International Journal of Innovative
Computing, Information and Control
Volume **8**, Number **1(B)**, January **2012**

ICIC International ©2012 ISSN 1349-4198
pp. 647–657

MINING SEQUENCE MOTIFS FROM PROTEIN DATABASES BASED ON A BIT PATTERN APPROACH

Ye-In Chang, Chen-Chang Wu, Jiun-Rung Chen and Yin-Han Jeng

Department of Computer Science and Engineering
National Sun Yat-Sen University
No. 70, Lienhai Rd., Kaohsiung 80424, Taiwan
changyi@cse.nsysu.edu.tw; { wucc; chenjr; jengyh }@db.cse.nsysu.edu.tw

Received September 2010; revised April 2011

Abstract. *Proteins are the structural components of living cells and tissues, and thus an important building block in all living organisms. Sequence motifs in proteins are some subsequences which appear frequently. Motifs often denote important functional regions in proteins and can be used to characterize a protein family or discover the function of proteins. The SP-index algorithm was proposed to find sequence motifs containing gaps of arbitrary size. To find motifs, it constructs B-trees for indexing the occurring positions of short segments. Then, to check whether a long pattern composed of short segments appears frequently, the SP-index algorithm needs to test a large number of nodes of those B-trees, which may not be efficient. Therefore, in this paper, we propose the Bit-Pattern-based (BP) algorithm to improve the efficiency of the SP-index algorithm. First, the BP algorithm transforms the protein sequences into bit patterns. Then, instead of testing a large number of nodes in the SP-index algorithm, the BP algorithm utilizes bit operations, i.e., AND, OR, shifting and masking, to efficiently find sequence motifs. The BP algorithm also performs a pruning step to reduce the processing time. From the experimental results on biological and synthetic data sets, we show that the BP algorithm needs shorter processing time than the SP-index algorithm.*
Keywords: Bit pattern, Data mining, Motif, Protein, Sequential pattern

1. **Introduction.** Proteins are the structural components of living cells and tissues, and thus an important building block in all living organisms. The normal protein size is a hundred amino acids, while large proteins can reach over a thousand amino acids [1, 12]. Only 20 different amino acids, i.e., {A, C, D, E, F, G, H, I, K, L, M, N, P, Q, R, S, T, V, W, Y}, make up the diverse array of proteins found in living things [2]. Protein sequences have the form of $X_1 * X_2 * \ldots * X_n$, where each X_k is a short consecutive items, called a segment, and $*$ denotes a variable length gap [8, 15]. One of the important tasks is the discovery of sequence motifs from the primary structure of proteins [7]. Sequence motifs are frequent patterns in the protein sequences, which typically correspond to residues conserved during evolution due to an important structural or functional role. Finding frequent patterns is often the first step in sequence analysis such as classifying protein sequences, extracting protein features, identifying transcription factor binding sites and predicting protein structures [15, 17]. In this paper, we will focus on how to efficiently find frequent patterns with gaps of arbitrary size from a large database of protein sequences.

There were two approaches proposed to find deterministic motifs: sequence driven (SD) and pattern driven (PD). For algorithms based on the SD approach, they do global multiple sequence alignments to find the frequent patterns in protein sequences. However, finding the optimal multiple alignment is an NP-hard problem [17]. Therefore, SD-based algorithms often use heuristic algorithms to obtain the approximate multiple alignments

to improve the efficiency, and cannot guarantee to find the optimal results [6]. Many SD-based algorithms improve the string matching skills to solve the problem, e.g., suffix trees and the BLAST algorithm [4].

For algorithms based on the PD approach, they often use the data mining skills. When mining protein sequence patterns, we need to consider gaps and order in the patterns. Therefore, the problem can be considered as a sequential patterns mining (SPM) problem [3]. The original SPM is studied in market-basket analysis, where a transaction sequence can be a purchase sequence, a web link stream, etc. [3, 10, 13, 14, 16]. Traditional algorithms of SPM were usually designed for short sequences over a large alphabet, but may not be efficient for protein sequences with a long length and a small alphabet. Moreover, in SPM, a sequential pattern does not consider that one item is purchased more than once at the same time, while in protein sequence discovering, multiple appearances of one animo acid at the same time, e.g., "MM∗NN", are allowed. Therefore, skills used in algorithms of SPM, e.g., the SPAM algorithm [5], need to be improved to be adaptable to the problem of protein sequence discovering for PD-based algorithms.

Based on the traditional SPM algorithms, Wang et al. proposed the SP-index algorithm to find the longest sequential pattern with gaps of arbitrary size [15]. The SP-index algorithm considers the multiple-appearance problem and the characteristics of bioinformatics. Although the SP-index algorithm considers the characteristics of bioinformatics, it still contains some time-consuming steps. The first phase in the SP-index algorithm is producing all segments. Then, these segments are used to find the longest pattern in the second phase. There are too many unnecessary segments for producing the longest pattern. The SP-index algorithm uses the base segments to produce the SP-tree. The more the base segments are, the longer the time of tracing the tree is. In the second phase, the SP-index algorithm uses the SP-tree to check whether segment A can be linked after segment B or not. This step needs long time, since it has to trace many nodes to get the result. If there are many unnecessary segments in the first phase, the second phase is time-consuming, since the number of segments is too large.

Therefore, to avoid the above disadvantages, we propose the *Bit-Pattern-Based* (BP) algorithm to mine sequential patterns in a protein database. In our proposed algorithm, first, we transform the protein sequences into bit sequences. Second, we use shifting and the AND operator to get frequent segments. After getting frequent segments, we prune unnecessary segments. Finally, we use OR and mask operators to get sequential patterns. Because we reduce the number of candidate segments and apply bit operations, our algorithm will be more efficient than the SP-index algorithm. From our performance study based on the biological data and the synthetic data, we show that our proposed algorithm is more efficient than the SP-index algorithm.

The rest of this paper is organized as follows: Section 2 briefly introduces the SP-index algorithm; Section 3 presents the proposed BP algorithm; in Section 4, we study the performance of the BP algorithm and make a comparison with the SP-index algorithm; finally, we give conclusions in Section 5.

2. **The SP-index Algorithm.**

The SP-index algorithm [15] uses a scalable two-phase algorithm to deal with mining frequent patterns from protein sequences. First, the segment phase searches for frequent segments containing no gap, and generates base segments from these frequent segments. An SP-index is built to index the occurring positions of these base segments. Then, by utilizing the SP-index, the pattern phase searches for the long frequent patterns containing multiple frequent segments separated by gaps with variable lengths. Table 1 shows an example database of protein sequences, DB.

TABLE 1. The example database

ID	Sequence
S_1	ABDACDAB
S_2	ABACDA
S_3	AACDC

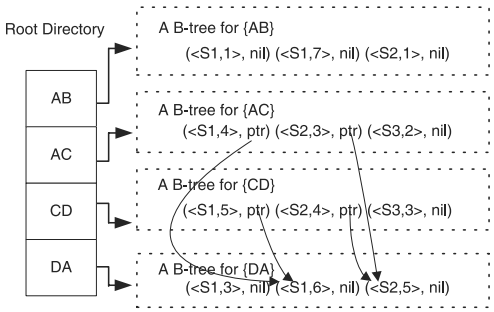

FIGURE 1. An example of the SP-index

Assuming that the minimal support is 2/3, the frequent segments are {AB, AC, CD, DA, ACD, CDA, ACDA}. Base segments are those frequent segments with length equal to parameter $MinLen$. Figure 1 shows the SP-index with $MinLen = 2$, where each base segment has a B-tree for indexing its occurring positions, and only the leaf nodes of each B-tree are shown here. After building the SP-index, a query $Q(X, s, i)$ can find the smallest position of subsequence X appearing after position i of sequence s.

In the pattern phase, the SP-index algorithm generates all frequent patterns $X_1 * \ldots * X_k$ by using frequent segments X_i found in the segment phase. Basically, it enumerates the possible combinations of all frequent segments. Then, for each combination of frequent segments, it performs several queries $Q(X, s, i)$ to check whether the pattern of this combination is frequent. For example, we want to check whether pattern {AC}∗{AB} is frequent. According to the occurring positions of the first segment {AC} in Figure 1, this algorithm tests three queries, i.e., $Q(\text{AB}, S_1, 4)$, $Q(\text{AB}, S_2, 3)$ and $Q(\text{AB}, S_3, 2)$. Then, it counts the number of sequences appearing in those queries which return a position result. Only if this number is not less than the minimal support, this pattern is frequent. In the above example, only the first query returns a position result. Therefore, pattern {AC}∗{AB} is not frequent. In this case, {AB} is added to a set, {AC}.*dead*, to indicate that segment {AB} can not appear after segment {AC}.

From the above example, we could see that in the SP-index algorithm, as the size of the B-tree of each base segment increases, the number of nodes needed to be checked during the testing process of queries also increases. Moreover, there exist many frequent segments which do not need to be considered when generating long patterns. For example, assume that there are two frequent segments, {DA} and {CDA}, with the same number of appearances. If pattern {CDA}∗X is frequent, we do not need to consider pattern {DA}∗X. This is because we only focus on the longest pattern.

3. **The BP Algorithm.** In this section, we present the Bit-Pattern-based (BP) algorithm. First, we formally define the problem as follows. A pattern "$X_1 * X_2 * \ldots * X_n$" matches one sequence if each segment X_i matches itself in order and each $*$ can be substituted for zero or more items [9]. The support of one pattern is the percentage of sequences matched by this pattern in a database DB. Given a minimal segment length $MinLen$ and a minimal support $MinSup$, pattern $X_1 * X_2 * \ldots * X_n$ is frequent, if $\mid X_i \mid \geq MinLen$

TABLE 2. Table BM

	S_1								S_2						S_3				
	A	B	D	A	C	D	A	B	A	B	A	C	D	A	A	A	C	D	C
$BS_{\{A\}}$	1	0	0	1	0	0	1	0	1	0	1	0	0	1	1	1	0	0	0
$BS_{\{B\}}$	0	1	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0
$BS_{\{C\}}$	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0
$BS_{\{D\}}$	0	0	1	0	0	1	0	0	0	0	0	0	1	0	0	0	0	1	0

for $1 \leq i \leq n$ and the support of this pattern is at least TH, where TH is the threshold in the BP algorithm. The goal is to find the longest frequent pattern from all protein sequences. Basically, the BP algorithm contains three steps: (1) transforming protein sequences into bit patterns; (2) finding the frequent segments and pruning the unnecessary frequent segments; and (3) finding the longest pattern. We will use an example to demonstrate the proposed BP algorithm later. Let DB be a collection of sequences $\{S_1, S_2, S_3\}$. Each sequence S_i is composed of 4 alphabets: $\{A, B, C, D\}$. Database DB is shown in Table 1. TH is 2. The minimal length of a segment, $MinLen$, is 2.

In Step 1, we transform the protein sequences into bit patterns. We will transform the bits corresponding to the occurring positions of one code in a protein sequence to '1'. We store this information in table BM, as shown in Table 2. For example, the protein sequence S_1 is "$ABDACDAB$". Code A appears in the first, the fourth, and the seventh positions. Therefore, the bit pattern of code A is 10010010.

In Step 2, there are two parts: (1) using the AND operator to find all frequent segments; and (2) pruning unnecessary segments. A segment refers to one or more codes at consecutive positions in a sequence. When the number of the segment found in protein sequences is at least TH, we call it a frequent segment.

Definition 3.1. *If a segment F starts at t-th position of a sequence S, we let the t-th bit of the bit pattern of F be 1.*

Consider the characteristic described as follows. For protein sequence S, assume that there are two segments whose lengths are k and 1, respectively. If the longer segment ends at position t and the shorter one starts at position $(t+1)$, there exists a segment with length $(k+1)$ in sequence S, which is composed of these two segments.

By this characteristic, any segment can be composed of two subsegments. If two subsegments can form a long segment, one segment must end at position t and the other segment must start at position $(t+1)$. When we want to find a long segment, we can find two subsegments in which one subsegment occurs from position t to position $(t+k)$, and the other subsegment occurs at position $(t+k+1)$. Now, we consider how to use these bit patterns to get frequent segments by this characteristic.

Property 1. Consider a protein sequence, $Seq = f_1 f_2 \dots f_k$, where f_i is an acid amino for $1 \leq i \leq k$. There are two segments F and F' in sequence Seq, where $F = f_{\underline{t}} f_{(t+1)} \dots f_{(t+m-1)}$ (whose length is m) and $F' = f_{(\underline{t+m})}$ (whose length is 1). We have two bit patterns for segments F and F', denoted as BS_F and $BS_{F'}$ respectively, where $BS_F = 0_1 \dots 0_{(t-1)} 1_{\underline{t}} 0_{(t+1)} \dots 0_k$ and $BS_{F'} = 0_1 \dots 0_{(t+m-1)} 1_{(\underline{t+m})} 0_{(t+m+1)} \dots 0_k$. If we shift the bits of $BS_{F'}$ to left by m bits (where m is the length of segment F), the new $BS_{F'}$ will be $0_1 \dots 0_{(t-1)} 1_t 0_{(t+1)} \dots 0_k$, and bit '1' in BS_F and $BS_{F'}$ will appear at the same position t.

Bit '1' in a bit pattern shows the start position of one segment. By Property 1, if two subsegments (i.e., F and F') can construct a long segment, bit '1' in their bit patterns will appear at the same position after shifting the bit pattern of the second segment (i.e., F')

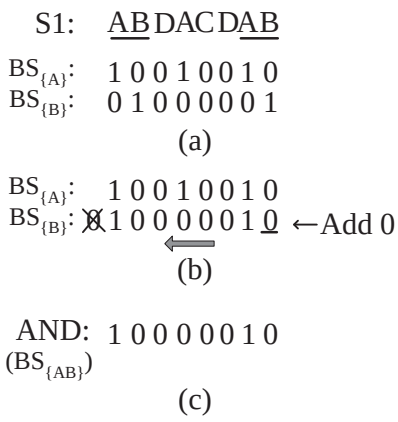

FIGURE 2. The process of finding the bit pattern of segment {AB} in sequence S_1: (a) bit patterns of segments {A} and {B}; (b) shifting the bit pattern of segment {B} to left by one position; (c) the resulting bit pattern of segment {AB} after applying the AND operation

to left by m bits. Therefore, after performing shifting, we can use the "AND" operation to construct a new bit pattern for the long segment composed of those two subsegments (i.e., F and F').

We use two examples to explain the process. The first example is finding whether segment {AB} appears in protein sequence S_1, as shown in Figure 2. We can consider segment {AB} as segment {A} + segment {B}. The length of segment {A} is 1, i.e., $m = 1$. Figure 2(a) shows the bit pattern of segment {A}, i.e., $BS_{\{A\}}$ = "10010010", and the bit pattern of segment {B}, i.e., $BS_{\{B\}}$ = "01000001", in protein sequence S_1. Figure 2(b) shows the result of shifting $BS_{\{B\}}$ to left by $m = 1$ bit. Figure 2(c) shows the resulting bit pattern of segment {AB}, by applying the AND operation on $BS_{\{A\}}$ and the shifting result of $BS_{\{B\}}$. From this bit pattern, we know that segment {AB} appears at positions 1 and 7 in protein sequence S_1. In the second example, we want to know whether segment {ACD} appears in protein sequence S_1 by considering segment {ACD} as segment {AC} + segment {D}. $BS_{\{AC\}}$ in sequence S_1 is "00010000". $BS_{\{D\}}$ in sequence S_1 is "00100100". The length of segment {AC} is 2, i.e., $m = 2$. We shift $BS_{\{D\}}$ to left by $m = 2$ bits and the result of $BS_{\{D\}}$ is "10010000". Finally, we apply the AND operation on $BS_{\{AC\}}$, i.e., "00010000", and this shifting result, i.e., "10010000". The AND result, i.e., "00010000", is the bit pattern of segment {ACD}, which means segment {ACD} appears at position 4 in sequence S_1.

By this way, we can derive whether one segment appears in one sequence. If one segment appears in at least TH protein sequences, this segment is a frequent segment. We use this way to find all frequent segments, where the length of each frequent segment is at least $MinLen = 2$. Then, these frequent segments and their bit patterns are stored in table BPS. For the database shown in Table 1, the resulting table BPS is shown in Table 3. These frequent segments will be the basic elements to produce frequent patterns later.

Up to this point, we have found all frequent segments stored in table BPS. These segments are not all necessary for producing the longest pattern. For example, segment {AC} is a subset of segment {ACD}, and their numbers of appearances are both 3. That is, when segment {AC} appears in a protein sequence, segment {ACD} appears at the same position of the same protein sequence. Because we want to find the longest pattern, we just keep segment {ACD} and prune segment {AC}. Pruning unnecessary frequent segments can help us reduce the storage space and the processing time.

TABLE 3. Table BPS

	S_1								S_2						S_3				
	A	B	D	A	C	D	A	B	A	B	A	C	D	A	A	A	C	D	C
$BS_{\{AB\}}$	1	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0
$BS_{\{AC\}}$	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0
$BS_{\{CD\}}$	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	1	0	0
$BS_{\{DA\}}$	0	0	1	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0
$BS_{\{ACD\}}$	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0
$BS_{\{CDA\}}$	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0
$BS_{\{ACDA\}}$	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0

TABLE 4. Table BPS after pruning

	S_1								S_2						S_3				
	A	B	D	A	C	D	A	B	A	B	A	C	D	A	A	A	C	D	C
$BS_{\{AB\}}$	1	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0
$BS_{\{ACD\}}$	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0
$BS_{\{ACDA\}}$	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0

The idea of pruning is the same as that of closed itemsets in the problem of mining association rules. If segment t is the subset of some segments and their numbers of appearances are the same, we prune segment t. For example, for table BPS shown in Table 3, segment {CDA} appears at sequences S_1 and S_2, i.e., the number of appearances = 2. Segment {ACDA} is the superset of segment {CDA}, and also appears twice in sequences S_1 and S_2. Therefore, segment {CDA} is pruned. After we prune those unnecessary segments, only three segments are kept, i.e., {AB}, {ACD} and {ACDA}, in Table 4.

In Step 3, we use a depth-first-search way to combine the longest pattern. In the combining process, we will utilize the OR operation, which can help us know whether a bit pattern has been changed. For every two segments, first, we find the position of the first appearance of bit '1' in the bit pattern of the first segment. Assume that this position is i. For the first segment, we create a new bit pattern, $NewBS$, whose bits left to position i are all '1' and bits right to position i are all '0'. This idea is similar to "masking". That is, we want to find out whether the second segment appears after the first appearance of the first segment, i.e., position i. Therefore, for those bits left to position i, we set them to '1' to avoid affecting the OR result. For example, for segment {AB}, its original bit patterns in sequences S_1 and S_2 are "10000010" and "100000", respectively, as shown in Figure 3. Note that segment {AB} does not appear in sequence S_3, and we do not consider sequence S_3 here. The new bit patterns in sequences S_1 and S_2 will be "10000000" and "100000", respectively.

Next, we want to find out whether there exists any segment appearing after the first segment in all sequences. For each sequence, we apply the OR operation on $NewBS$ of the first segment and the original bit pattern of the second segment. If the bit pattern of the OR result is different from $NewBS$ of the first segment, it means that the second segment appears after the first segment in this sequence. Figure 3 shows examples of applying the OR operation to find the longest pattern based on those remaining segments in Table 4 after pruning. Figure 3(a) shows the example of finding out whether segment {AB} appears after segment {AB}. For sequence S_1, after applying the OR operation, the resulting bit pattern is "10000010", different from $NewBS$ of the first segment, i.e., "10000000". It means that segment {AB} appears after segment {AB} in sequence S_1.

	S_1								S_2						S_3				
	A	B	D	A	C	D	A	B	A	B	A	C	D	A	A	A	C	D	C
$NewBS_{\{AB\}}$	1	0	0	0	0	0	0	0	1	0	0	0	0	0					
$BS_{\{AB\}}$	1	0	0	0	0	0	1	0	1	0	0	0	0	0					
OR	1	0	0	0	0	0	1	0	1	0	0	0	0	0					

(a)

	S_1								S_2						S_3				
	A	B	D	A	C	D	A	B	A	B	A	C	D	A	A	A	C	D	C
$NewBS_{\{AB\}}$	1	0	0	0	0	0	0	0	1	0	0	0	0	0					
$BS_{\{ACD\}}$	0	0	0	1	0	0	0	0	0	0	1	0	0	0					
OR	1	0	0	1	0	0	0	0	1	0	1	0	0	0					

(b)

	S_1								S_2						S_3				
	A	B	D	A	C	D	A	B	A	B	A	C	D	A	A	A	C	D	C
$NewBS_{\{AB\}*\{ACD\}}$	1	1	1	1	0	0	0	0	1	1	1	0	0	0					
$BS_{\{ACD\}}$	0	0	0	1	0	0	0	0	0	0	1	0	0	0					
OR	1	1	1	1	0	0	0	0	1	1	1	0	0	0					

(c)

FIGURE 3. Examples of finding the longest pattern: (a) pattern {AB}∗{AB}; (b) pattern {AB}∗{ACD}; (c) pattern {AB}∗{ACD}∗{ACD}

For sequence S_2, the resulting bit pattern is "100000", the same as $NewBS$ of the first segment. Therefore, segment {AB} does not appear after segment {AB} in sequence S_2.

From these bit patterns of the OR result, we find that pattern {AB}∗{AB} only appears once (in sequence S_1), and the number of appearances of pattern {AB}∗{AB} is $1 < TH(= 2)$. Therefore, pattern {AB}∗{AB} is not frequent. We use a data structure, {AB}.*dead*, to record those segments which have been found to be unable to appear after segment {AB}. Therefore, {AB}.*dead* is set to {{AB}} now.

Since we perform a depth-first-search and segment {AB} has been found to be unable to appear after segment {AB}, we continue to find out whether there exists any other segment, e.g., segments {ACD} and {ACDA} in Table 4, appearing after segment {AB}. Figure 3(b) shows the resulting bit pattern of pattern {AB}∗{ACD}. Since these resulting bit patterns are all different from $NewBS$ of segment {AB} for sequences S_1 and S_2, it means that pattern {AB}∗{ACD} appears in both sequences S_1 and S_2. The number of appearances of this pattern is $2 \geq TH$, and pattern {AB}∗{ACD} is frequent. In this case, {AB}∗{ACD}.*dead* inherits the result of {AB}.*dead*. That is, {AB}∗{ACD}.*dead* is {{AB}}. In the depth-first-search way, next, we consider whether there exists any segment appearing after pattern {AB}∗{ACD}.

The resulting bit pattern of pattern {AB}∗{ACD} in sequence S_1 is "10010000". The position of the first appearance of segment {ACD} after segment {AB} is 4 in sequence S_1. Because we want to find whether there exists any segment appears after pattern {AB}∗{ACD}, we should not consider those segments whose positions of appearance are not larger than position 4. We set those bits of positions from 1 to 4 to '1', i.e., masking. (Note that if there is bit '1' in a bit pattern, there is bit '1' at the same positions of the bit pattern after applying the OR operation). Therefore, $NewBS$ of pattern {AB}∗{ACD} for sequence S_1 becomes "11110000". Similarly, $NewBS$ of pattern {AB}∗{ACD} for sequence S_2 becomes "111000".

After changing $NewBS$ of pattern {AB}∗{ACD}, we consider which segment can be combined after pattern {AB}∗{ACD}. Because segment {AB} is in {AB}∗{ACD}.*dead*, we do not need to consider pattern {AB}∗{ACD}∗{AB}. We just consider segment {ACD} and segment {ACDA}. We want to know whether pattern {AB}∗{ACD}∗{ACD} is frequent. The resulting bit patterns of pattern {AB}∗{ACD}∗{ACD} are shown in Figure 3(c). For sequences S_1 and S_2, the resulting bit patterns are the same as $NewBS$ of pattern {AB}∗{ACD}, which means that segment {ACD} does not appear after pattern {AB}∗{ACD}. Therefore, pattern {AB}∗{ACD}∗{ACD} is not frequent. Segment {ACD} is added to {AB}∗{ACD}.*dead*. Following the similar idea, we continue to consider the other possible patterns in a depth-first-search way. After considering all possible patterns, there are two frequent patterns found from the example database shown in Table 1, i.e., {AB}∗{ACD} and {AB}∗{ACDA}, where {AB}∗{ACDA} is the longest pattern.

4. **Performance.** In this section, we study the performance of the proposed BP algorithm, and make a comparison with the SP-index algorithm [15]. Our experiments were run on a Quad CPU Q6600 2.40GHz, 2GB RAM and running Windows XP. All experiments were written in Java and were compiled by JDK 1.6.

In order to evaluate the performance of the proposed algorithm, we first extracted real life protein sequences from the web site of National Center for Biotechnology Information (http://www.ncbi.nlm.nih.gov/) as the experimental data. The protein data set was extracted by the conjunction of searching: (1) category = "Protein"; (2) sequence length range = [100 : 150]; and (3) substance name = "human". We experimented three cases: (1) changing the value of the minimal support; (2) changing the number of sequences; and (3) changing the length of protein sequences. Figure 4 shows the experimental results of these three cases, where Figures 4(a)-4(c) are for the first case, the second case and the third case, respectively. The default number of sequences and the default minimal support are 1000 and 40%, respectively. From Figure 4, we show that the BP algorithm needs shorter processing time than the SP-index algorithm. Since in the real data sets, the length of one segment is usually short, the number of segments which can be pruned in the BP algorithm is also small. However, since the BP algorithm efficiently utilizes bit operations to find the longest pattern, the BP algorithm could outperform the SP-index algorithm in terms of processing time.

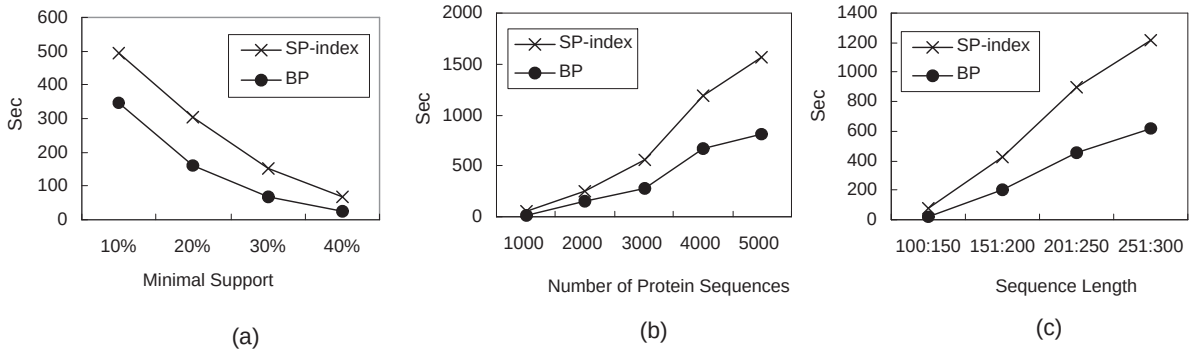

FIGURE 4. A comparison of processing time between SP-index and BP algorithms for real data: (a) changing the value of the minimal support; (b) changing the number of sequences; (c) changing the length of protein sequences

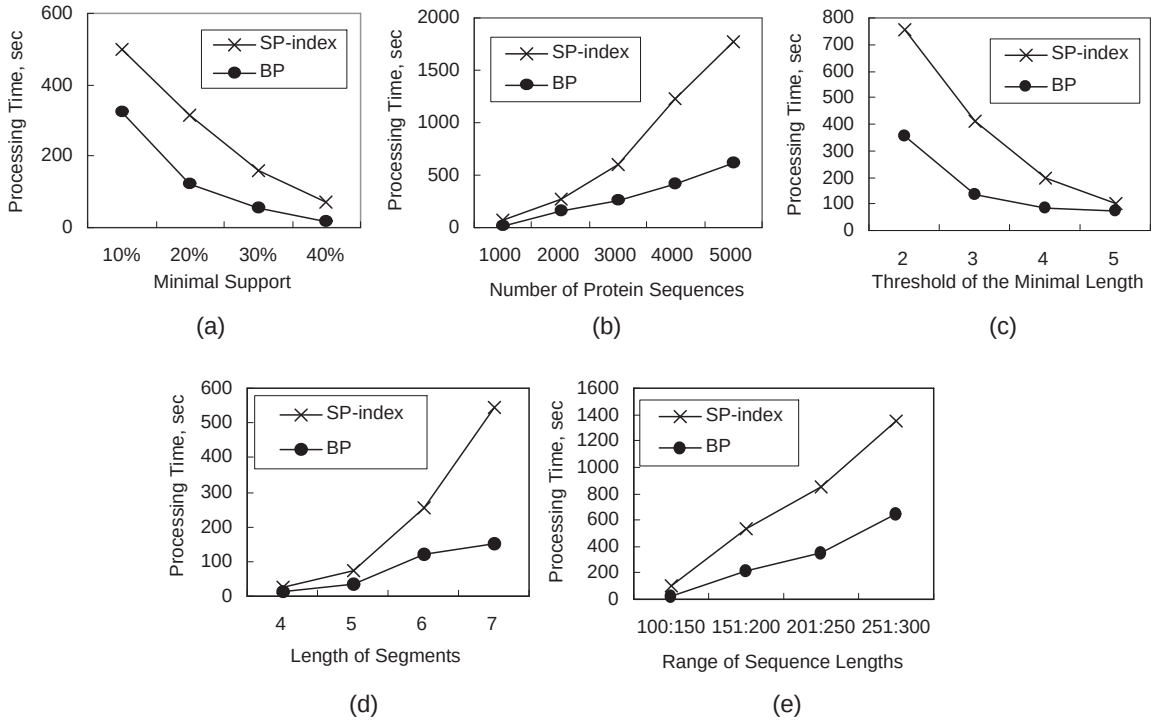

FIGURE 5. A comparison of processing time between SP-index and BP algorithms for synthetic data: (a) changing the value of $MinSup$; (b) changing the value of D; (c) changing the value of $MinLen$; (d) changing the value of $MaxSeg$; (e) changing the value of L

Next, to study the performance of the BP algorithm when the length of a segment is long, we generated the synthetic data as the experimental data. We applied the method of generating synthetic data in the SP-index algorithm [15]. That is, D sequences will be randomly generated with the codes of 20 amino acids. First, we generate $NumSeg$ (whose default value is 20) distinct segments randomly, where the length of each segment is at most $MaxSeg$. Next, we let the number of appearances of these segments be $F * D$, where F is the percentage of one frequent segment appearing in all sequences. That is, We randomly choose $F * D$ sequences and insert one segment into random positions of these sequences. The default value of F is 45%. We repeat the inserting process for all segments and avoid the case that the length of one sequence is out of range L. Finally, the rest of codes which have not been determined yet in each sequence are randomly generated. We experimented five cases by changing the values of: (1) the minimal support, $MinSup$; (2) D; (3) the threshold of the minimal length of a segment, $MinLen$; (4) $MaxSeg$ and (5) L, respectively. Figures 5(a)-5(e) are experimental results of Cases 1, 2, 3, 4 and 5, respectively. The default values of parameters are $MinSup = 40\%$, $D = 1000$, $Minlen = 2$, $MaxSeg = 5$ and $L = [100 : 150]$.

From Figure 5, we can observe that the BP algorithm needs shorter processing time than the SP-index algorithm for these 5 cases. In Case 1 shown in Figure 5(a), as the value of $MinSup$ increases, the processing time of both algorithms decreases. This is because as the value of $MinSup$ increases, the number of frequent segments which need to be processed decreases. On the other hand, as the value of $MinSup$ decreases, the number of frequent segments increases. The more the number of frequent segments is, the longer the time needed by the SP-index algorithm in its pattern phase is. Therefore, the

difference of the processing time between the SP-index algorithm and the BP algorithm becomes large when the value of $MinSup$ becomes small.

In Case 2 shown in Figure 5(b), as the number of protein sequences increases, the processing time of both algorithms increases. This is because both algorithms need more time for processing protein sequences. In Case 3 shown in Figure 5(c), as the value of $MinLen$ increases, the processing time of both algorithms decreases. This is because as the value of $MinLen$ increases, the number of segments which can pass this threshold (i.e., $MinLen$) decreases. In Case 4 shown in Figure 5(d), as the length of each segment, $MaxSeg$, increases, the processing time of both algorithms increases. This is because the larger the value of $MaxSeg$ is, the larger the total number of frequent segments is. Moreover, the larger the value of $MaxSeg$, the more the unnecessary frequent segments we can prune. The process time of the BP algorithm increases slowly, while that of the SP-index algorithm increases rapidly. In Case 5 shown in Figure 5(e), as L increases, the processing time of both algorithms increases. Because as the range of lengths of sequences, L, increases, both algorithms need more time to process each sequence.

5. **Conclusions.** Sequence motifs often denote important functional regions in proteins, and can be used to discover the function of proteins. In this paper, we have proposed the BP algorithm to mine the longest sequential pattern with gaps of arbitrary size from protein sequences. In the BP algorithm, first, protein sequences are transformed into bit patterns. Next, by applying various bit operations, i.e., AND, OR, shifting and masking, on those bit patterns, the BP algorithm can find frequent segments and the longest sequential pattern efficiently. From the experimental results on real and synthetic data sets, we have shown that the BP algorithm is more efficient than the SP-index algorithm in terms of processing time.

Acknowledgment. This research was supported in part by the National Science Council of Taiwan under Grant No. NSC-99-2221-E-110-080-MY3.

REFERENCES

[1] *http://www.csie.ncnu.edu.tw/∼rctlee/biology.html*, 2009.
[2] *http://en.wikipedia.org/wiki/Protein*, 2009.
[3] R. Agrawal and R. Srikant, Mining sequential patterns, *Proc. of the 11th Int. Conf. Data Engineering*, Taipei, Taiwan, pp.3-14, 1995.
[4] S. F. Altschul, W. Gish, W. Miller, E. W. Myers and D. J. Lipman, Basic local alignment search tool, *J. Mol. Biol.*, vol.215, no.3, pp.403-410, 1990.
[5] J. Ayres, J. Flannick, J. Gehrke and T. Yiu, Sequential pattern mining using a bitmap representation, *Proc. of the 8th ACM SIGKDD Int. Conf. Knowledge Discovery Data Mining*, Edmonton, Canada, pp.429-435, 2002.
[6] A. Brazma, I. Eidhammer and D. Gilbert, Approaches to the automatic discovery of patterns in biosequences, *J. Comput. Biol.*, vol.5, no.2, pp.279-305, 1998.
[7] P. G. Ferreira and P. J. Azevedo, Query driven sequence pattern mining, *Proc. of Simpósio Brasileiro de Banco de Dados*, Santa Catarina, Brasil, pp.1-15, 2006.
[8] P. G. Ferreira and P. J. Azevedo, Evaluating deterministic motif significance measures in protein databases, *Algorithms Mol. Biol.*, vol.2, no.16, pp.1-20, 2007.
[9] V. Guralnik and G. Karypis, A scalable algorithm for clustering protein sequences, *Proc. of Workshop Data Mining Bioinformatics*, San Francisco, USA, pp.73-80, 2001.
[10] K.-F. Jea, K.-C. Lin and I.-E. Liao, Mining hybrid sequential patterns by hierarchical mining technique, *International Journal of Innovative Computing, Information and Control*, vol.5, no.8, pp.2351-2367, 2009.
[11] J. Pei et al., PrefixSpan: Mining sequential patterns efficiently by prefix-projected pattern growth, *Proc. of the 17th Int. Conf. Data Engineering*, Heidelberg, Germany, pp.215-226, 2001.

[12] X. Shang, Z. Li and W. Li, Mining functional associated patterns from biological network data, *Proc. of ACM Symp. Applied Computing*, Hawaii, USA, pp.1488-1489, 2009.
[13] Z. Sui, Y. Liu and Y. Hu, Extracting hyponymy relation between Chinese terms based on term types' commonality and sequential patterns, *ICIC Express Letters*, vol.3, no.4(B), pp.1233-1238, 2009.
[14] C.-Y. Tsai and P.-H. Lo, A sequential pattern based route suggestion system, *International Journal of Innovative Computing, Information and Control*, vol.6, no.10, pp.4389-4408, 2010.
[15] K. Wang, Y. Xu and J. X. Yu, Scalable sequential pattern mining for biological sequences, *Proc. of the 13th ACM Int. Conf. Information Knowledge Management*, Washington, USA, pp.178-187, 2004.
[16] Y. Wang, W. Zhang, C. Cheng, H. Ling and F. Zhao, Establishing website navigation support systems by mining sequential patterns, *ICIC Express Letters*, vol.4, no.2, pp.395-400, 2010.
[17] J. Yang, J. S. Deogun and Z. Sun, A new scheme for protein sequence motif extraction, *Proc. of the 38th Annual Hawii Int. Conf. System Sciences*, Hawaii, USA, 2005.