Mining frequent biological sequences based on bitmap without candidate sequence generation

Qian Wang [a,b,c,*] , Darryl N Davis [c], Jiadong Ren [a,b]

[a] College of Information Science and Engineering, Yanshan University, Qianhuangdao, Hebei, China

[b] Computer Virtual Technology and System Integration Laboratory of Hebei Province, China

[c] Department of Computer Science, University of Hull, Hull, UK

Abstract

Biological sequences carry a lot of important genetic information of organisms. Furthermore, there is an inheritance law related to protein function and structure which is useful for applications such as disease prediction. Frequent sequence mining is a core technique for association rule discovery, but existing algorithms suffer from low efficiency or poor error rate because biological sequences differ from general sequences with more characteristics. In this paper, an algorithm for mining **F**requent **B**iological **S**equence based on **B**itmap, **FBSB**, is proposed. FBSB uses bitmaps as the simple data structure and transforms each row into a quicksort list QS-list for sequence growth. For the continuity and accuracy requirement of biological sequence mining, tested sequences used during the mining process of FBSB are real ones instead of generated candidates, and all the frequent sequences can be mined without any errors. Comparing with other algorithms, the experimental results show that FBSB can achieve a better performance on both run time and scalability.

Keywords: Biological sequence, frequent pattern, Bitmap, Quicksort list

1. Introduction

* Corresponding author at: College of Information Science and Engineering, Yanshan University, Qianhuangdao, Hebei, China, 066000. Tel.:+8613731767789.
E-mail addresses:wanqianysu@163.com(Q. Wang)

Biosequence mining can help people recognize interesting and important relationships between biological sequences for human genome research. There are usually different functions in the sequences, some of them are because of a special element, and some of them are the result of the interaction of a few elements. Biosequence mining is one of the key technologies to discover the single and mutual functions of the elements or the sequences. It can give reasonable prediction and guidance for making human nucleic acid, protein and other biological data. By identifying the protein-coding genes from DNA sequences, it can be found that some gene combination mode is related to drug allergy or appears frequently in some disease.

Bioinformatics refers to several subjects such as computer science, information science, and mathematics. Computer processing is a major part in the analysing of biological data[1]. Even at the beginning of the biological research and development, many pattern mining algorithms for biosequence were proposed. Brazma[2] surveyed approaches to the pattern discovery in biosequences and placed these approaches within a formal framework that systematises the types for algorithm comparison. It is found that most of the algorithms face efficiency problems when the pattern space grows rapidly.

Mining frequent patterns is an indispensable component in many data mining tasks such as association rule mining. These association rule mining algorithms can be partitioned into different categories. Apriori[3] and FP-growth[4] are two classic algorithms. Apriori uses candidate generate-test technique and FP-growth is based on a tree structure which records the sequence paths without candidate generation. The algorithms based on them can be classified in terms of whether there is candidate generation. Another classic algorithm is Eclat[5]. It is different because it expresses the dataset vertically. Algorithms can also be classified by horizontal or vertical dataset expression forms. BitTableFI[6] is BitTable-based and the BitTable is horizontally and vertically indexed. Although efficient

bit wise operations are used, candidate generation and test ensure that BitTableFI suffers with high computational costs. Index-BitTableFI[7] is proposed to solve this problem. Index array and the correlative calculate method are applied in using BitTable horizontally. A breadth-first search strategy is used for quick identification of the co-occurrence items and depth-first search is for mining all the frequent itemsets in different levels. DBV-Miner[8] is presented by Bay et al. and developed for mining frequent closed itemsets. Dynamic Bit-Vector(DBV) and a lookup table are used, and the support of itemsets can be quickly computed by the intersection between two DBVs. Bay et al. presented more algorithms based on various data structures, like lattice-based algorithms[9-11] and N-list based algorithm[12]. Sequential pattern mining is originally put forward by Agrawal et al.[13], who also presented three sequential mining algorithms. Afshar formally defined the maximal frequent sequence and proposed the corresponding algorithm MaxSequence[14]. MaxSequence needs to generate and test candidate maximal sequences, and a compressed prefix tree is used to store maximal frequent sequences. When the database is larger, the overhead for maintaining and storing the prefix tree compromises the scalability of the algorithm. A highly compressed lattice storage structure and a breadth-first approach are used by FMMSP[15], and maximal frequent sequences and closed frequent sequences are mined quickly without candidate generation. Because of the nature of biology data, the above algorithms are not necessarily suitable for biological data mining. There are more algorithms designed for biosequences these days. TRFinder[16] algorithm seeks tandem repeats which can cause human disease and consist of two or more copies of nucleotide patterns. A probabilistic model and a statistical criteria collection are used to detect tandem repeats. REPuter[17] is based on a suffix tree and a sequence alignment technique for detecting various types of repeats in DNA, it circumscribes the wide scope of repeat analysis, but it is not efficient for mining

frequent repeats. BioPM algorithm[18] was developed for protein sequence mining and it introduces the concept of multiple supports so as to improve performance and efficiency. The mMbioPM[19] algorithm optimizes the structure of hash lists to improve the efficiency of BioPM and reduces the run time. However, the efficiency of the BioPM algorithm and its improvements are not quite ideal because of the large scale of the projected database when the minimal support is lower. MSPM[20] is based on the prefix-tree structure, and it presents the concept of primary pattern which makes the degree of the prefix tree a constant. By avoiding too many short patterns generations, the scale of the prefix-tree will not be too large. Its efficiency is much better but it may miss frequent sequences. An index-based approach[21] proposes an interesting measure for meaningful biological information. Each leaf node in the tree structure is an array whose length is variable. The transaction ID and starting position of each sequence are stored in the arrays for memory reduction. CBFMM[22] mines nucleotide and protein sequence with a variety of FP-tree-based model definitions. Repetition detection in biological data is required for potential malfunction and disease identification. DPMine[23] is designed for colossal sequence discovery from biological dataset. It integrates a DPT+ tree, a doubleton data matrix and a one-dimensional array to find doubleton patterns which may further generate colossal sequences. For irrelevant regions in biological sequence evolution such as mutations, gap constraints need to be considered. DFSG[24] is designed for the sequences which are not conserved. For biological network analysis, graph simplification technology is used, aiming to reduce the graph size[25]. An overview is given in [26] to illustrate how the various frequent pattern mining algorithms can be used for human bioinformatic applications.

The FBSB algorithm, reported here, is proposed to mine frequent biosequences with a high efficiency and less memory space requirements. It first calculates the supports of the

items for frequency mining and records the end position value of each 2-sequence to form a bitmap which is further used for frequent 2-sequence mining and pattern growth. A quicksort list is created for fast connecting sequences in the same biosequence. Two sequences in the quicksort with the position values next to each other in an ascending order can be connected easily by adding the last item of the second sequence to the first sequence, and the position value of the second sequence is used as the position value of the new sequence. The bitmap is updated and the storage space is released constantly. There is no candidate generation and every frequent sequence can be obtained. The algorithm can satisfy the requirements of biological data mining because of its good efficiency and high quality results. Experimental results show that FBSB is much better when compared with other methods.

The remainder of the paper is organized as follows. Section 2 introduces the problem definition about biological sequences and bitmap formation. Section 3 develops the FBSB algorithm and gives some examples. Section 4 presents the performance study of FBSB algorithm. Section 5 contains the concluding remarks.

2. Preliminaries and Problem Definitions

2.1. Problem definition

DNA and protein sequences are two typical types of biosequences. It should be noted that there are differences between biosequences and general sequences, so some definitions are given as follows.

Definition 1. Let Σ be an alphabet, a sequence $S=<s_1s_2\ldots s_m>$ with $s_i \in \Sigma (i=1,\ldots,m)$ is called a DNA sequence if $\Sigma=\{A,C,G,T\}$ consists of four nucleotides, or it is called a protein sequence if Σ consists of the 20 symbols for amino acids. A sequence can be called a k-sequence if it contains k nucleotides or amino acid symbols.

Example 1. Let Σ={a,b,c} and S=<bacaab>, S is a 6-sequence because there are six items in S.

Definition 2. Let sequence S_1=<a_1,a_2,…,a_m> and sequence S_2=<b_1,b_2,…,b_n> be two sequences(m<n) on the alphabet Σ. S_1 is a subsequence of S_2 if there exist integers i_1, i_2,…,i_m, such that $1 \le i_1 < i_2 < \ldots < i_m \le n$ and $a_1=b_{i1}$, $a_2=b_{i2}$, …, $a_m=b_{im}$. It can also be said that S_2 is a super sequence of S_1.

Example 2. Let S_1=<bacaab>, S_2=<aca>, S_3=<bab>. S_2 is a subsequence of S_1, but S_3 is not a subsequence of S_1. This is different from general sequences; as the items of the subsequence must occur contiguously in the super sequence.

Definition 3. Given a database of biosequences D and a biosequence S. The support of the biosequence S in D, denoted as sup(S), is the number of the sequences in D which contains S as its subsequence.

Definition 4. Given a biosequence database D={S_1,S_2,S_3,...,$S_{|D|}$} and a user-defined threshold ζ, where |D| is the number of sequences in D. The minimal support min-sup can be calculated as min-sup=ζ*|D|. If the support of a biosequence S satisfies sup(S)≥min-sup, S is called a frequent sequence in D. If S is a k-sequence, it can be called a frequent k-sequence.

Property 1. For a sequence S, if S is an infrequent sequence, then any of its super sequences are also infrequent.

2.2. Bitmap of the sequence

For efficient mining process, all the position values of the 2-sequence occurrences are recorded. A bitmap is constructed to store the values and can be updated as the frequent sequences grow. A sequence may occur several times in the same database transaction, and all its occurrence positions should be put into a position array.

Definition 5. A bitmap is a two-dimensional table, where each row represents the ID of a sequence in the database and each column represents a sequence. A bitmap cell is denoted as $Pos_i(S)$, where i is the corresponding row number and S is the corresponding column sequence. $Pos_i(S)$ is an array of position values for the occurrences of S in the ith sequence.

If a subsequence does not occur in a database transaction, its position array is not established, but here '∅' is used to indicate that no occurrence.

Example 3. Table 1 shows a sequential database and Table 2 is the corresponding bitmap.

Table 1. Sequential database D

ID	S
1	<abcbac>
2	<acbcab>
3	<bcbabc>
4	<acbabc>

Table 2. Bitmap of database D

S / ID	ab	bc	cb	ba	ac	ca
1	2	3	4	5	6	∅
2	6	4	3	∅	2	5
3	5	2,6	3	4	∅	∅
4	5	6	3	4	2	∅

Taking $Pos_3(bc)=\{2,6\}$ for example, sequence bc occurs twice in the third database transaction, and the position values are 2 and 6. Sequence bc occurs in all the 4 transactions, so sup(bc) is 4. If ζ is set to be 50%, the min-sup can be calculated by ζ*|D|, and min-sup=50%*2=1. Then sequence bc is a frequent 2-sequence.

Definition 6. QS-list is a quicksort list of all the SP-codes for a row of the bitmap. In a QS-list, SP-codes are arranged in position value ascending order. Each SP-code is denoted by SP=<seq,pos>, where seq is the sequence name and pos is the position value

of seq. QS-list of a bitmap row is denoted by $\{SP_1,SP_2,...,SP_n\}$, and $SP_1.pos< SP_2.pos< \ldots < SP_n.pos$.

Property 2 (Ancestor-descendant relationship of SP-codes). Given SP_i and SP_j are two SP-codes, SP_i is an ancestor of SP_j if and only if $SP_i.pos+1$ are equal to $SP_j.pos$.

Example 4. QS-list for the third row of the bitmap in Table 2 is as follows.

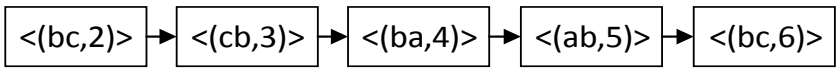

Fig.1. QS-list of the 2-patterns in the third transaction

There are 5 SP-codes. $Pos_3(bc)=\{2,6\}$ can produce two SP-codes as <bc,2> and <bc,6>. SP[1]=<bc,2>, so SP[1].seq=bc and SP[1].pos=2. All the SP-codes are in a pos ascending order, SP[1].pos=2<SP[2].pos=3<…<SP[5].pos=6.

SP[1] is the ancestor of SP[2] for SP[1].pos+1=2+1=3 is equal to SP[2].pos=3. And it can also be said that SP[2] is the descendant of SP[1]. SP[1] is not the ancestor of SP[3] or others; the descendant must be next to its ancestor.

3. FBSB algorithm

3.1. The process of the FBSB algorithm

To design an effective data structure and mine frequent biological sequences accurately, the FBSB algorithm can be executed based on the following observations. First, by scanning the database once, the supports of all the items are calculated, all the position values of the 2-sequences are put into arrays and the initial bitmap is formed. Frequent items are mined directly. Second, frequent k-sequences are mined and (k+1)-sequences are generated by connecting frequent k-sequences in the same transaction with a position value ascending order. The two k-sequences can be connected only when the position value of the second sequence is 1 more than the first one, then frequent (k+1)-sequences are obtained and bitmap is updated. Third, no candidate sequences are generated, all the sequences really do occur and the precision is 100%.

Algorithm 1. Bitmap initiation and frequent items generation

Input: A sequence database D, threshold ζ

Output: Bitmap Pos and frequent itemset L_1

min-sup=ζ*|D|;

Pos=null;

for each database sequence S_i

{G=null;

for each item $e_j \in S_i$

{if($e_j \notin G$)

{creat e_j;

sup(e_j)++;}

if($e_j e_{j+1} \in$Pos)

{$Pos_i(e_j e_{j+1})$.add(e_{j+1}.pos);}

else

{create $e_j e_{j+1}$;

$Pos_i(e_j e_{j+1})$.add(e_{j+1}.pos)}}}

for each sup(e_j)

{if(sup(e_j)$\geq$min-sup)

{$L_1=L_1$.add(e_j);}}

Output Pos and L_1;

Line 1 gives the minimal support min-sup. Line 2 initiates the null bitmap Pos. Lines 3-8 are to calculate the supports of the items, even if an item occurs several time in one transaction, its support can be added at most 1 for one transaction. Lines 9-13 form the bitmap for all the 2-sequences, the position value of the last item of a sequence is used as the position value of the sequence. Lines 14-16 mine frequent items whose supports are

no less than the minimal support min-sup and insert the frequent items into frequent itemset L_1. Line 17 outputs the bitmap Pos and the frequent 1-item set L_1.

Algorithm 2. Frequent k-sequence mining and (k+1)-sequence generation

Input: Bitmap Pos for k-sequences, minimal support min-sup

Output: Bitmap NewPos for (k+1)-sequences

for each sequence S_j in Pos

{if(sup(S_j)≥min-sup)

{L_k.add(S_j);}

else

{delete Pos(S_j);}}

for each Pos_i

{QS-list=null;

for each S_j in Pos

{if(Pos_i(S_j)!=null)

{QS-list.add(S_j.SP);}}

if(QS-list==null || |QS-list|==1)

{delete Pos_i;

continue;}

else

{sort QS-list by SP.pos;}

for each SP in QS-list

{if(SP_m.pos+1==SP_{m+1}.pos)

{Newseq=SP_m.seqSP_{m+1}.seq[k]);

if(Newseq∈NewPos)

{$NewPos_i$(Newseq).add(SP_{m+1}.pos);}

else

{Create $NewPos_i$(Newseq);

$NewPos_i$(Newseq).add(SP_{m+1}.pos);}}}}

Output NewPos and L_k;

Lines 1-5 mine the frequent k-sequences whose support are no less than minimal support min-sup. Lines 6-15 form the QS-list for the ith row in Pos. If the QS-list is null or its length |QS-list| is 1, delete the ith row of the Pos and form the QS-list for the next row in Pos. Lines 16-23 connect frequent k-sequences to generate (k+1)-sequences, the two sequences can be connected only when the position value of the second sequence is 1 more than the position value of the first sequence. The new connected (k+1)-sequence Newseq can be generated by adding the last item of the second sequence to the first sequence. Line 24 outputs the updated bitmap for (k+1)-sequences and the frequent k-sequence set L_k.

3.2. An illustrative example

The process of mining frequent biosequences from the database D in Table 1 is as follows. Let ζ=50%, then min-sup=ζ*|D|=50%*4=2.

The initial bitmap for 2-sequences are first formed in Table 2. All the supports of the items can be calculated. sup(a)=4, sup(b)=4, sup(c)=4. Items a, b, c are all frequent items. Frequent 2-sequences are mined, as sup(ab)=4, sup(bc)=4, sup(cb)=4, sup(ba)=3, sup(ac)=3 and sup(ca)=1. 2-sequence ca and Pos(ca) are deleted because Sup(ca) is smaller than min-sup. Frequent 2-sequences are ab, bc, cb, ba, ac.

Fig.2 shows the QS-list of the frequent 2-sequences in the second transaction(ac has been removed). 3-sequences are easy to be obtained. Just add the last item of cb to ac, and add the last item of bc to cb. Then Pos_2(acb)=3, Pos_2(cbc)=4, ab cannot be connected because

the position value of ab is not next to bc, and connection stops for there are no more sequences after ab. The new bitmap for 3-sequences are shown in Table 3.

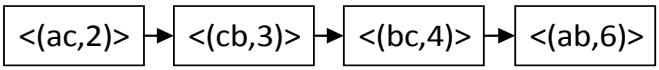

Fig.2. QS-list of the frequent 2-sequences in the second transaction

Table 3. Bitmap for 3-sequences

S / ID	abc	bcb	cba	bac	acb	cbc	bab
1	3	4	5	6	∅	∅	∅
2	∅	∅	∅	∅	3	4	∅
3	6	3	4	∅	∅	∅	5
4	6	∅	4	∅	3	∅	5

sup(abc)=3, sup(bcb)=2, sup(cba)=3, sup(bac)=1, sup(acb)=2, sup(cbc)=1, sup(bab)=2. Frequent 3-sequences are abc, bcb, cba, acb, bab. The new bitmap for 4-sequences are shown in Table 4. The final results for all the frequent sequences are in Table 5.

Table 4. Bitmap for 4-sequences

S / ID	abcb	bcba	cbac	acbc	cbab	babc	acba
1	4	5	6	∅	∅	∅	∅
2	∅	∅	∅	4	∅	∅	∅
3	∅	4	∅	∅	5	6	∅
4	∅	∅	∅	∅	5	6	4

Table 5. Final results for all the frequent sequences

Frequent sequence set	Patterns
L_1	a,b,c
L_2	ab,bc,cb,ba,ac
L_3	abc,bcb,cba,acb,bab
L_4	bcba,cbab,babc
L_5	cbabc

4. Experiments and performance evaluation

Experiments are performed on a PC with Intel(R) Core(TM) 3.6 GHz CPU and 16G main memory, running on Windows 8. We evaluate the run time and scalability of the

algorithm FBSB, and compare it with BioPM[18] and MSPM[20]. There is no comparison about precision or error rate because FBSB algorithm is always 100% accurate. To test the algorithms in the same coding environment, all the programs are written in C++ using Visual Studio 2013.

Dataset used in the experiments has 10 protein families taken from the pfam database[27] as shown in Table 6. The sequence number and the sequence length are different in different families.

Table 6. Protein families in pfam as test dataset

Protein family	pfam accession number	Number of sequences	Average length	Number of sequences tested
Adh_short	PF00106	406662	189.80	195
Lectin_legB	PF00139	3428	219.20	249
Glyco_hydro_19	PF00182	5589	160.50	232
G-alpha	PF00503	4840	306.10	377
TatC	PF00902	16250	205.30	205
Calici_Coat	PF00915	14558	130.80	291
SBP_bac_9	PF01297	32759	270.80	252
Metalloenzyme	PF01676	28310	423.90	245
Birna VP2	PF01766	2744	236.30	442
Peptidase_S66	PF02016	9457	283.70	259

4.1. Test of run time

To test the run time performance of the FBSB algorithm, we chose 2747 sequences from the protein families in the pfam database for the experiment; the average length is 245.50. The sequences are divided into three sets, each experiment is performed three times for different sets and the run time is the average value. In order to make the comparison fair for BioPM and MSPM, each algorithm is tested on the same biological sequence dataset and with the same minimal support threshold min-sup. The run time starts from inputting the dataset and ends with the outputted results, including the processing time. Fig.3 shows the comparison of the average run time of the three algorithms with different minimal support thresholds.

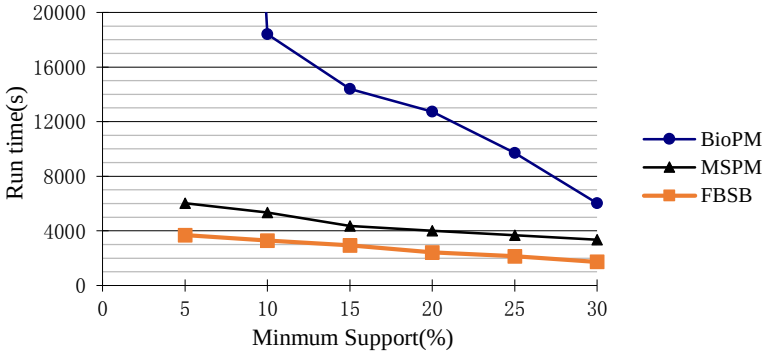

Fig.3. Comparison of the run time with different minimal supports

Under the same conditions, FBSB usually runs faster than the other two algorithms. When the minimal support threshold is small, the run time of BioSM and MSPM are much higher and BioPM seems less stable than MSPM and FBSB. This is because BioPM needs more time to continuously build projective databases, although MSPM significantly reduces time in the main mining section, it needs extra time to divide the original sequences to form primary patterns and the primary pattern tree. Also, after the main mining process, some of the leaves should be checked for further joining and sometimes it may miss a few frequent sequences. FBSB uses the QS-list which can ensure all the tested sequences during the mining process really do occur and all the subsequences of the growth sequences are frequent. It can avoid doing unnecessary connections which may generate candidate sequences, and this again saves in testing time.

4.2. Test of scalability

To test the scalability of the three algorithms, we use the same datasets as testing the run time of FBSB and there are 915 sequences in each datasets. We set the minimal support threshold as 15% when the sequence number grows from 100 to 600. Fig.4 shows the comparison of the run time with different number of sequences.

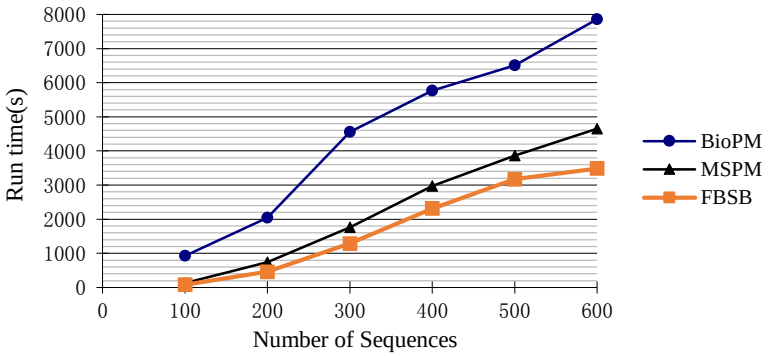

Fig.4. Comparison of the run time with different number of sequences (min-sup=15%)

It can be observed that the run time of the three algorithms increase when there are more sequences. MSPM and FBSB are nearly linear increased. This indicates both MSPM and FBSB have a good scalability, but FBSB always runs faster and its run time increases more gently. Constructing projective database continuously is again the reason why BioPM needs more run time. Sometimes the run time increases significantly if there are more sequences with longer length, so the slope changes more at the point of 200 sequences. MSPM displays superior run time performance with longer sequences, because it usually uses primary patterns without producing lots of short ones. However, the more divisions it does, the greater the risk of frequent sequences being missed. FBSB also mines a lot of short sequences, but all the sequences really do occur and all their subsequences are frequent sequences. There are no candidate generation or useless connections. As sequence length grows, the length of the QS-list will become shorter and the number of QS-list will become smaller. That contributes significantly to the reduction of run time for FBSBS.

5. Conclusion

Since biosequences are based on an alphabet Σ with only 4 or 20 different characters, the number of items in a biosequence is limited and accuracy becomes a most important issue. The FBSB algorithm uses bitmaps to record the sequence position value in each transaction, and a quicksort list QS-list is created for quick sequence connection. They

display a compression characteristic and efficient computation during the mining process, and FBSB will not blindly join the frequent sequences to generate candidate sequences. All the sequences tested really do occur because FBSB algorithm uses position value to guide the connection, and it ensures that the subsequences of the tested sequences are frequent. So less calculating time and memory space are required. After analysing the experimental results, it is shown that FBSB algorithm is more effective with a good scalability. It is faster than the comparison algorithms and will not miss any frequent sequences.

6. Acknowledgment

This work is supported by the National Natural Science Foundation of China under Grant No. 61572420, No. 61472341 and the Natural Science Foundation of Hebei Province of China under Grant No. F2013203324, No. F2014203152 and No. F2015203326. It is also supported by China Scholarship Council. The authors are grateful to valuable comments and suggestions of the reviewers.

Reference

[1] N.M. Luscombe, D. Greenbaum, M. Gerstein, What is bioinformatics? A proposed definition and overview of the field, Methods Inf. Med. 40 (2001) 346-358. doi:10.1053/j.ro.2009.03.010.

[2] A. Brazma, I. Jonassen, I. Eidhammer, D. Gilbert, Approaches to the automatic discovery of patterns in biosequences., J. Comput. Biol. 5 (1998) 279-305. doi:10.1089/cmb.1998.5.279.

[3] R. Agrawal, Fast algorithms for mining association rules, Proc. 20th Int. Conf. Very Large Data Bases. (1994). doi:10.1.1.40.6757.

[4] J. Han, J. Pei, Y. Yin, Mining frequent patterns without candidate generation, Networks. (n.d.) 1-12.

[5] M.J. Zaki, Scalable algorithms for association mining, IEEE Trans. Knowl. Data Eng. 12 (2000) 372-390. doi:10.1109/69.846291.

[6] J. Dong, M. Han, BitTableFI: An efficient mining frequent itemsets algorithm, Knowledge-Based Syst. 20 (2007) 329-335. doi:10.1016/j.knosys.2006.08.005.

[7] W. Song, B. Yang, Z. Xu, Index-BitTableFI: An improved algorithm for mining frequent itemsets, Knowledge-Based Syst. 21 (2008) 507-513. doi:10.1016/j.knosys.2008.03.011.

[8] B. Vo, T.-P. Hong, B. Le, DBV-Miner: A Dynamic Bit-Vector approach for fast mining frequent closed itemsets, Expert Syst. Appl. 39 (2012) 7196-7206. doi:10.1016/j.eswa.2012.01.062.

[9] B. Vo, T.P. Hong, B. Le, A lattice-based approach for mining most generalization association rules, Knowledge-Based Syst. 45 (2013) 20-30. doi:10.1016/j.knosys.2013.02.003.

[10] B. Vo, T. Le, T. Hong, B. Le, Maintenance of a frequent-itemset lattice based on pre-large concept, 245 (2014) 295-305. doi:10.1007/978-3-319-02821-7.

[11] B. Vo, T. Le, T.-P. Hong, B. Le, An effective approach for maintenance of pre-large-based frequent-itemset lattice in incremental mining, Appl. Intell. 41 (2014) 759-775. doi:10.1007/s10489-014-0551-z.

[12] B. Vo, T. Le, F. Coenen, T.-P. Hong, Mining frequent itemsets using the N-list and subsume concepts, Int. J. Mach. Learn. Cybern. (2014). doi:10.1007/s13042-014-0252-2.

[13] R. Agrawal, R. Srikant, Mining sequential patterns, Proc. 11th Int. Conf. Data Eng. (1995) 3-14. doi:10.1016/j.jbi.2007.05.004.

[14] R. Afshar, Mining frequent max and closed sequential, (2002).

[15] N.P. Lin, W. Hao, H. Chen, Fast mining maximal sequential patterns, (2007) 404-407.

[16] G. Benson, Tandem repeats finder: a program to analyze DNA sequences., Nucleic Acids Res. 27 (1999) 573-80. doi:10.1093/nar/27.2.573.

[17] S. Kurtz, J. V Choudhuri, E. Ohlebusch, C. Schleiermacher, J. Stoye, R. Giegerich, REPuter: the manifold applications of repeat analysis on a genomic scale., Nucleic Acids Res. 29 (2001) 4633-4642. doi:http://dx.doi.org/10.1093/nar/29.22.4633.

[18] X. Yun, BioPM : An efficient algorithm for protein motif mining, (2007) 394-397.

[19] Q. Zhou, Q. Jiang, S. Li, X. Xie, L. Lin, An efficient algorithm for protein sequence pattern mining, 2010 5th Int. Conf. Comput. Sci. Educ. (2010) 1876-1881. doi:10.1109/ICCSE.2010.5593815.

[20] L. Chen, W. Liu, Frequent patterns mining in multiple biological sequences, Comput. Biol. Med. 43 (2013) 1444-1452. doi:10.1016/j.compbiomed.2013.07.009.

[21] M. Rashid, R. Karim, Efficient mining of Interesting patterns in large biological sequences, 10 (2012) 44-50.

[22] Ramachandra.V.Pujeri, G.M.Karthik, Frequent periodic cryptic sequence mining in biological data, J. Comput. Eng. Technol. 1 (2013) 46-60.

[23] K. Prasanna, M. Seetha, Efficient and accurate discovery of colossal pattern sequences from biological datasets: a Doubleton Pattern Mining Strategy (DPMine), Procedia Comput. Sci. 54 (2015) 412-421. doi:10.1016/j.procs.2015.06.048.

[24] V. Liao, M.-S. Chen, Efficient mining gapped sequential patterns for motifs in biological sequences., BMC Syst. Biol. 7 Suppl 4 (2013) S7. doi:10.1186/1752-0509-7-S4-S7.

[25] S. Singh, An efficient framework for mining biological network, (2015) 387-392.

[26] S. Naulaerts, P. Meysman, W. Bittremieux, T.N. Vu, W. Vanden Berghe, B. Goethals, et al., A primer to frequent itemset mining for bioinformatics, Brief. Bioinform. 16 (2015) 216-231. doi:10.1093/bib/bbt074.

[27] <http://pfam.sanger.ac.uk/>.