Knowl Inf Syst (2014) 38:623–639
DOI 10.1007/s10115-012-0602-x

REGULAR PAPER

DFSP: a Depth-First SPelling algorithm for sequential pattern mining of biological sequences

Vance Chiang-Chi Liao · Ming-Syan Chen

Received: 13 July 2011 / Revised: 19 August 2012 / Accepted: 19 December 2012 /
Published online: 26 January 2013

Abstract Scientific progress in recent years has led to the generation of huge amounts of biological data, most of which remains unanalyzed. Mining the data may provide insights into various realms of biology, such as finding co-occurring biosequences, which are essential for biological data mining and analysis. Data mining techniques like sequential pattern mining may reveal implicitly meaningful patterns among the DNA or protein sequences. If biologists hope to unlock the potential of sequential pattern mining in their field, it is necessary to move away from traditional sequential pattern mining algorithms, because they have difficulty handling a small number of items and long sequences in biological data, such as gene and protein sequences. To address the problem, we propose an approach called *Depth-First SPelling* (*DFSP*) algorithm for mining sequential patterns in biological sequences. The algorithm's processing speed is faster than that of *PrefixSpan*, its leading competitor, and it is superior to other sequential pattern mining algorithms for biological sequences.

Keywords Sequential patterns · Pattern mining · Data mining · Bioinformatics

1 Introduction

Biological experiments have generated vast amounts of data, most of which remains unanalyzed. How to process the data and utilize the results in practical applications are therefore important areas of research. Data mining is a step in process used to discover knowledge in a database. In general, the objective of data mining is to discover implicit knowledge and special relationships automatically. Mining the biological data can help biologists discover implicit biological knowledge. Data mining techniques include association

V. C.-C. Liao (✉) · M.-S. Chen
Department of Electrical Engineering, National Taiwan University,
Taipei, Taiwan
e-mail: chiangchi@arbor.ee.ntu.edu.tw

M.-S. Chen
e-mail: mschen@cc.ee.ntu.edu.tw

rules, sequential patterns, classification, clustering, and time series [8,12]. Applying these techniques to process numerous biological data are important issues.

There are some important issues about biological data analysis or biological data mining, such as finding co-occurring biological sequences, effective classification of biological sequences, and cluster analysis of biological sequences [6,11,39]. Sequential pattern mining algorithms facilitate identification of co-occurring biological sequences and the discovery of relationships in DNA or protein sequences [17,18]. In the field of biology, it has been shown that sequential pattern mining is useful for identifying hot regions in protein–protein interactions, predicting transcription factor binding sites, selecting representative characteristics, classifying biosequences, compiling gene regulatory expression profiles, and recognizing protein folds.

Frequent pattern mining and sequential pattern mining have broad applications, such as mining web log patterns, biological patterns, purchasing patterns, and so on. Here are some recent relevant papers, which are published in 2011. Generating web navigation patterns is integrated with semantic information [33]. The text mining and association rule mining methods are used to handle the warranty data [31]. For describing the data, the most interesting itemsets are discovered [23]. For interval-based data, closed temporal patterns are generated efficiently [9]. The specific graph structure is used to generate top-k maximal frequent itemsets [32]. The video event is detected by high-level semantic trajectory [36]. The unified view of the various apriori-based frequency counting methods is proposed for serial episodes [1]. Mining sequential patterns has generated a great deal of interest in the academic community because of its perceived usefulness. As a result, a large number of approaches have been proposed, for example, traditional sequential pattern mining [2,4,5,13,14,27,34,40,41], incremental sequential pattern mining [10,21,25,26], progressive sequential pattern mining [19], closed sequential pattern mining [35,38], maximal sequential pattern mining [22], and sequential pattern mining of data streams [16,24]. In general, traditional sequential pattern mining algorithms have two famous types, *apriori-based* approaches [5,34,41] and *projection-based* approaches [14,27,30] for technique aspects. The two types of algorithms have numerous variations and applications.

In traditional frequent pattern mining methods, *WAP-tree* [29] does not suit for the biological sequential pattern mining problem, because these sequences will make it too much branches. The mining process will cost a lot of memory and execution time. The header table of *H-struct* [28] cannot handle the repeatable items of sequences in sequential pattern mining problem. In the traditional sequential pattern mining problem, a large number of items and shorter sequences are in opposition to biological sequences. Gene and protein sequences have two distinctive features. First, gene and protein sequences are made up of combinations of four letters and twenty letters, respectively. Second, they are usually hundreds or thousands of length. Traditional sequential pattern mining algorithms have difficulty handling a small number of items and long sequences in biological data. Consequently, they are not effective in mining biological sequences. The projection-based pattern growth algorithms mentioned above in traditional sequential pattern mining are used to process long sequences, but the running time is excessive. They need to construct and scan corresponding projected databases many times to generate long sequential patterns. Apriori-based algorithms, the other type of algorithm used in traditional sequential pattern mining, have the same limitation, that is, a long processing time. Furthermore, both types of algorithms are designed to handle a large number of items and shorter sequences, such as transaction sequences; therefore, they cannot process biological sequences efficiently.

In an attempt to move away from traditional sequential pattern mining algorithms and solve the problems they pose, we propose a novel approach called the *Depth-First SPelling* (*DFSP*) algorithm for sequential pattern mining of biological sequences. Compared with the above-mentioned traditional algorithms, DFSP has a shorter running time when mining biological sequences. There are two reasons for its superior performance. First, *the three-dimensional list* of sequences, which has three dimensions, is constructed by only scanning the database once. Second, sequential patterns are generated by our *DFSP-Expansion algorithm,* which involves two steps: (a) depth-first spelling the candidate sequential patterns and (b) verifying the patterns by using direct access to the first two dimensions of the three-dimensional list, and a binary search (finding a larger minimum value) for index in the third dimension. Then, the present sites are written to the counting sequence site, which can be counted for the pattern support. If the candidate patterns satisfy the support constraint, they are regarded as sequential patterns. The DFSP-Expansion component utilizes a depth-first recursive process and keeps running recursively until all the spelling processes stop.

Some traditional biological research and biological computing problems [3,7,15,20] are related to sequential pattern mining problems. In the biological computing domain, motif finding problems involve finding a set of *l-mers* that represent strings of length l,while sequence alignment problems involve computing sequence similarities. Sequential patterns of biological sequences represent to have been conserved in biological sequences during long evolution which may be useful functions in biology. The *2PDF* method [37] was the first sequential pattern mining algorithm designed specifically for biological sequences. In contrast to the complete set of patterns in traditional sequential pattern mining problems, the 2PDF method generates new and different types of patterns, which have the form "$P_1 * P_2 * \cdots * P_k * \cdots * P_{n-1} * P_n$." "$P_i$" denotes a frequent segment. A segment that is longer than *MinLen* (minimum segment length) is called a frequent segment. The symbol "$*$" represents the arbitrary lengths of items or gaps. Segments are extracted from all sequences by a generalized suffix tree. The segment tree (composed of the segments) is used to generate the pattern tree in the 2PDF method. Only setting *MinLen* =1 for the 2PDF method represents that the method mines the complete set of sequential patterns. The complete set of sequential patterns means that the complete set of all length sequential patterns such as $\{\langle A\rangle, \langle T\rangle, \langle C\rangle, \langle G\rangle\}$ may be the complete set of length 1 sequential patterns in DNA sequences. The segment tree in the 2PDF method is too large when *MinLen* =1, and the pattern tree in the method is generated by a combinatorial method; thus, these techniques generate too many patterns (all combinations of the "*" site). For example, if the *DFSP*, *SPAM* [5], or *PrefixSpan* [27] merely generates the pattern *"a*b*c*d,"* the 2PDF method may generate the patterns *"abc*d," "ab*cd," "a*bcd," "ab*c*d," "a*bc*d," "a*b*cd,"* and *"a*b*c*d."* This example obviously shows that the 2PDF method mines too many patterns for biological sequences.

Our complexity analysis and experimental results show that the DFSP algorithm is one of the fastest known algorithms for sequential pattern mining of biological sequences. It is faster than PrefixSpan [27], one of the fastest traditional-type sequential pattern mining algorithms. We find that PrefixSpan is faster and requires less space than SPAM [5], 2PDF-Index, and 2PDF-Compression [37] when mining the complete set of sequential patterns. This experimental result is shown in Fig. 8 in [37]. To evaluate the DFSP algorithm, we conducted a series of experiments. First, we used DFSP and PrefixSpan to process simulated DNA and protein sequences and compared the results. Then, we compared the algorithms' performance in processing various parameters in simulated biological sequences and processing actual biological sequences for validation. The results show that DFSP algorithm processes biological sequences faster than PrefixSpan and it is more scalable.

Several reasons account for DFSP's superior processing speed and scalability. We compare DFSP with projection-based pattern growth algorithms and the 2PDF method. First, unlike traditional projection-based pattern growth algorithms like PrefixSpan, DFSP algorithm does not need to construct corresponding projected databases, so it also does not have to scan that databases. Thus, it saves recursively projecting and scanning time. For example, for a biological sequence $\{X : atacgat, Y : atcacga, Z : taacgca\}$ with minimum support λ equal to *3*, PrefixSpan must first scan all sequences to generate frequent items {*"a," "t," "c," "g"*}. Then, it generates projected databases for *"a," "t," "c,"* and "*g*" individually. The projected databases are $\{tacgat, tcacga, acgca\}$, $\{acgat, cacga, aacgca\}$, $\{gat, acga, gca\}$, and $\{at, a, ca\}$, respectively. After projecting projected database of *"a,"* PrefixSpan scans all the sequences in the projected database of "*a*" to generate frequent items {*"a," "c," "g,"*} and then generates projected databases for *"aa," "ac,"* and *"ag"* individually. PrefixSpan projects the databases recursively until it cannot generate any more frequent items, as shown in Fig. 1.

```
Algorithm DFSP-Expansion (W, C, P)
Require: The three-dimensional list P and the support threshold λ.
Ensure: The complete set of sequential patterns
1.  for S = 1 to N do   /* N is the number of sequences.*/
2.      Initialize flag = false;
3.      Initialize first = 1;
4.      Initialize last = the length of P(W, S) in the list;
5.          /* W is the letter. */
6.          Initialize count = 0;
7.      if C(S) has a value then
8.          /* C is the counting sequence site. */
9.          while first ≦ last
10.             m = (first + last) / 2;
11.             if C(S) less than P(W, S, m) then
12.                 near = P(W, S, m);
13.             last = m - 1;
14.             flag = true;
15.             else
16.                 first = m + 1;
17.             end if
18.         end while
19.     end if
20.     if flag equals true then
21.         C(S) = near;
22.         count = count + 1;
23.     else
24.         C(S) = null;
25.     end if
26.         if count ≧ λ then
27.         break;
28.         end if
29. end for
30. if count ≧ λ then
31.     for each letter W do,
32.         call DFSP-Expansion (W, C, P);
33.     end for
34. end if
```

Second, unlike the 2PDF-Index and 2PDF-Compression algorithms, DFSP does not need to construct a generalized suffix tree. Furthermore, it does not need to link entries, construct segment trees, or maintain auxiliary sets. For example, in the biological sequence $\{X : atacgat, Y : atcacga, Z : taacgca\}$, whose minimum segment length is 1 and minimum support λ is 2, { "*a*," "*t*," "*c*," "*g*," "*at*," "*ac*," "*ta*," "*ca*," "*cg*," "*ga*," "*acg*," "*cga*," and

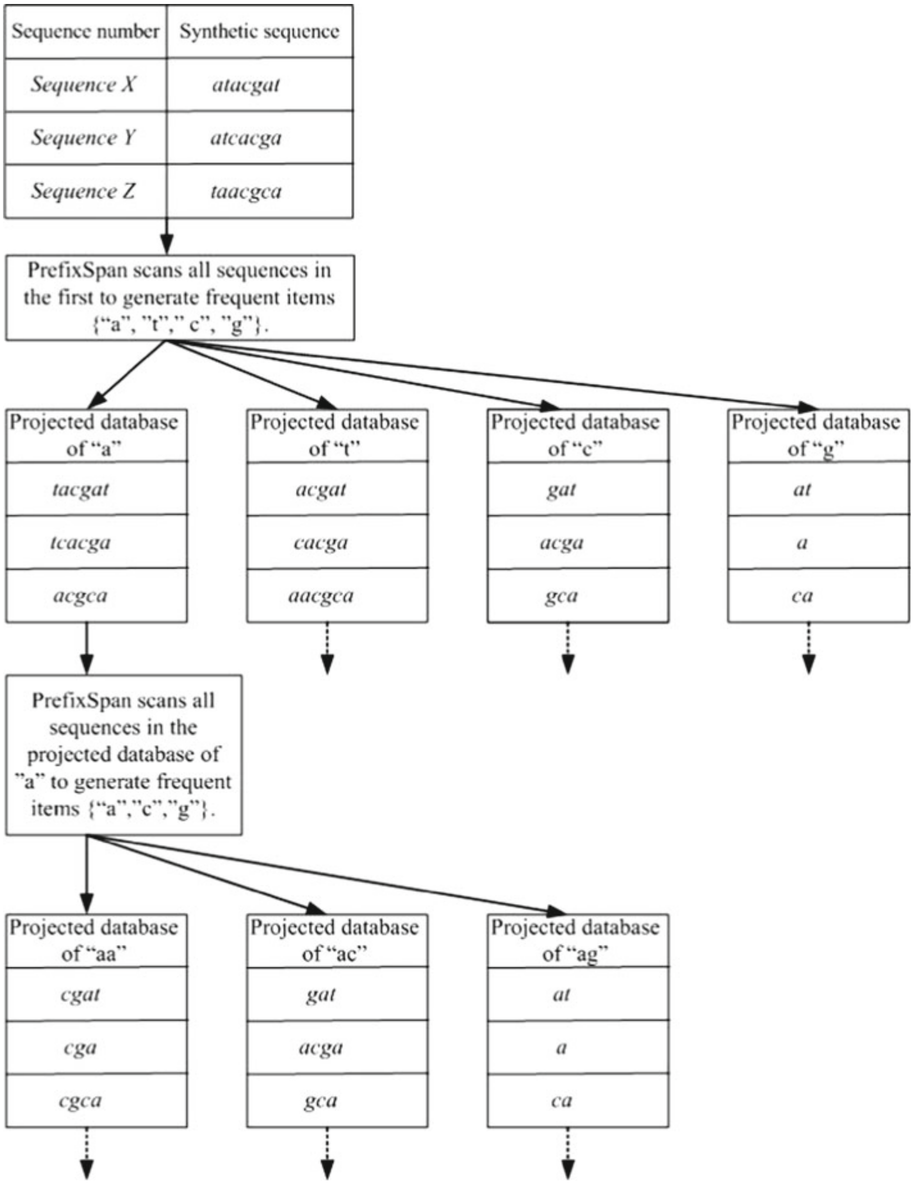
Sequence number
Synthetic sequence
Sequence X
atacgat
Sequence Y
atcacga
Sequence Z
taacgca
PrefixSpan scans all sequences in the first to generate frequent items {"a", "t", "c", "g"}.
Projected database of "a"
tacgat
tcacga
acgca
Projected database of "t"
acgat
cacga
aacgca
Projected database of "c"
gat
acga
gca
Projected database of "g"
at
a
ca
PrefixSpan scans all sequences in the projected database of "a" to generate frequent items {"a","c","g"}.
Projected database of "aa"
cgat
cga
cgca
Projected database of "ac"
gat
acga
gca
Projected database of "ag"
at
a
ca

Fig. 1 An example of the partial PrefixSpan process

"*acga*"} are frequent segments generated from the constructed generalized suffix tree by 2PDF-Index and 2PDF-Compression algorithms. The actions of linking entries for 2PDF-Index are in the SP-index. For example, the leaf entries for the SP-index of the base segment "*a*" are (⟨*s*1, *1*⟩, *ptr*), (⟨*s*1, *3*⟩, *ptr*), (⟨*s*1, *6*⟩, *nil*), (⟨*s*2, *1*⟩, *ptr*), (⟨*s*2, *4*⟩, *ptr*), (⟨*s*2, *7*⟩, *nil*), (⟨*s*3, *2*⟩, *ptr*), (⟨*s*3, *3*⟩, *ptr*), and (⟨*s*3, *7*⟩, *nil*). Each *"ptr"* links to the leaf entry for next base segment in the SP-index, as shown in Fig. 2. Then, the 2PDF method organizes { "*a*," "*t*," "*c*," "*g*," "*at*," "*ac*," "*ta*," "*ca*," "*cg*," "*ga*," "*acg*," "*cga*," "*acga*"} into the segment tree

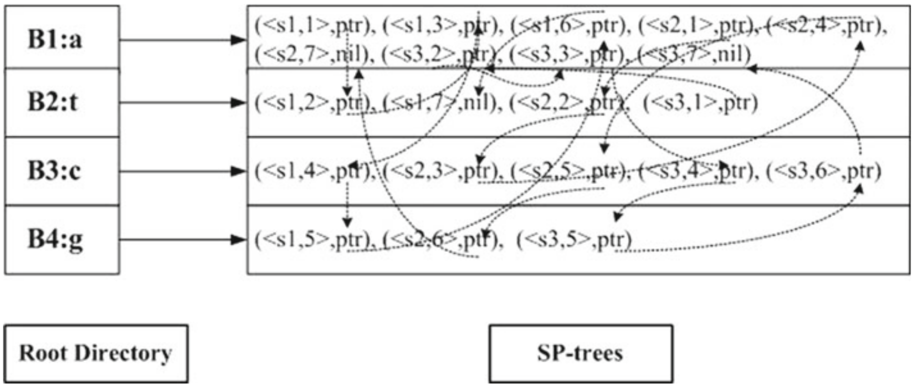
B1:a
B2:t
B3:c
B4:g
(<s1,1>,ptr), (<s1,3>,ptr), (<s1,6>,ptr), (<s2,1>,ptr), (<s2,4>,ptr),
(<s2,7>,nil), (<s3,2>,ptr), (<s3,3>,ptr), (<s3,7>,nil)
(<s1,2>,ptr), (<s1,7>,nil), (<s2,2>,ptr), (<s3,1>,ptr)
(<s1,4>,ptr), (<s2,3>,ptr), (<s2,5>,ptr), (<s3,4>,ptr), (<s3,6>,ptr)
(<s1,5>,ptr), (<s2,6>,ptr), (<s3,5>,ptr)
Root Directory
SP-trees

Fig. 2 An example of the SP-index for 2PDF-index

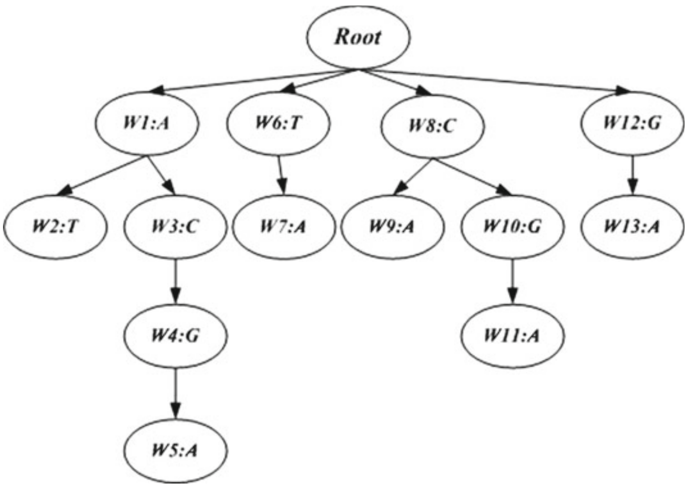

Fig. 3 An example of the segment tree in the 2PDF method

by the prefix order relation shown in Fig. 3 and uses all the segments to recursively generate the frequent patterns shown in Fig. 4. The auxiliary sets of the 2PDF method maintain the items of the set in this branch, which would not occur in the lower level of this branch, but the auxiliary sets need a great deal of maintenance and verification time.

The remainder of this paper is organized as follows. Sect. 2 defines the problem of the sequential pattern mining for biological sequences. The DFSP algorithm and complexity analysis are presented in Sect. 3; the experimental results are discussed in Sect. 4. Section 5 contains some concluding remarks.

2 Problem definition

Let I $= \{i_1, i_2, \ldots, i_A\}$ be the set of all letters. In addition, let $A = 4$ be a simulated DNA sequence and $A = 20$ be a simulated protein sequence. A sequence is an ordered list of letters. A sequence $\boldsymbol{s}$ is denoted by $\{s_1, s_2, s_3, \ldots, s_l\}$, where s_j is a letter. In general, biological

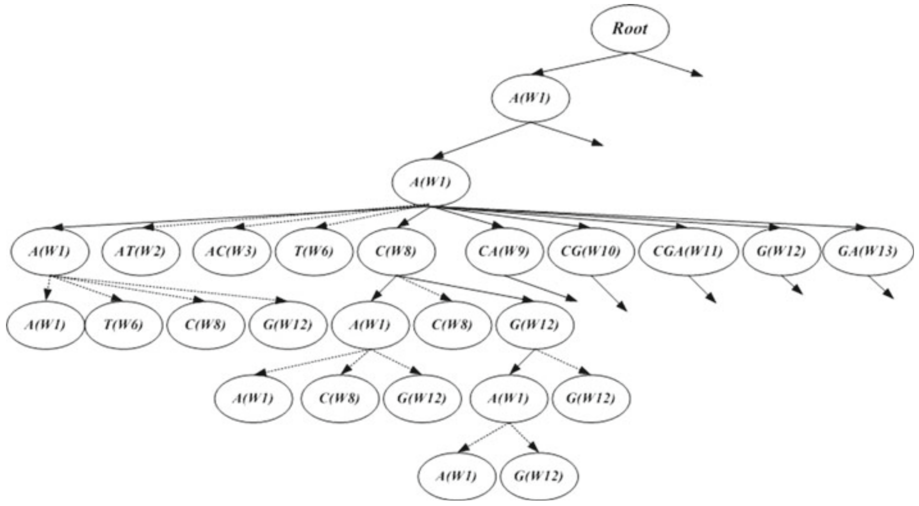

Fig. 4 An example of a partial pattern tree in the 2PDF method

sequences are long, and a letter can appear several times in a sequence. The support count of a sequence α is the number of tuples containing α in the biological DB. If the support count is larger than the minimum support count, then the sequence α is called a sequential pattern. Basically, the problem is not limited with any kinds of biological sequences. The following is an example of a biological sequence: $\{X : atacgat, Y : atcacga, Z : taacgca\}$.

3 The DFSP algorithm and its complexity

In this section, we introduce the Depth-First SPelling (DFSP) algorithm for sequential pattern mining of biological sequences. First, we describe in more detail what the DFSP algorithm is in Sect. 3.1 and then provide an example of the algorithm in Sect. 3.2. Finally, we analyze its complexity.

3.1 The DFSP algorithm

The DFSP algorithm is executed in two steps. First, a three-dimensional list is constructed by scanning the given biological database once, and then the DFSP-Expansion operation generates sequential patterns. Spelling candidate sequential patterns and verifying the patterns are contained in DFSP-Expansion. Direct access and the binary search with the three-dimensional list are included in the verification process. The site of the next occurrence of each letter depends on the letter's prefix in each pattern generating process. Consequently, we design the three-dimensional list and the counting sequence site for DFSP-Expansion. For the counting sequence site, the site of the next occurrence cannot be stored in advance because it is not known and the total number of possible sites for the next occurrence is too large.

Definition 1 (*The three-dimensional list*). The three-dimensional list has three dimensions: the numbers of letter A_i, the numbers of sequence E_j, and the numbers of the site W_k,

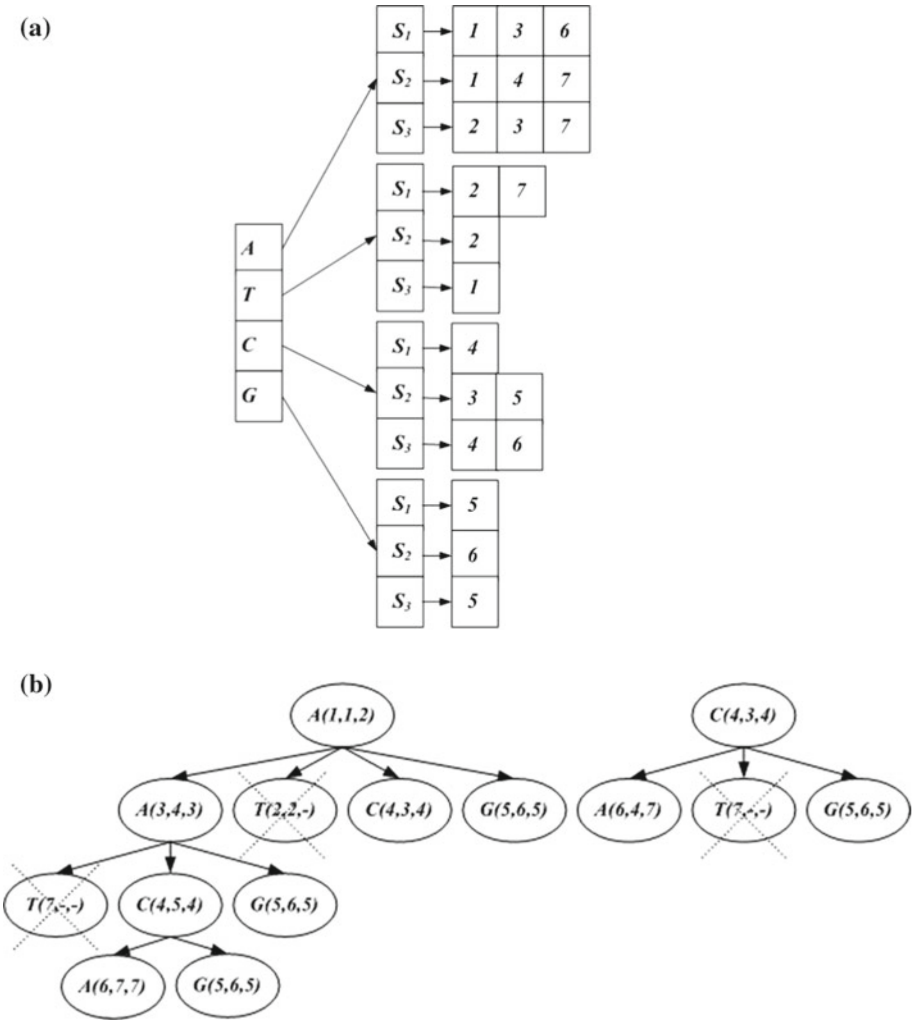

Fig. 5 **a** An example of a three-dimensional list. **b** An example of a partial DFSP-Expansion

which represents the letter A_i of the first dimension occurring in the sequences of the second dimension E_j.

The sequence number E_j is the number of each sequence. The site number W_k is the number of each position. Figure 5a shows an example of a three-dimensional list.

Definition 2 (*Counting sequence site in the three-dimensional list*). The counting sequence site C_l represents W_k in each E_j in the spelling sequential pattern process. Let $\alpha = \langle t_1 t_2 \ldots t_{n-1} \rangle$ be a sequential pattern in a biological database D, and let $\beta = \langle t_1 t_2 \ldots t_{n-1} t_n \rangle$ be a sequence with the prefix α. W_k represents the counting sequence site C_l of β. (W_k may not have a value in each E_j). The value of W_k, which is determined by using the letter $\langle t_n \rangle$ in each E_j on the three-dimensional list, must be larger than the counting sequence site C_l of α.

For the pattern, the counting sequence site records the positions of the latest item in each sequence. The value of each position for the counting sequence site represents that it must be instead by the smallest one of lager positions in the next time. If it can find larger one, it represents contributing one support count to the candidate pattern; otherwise, it does not need to search the sequence below this pattern hereafter.

Definition 3 (*Support count for the DFSP algorithm*). The support count of β is the number of the attributes that have value in counting sequence site C_l. A pattern β is regarded as a sequential pattern only if the support count γ is larger than the minimum support count.

The DFSP algorithm is assured to generate the complete set of sequential patterns. DFSP's pattern generating method is clearly different to that of PrefixSpan, but both algorithms generate the complete set of sequential patterns in the same order. Similar to the proof of PrefixSpan, the problem partitioning technique is used to show that the DFSP generates the complete set of sequential patterns.

3.2 Example: the DFSP algorithm

The following example illustrates the execution of the DFSP algorithm. In this example, DFSP mines a set of letters $\{a, t, c, g\}$ in the sequence database D, $\{X : atacgat, Y : atcacga, Z : taacgca\}$ with a minimum support $\lambda = 3$. The steps are as follows:

1. **Scan D once to build a three-dimensional list, as shown in Fig. 5a.**

DFSP scans the *sequence S* and put the scanned letter A_i into the three-dimensional list with the value W_k.

2. **Find sequential patterns using DFSP-Expansion, as shown in Fig. 5b:**

   (i) DFSP depth-firstly process spells letter I in order to enumerate candidate sequential patterns α.
   (ii) DFSP verifies that the support of candidate pattern α is larger than the minimum support by using the three-dimensional list and the counting sequence site C_l.
   (iii) If the support of a candidate pattern α is larger than the minimum support, the pattern is a real sequential pattern and the process keeps running recursively.

Let us look at the above steps in more detail. In the example shown in Fig. 5a, "T" occurs in positions 2 and 7 of the sequence *X, which* represents S_1. In Fig. 5b, the depth-firstly process spells the candidate pattern "CA." The values in dimension "A" of the three-dimensional list are searched in order to find a minimum value that is larger than the value in the *Cl* site. This search technique is the binary search. Its pseudo codes are from line 9 to line 18 in Algorithm DFSP-Expansion. We find that 6 is larger than 4 in S_1, 4 is larger than 3 in S_2, and 7 is larger than 4 in S_3. The new *Cl* (6, 4, 7) is larger than the previous *Cl* (4, 3, 4). The *support* is greater than the *minimum support* 3, so the candidate pattern "CA" is a sequential pattern (*support* = 3). The process continues to depth-firstly spell and verify candidate sequential patterns. Then, another candidate pattern "AT" is looked at. The values in dimension "T" of the three-dimensional list are searched. No value is larger than 2 in S_3, so the new *Cl* is (2, 2, -) and the *support* is 2. Thus, the candidate pattern "AT" is not a sequential pattern. Since this spelling candidate pattern fails, the process will not continue to spell subsequent candidate patterns after this candidate pattern. The counting sequence site and the binary search technologies are effective especially in longer candidate patterns and more sequences. DFSP outperforms PrefixSpan in terms of mining sequential patterns of the complete set in biological sequences because of its conciseness, suitability (for few letters and long sequence lengths), and fewer redundant actions (in opposition to other sequential pattern mining algorithms for biological sequences).

3.3 Complexity analysis

PrefixSpan, one of the fastest traditional algorithms, mines the complete set of sequential patterns faster than SPAM, 2PDF-Index, and 2PDF-Compression. We therefore analyze and compare the time complexity of the proposed DFSP algorithm with that of PrefixSpan as opposed to any of the others. The sequential patterns identified by the DFSP are the same as PrefixSpan. In the following discussion, "L" denotes the length of a sequence, "A" is the number of letters, "N" is the number of sequences, and "P" is the projecting time of an item. We make two assumptions: (i) The minimum support count is 1 and (ii) the items are in uniform distribution. The assumptions do not have loss of fairness to compare the complexity of DFSP with that of PrefixSpan.

The process for deriving the complexity of our algorithm is explained below. The time complexity of scanning a database to build the three-dimensional list is $O(LN)$; the number of building nodes for DFSP-Expansion is less than $O(2^L - 2^{L-A}) = O(2^L)$; and the number of next level nodes created in the spelling process is less than $O(A)$. The average value for the numbers of site number in the three-dimensional list is (L/A). Each verification of a candidate sequential pattern with the three-dimensional list takes $O(N(\log_2 \frac{L}{A}))$ time. The total time complexity of DFSP is $O(LN) + O(2^L) \times O(A) \times O(N(\log_2 \frac{L}{A})) = O(LN + 2^L AN(\log_2 \frac{L}{A}))$. For biological sequences where L is large and A is small, the total time complexity of DFSP is $O(2^L N(\log_2 L))$.

PrefixSpan requires $O(LN)$ time to scan a database and count the number of frequent items. The number of building projected databases for PrefixSpan is less than $O(2^L - 2^{L-A}) = O(2^L)$. The algorithm scans each sequence to count frequent items and project them to the next level of process; thus, the time cost is $O(NLA + PNLA)$. $O(LN) + O(2^L) \times O(NLA + PNLA) = O(LN + 2^L ANL + 2^L PANL)$ is the total time complexity of PrefixSpan. $O(2^L NL + 2^L PNL)$ is the total time complexity of PrefixSpan for biological sequences in which L is large and A is small.

The above complexities, $O(LN + 2^L AN(\log_2 \frac{L}{A}))$ and $O(LN + 2^L ANL + 2^L PANL)$, show that DFSP algorithm can mine a database faster than PrefixSpan when L is large and A is small. In biological sequences, L is large and A is small; thus, the time complexity is $O(2^L N(\log_2 L))$ and $O(2^L NL + 2^L PNL)$ for DFSP and PrefixSpan, respectively. $O(\log_2 L)$ and $O(L + PL)$ is the major difference between these two algorithms, so DFSP can mine biological sequences much faster than PrefixSpan.

4 Performance evaluation

We conducted a number of experiments to evaluate the performance of DFSP. In Sect. 4.1, we compare DFSP's performance with that of PrefixSpan in mining simulated DNA and protein sequences. In Sect. 4.2, we test various parameters and real biological data for the two algorithms; in Sect. 4.3, we explain why DFSP outperforms PrefixSpan. All the experiments were performed on a 3.20 GHz Pentium(R) 4 PC with 1 GB of RAM. The operating system was Microsoft Windows XP Professional (2002); the programs were written in Microsoft Visual C++ 6.0.

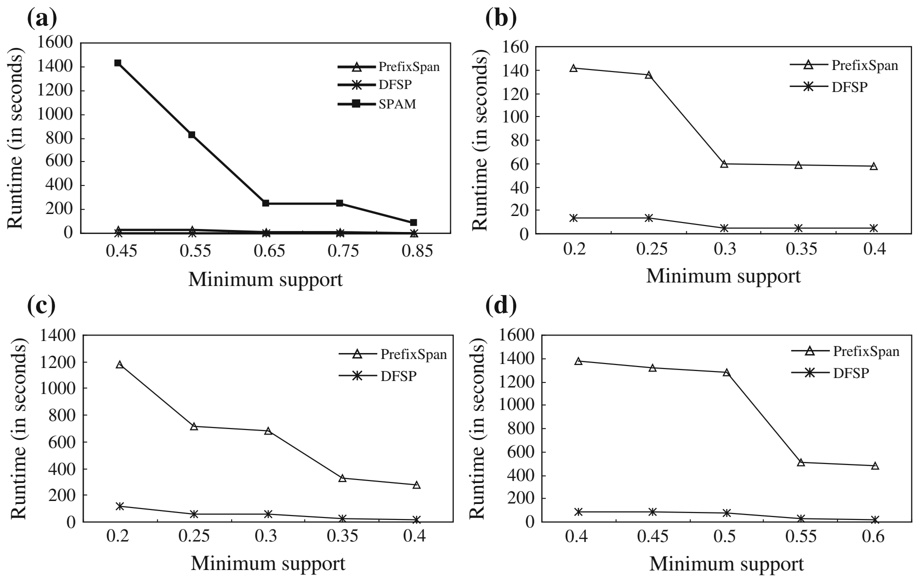

Fig. 6 **a** Execution Time ($L = 25$, $A = 4$, and $N = 1{,}000$). **b** Execution Time ($L = 25$, $A = 4$, and $N = 1{,}000$). **c** Execution Time ($L = 30$, $A = 4$, and $N = 1{,}000$). **d** Execution Time ($L = 35$, $A = 4$, and $N = 1{,}000$)

4.1 Simulated DNA and protein sequences

The synthetic DNA and protein sequence data used in the experiments followed [34]. In the following discussion, L represents the length of a sequence, A is the number of letters, N is the number of sequences, and S is the minimum support.

When $A = 4$ (for simulated DNA sequences), L is increased from 25 to 35, as shown in Fig. 6a–d. According to Fig. 6a, both DFSP and PrefixSpan mine the data much faster than SPAM. However, Fig. 6b–d show that DFSP outperforms PrefixSpan on the datasets. In each figure, the horizontal axis represents the minimum support, and the vertical axis represents the runtime (in seconds). The number of letters, which are simulated DNA sequences in this experiment, is 4; the number of sequences is 1,000; and the lengths of the sequences are 25, 30, and 35. The runtime rate is equal to PrefixSpan's runtime divided by that of DFSP. The runtime rates in Fig. 6d are 15.41, 16.03, 16.07, 20.33, and 20.38. The rate increases as the minimum support gets larger. In other words, the runtime of one is lower than the other.

When $A = 20$ (for simulated protein sequences), L is increased from 25 to 35, as shown in Fig. 7a–d. Figure 7a shows that DFSP and PrefixSpan perform much faster than SPAM. However, as shown in Figs. 7b–d, DFSP outperforms PrefixSpan. The number of letters, which are simulated protein sequences in this experiment, is 20; the number of sequences is 1,000; and the lengths of the sequences are 25, 30, and 35. Compared to the results in Figs. 6b–d, when A gets larger and L remains the same, the runtime is shorter. Both DFSP and PrefixSpan generate far fewer sequential patterns where A increases, but L and S remain the same. In situations where the minimum support count is 1, A gets larger, and$L$ remains the same, the runtimes of DFSP and PrefixSpan are longer than in situations where the minimum support is large enough. This is because the algorithms have to generate more sequential patterns. The runtime rates in Fig. 7d are 4.95, 5.06, 5.10, 5.27, and 5.52. The rate increases

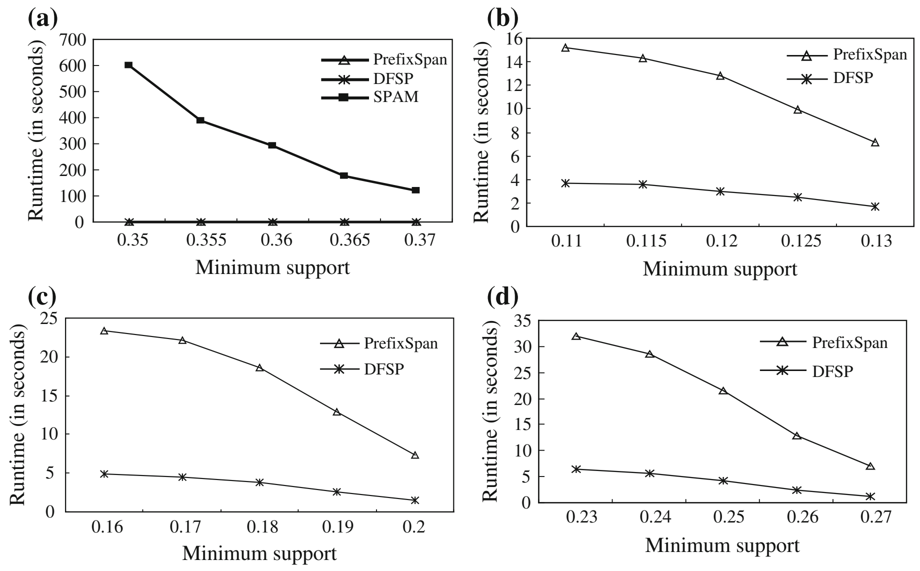

Fig. 7 **a** Execution Time ($L = 25$, $A = 20$, and $N = 1{,}000$). **b** Execution Time ($L = 25$, $A = 20$, and $N = 1{,}000$). **c** Execution Time ($L = 30$, $A = 20$, and $N = 1{,}000$). **d** Execution Time ($L = 35$, $A = 20$, and $N = 1{,}000$)

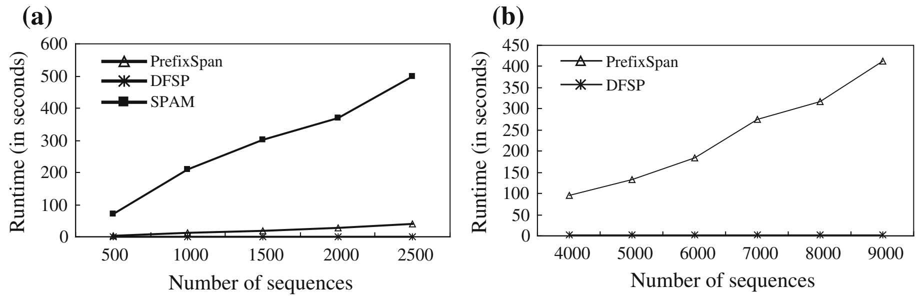

Fig. 8 **a** Execution Time ($L = 30$, $A = 4$, and $S = 0.9$). **b** Execution Time ($L = 30$, $A = 4$, and $S = 0.9$)

as the minimum support gets larger. With A = 4 and A = 20, the runtime rate is lower when the minimum support is lower, or A is larger. The larger the value of A, the lower will be the runtime rate.

4.2 Various parameters and real data

We used various parameters and real biological sequences to compare the performance of DFSP with that of PrefixSpan. **When N is increased**, both algorithms mine much faster than SPAM as shown in Fig. 8a. However, DFSP outperforms PrefixSpan when N is larger (Fig. 8b). The length of the sequences is 30; the number of simulated DNA sequences is 4; the minimum support is 0.9; and the numbers of sequences are 4,000, 5,000, 6,000, 7,000, 8,000, and 9,000. Moreover, although PrefixSpan is scalable [14,27], DFSP is more scalable. In Fig. 8b, DFSP's runtimes look like a straight line due to the proportional scale. The runtimes are 1.421 s (4,000 sequences), 1.640 s (5,000 sequences), 1.953 s (6,000 sequences), 2.593 s

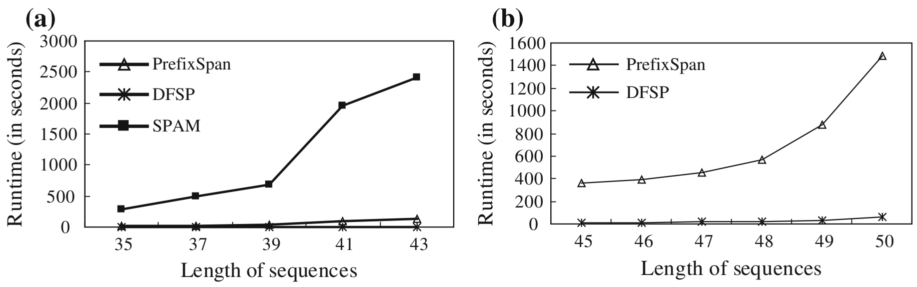

Fig. 9 **a** Execution Time ($A = 4$, $N = 500$, and $S = 0.9$). **b** Execution Time ($A = 4$, $N = 500$, and $S = 0.9$)

(7,000 sequences), 2.671 s (8,000 sequences), and 3.125 s (9,000 sequences). The variations are not as obvious in Fig. 8b as they are from the runtime rates (67.44, 81.48, 94.57, 106.08, 118.67, and 131.71). The runtime rate increases as the number of sequences increases.

When *L* is increased, DFSP and PrefixSpan perform much faster than SPAM, as shown in Fig. 9a. However, DFSP algorithm mines much faster than PrefixSpan when L is larger (Fig. 9b). The number of simulated DNA sequences is 4; the number of sequences is 500; the minimum support is 0.9; and the lengths of the sequences are 45, 46, 47, 48, 49, and 50. The runtime of DFSP shows a steady rise in runtime as L increases when compared to PrefixSpan. In the experiment, the maximum sequence length is 50, because we have already shown that DFSP is significantly faster than PrefixSpan and the latter's runtime increases exponentially. Additionally, if the maximum sequence length is larger than 50, the runtime of DFSP will be looked like a straight line in the figure due to the proportional scale. Thus, we will hardly see the differences of runtimes for DFSP among different length of sequences in the figure.

To evaluate the algorithms, we used real DNA sequences obtained from the National Center for Biotechnology Information (NCBI). **When *A* = 4 (real DNA sequences), *L* is increased from 25 to 35.** As shown in Fig. 10a, DFSP and PrefixSpan mine sequences much faster than SPAM. However, DFSP outperforms PrefixSpan as shown in Fig. 10b–d. Here, the number of real DNA sequences is 4; the number of sequences is 1,000; and the lengths of the sequences are 25, 30, and 35. DFSP mines real DNA sequences even faster than it mines synthetic ones. The runtime rates in Fig. 10d are 16.05, 17.62, 19.30, 20.68, and 22.57. Invariably, the rate increases as the minimum support increases. The runtime rates for the real DNA sequences are almost the same as those for the synthetic ones in the previous experiment.

***N* is increased from** 200 to 900 k to assess the scalability of DFSP. The numbers of sequences are 200, 300, 400, 500, 600, 700, 800, and 900 k, and DFSP's runtimes are 98.16, 142.44, 187.86, 231.66, 277.47, 328.47, 375.19, and 421.34 s respectively. The results show that DFSP is more scalable when the datasets are larger. The growth rate of the execution time is steady and small. Here, the length of the sequences is 30; the number of letters in the simulated DNA sequences is 4; and the minimum support is 0.9. The experimental results show that DFSP outperforms PrefixSpan on real biological sequences and in various parameters including scalability for simulated sequences.

4.3 Performance analysis

DFSP's time cost is substantially lower than that of PrefixSpan because of its conciseness. There are three reasons for its superior performance. First, unlike PrefixSpan, our algorithm

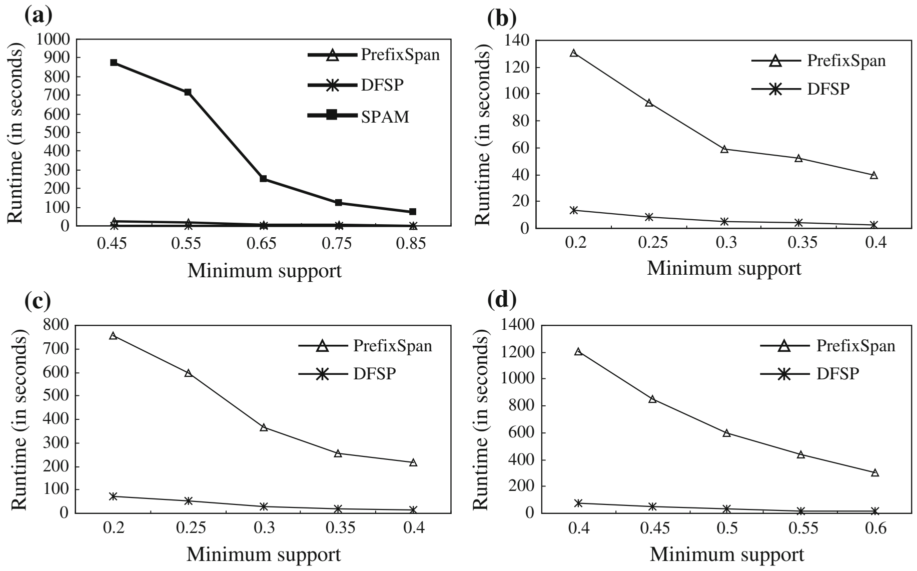

Fig. 10 **a** Execution Time ($L = 25$, $A = 4$, and $N = 1{,}000$). **b** Execution Time ($L = 25$, $A = 4$, and $N = 1{,}000$). **c** Execution Time ($L = 30$, $A = 4$, and $N = 1{,}000$). **d** Execution Time ($L = 35$, $A = 4$, and $N = 1{,}000$)

does not need to construct and scan corresponding projected databases. Consequently, it does not require recursively projecting and scanning corresponding databases, thereby saving on execution time. The DFSP algorithm spells the candidate patterns directly and verifies them by using the three-dimensional list. The combination of the three-dimensional list with simultaneous spelling and verification enables the algorithm to mine significantly faster than other sequential pattern mining algorithms.

The second reason is that DFSP mines much faster than PrefixSpan when the number of sequences is large. Additionally, the runtime rate (runtime of PrefixSpan/runtime of DFSP) is high when the minimum support is large. Because our spelling method of DFSP algorithm takes almost as few frequencies as projected database of PrefixSpan does when the support of sequences is high, but PrefixSpan almost scans the same long length of lower-level projected database as support is small in each projected process. In other words, when the minimum support or the number of sequences is large, there are fewer sequential patterns. This situation reduces the number of building projected DB for PrefixSpan and represents having the lower layers of projected databases, which have longer sequences than high layers of projected databases.

Third, the DFSP algorithm mines the complete set of biological sequential patterns faster than the 2PDF-Index and 2PDF-Compression algorithms. Mining the complete set of sequential patterns is the original prototype in sequential pattern mining. Sequential pattern mining for the complete set is more flexible (a generalized model) because it has a lot of variations to mining specific sequential patterns and it is intuitional to achieve the basic variations. On the other hand, a new constraint, *MinLen,* is incorporated into 2PDF-Index and 2PDF-Compression, but the authors do not explain why they use *MinLen* to generate new types of patterns. Before mining a dataset, the 2PDF-Index and 2PDF-Compression algorithms first make generalized suffix trees and then add the *MinLen* constraint. In contrast, DFSP does not need to (1) construct generalized suffix trees; (2) apply constraints like *MinLen* (single item versus base/frequent segment); (3) link entries; or (4) construct segment trees. Moreover, it does not need to consider the segment length or maintain sets like *v.dead*, which means that items of the set in this branch would not occur in the lower level of this branch but much maintaining and verifying time is needed for the sets. The DFSP algorithm does away with these features of previous algorithms by way of spelling patterns instead of generating pattern trees using segment trees. It first creates the three-dimensional list instead of operating on the complex structures of multiple modified *B-trees* (or *SP tree* in certain literature). It accesses the three-dimensional list directly for the dataset and performs a binary search on the index. These modifications yield a difference in complexity that reduces running time substantially. Constructing auxiliary structures (*the three-dimensional list* and *SP-index*) and searching have a significant influence on the running time, as the complexity of DFSP and that of the 2PDF method demonstrate: $O(LN+\log_2 \frac{L}{A})$ and $O(2LN+LNT\log_T \frac{LN}{A}+AT\log_T \frac{LN}{A})$, respectively, where “$A$” denotes the number of letters and “T” is the minimum degree of the B-tree.

5 Conclusion

The proposed DFSP algorithm can mine biological (DNA and protein) sequences with a small number of letters and long sequence lengths faster than PrefixSpan. This is because it uses a direct spelling method to enumerate candidate patterns, the three-dimensional list to verify the patterns, and a counting sequence site to prune the searching space. To process small items and long sequences such as biological sequences is suited for the DFSP because of these above technologies. The implicit knowledge in biological sequences is valuable, for example, it may facilitate the identification of hot regions in protein–protein interactions, among other things. In our future research, we will design faster algorithms and other algorithms that aid in mining biological data.

References

1. Achar A, Laxman S, Sastry PS (2011) A unified view of the apriori-based algorithms for frequent episode discovery. Knowl Inf Syst 31: 223–250
2. Agrawal R and Srikant R (1995) Mining sequential patterns. Proceedings of the 11th international conference on data, engineering, pp 3–14
3. Altschul SF, Gish W, Miller W, Myers EW, Lipman DJ (1990) Basic local alignment search tool. J Mol Biol 215(3):403–410
4. Aseervatham S, Osmani A, Viennet E (2006) bitSPADE: A lattice-based sequential pattern mining algorithm using bitmap representation. In: Proceedings of 6th international conference on data mining, pp 792–797
5. Ayres J, Gehrke J, Yiu T, Flannick J (2002), Sequential pattern mining using a bitmap representation. In: Proceedings of 8th ACM SIGKDD international conference on knowledge discovery and data mining, pp 429–435, July 2002
6. Bajcsy P, Han J, Liu L, Young J (2004) Survey of biodata analysis from a data mining perspective. Wang JTL, Zaki MJ, Toivonen HTT, and Shasha D (eds) Data Mining in Bioinformatics, Chapter 2. Springer, pp 9–39
7. BLAST, http://www.ncbi.nlm.nih.gov/Education/BLASTinfo/
8. Chen M-S, Han J, Yu PS (1996) Data mining: an overview from database perspective. IEEE Trans Knowl Data Eng 5(1):866–883
9. Chen Y-C, Peng W-C, Lee S-Y (2011) CEMiner—An efficient algorithms for mining closed patterns from interval-based data. In: Proceedings of the 11th IEEE international conference on data mining (ICDM). Vancouver, Canada, pp 121–130, Dec 11–14

10. Cheng H, Yan X, Han J (2004) Incspan: incremental mining of sequential patterns in large database. In: Proceedings of 10th ACM SIGKDD international conference on knowledge discovery and data mining, pp 527–532
11. Han J (2002) How can data mining help bio-data analysis? In: Proceedings of the workshop on data mining in bioinformatics (BIOKDD'02 with SIGKDD'02 conference. Edmonton, Canada), pp 1–4
12. Han J, Kamber M (2000) Data mining: concepts and techniques. Morgan Kaufmann, San Francisco
13. Han J, Pei J, Mortazavi-Asl B, Chen Q, Dayal U, Hsu MC (2000) Freespan: frequent pattern-projected sequential pattern mining. In: Proceedings of the 6th ACM SIGKDD international conference on knowledge discovery and data mining, pp 355–359
14. Han J, Pei J, Yan X (2005) Sequential pattern mining by pattern-growth: principles and extensions. In: Chu WW, lIN TY (eds) Recent advances in data mining and granular computing. Springer, Berlin, pp 183–220
15. Hirosawa M, Totoki Y, Hoshida M, Ishikawa M (1995) Comprehensive study on iterative algorithms of multiple sequence alignment. Bioinformatics 11:13–18
16. Ho C-C, Li H-F, Kuo F-F, Lee S-Y (2006) Incremental mining of sequential patterns over a stream sliding window. In: Proceedings of IEEE international workshop on mining evolving and streaming data (IWMESD-2006), pp 677–681, Dec 2006
17. Hsu C-M, Chen C-Y, Liu B-J (2006) MAGIIC-PRO: detecting functional signatures by efficient discovery of long patterns in protein sequences. Nucleic Acids Res, W356–W361
18. Hsu C-M, Chen CY, Liu BJ, Huang CC, Laio MH, Lin CC, Wu TL (2007) Identification of hot regions in protein-protein interactions by sequential pattern mining. BMC Bioinform 8(Suppl. 5):S8. doi:10.1186/1471-2105-8-S5-S8
19. Huang J-W, Tseng C-Y, Ou J-C, Chen M-S (2008) A General Model for Sequential Pattern Mining with a Progressive Database. IEEE Trans Knowl Data Eng, 20: 1153–1167, 11 Feb 2008
20. Jones N, Pevzner P (2004) An introduction to bioinformatics algorithms. MIT Press, Cambridge
21. Lin M-Y, Lee S-Y (2004) Incremental update on sequential patterns in large databases by implicit merging and efficient counting. Inf Syst 29(5):385–404
22. Luo C, Chung SM (2005) Efficient mining of maximal sequential patterns using multiple samples. In: Proceedings of the 5th SIAM international conference on data mining (SDM'05), pp 415–426
23. Mampaey M, Tatti N, Vreeken J (2011) Tell me what I need to know: succinctly summarizing data with itemsets. In: Proceedings of 17th ACM SIGKDD international conference on knowledge discovery and data mining, pp 573–581
24. Marascu A, Masseglia F (2006) Mining sequential patterns from data streams: a centroid approach. J Intell Inf Syst (JIIS) 27(3):291–307
25. Nguyen S, Sun X, Orlowska M (2005) Improvements of IncSpan: incremental mining of sequential patterns in large database. In: Proceedings of the 9th Pacific-Asia conference on knowledge discovery and data mining, pp 442–451
26. Parthasarathy S, Zaki MJ, Ogihara M, Dwarkadas S (1999) Incremental and interactive sequence mining. In: Proceedings of the 8th international conference on information and, knowledge management, pp 251–258
27. Pei J, Han J, 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, pp 1424–1440, Oct 2004
28. Pei J, Han J, Lu H, Nishio S, Tang S, Yang D (2001) H-mine: hyper-structure mining of frequent patterns in large databases. In: Proceedings of the 2001 IEEE international conference on data mining (ICDM'01), San Jose, California, pp 441–448, Nov 29–Dec 2
29. Pei J, Han J, Mortazavi-Asl B, Zhu H (2000) Mining access patterns efficiently from Web logs. In: Proceedings of the 2000 Pacific-Asia conference on knowledge discovery and data mining (PAKDD'00). Kyoto, Japan, April, pp 396–407
30. Pei J, Han J, Wang W (2007) Constraint-based sequential pattern mining: the pattern-growth methods. J Intell Inf Syst 28(2):133–160
31. Rajpathak D, Chougule R, Bandyopadhyay P (2011) A domain-specific decision support system for knowledge discovery using association and text mining. Knowl Inf Syst, pp 405–432
32. Salam A, Sikandar Hayat Khayal M (2011) Mining top-k frequent patterns without minimum support threshold. Knowl Inf Syst, pp 57–86
33. Senkul P, Salin S (2011) Improving pattern quality in web usage mining by using semantic information. Knowl Inf Syst, pp 527–541
34. Srikant R and Agrawal R (1996) Mining sequential patterns: generalizations and performance improvements. In: Proceedings of 5th international conference extending database technology (EDBT), vol 1057, Springer, pp 3–17

35. Wang J, Han J (2004) Bide: efficient mining of frequent closed sequences. In: Proceedings of the 20th international conference on data, engineering, pp 79–91
36. Wang X, Li G, Jiang G, Shi Z (2011) Semantic trajectory-based event detection and event pattern mining. Knowl Inf Syst, pp 1–25
37. Wang K, Xu Y, Yu J (2004) Scalable sequential pattern mining for biological sequences. In: Proceedings of conference information and, knowledge management, pp 178–187
38. Yan X, Han J, Afshar R (2003) Clospan: Mining closed sequential patterns in large datasets. In: Proceedings of the 3rd SIAM international conference on data mining (SDM'03), pp 166–177, May 2003
39. Yang J, Wang W, Yu PS, Han J (2002) Mining long sequential patterns in a noisy environment, In: Proceedings 2002 ACM-SIGMOD I international conference. Management of data (SIGMOD '02), pp 406–417, June 2002
40. Zaki MJ (1998) Efficient enumeration of frequent sequences. In: Proceedings of the 7th international conference on information and knowledge management, pp 68–75, Nov 1998
41. Zaki MJ (2001) SPADE: an efficient algorithm for mining frequent sequences. Machine learning 42(1/2):31–60

Author Biographies

Vance Chiang-Chi Liao is currently a Ph.D. candidate in the Department of Electrical Engineering, National Taiwan University, Taipei, Taiwan. His research interests include data mining, sequential pattern mining, frequent pattern mining, bioinformatics, social networks, and cloud computing.

Ming-Syan Chen received the B.S. degree in electrical engineering from National Taiwan University, Taipei, Taiwan, and the M.S. and Ph.D. degrees in Computer, Information, and Control Engineering from The University of Michigan, Ann Arbor, MI, USA, in 1985 and 1988, respectively. He is now a Distinguished Research Fellow and the Director of Research Center of Information Technology Innovation (CITI) in the Academia Sinica, Taiwan, and is also a Distinguished Professor jointly appointed by EE Department, CSIE Department, and Graduate Institute of Communication Eng. (GICE) at National Taiwan University. He was a research staff member at IBM Thomas J. Watson Research Center, Yorktown Heights, NY, USA, the Director of GICE, and also the President/CEO of Institute for Information Industry (III). His research interests include databases, data mining, cloud computing, and multimedia networking, and he has published more than 300 papers in his research areas.