A[0,1]A", although it matches the pattern "T$*$A$*$A", defined using a self-adaptive gap. Hence, a scheme based on a self-adaptive gap can find more valuable information.

From the perspective of the repetition of subsequences, SPM methods can be categorized as *repetition ignored* [11, 59, 60], *no-condition* [24, 61, 62], *one-off condition* [35, 56, 63, 64], and *nonoverlapping condition* [25–27] approaches. The 'repetition ignored' approach focuses on whether a pattern occurs, rather than counting how many times it occurs. 'No-condition' means that the same characters can be reused unlimited times. In Example 1, all the four occurrences of the pattern are found under no-condition. No-condition SPM does not satisfy the Apriori property [65], and therefore the search space needs to be enlarged to find all the patterns [13, 24]. Under the one-off condition, each sequence character can be used at most once. In Example 1, there is only one one-off occurrence, i.e. $< 1, 2, 4 >$. Although SPM under the one-off condition satisfies the Apriori property, it is an NP-hard problem [66]. Approximate algorithms therefore have been proposed, which means that some patterns may be missed. The nonoverlapping condition means that the same sequence characters can be matched at different positions for different occurrences. In Example 1, $< 1, 2, 4 >$ and $< 3, 4, 5 >$ are two nonoverlapping occurrences, since position s_4 matches p_3 and p_2, respectively.

It should be noticed that the overlapping condition can be seen as no-condition. For example, in Example 1, $< 1, 2, 4 >$, $< 1, 2, 5 >$ and $< 1, 4, 5 >$ are three overlapping occurrences, since s_1 is used multiple times by p_1 in the three occurrences. Similarly, all overlapping occurrences of pattern **p** in sequence **s** are $< 1, 2, 4 >$, $< 1, 2, 5 >$, $< 1, 4, 5 >$, and $< 3, 4, 5 >$, which are the same as no-condition occurrences. Therefore, overlapping SPM can be regarded as no-condition SPM. In Example 1, pattern "A" only has two occurrences in sequence **s**, i.e. $< 1 >$ and $< 3 >$, while its super-pattern "AT" has five overlapping occurrences, i.e. $< 1, 2 >$, $< 1, 4 >$, $< 1, 5 >$, $< 3, 4 >$, and $< 3, 5 >$. This shows that overlapping SPM does not satisfy the Apriori property. Fortunately, nonoverlapping SPM satisfies the Apriori property, and efficient algorithm can be developed to find a complete set of patterns. Since nonoverlapping SPM is neither as loose as no-condition SPM (overlapping SPM) nor as tight as one-off SPM, it therefore outperforms the repetition ignored, no-condition (overlapping) and one-off condition schemes [26].

From the perspective of support calculating methods, some classical SPM methods employ FP-Tree structure [4], list structure [48, 67] and Nettree structure [26]. FP-Tree is a tree structure containing one root labeled as "null", a set of item-prefix subtrees as the children of the root, and a frequent-item-header table, which is designed for mining patterns in transaction database. SPM methods using this structure do not need to generate candidate patterns, which are one-phase algorithms. However, the latter two structures are designed for mining patterns in sequence database, where list structure is a list of all input-sequence (sid) and event identifier (eid) pairs including the patterns, and Nettree structure is an extended tree structure with multiple roots and parents. SPM method using list and Nettree structure need to generate candidate patterns first and then calculate their supports, i.e. two-phase algorithms. However, FP-Tree and list structures only record whether the patterns appear, which are suitable for the 'repetition ignored' approach, while Nettree structure can keep specific number of occurrences of patterns. In other words, Nettree can be used for nonoverlapping condition scheme. Unfortunately, the above three structures are not suitable for self-adaptive gap SPM problem. We list the main differences between the above structures in Table 1.

Inspired by two prior studies [35] and [26], this paper proposes a novel solution for self-adaptive nonoverlapping sequential pattern mining. Although the schemes in both [35] and the present paper discover patterns using a self-adaptive gap, the mining conditions are different: the former employs the one-off condition, while this paper considers the nonoverlapping condition. The difference between the scheme in [26] and the one presented here is that the former requires a user-specified gap constraint, while the approach presented in this manuscript can identify a suitable gap constraint based on the characteristics of the sequence. In addition, our scheme has a significantly improved mining speed, since it employs the InNet algorithm to calculate the support, and does not need to create and prune invalid nodes, thus avoiding redundant calculations.

3 Problem definition

This section presents definitions and describes the problem of SNP mining.

Table 1 Comparison of FP-tree, list, nettree and incomplete nettree

	Repetition of subsequences	Structure	Candidate generation	Calculating method	Goal
FP-Tree [4]	Repetition ignored	One "null" root, a set of item-prefix subtrees as the children of the root, and a frequent-item-header table	No (one-phase algorithm)	The frequent patterns and their supports are obtained according to the conditional pattern base in FP-Tree.	Frequent patterns and support
List [67]	Repetition ignored	A list with pattern name, sid and eid	Yes (two-phase algorithm)	The number of sid/eid in list is the support of pattern **p** in database *D*.	Support
Nettree [26]	Nonoverlapping condition	An extended tree structure with multiple roots and parents	Yes (two-phase algorithm)	The number of root-leaf paths is the support of pattern **p** in sequence **s**.	Specific number of occurrences
Incompete Nettree (this paper)	Nonoverlapping condition	The last level of a single-branch Nettree, which can be stored in an array	Yes (two-phase algorithm)	The number of nodes in the *m*-th level is the support of pattern **p** in sequence **s**.	Specific number of occurrences

Definition 1 A sequence with length n can be written as $\mathbf{s}=s_1 s_2 \cdots s_n$, where $s_i (1 \leq i \leq n) \in E$. E is a collection of different characters, and $|E|$ is the size (number of different characters).

Definition 2 A self-adaptive pattern **p** with length m can be expressed as $\mathbf{p}=p_1 * p_2 * \cdots p_m$, where $*$ denotes any number of wildcard characters.

Definition 3 A list of positions $\mathbf{l}=< l_1, l_2, \cdots, l_m >$ denotes an occurrence of **p** in **s**, if and only if $p_j = s_{l_j}$ $(1 \leq j \leq m$ and $1 \leq l_j \leq n)$. Suppose that $\mathbf{l'}=< l'_1, l'_2, \cdots, l'_m >$ is another occurrence. Then, **l** and **l'** are two nonoverlapping occurrences if and only if $\forall 1 \leq j \leq m$ and $l_j \neq l'_j$. The maximum number of nonoverlapping occurrences of a self-adaptive pattern **p** in a sequence **s** is called its support and is denoted as $sup(\mathbf{p}, \mathbf{s})$.

Definition 4 If $sup(\mathbf{p}, \mathbf{s})$ is no less than the support threshold $minsup$, then pattern **p** is called an SNP.

The problem of SNP mining involves discovering all self-adaptive nonoverlapping patterns (SNPs) in some sequences. A pattern is an SNP if its support is no less than a minimum support threshold $minsup$.

Example 2 In Example 1, $n = 5$, $E = \{A, T\}$, and $|E| = 2$. For the self-adaptive pattern **p**=T∗A∗A and the support threshold *minsup*=2, the nonoverlapping occurrences of **p** in **s** are $< 1, 2, 4 >$ and $< 3, 4, 5 >$. Because $sup(\mathbf{p}, \mathbf{s}) = 2 \geq minsup$, **p** is an SNP. In sequence "TATAA", the SNPs are "A", "T", "AA", "TA", and "TAA".

The main symbols used in this paper are listed in Table 2.

4 Algorithm design

To mine SNPs effectively, this paper proposes a two-phase algorithm called SNP-Miner, since it involves two main tasks: candidate generation and support calculation. Section 4.1 introduces a candidate generation method based

Table 2 Notation

Symbol	Description
E	A set of characters $\{e_1, e_2, \cdots, e_t\}$
s	A sequence $s_1 s_2 \cdots s_n$ with length n
p	A pattern $p_1 * p_2 * \cdots p_m$ with length m
$sup(\mathbf{p}, \mathbf{s})$	The support of pattern **p** in sequence **s**
minsup	The support threshold
$\widehat{\mathcal{P}}_x$	The position set of character x
$\widehat{\mathcal{T}}_\mathbf{p}/\widehat{\mathcal{T}}_\mathbf{q}$	The incomplete Nettree of **p/q**

on pattern join strategy. Section 4.2 proposes two efficient structures for calculating the support of patterns. The pseudocode of SNP-Miner and a complexity analysis are presented in Section 4.3.

4.1 Candidate generation

Traditional SPM methods apply a breadth-first or depth-first search whose runtime and memory consumption could be very large. To address this problem, a pattern join strategy is employed, which reduces the number of redundant candidate patterns generated.

Pattern join strategy: Consider a pattern $\mathbf{p}=p_1p_2\cdots p_m$ and two characters a and b. If $\mathbf{q}=\mathbf{p}b$, then $\mathbf{p}$ is the prefix sub-pattern of $\mathbf{q}$, which is denoted as $prefix(\mathbf{q})=\mathbf{p}$, and $\mathbf{q}$ is a super-pattern of $\mathbf{p}$. Similarly, if $\mathbf{r}=a\mathbf{p}$, then $\mathbf{p}$ is a suffix sub-pattern of $\mathbf{r}$, denoted as $suffix(\mathbf{r})=\mathbf{p}$. If $prefix(\mathbf{q})=suffix(\mathbf{r})=\mathbf{p}$, then $\mathbf{q}$ and $\mathbf{r}$ can be joined into a super-pattern $\mathbf{t}$ with length $m+2$, where $\mathbf{t}=\mathbf{r}\oplus\mathbf{q}=a\mathbf{p}b$. The method for generating these super-patterns is called pattern join [68].

Theorem 1 *SNP satisfies the Apriori (anti-monotonicity) property. In other words, if a pattern is not an SNP, its super-patterns are not SNPs.*

Proof Suppose we have a pattern $\mathbf{q}$ with length m, and there are k nonoverlapping occurrences of pattern $\mathbf{q}$ in sequence $\mathbf{s}$, which are $< g_{1,1}, g_{1,2}, \cdots, g_{1,m} >$, $< g_{2,1}, g_{2,2}, \cdots, g_{2,m} >, \cdots, < g_{k,1}, g_{k,2}, \cdots, g_{k,m} >$, i.e. $sup(\mathbf{q}, \mathbf{s}) = k$. If $\mathbf{p}$ is the prefix pattern of $\mathbf{q}$, we can safely say that there are at least k nonoverlapping occurrences of $\mathbf{p}$ in $\mathbf{s}$, which are $< g_{1,1}, g_{1,2}, \cdots, g_{1,m-1} >$, $< g_{2,1}, g_{2,2}, \cdots, g_{2,m-1} >, \cdots, < g_{k,1}, g_{k,2}, \cdots, g_{k,m-1} >$, i.e. $sup(\mathbf{p}, \mathbf{s}) \geq k$. Similarly, if $\mathbf{p}$ is the suffix pattern of $\mathbf{q}$, $sup(\mathbf{p}, \mathbf{s})$ is no less than k. Thus, $sup(\mathbf{p}, \mathbf{s}) \geq sup(\mathbf{q}, \mathbf{s})$. Hence, SNP mining satisfies the anti-monotonicity. □

We give an example below to demonstrate that the pattern join strategy outperforms the breadth-first and depth-first strategies.

Example 3 Consider the sequence **s**=TATAAACC and *minsup*=2.

There are four SNPs with length two: {"A∗A", "A∗C", "T∗A", "T∗C"}. Using the breadth-first and depth-first strategies, there are 4×3=12 candidate patterns with length three, since each SNP with length two will generate three candidate patterns. For example, "A∗A" will generate "A∗A∗A", "A∗A∗C" and "A∗A∗T". Nevertheless, since "A∗T" is not an SNP, its super-pattern "A∗A∗T" is not an SNP according to Theorem 1. Hence, the pattern join strategy gives four candidate patterns: {"A∗A∗A", "A∗A∗C", "T∗A∗A", "T∗A∗C"}. This shows that the pattern join strategy is more effective than the breadth-first and depth-first strategies.

4.2 Support calculation

The FP-Tree and list structure only record whether patterns appear or not, which is difficult to obtain the exact number of patterns in a sequence. Although using Nettree can get the accurate result, it needs to create a large number of invalid nodes and parent-child relationships with a self-adaptive gap, which is time-consuming. To tackle the above problems and efficiently calculate the support with a self-adaptive gap, a compact data structure called a single-branch Nettree is designed. Then, based on this, a more efficient data structure called an incomplete Nettree is proposed, which can store the results of a sub-pattern to avoid redundant calculations.

4.2.1 Single-branch Nettree

The occurrences of a gap constraint pattern can be represented by a Nettree structure [26], which is an extended tree structure with multiple roots and parents. Nettree nodes with the same ID can appear multiple times. The notation n_j^i refers to the node with ID i on the j-th level of the Nettree. A path from a root node to a leaf node corresponds to an occurrence of a pattern in a sequence. NETLAP [19] is a complete method that employs a Nettree to obtain nonoverlapping occurrences, as illustrated in Example 4.

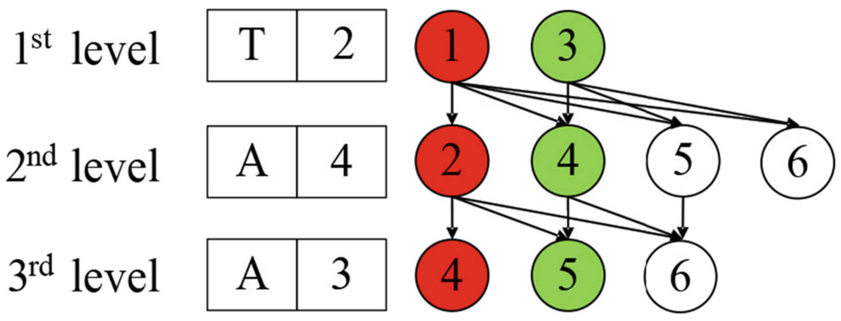


Fig. 2 A Nettree, in which each node may have many children and many parents. For example, node n_2^4 has two children, n_3^5 and n_3^6, and two parents, n_1^1 and n_1^3

Example 4 Given a sequence $\mathbf{s} = s_1s_2s_3s_4s_5s_6$ = TATAAA, a pattern $\mathbf{p}$ = T∗A∗A under the nonoverlapping condition, NETLAP creates the Nettree shown in Fig. 2. The sequence character s_4 is used as an example to illustrate the process of creating a Nettree. Node n_2^4 can be created since $s_4 = p_2$ =“A”, and it has two parents, n_1^1 and n_1^3, which satisfy the self-adaptive gap. Hence, the parent-child relationships between nodes n_1^1 and n_2^4, and n_1^3 and n_2^4 can be created. Similarly, leaf n_3^4 and the parent-child relationship between nodes n_2^2 and n_3^4 can be created. There are two nodes with ID 4, i.e. n_2^4 and n_3^4. In Fig. 2, the root-leaf paths $< n_1^1, n_2^2, n_3^4 >$ shown in red and $< n_1^3, n_2^4, n_3^5 >$ shown in green correspond to the two nonoverlapping occurrences of $< 1, 2, 4 >$ and $< 3, 4, 5 >$, respectively.

Although NETLAP can be used to solve the self-adaptive gap problem, this method is relatively inefficient, since it needs to create all of the nodes and parent-child relationships in the Nettree. Based on the above observation, we propose a single-branch Nettree structure which is stored in m arrays and more efficient.

Definition 5 A single-branch Nettree is similar to a Nettree, and consists of multiple roots. However, the nodes of a single-branch Nettree can have at most one parent or one child. That is, when $s_i = p_1$, the first-level node n_1^i can be created. When $s_i = p_j (j \geq 2)$, node n_j^i can be created if and only if the $(j-1)$-level node n_{j-1}^u has no child and $u < i$.

Example 5 demonstrates the principle of a single-branch Nettree.

Example 5 We use the same sequence and pattern as in Example 4. The corresponding single-branch Nettree is shown in Fig. 3. Since $s_1 = p_1$, node n_1^1 is created. Node n_2^2 is created as a child of n_1^1, since $s_2 = p_2$, node n_1^1 has no child and $1 < 2$. In the same way, n_1^3, n_2^4, n_3^4 and n_3^5 are created. Although $s_5 = p_2$, node n_2^5 cannot be created, since n_1^3 has a child, n_2^4. There are two nonoverlapping occurrences $< 1, 2, 4 >$ and $< 3, 4, 5 >$. Hence, the support of $\mathbf{p}$ in $\mathbf{s}$ is two.

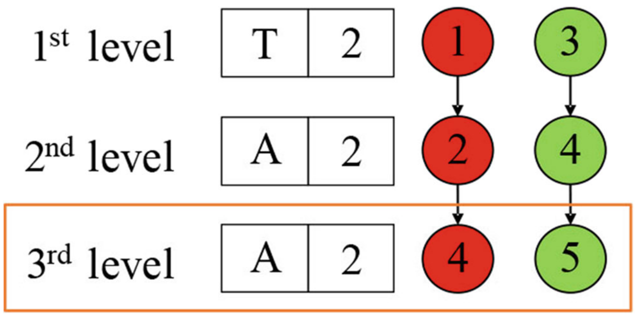


Fig. 3 A single-branch Nettree, in which each node has at most one child and one parent. For example, node n_2^2 has one child, n_3^4, and one parent, n_1^1

From Examples 4 and 5, it can be seen that the two schemes find the same nonoverlapping occurrences with a self-adaptive gap. However, a Nettree is more complex than the corresponding single-branch Nettree. For example, Nettree creates three invalid nodes n_2^5, n_2^6 and n_3^6, and nine invalid parent-child relationships, while the single-branch Nettree does not, which reduces the space and time complexities.

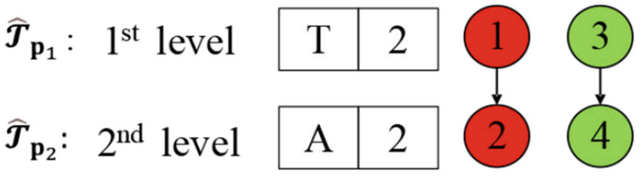
4.2.2 Incomplete Nettree structure

Compared with a Nettree, a single-branch Nettree is a more efficient structure. However, it still carries out redundant calculations to obtain the support of a pattern $\mathbf{p} \oplus x (x \in E)$, since the single-branch Nettree of sub-pattern $\mathbf{p}$ is created. Therefore, a new structure called an incomplete Nettree is proposed, which stores the results of sub-pattern $\mathbf{p}$ in an array. Using this structure, the support of the super-pattern $\mathbf{p} \oplus x$ can be obtained based on the sub-pattern $\mathbf{p}$.

Definition 6 The last level of a single-branch Nettree is called an incomplete Nettree, and can be stored in an array. The incomplete Nettree of pattern $\mathbf{p}$ is denoted as $\widehat{\mathcal{T}}_{\mathbf{p}}$.

In Example 5, the third level in Fig. 3 is the $\widehat{\mathcal{T}}_{\mathbf{p}}$ of pattern $\mathbf{p}$=T∗A∗A. To further improve the efficiency, the position set of character x (denoted as $\widehat{\mathcal{P}}_x$) is also employed to avoid rescanning the database. An example is given below.

Example 6 Consider a sequence $\mathbf{s}=s_1s_2s_3s_4s_5s_6$=TATAAA, a pattern $\mathbf{p}_1$=T, and a pattern $\mathbf{p}_2$=T∗A. The sequence is scanned once to obtain the position sets of all characters, i.e. $\widehat{\mathcal{P}}_{\mathrm{A}}$={2,4,5,6}, $\widehat{\mathcal{P}}_{\mathrm{T}}$={1,3}. Then $\widehat{\mathcal{T}}_{\mathbf{p}_1}$ can be obtained using $\widehat{\mathcal{P}}_{\mathrm{T}}$, that is, $\widehat{\mathcal{T}}_{\mathbf{p}_1} = \{n_1^1, n_1^3\}$. To calculate the support of the super-pattern $\mathbf{p}_2$ = T∗A, nodes are only created in the second level based on $\widehat{\mathcal{T}}_{\mathbf{p}_1}$. Since j=2 and node n_1^1 has no child, the first position that is greater than one from $\widehat{\mathcal{P}}_{\mathrm{A}}$ is chosen as a


$\widehat{\mathcal{T}}_{\mathbf{p}_1}$: 1st level
T
2
1
3
$\widehat{\mathcal{T}}_{\mathbf{p}_2}$: 2nd level
A
2
2
4
Fig. 4 Incomplete Nettrees of $\widehat{\mathcal{T}}_{\mathbf{p}_1}$ and $\widehat{\mathcal{T}}_{\mathbf{p}_2}$

child of n_1^1, i.e. n_2^2. In the same way, n_2^4 is created as a child of n_1^3. Since there is no other node on the first level, creation of the incomplete Nettree ends, and $\widehat{\mathcal{T}}_{\mathbf{p}_2} = \{n_2^2, n_2^4\}$. Hence, the supports of $\mathbf{p}_1$ and $\mathbf{p}_2$ are two. $\widehat{\mathcal{T}}_{\mathbf{p}_1}$ and $\widehat{\mathcal{T}}_{\mathbf{p}_2}$ are shown in Fig. 4.

4.3 The SNP-Miner algorithm

This section presents the pseudocode of the proposed SNP-Miner algorithm.

Step 1. Scan the database to obtain the character set E and the position sets of all characters $\widehat{\mathcal{P}}_x$, and store SNPs with length one and their incomplete Nettrees in a SNP set, called S_1.

Step 2. Generate candidate patterns with length m+1 using the SNP set $S_m (m \geq 1)$.

Step 3. Calculate the support of candidate pattern $\mathbf{q}$ ($sup(\mathbf{q}, D)$) using the InNet algorithm. If $sup(\mathbf{q}, D) \geq minsup$, then store $\mathbf{q}$ and its incomplete Nettree $\widehat{\mathcal{T}}_{\mathbf{q}}$ in S_{m+1}.

Step 4. Iterate Steps 2 and 3, until no new candidate patterns can be generated.

The detailed pseudocode of SNP-Miner is given in Algorithm 1.

Algorithm 1 SNP-Miner.

```
Input: Sequence database $D$, $minsup$
Output: SNP set $S$
1: Scan $D$ to obtain the character set $E$ and the position
   sets of all characters $\widehat{\mathcal{P}}_x$;
2: Calculate the support of each character, and store the
   SNPs with length one and their incomplete Nettree in
   $S_1$;
3: $m \leftarrow 1$;
4: $G \leftarrow Patternjoin(S_1)$;
5: while $G$ <> NULL do
6:     for each $\mathbf{q}$ in $G$ do
7:         $sup(\mathbf{q}, D) \leftarrow$ InNet($\widehat{\mathcal{T}}_{\mathbf{p}}$, $\widehat{\mathcal{P}}_x$, $D$); // Suppose
           $\mathbf{q}=\mathbf{p}\oplus x$
8:         if $sup(\mathbf{q}, D) \geq minsup$ then
9:             Store $\mathbf{q}$ and its incomplete Nettree $\widehat{\mathcal{T}}_{\mathbf{q}}$ in
               $S_{m+1}$;
10:        end if
11:    end for
12:    $G \leftarrow Patternjoin(S_{m+1})$;
13:    $m \leftarrow m + 1$;
14: end while
15: return $S$;
```

Algorithm InNet, shown in Algorithm 2, uses an incomplete Nettree to calculate the support of a pattern.

Algorithm 2 InNet($\widehat{\mathcal{T}}_{\mathbf{p}}$, $\widehat{\mathcal{P}}_x$, D).

```
Input: $\widehat{\mathcal{T}}_{\mathbf{p}}$, $\widehat{\mathcal{P}}_x$, $D$
Output: $sup(\mathbf{q}, D)$ and $\widehat{\mathcal{T}}_{\mathbf{q}}$
1: for $i$=1 to $|D|$ do
2:     for each node $y$ in $\widehat{\mathcal{T}}_{\mathbf{p}\_i}$ do
3:         Find the first position $z$ from $\widehat{\mathcal{P}}_{x\_i}$ which is
           greater than $y$, and store $z$ in $\widehat{\mathcal{T}}_{\mathbf{q}\_i}$;
4:     end for
5:     $sup(\mathbf{q}, D) \leftarrow sup(\mathbf{q}, D) + |\widehat{\mathcal{T}}_{\mathbf{q}\_i}|$;
6: end for
7: return $sup(\mathbf{q}, D)$ and $\widehat{\mathcal{T}}_{\mathbf{q}}$;
```

Theorem 2 *The average space complexity of SNP-Miner is $O(t \times N/r)$, where t, N and r are the maximum number of candidate patterns in candidate set G, the size of database D, and the size of character set E, respectively.*

Proof Obviously, the space complexity of the SNPs can be neglected, since it is far less than the size of the database D. Since there are r characters in E, the average size of each incomplete Nettree is $O(N/r)$. The space complexity of the SNPs and candidate patterns in an incomplete Nettree is therefore $O(t \times N/r)$, and the space complexity of SNP-Miner is $O(t \times N/r + N) = O(t \times N/r)$. □

Theorem 3 *The time complexity of SNP-Miner is $O(T \times N/r)$, where T is the number of candidate patterns.*

Proof The time needed to generate an incomplete Nettree of a super-pattern is $O(N/r)$. Since there are T candidate patterns that need to be calculated, the time complexity of SNP-Miner is $O(T \times N/r)$. □

5 Performance evaluation

Section 5.1 introduces the benchmark datasets and baseline methods used in the experiments. Section 5.2 reports the efficiency of different strategies, including candidate generation and support calculation. Section 5.3 compares and analyzes the mining ability of our method. Section 5.4 shows and analyzes the self-adaptive ability of the proposed method, and Section 5.5 reports the results of a travel route mining.

All the experiments are conducted on a computer with an Intel(R) Core (TM) i5-8259U 2.50 GHz CPU, 16 GB of memory, and the Windows 10 operating system. All the algorithms are implemented and compiled

Table 3 Benchmark datasets

Dataset	Type	From	Total length
SDB1 [1]	protein	ASTRAL95_1_161	91 875
SDB2	protein	ASTRAL95_1_161	62 985
SDB3	protein	ASTRAL95_1_171	109 424
SDB4	protein	ASTRAL95_1_171	73 425
SDB5	protein	ASTRAL95_2_171	96 394
SDB6	protein	ASTRAL95_2_171	97 062
Baby [2]	Sales	Sales of baby products	32 640
Super [3]	Sales	SuperStore	40 180
Travelling [4]	Travelling	Recommended tourist routes	1 390

[1]The SDB1-6 datasets were studied in [69] and is from http://scop.mrc-lmb.cam.ac.uk/.

[2]Baby is a sales dataset for infant products, which is from https://tianchi.aliyun.com/dataset/dataDetail?dataId=45.

[3]Super is a sales dataset from SuperStore, which is download from https://tianchi.aliyun.com/dataset/dataDetail?dataId=93285.

[4]Travelling is a dataset of routes recommended by various tourists, and is downloaded from http://www.mafengwo.cn/mdd/route/10035.html.

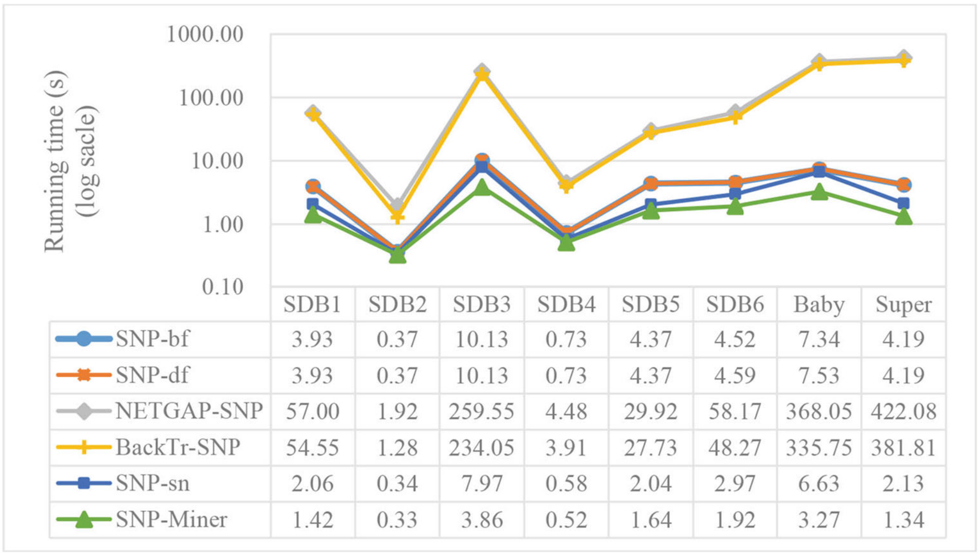
using the VC++6.0 development environment. Algorithms and data can be downloaded from https://github.com/wuc567/Pattern-Mining/tree/master/SNP-Miner. Moreover, a Java version of SNP-Miner can be found at http://www.philippe-fournier-viger.com/spmf/index.php?link=algorithms.php.

5.1 Benchmark datasets and Baseline methods

To evaluate the performance of SNP-Miner, we choose nine benchmark datasets which are summarized in Table 3.

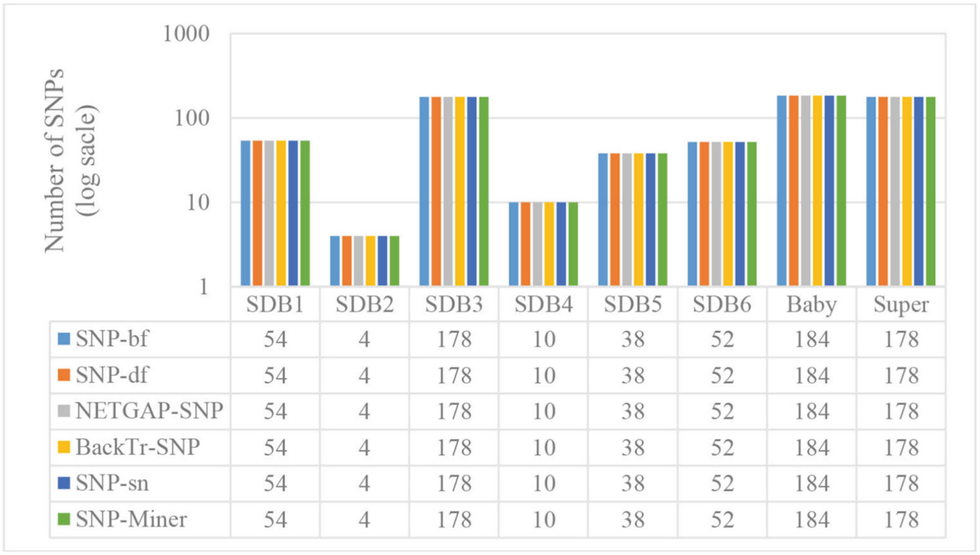
To evaluate the performance of SNP-Miner, four experiments are designed. The first aims to evaluate the efficiency of the algorithm, while the second is designed to verify its mining ability. The third experiment is done to evaluate the advantage of using a self-adaptive gap for SPM. In the last experiment, a database of tourist routes is used to explore the application of SNP-Miner to real-world scenarios. A brief introduction of the competitive algorithms is given below.

1. SNP-bf and SNP-df: To evaluate the efficiency of the pattern join strategy, SNP-bf and SNP-df are developed. They adopt the breadth-first and depth-first strategies to generate candidate patterns, respectively.
2. NETGAP-SNP, BackTr-SNP, and SNP-sn: To validate the efficiency of SNP-Miner which uses an incomplete



	SDB1	SDB2	SDB3	SDB4	SDB5	SDB6	Baby	Super
SNP-bf	3.93	0.37	10.13	0.73	4.37	4.52	7.34	4.19
SNP-df	3.93	0.37	10.13	0.73	4.37	4.59	7.53	4.19
NETGAP-SNP	57.00	1.92	259.55	4.48	29.92	58.17	368.05	422.08
BackTr-SNP	54.55	1.28	234.05	3.91	27.73	48.27	335.75	381.81
SNP-sn	2.06	0.34	7.97	0.58	2.04	2.97	6.63	2.13
SNP-Miner	1.42	0.33	3.86	0.52	1.64	1.92	3.27	1.34

Fig. 5 Comparison of running time



	SDB1	SDB2	SDB3	SDB4	SDB5	SDB6	Baby	Super
🟦 SNP-bf	54	4	178	10	38	52	184	178
🟧 SNP-df	54	4	178	10	38	52	184	178
⬜ NETGAP-SNP	54	4	178	10	38	52	184	178
🟨 BackTr-SNP	54	4	178	10	38	52	184	178
🟦 SNP-sn	54	4	178	10	38	52	184	178
🟩 SNP-Miner	54	4	178	10	38	52	184	178

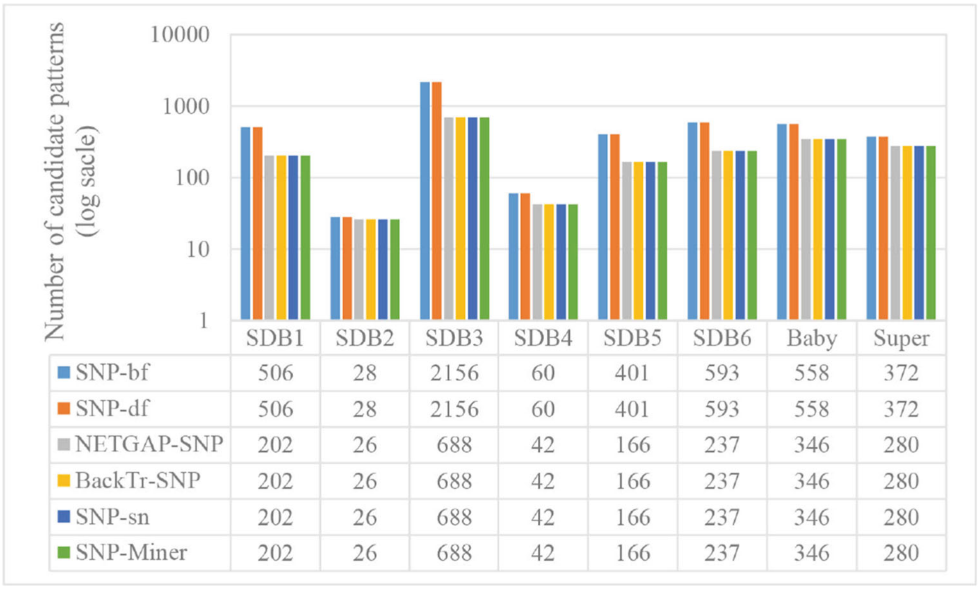
Fig. 6 Comparison of number of SNPs

Nettree structure to calculate the support, NETGAP-SNP, BackTr-SNP, and SNP-sn are developed. They employ NETGAP [26], BackTr [14], and a single-branch Nettree to calculate the support, respectively.

3. SNP-nogap: To verify the effect of the gap constraint on SNP-Miner, SNP-nogap is developed to mine continuous SNPs without gaps.
4. PMBC: To analyze the mining ability of SNP-Miner under the nonoverlapping condition, PMBC [35] is selected as a competitive algorithm to mine self-adaptive patterns under the one-off condition.
5. GSgrow, NOSEP and BackTr-Pro: To validate the self-adaptive ability of SNP-Miner, GSgrow [25] and NOSEP [26] are selected as the competitive algorithms, and BackTr-Pro is proposed which uses BackTr [14] to calculate the support and mines patterns with gap constraint.

5.2 Effectiveness

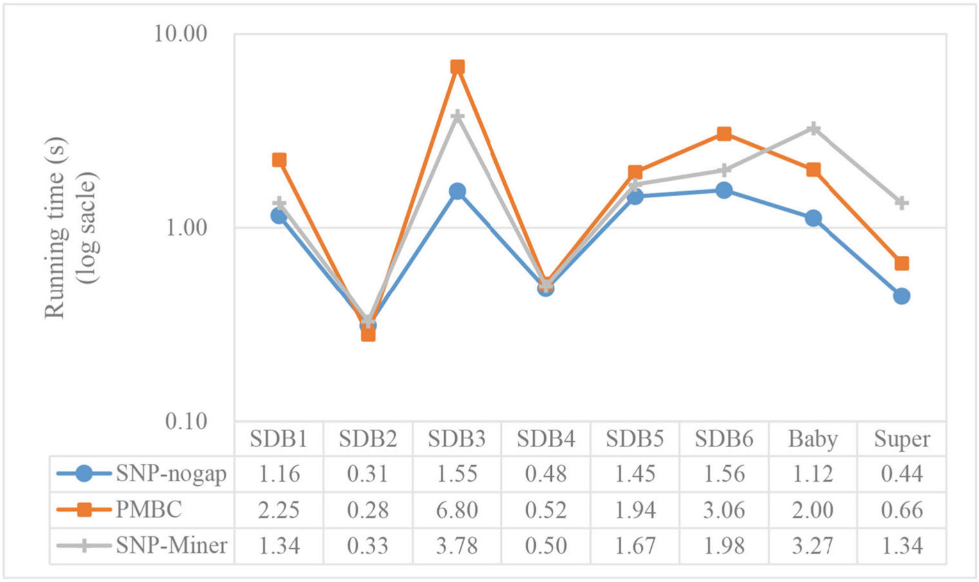
In this subsection, we evaluate the mining performance of SNP-Miner using different strategies. Five competive algorithms are selected: SNP-bf, SNP-df, NETGAP-SNP, BackTr-SNP and SNP-sn. Eight datasets are used: SDB1, SDB2, SDB3, SDB4, SDB5, SDB6, Baby and Super. The experiments are conducted with $minsup$ = 5 000. The running time, number of SNPs and number of candidate patterns are shown in Figs. 5, 6 and 7, respectively.



	SDB1	SDB2	SDB3	SDB4	SDB5	SDB6	Baby	Super
🟦 SNP-bf	506	28	2156	60	401	593	558	372
🟧 SNP-df	506	28	2156	60	401	593	558	372
⬜ NETGAP-SNP	202	26	688	42	166	237	346	280
🟨 BackTr-SNP	202	26	688	42	166	237	346	280
🟦 SNP-sn	202	26	688	42	166	237	346	280
🟩 SNP-Miner	202	26	688	42	166	237	346	280

Fig. 7 Comparison of number of candidate patterns

Fig. 8 Comparison of running time

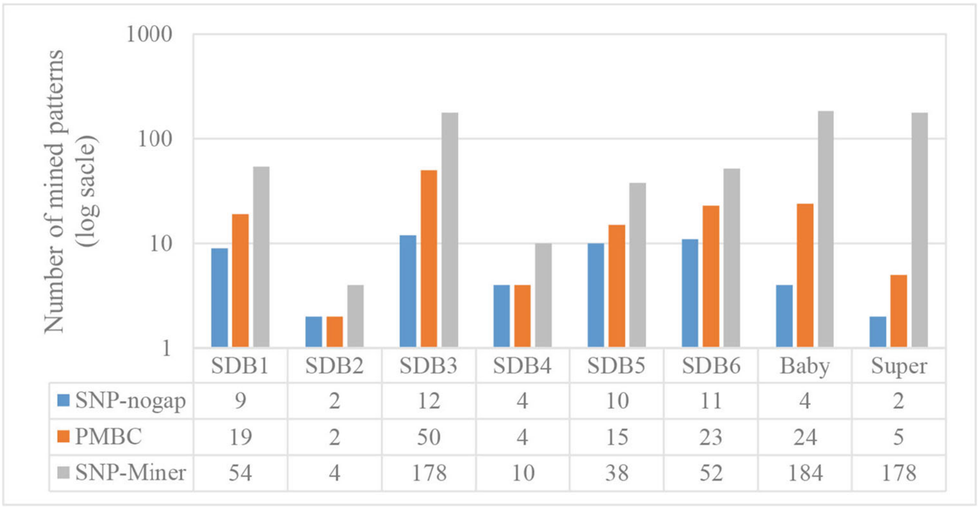


	SDB1	SDB2	SDB3	SDB4	SDB5	SDB6	Baby	Super
● SNP-nogap	1.16	0.31	1.55	0.48	1.45	1.56	1.12	0.44
■ PMBC	2.25	0.28	6.80	0.52	1.94	3.06	2.00	0.66
+ SNP-Miner	1.34	0.33	3.78	0.50	1.67	1.98	3.27	1.34

We use the average running time per 10 000 characters length (ART) to evaluate the running performance of SNP-bf, SNP-df, NETGAP-SNP, BackTr-SNP, SNP-sn and SNP-Miner. The ART of the six algorithms is 7.16, 7.24, 322.29, 292.35, 5.21 and 2.92 s, respectively. It can be seen that SNP-Miner is about 2.45, 2.48, 110.37, 100.12 and 1.78 times faster than SNP-bf, SNP-df, NETGAP-SNP, BackTr-SNP and SNP-sn, respectively. We show the reasons as follows.

1. SNP-Miner outperforms SNP-bf and SNP-df. From Figs. 5, 6 and 7, SNP-Miner runs faster than both SNP-bf and SNP-df, although the three algorithms discover the same number of SNPs. For example, on SDB6, the running time of SNP-Miner is 1.92 s, while the running time of SNP-bf and SNP-df is 4.52 and 4.59 s, respectively. SNP-Miner generates 237 candidate patterns, while SNP-bf and SNP-df generate 593, and all the three algorithms discover 52 patterns. The reason for this is that SNP-Miner employs a pattern join strategy to generate the candidate patterns, while SNP-bf and SNP-df employ the breadth-first and depth-first strategies, respectively. The analysis in Section 4.2 shows that the pattern join strategy is more effective than the breadth-first and depth-first strategies, and the results verify its correctness. Hence, SNP-Miner has better performance than SNP-bf and SNP-df.
2. SNP-Miner outperforms NETGAP-SNP, BackTr-SNP and SNP-sn. From Figs. 5–7, it can be seen that SNP-Miner, NETGAP-SNP, BackTr-SNP and SNP-sn discover the same number of SNPs and generate the

Fig. 9 Comparison of number of mined patterns



	SDB1	SDB2	SDB3	SDB4	SDB5	SDB6	Baby	Super
■ SNP-nogap	9	2	12	4	10	11	4	2
■ PMBC	19	2	50	4	15	23	24	5
■ SNP-Miner	54	4	178	10	38	52	184	178

same number of candidate patterns, but SNP-Miner runs faster than NETGAP-SNP, BackTr-SNP and SNP-sn. For example, on SDB3, all the four algorithms discover 178 SNPs and generate 688 candidate patterns, but the running time of SNP-Miner, NETGAP-SNP, BackTr-SNP and SNP-sn is 3.86, 259.55, 234.05 and 7.97 s, respectively. The reason for this is that all the four algorithms adopt the same candidate pattern generation strategy but different support calculation methods. The number of candidate patterns and SNPs are therefore the same for all the four algorithms, but SNP-Miner applies InNet to calculate the support based on sub-patterns, and does not need to create and prune invalid nodes. Hence, SNP-Miner is more effective than NETGAP-SNP, BackTr-SNP and SNP-sn, and has better performance than other competitive algorithms.

In summary, SNP-Miner achieves better performance than other competitive algorithms.

5.3 Mining ability

In this subsection, SNP-nogap and PMBC are selected as the competitive algorithms to explore the mining ability of SNP-Miner on different datasets. Eight datasets are used: SDB1, SDB2, SDB3, SDB4, SDB5, SDB6, Baby and Super. The experiments are conducted with $minsup$=5 000. The running time and number of patterns are shown in Figs. 8 and 9, respectively.

The results give rise to the following observations.

1. The use of a gap constraint can effectively improve the mining ability of the algorithm. Although SNP-nogap runs faster than SNP-Miner, fewer SNPs are found. For example, from Fig. 9, it can be seen that SNP-nogap mines nine patterns on SDB1, while SNP-Miner mines 54. The reason for this lies in the fact that SNP-Miner applies a gap constraint, meaning that more patterns can be mined, while SNP-nogap mines only continuous patterns without a gap constraint.
2. The mining ability of the algorithm under the nonoverlapping condition is better than the one-off condition. Although PMBC runs faster than SNP-Miner on Baby and Super, it discovers fewer patterns than SNP-Miner. Except for these two datasets, SNP-Miner not only discovers more interesting SNPs than PMBC, but also runs faster on most cases. For example, on SDB6, SNP-Miner runs 1.98 s to mine 52 SNPs and PMBC runs 3.06 s to mine 23 patterns. The reason for this is that SNP-Miner mines patterns under the nonoverlapping condition, which is looser than the one-off condition. More importantly, SNP-Miner employs a pattern join strategy to generate candidates and adopts an incomplete Nettree structure to calculate the support, which significantly improves the efficiency and reduces the running time. Hence, SNP-Miner not only can mine more patterns than PMBC, but also runs faster.

In conclusion, SNP-Miner has better mining ability than all the competitive algorithms.

5.4 Self-adaptive ability

In this subsection, we compare the mining results of GSgrow, NOSEP, BackTr-Pro and SNP-Miner to demonstrate the advantage of using a self-adaptive gap in SPM. Eight datasets are used: SDB1, SDB2, SDB3, SDB4, SDB5, SDB6, Baby and Super. The experiments are conducted with $minsup$=5 000. The running time and number of SNPs are shown in Tables 4 and 5, respectively.

The results can be summarized as follows.

The mining ability of SNP-Miner is superior to that of GSgrow, NOSEP and BackTr-Pro, since it not only runs faster, but also discovers more patterns. From Table 4, the running time of SNP-Miner is 3.28 s on SDB3 with gap

Table 4 Comparison of running time for different gap constraints

	Gap	SDB1	SDB2	SBD3	SBD4	SBD5	SBD6	Baby	Super
Gsgrow	[0,10]	3.86	0.92	3.52	1.56	3.64	2.92	34.53	3.51
	[0,20]	5.14	0.97	5.13	1.89	5.00	4.24	45.16	4.03
	[0,30]	7.16	0.98	7.58	2.02	6.31	5.63	48.28	4.45
NOSEP	[0,10]	6.67	0.34	12.16	1.13	7.59	9.14	144.24	4.64
	[0,20]	8.08	0.38	19.63	1.44	9.17	11.45	224.42	51.91
	[0,30]	14.03	0.41	39.27	1.70	11.89	13.74	258.28	143.08
BackTr-Pro	[0,10]	6.45	0.33	10.50	1.03	6.53	7.86	137.63	4.56
	[0,20]	7.84	0.36	17.83	1.31	8.25	10.13	210.72	48.67
	[0,30]	13.78	0.39	37.92	1.56	10.83	12.91	240.41	134.78
SNP-Miner		**1.34**	**0.33**	**3.28**	**0.50**	**1.67**	**1.98**	**3.27**	**1.34**

Table 5 Comparison of number of patterns for different gap constraints

	Gap	SDB1	SDB2	SBD3	SBD4	SBD5	SBD6	Baby	Super
Gsgrow	[0,10]	11	2	14	4	11	12	140	15
	[0,20]	14	2	20	5	15	17	177	70
	[0,30]	21	2	29	6	19	22	182	138
NOSEP	[0,10]	11	2	14	4	11	12	152	15
	[0,20]	14	2	28	5	15	18	181	78
	[0,30]	27	2	58	6	21	24	184	152
BackTr-Pro	[0,10]	11	2	14	4	11	12	152	15
	[0,20]	14	2	28	5	15	18	181	78
	[0,30]	27	2	58	6	21	24	184	152
SNP-Miner		**54**	**4**	**178**	**10**	**38**	**52**	**184**	**178**

[0,10], while the running time of GSgrow, NOSEP and BackTr-Pro is 3.52, 12.16 and 10.50 s, respectively. Table 5 shows that SNP-Miner finds 178 SNPs, while GSgrow, NOSEP and BackTr-Pro find 14 patterns. The results are similar on other datasets. The reasons for this are as follows: (i) GSgrow, NOSEP and BackTr-Pro need to constantly check whether the gap meets the requirements, while SNP-Miner does not; (ii) some patterns will be missed if the gap is too small. More importantly, with the increase of the gap, the running time increases rapidly. Hence, SNP-Miner outperforms GSgrow, NOSEP and BackTr-Pro.

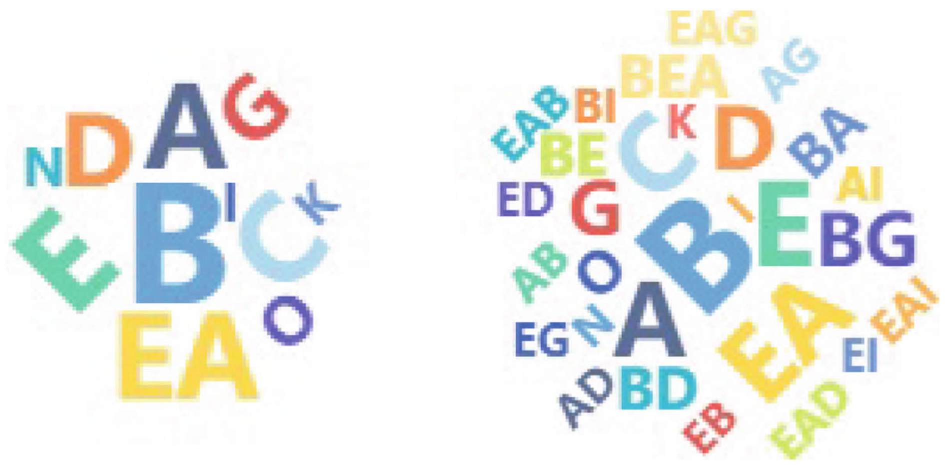
5.5 Case study

In this subsection, we explore the application of SNP-Miner to a dataset of travel routes. To find the most popular attractions and travel routes, experiments are conducted on the Travelling dataset, which contains many routes recommended by different tourists. There are no duplicate attractions between routes. To simplify the problem, symbols are used to denote the attractions and each tourist route is converted into a sequence of symbols. In this dataset, there are 26 common attractions, represented by characters A to Z. The experiments are conducted with $minsup$=50 and gap=[0,1] in NOSEP. Wordcloud maps [70] of the mining results are shown in Fig. 10.

The results give rise to the following observations.

1. In Fig. 10, the larger the font size is, the more popular is a pattern (sequence of attractions). It can be seen that attraction "B" is the most popular. Although attractions "A" and "E" are not the most popular attractions on an individual basis, a combination of "EA" is very popular. Websites or travel agencies should therefore recommend packages that include both attraction "B" and route "EA" to tourists.
2. SNP-Miner gives better mining results than NOSEP, since it can mine more interesting routes that can be recommended to tourists. For example, from Fig. 10a, it can be observed that although NOSEP mines many popular attractions, it only finds one popular tourist route, meaning that the choice is more limited.

Fig. 10 Comparison of Wordcloud maps of the mining results



(a) Wordcloud map for NOSEP (b) Wordcloud map for SNP-Miner

However, Fig. 10b shows that SNP-Miner can discover more tourist routes, leading to a more valuable result. The reason for this is that SNP-Miner adopts a self-adaptive gap that mines patterns based on sequence characters, while NOSEP employs a user-predefined gap to mine patterns. Hence, SNP-Miner is superior to NOSEP for real-world applications.

6 Conclusion

To discover meaningful patterns without a predefined gap constraint, and to reduce the mining time of gap constraint SPM, we present an effective self-adaptive pattern mining algorithm called SNP-Miner, which involves two main steps: candidate generation and support calculation. To effectively reduce the number of candidate patterns, SNP-Miner employs a pattern join strategy. To calculate the support more efficiently, the algorithm applies a novel single-branch Nettree structure to find self-adaptive occurrences without creating invalid nodes and relationships. However, redundancy still exists in terms of calculating the support of super-patterns. Thus, to further improve the efficiency of the support calculation, the InNet algorithm is presented, which adopts an incomplete Nettree structure to obtain the support of a pattern based on its sub-patterns with one-way scanning. SNP-Miner has a lower time complexity than the existing alternative methods. Extensive experiments are carried out on protein, sales sequence and travel route datasets. Results show that SNP-Miner is superior to other competitive algorithms and can discover frequent patterns without gap constraints. More importantly, SNP-Miner can be used for travel routes recommendation to help tourists make better travel plans.

Although SNP-miner algorithm adopts the incomplete Nettree structure to calculate the support, which has better performance, it still has some drawbacks. SNP-miner is a two-phase algorithm which needs to generate candidate patterns. In the future, we will focus on one-phase algorithm to avoid candidate generation and improve the mining efficiency. Furthermore, self-adaptive overlapping (no-condition) SPM is also interesting and has many applications, which also requires further exploration in the future.

Acknowledgements This work was supported by National Social Science Fund of China under grant number 18BGL191.

References

1. Fournier-Viger P, Gomariz A, Gueniche T, Soltani A, Wu C-W, Tseng VS (2014) SPMF: A java open-source pattern mining library. J Mach Learn Res 15(1):3389–3393
2. Kim J, Yun U, Yoon E, Lin JC-W, Fournier-Viger P (2020) One scan based high average-utility pattern mining in static and dynamic databases. Futur Gener Comput Syst 111:143–158
3. Fournier-Viger P, Lin JC-W, Kiran RU, Koh YS, Thomas R (2017) A survey of sequential pattern mining. Data Sci Pattern Recogn 1(1):54–77
4. Han J, Pei J, Yin Y, Mao R (2004) Mining frequent patterns without candidate generation. Data Min Knowl Discov 8(1):53–87
5. Wu M, Wu X (2019) On big wisdom. Knowl Inf Syst 58(1):1–8
6. Xie F, Wu X, Zhu X (2017) Efficient sequential pattern mining with wildcards for keyphrase extraction. Knowl Based Syst 115:27–39
7. Yun U, Lee G, Yoon E (2019) Advanced approach of sliding window based erasable pattern mining with list structure of industrial fields. Inf Sci 494:37–59
8. Lin JC-W, Pirouz M, Djenouri Y, Cheng C-F, Ahmed U (2020) Incrementally updating the high average-utility patterns with pre-large concept. Appl Intell 50(11):3788–3807
9. Lin JC-W, Shao Y, Djenouri Y, Yun U (2021) ASRNN: A recurrent neural network with an attention model for sequence labeling. Knowl Based Syst 212(5):106548
10. Srivastava G, Lin JC-W, Pirouz M, Li Y, Yu U (2020) A pre-large weighted-fusion system of sensed high-utility patterns. IEEE Sensors Journal. https://doi.org/10.1109/JSEN.2020.2991045
11. Srikant R, Agrawal R (1995) Mining sequential patterns. Proc 11th Int Conf Data Eng 1995:3–14
12. Truong T, Duong H, Le B, Fournier-Viger P, Yun U (2019) Efficient high average-utility itemset mining using novel vertical weak upper-bounds. Knowl Based Syst 183(1):104847
13. Wu Y, Wang Y, Liu J, Yu M, Liu J, Li Y (2019) Mining distinguishing subsequence patterns with nonoverlapping condition. Clust Comput 22:5905–5917
14. Wu Y, Zhu C, Li Y, Guo L, Wu X (2020) NetNCSP: Nonoverlapping closed sequential pattern mining. Knowl Based Syst 196(105812)
15. Ji X, Bailey J, Dong G (2005) Mining minimal distinguishing subsequence patterns with gap constraints. Proc 5th IEEE Int Conf Data Min (ICDM) 2005:194–201
16. Wu Y, Fu S, Jiang H, Wu X (2015) Strict approximate pattern matching with general gaps. Appl Intell 42(3):566–580
17. Dong X, Gong Y, Cao L (2020) e-RNSP: An efficient method for mining repetition negative sequential patterns. IEEE Trans Cybern 50(5):2084–2096
18. Dong X, Qiu P, Lü J, Cao L (2019) Mining top-*k* useful negative sequential patterns via learning. IEEE Trans Neural Netw Learn Syst 30(9):2764–2778
19. Wu Y, Shen C, Jiang H, Wu X (2017) Strict pattern matching under non-overlapping condition. Sci China Inf Sci 60(1):012101
20. Gan W, Lin JC-W, Fournier-Viger P, Chao H-C, Yu PS (2019) A survey of parallel sequential pattern mining. ACM Trans Knowl Discov Data 13(3):25:1–25,34
21. Nam H, Yun U, Yoon E, Lin JC-W (2020) Efficient approach of recent high utility stream pattern mining with indexed list structure and pruning strategy considering arrival times of transactions. Inf Sci 529:1–27
22. Lv Z, Qiao L (2020) Analysis of healthcare big data. Futur Gener Comput Syst 109:103–110
23. Gan W, Lin JC-W, Fournier-Viger P, Chao H-C, Tseng VS, Yu PS (2021) A survey of utility-oriented pattern mining. IEEE Trans Knowl Data Eng 33(4):1306–1327
24. Zhang M, Kao B, Cheung DW, Yip KY (2007) Mining periodic patterns with gap requirement from sequences. ACM Trans Knowl Discov Data 1(2):7
25. Ding B, Lo D, Han J, Khoo S (2009) Efficient mining of closed repetitive gapped subsequences from a sequence database. IEEE 25th Int Conf Data Eng 2009:1024–1035

26. Wu Y, Tong Y, Zhu X, Wu X (2018) NOSEP: Nonoverlapping Sequence pattern mining with gap constraints. IEEE Trans Cybern 48(10):2809–2822
27. Shi Q, Shan J, Yan W, Wu Y, Wu X (2020) NetNPG: Nonoverlapping pattern matching with general gap constraints. Appl Intell 50(6):1832–1845
28. Wu Y, Liu X, Yan W, Guo L, Wu X (2021) Efficient solving algorithm for strict pattern matching under nonoverlapping condition. Journal of Software. https://doi.org/10.13328/j.cnki.jos.006054
29. Min F, Zhang Z, Zhai W, Shen R (2020) Frequent pattern discovery with tri-partition alphabets. Inf Sci 507:715–732
30. Huang J-W, Jaysawal B, Chen K-Y, Wu Y-B (2019) Mining frequent and top-K high utility time interval-based events with duration patterns. Knowl Inf Syst 61(3):1331–1359
31. Renz-Wieland A, Bertsch M, Gemull R (2019) Scalable frequent sequence mining with flexible subsequence constraints. IEEE 35th Int Conf Data Eng 2019:1490–1501
32. Truong T, Duong H, Le B, Fournier-Viger P, Yun U, Fujita H (2021) Efficient algorithms for mining frequent high utility sequences with constraints. Inf Sci 568:239–264
33. Okolica J, Peterson G, Mills R, Grimaila M (2020) Sequence pattern mining with variables. IEEE Trans Knowl Data Eng 32(1):177–187
34. Fournier-Viger P, Li Z, Lin JC-W, Kiran RU, Fujita H (2019) Efficient algorithms to identify periodic patterns in multiple sequences. Inf Sci 489:205–226
35. Wu X, Zhu X, He Y, Zhao P, Arslan AN (2013) PMBC: Pattern Mining from biological sequences with wildcard constraints. Comput Biol Med 43(5):481–492
36. Wu X, Zhu X, Wu GQ, Ding W (2014) Data mining with big data. IEEE Trans Knowl Data Eng 26(1):97–107
37. Fournier-Viger P, Li J, Lin JC-W, Truong T, Kiran RU (2020) Mining cost-effective patterns in event logs. Knowl Based Syst 191(105241)
38. Yu K, Liu L, Li J, Ding W, Le T (2020) Multi-source causal feature selection. IEEE Trans Pattern Anal Mach Intell 42(9):2240–2256
39. Li C, Yang Q, Wang J, Li M (2012) Efficient mining of gap-constrained subsequences and its various applications. ACM Trans Knowl Discov Data (TKDD) 6(1):2:1–2:39
40. Xu T, Li T, Dong X (2018) Efficient high utility negative sequential patterns mining in smart campus. IEEE Access 6:23839–23847
41. Zhang L, Luo P, Tang L, Chen E, Liu Q, Wang M, Xiong H (2015) Occupancy-based frequent pattern mining. ACM Trans Knowl Discov Data 10(2):14:1–14:33
42. Gan W, Lin JC-W, Zhang J, Yu PS (2020) Utility mining across multi-sequences with individualized thresholds. ACM/IMS Trans Data Sci 1(2):18:1–18:29
43. Srivastava G, Lin JC-W, Jolfaei A, Li Y, Djenouri Y (2020) Uncertain-driven analytics of sequence data in IoCV environments. IEEE Transactions on Intelligent Transportation Systems. https://doi.org/10.1109/TITS.2020.3012387
44. Wu Y, Luo L, Li Y, Guo L, Fournier-Viger P, Zhu X, Wu X (2021) NTP-Miner: Nonoverlapping three-way sequential pattern mining. ACM Trans Knowl Discov Data 16(3):51
45. Cheng S, Wu Y, Li Y, Yao F, Min F (2021) TWD-SFNN: Three-way decisions with a single hidden layer feedforward neural network. Information Sciences. https://doi.org/10.1016/j.ins.2021.07.091
46. Wu Y, Geng M, Li Y, Guo L, Li Z, Fournier-Viger P, Zhu X, Wu X (2021) HANP-Miner: High average utility nonoverlapping sequential pattern mining. Knowledge-Based Systems. https://doi.org/10.1016/j.knosys.2021.107361
47. Srivastava G, Lin JC-W, Zhang X, Li Y (2020) Large-scale high-utility sequential pattern analytics in Internet of things. IEEE Internet of Things Journal. https://doi.org/10.1109/JIOT.2020.3026826
48. Kim H, Yun U, Baek Y, Kim J, Vo B, Yoon E, Fujita H (2021) Efficient list based mining of high average utility patterns with maximum average pruning strategies. Inf Sci 543(8):85–105
49. Yun U, Kim D, Yoon E, Fujita H (2018) Damped window based high average utility pattern mining over data streams. Knowl-Based Syst 144(15):188–205
50. Wu Y, Wang Y, Li Y, Zhu X, Wu X (2021) Top-k self-adaptive contrast sequential pattern mining. IEEE Transactions on Cybernetics. https://doi.org/10.1109/TCYB.2021.3082114
51. Chen X, Rao Y, Xie H, Wang FL, Zhao Y, Yin J (2019) Sentiment classification using negative and intensive sentiment supplement information. Data Sci Eng 4:109–118
52. Gan W, Lin JC-W, Fournier-Viger P, Chao H-C, Yu PS (2020) HUOPM: High-Utility occupancy pattern mining. IEEE Trans Cybern 50(3):1195–1208
53. Gan W, Lin JC-W, Zhang J, Chao H-C, Fujita H, Yu PS (2020) ProUM: Projection-based utility mining on sequence data. Inf Sci 513:222–240
54. Wu Y, Fan J, Li Y, Guo L, Wu X (2020) NetDAP: (δ, γ)-approximate pattern matching with length constraints. Appl Intell 50(11):4094–4116
55. Wang H, Duan L, Zuo J, Wang W, Li Z, Tang C (2016) Efficient mining of distinguishing sequential patterns without a predefined gap constraint. Chin J Comput 39(10):19791991
56. Wu Y, Lei R, Li Y, Guo L, Wu X (2021) HAOP-Miner: Self-adaptive high-average utility one-off sequential pattern mining. Expert Systems with Applications. https://doi.org/10.1016/j.eswa.2021.115449
57. Dinh D-T, Le B, Fournier-Viger P, Huynh V-N (2018) An efficient algorithm for mining periodic high-utility sequential patterns. Appl Intell 48(12):4694–4714
58. Lin JC-W, Li T, Pirouz M, Zhang J, Fournier-Viger P (2020) High average-utility sequential pattern mining based on uncertain databases. Knowl Inf Syst 62(3):1199–1228
59. Wang J, Han J, Li C (2007) Frequent closed sequence mining without candidate maintenance. IEEE Trans Knowl Data Eng 19(8):1042–1056
60. Yun U, Nam H, Kim J, Kim H, Pedrycz W (2020) Efficient transaction deleting approach of pre-large based high utility pattern mining in dynamic databases. Futur Gener Comput Syst 103:58–78
61. Pei J, Han J, Wang W (2007) Constraint-based sequential pattern mining: the pattern-growth methods. J Intell Inf Syst 28(2):133–160
62. Min F, Wu Y, Wu X (2010) The Apriori property of sequence pattern mining with wildcard gaps. IEEE Int Conf Bioinform Biomed Workshop 2010:138–143
63. Guo D, Hu X, Xie F, Wu X (2013) Pattern matching with wildcards and gap-length constraints based on a centrality-degree graph. Appl Intell 39(1):57–74
64. Wu X, Wang X, Li Y, Guo L, Li Z, Zhang J, Wu X (2021) OWSP-Miner: Self-adaptive one-off weak-gap strong pattern mining. ACM Transactions on Management Information Systems. https://doi.org/10.1145/3476247
65. Hoang T, Mörchen F, Fradkin D, Calders T (2014) Mining compressing sequential patterns. Stat Anal Data Min 7(1):34–52
66. Liu H, Liu Z, Huang H, Wu X (2018) Sequential pattern matching with general gap and one-off condition. J Softw 29:363–382
67. Zaki MJ (2001) SPADE: An efficient algorithm for mining frequent sequences. Mach Learn 42:31–60
68. Pei J, Han J, Mortazavi-Asl B, Wang J, Pinto H, Chen Q, Dayal U, Hsu M (2004) Mining sequential patterns by pattern-growth: The

prefixspan approach. IEEE Trans Knowl Data Eng 16(11):1424–1440

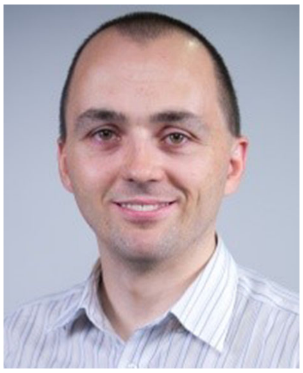
69. Wittkop T, Baumbach J, Lobo F, Rahmann S (2007) Large scale clustering of protein sequences with FORCE-a layout based heuristic for weighted cluster editing. BMC Bioinform 8(1):396
70. Heimerl F, Lohmann S, Lange S, Ertl T (2014) Word cloud explorer: Text analytics based on word clouds. 2014 47th Hawaii Int Conf Syst Sci 2014:1833–1842

Yuehua Wang received the B.S. degree in Computer Science and Technology from Langfang Normal University, Langfang, China. She is a Ph.D. candidate with the Department of Management Science and Engineering, Hebei University of Technology, Tianjin, China. Her current research interests include data mining and machine learning.

Youxi Wu received the Ph.D. degree in Theory and New Technology of Electrical Engineering from the Hebei University of Technology, Tianjin, China. He is currently a Ph.D. Supervisor and a Professor with the Hebei University of Technology. He has published more than 30 research papers in some journals, such as IEEE TCYB, ACM TKDD, ACM TMIS, SCIS, INS, JCST, KBS, EWSA, JIS, Neurocomputing, and APIN. He is a senior member of CCF and a member of IEEE. His current research interests include data mining and machine learning.

Yan Li received the Ph.D. degree in Management Science and Engineering from Tianjin University, Tianjin, China. She is an Associate Professor with the Hebei University of Technology, Tianjin. Her current research interests include data mining and supply chain management.

Fang Yao received the Ph.D. degree in Electrical Electronics from the Hebei University of Technology, Tianjin, China. She is currently a Professor with the Hebei University of Technology. Her current research interests include Reliability of electrical equipment and machine learning.



Philippe Fournier-Viger (Ph.D) is professor at the Shenzhen University, China. He received a Ph.D. in Computer Science at the University of Quebec in Montreal (2010). He has published more than 300 research papers related to data mining, intelligent systems and applications, which have received more than 7500 citations. He is associate editor-in-chief of the Applied Intelligence journal and editor-in-chief of Data Science and Pattern Recognition. He is the founder of the popular SPMF data mining library. He is also co-founder of the UDML and MLiSE series of workshops, held at the ICDM, KDD and PKDD conferences.

Xindong Wu received the Ph.D. degree from the University of Edinburgh, Edinburgh, U.K. He is a Ph.D. Supervisor, a Professor, and a Yangtze River Scholar with the Hefei University of Technology, Hefei, China. His current research interests include data mining, big data analytics, knowledge based systems, and Web information exploration. Dr. Wu is the Steering Committee Chair of the IEEE International Conference on Data Mining and the Editor-in-Chief of Knowledge and Information Systems. He is a Fellow of the American Association for the Advancement of Science and IEEE fellow.

Affiliations

Yuehua Wang[1] · Youxi Wu[2,3,4] · Yan Li[1] · Fang Yao[3] · Philippe Fournier-Viger[5] · Xindong Wu[6,7]

Yuehua Wang
18822028908@163.com

Yan Li
lywuc@163.com

Fang Yao
yaofang@hebut.edu.cn

Philippe Fournier-Viger
philfv@szu.edu.cn

Xindong Wu
xwu@hfut.edu.cn

1 School of Economics and Management, Hebei University of Technology, Tianjin 300401, China

2 School of Artificial Intelligence, Hebei University of Technology, Tianjin 300401, China

3 State Key Laboratory of Reliability and Intelligence of Electrical Equipment, Hebei University of Technology, Tianjin 300401, China

4 Hebei Key Laboratory of Big Data Computing, Tianjin 300401, China

5 College of Computer Science and Software Engineering, Shenzhen University, Shenzhen 518060, China

6 Key Laboratory of Knowledge Engineering with Big Data (Hefei University of Technology), Ministry of Education, Hefei 230009, China

7 Mininglamp Academy of Sciences, Mininglamp Technology, Beijing 100084, China