An Efficient Sequential Pattern Mining Algorithm for Motifs with Gap Constraints

Vance Chiang-Chi Liao
Department of Electrical Engineering
National Taiwan University
Taipei, Taiwan
chiangchi@arbor.ee.ntu.edu.tw

Ming-Syan Chen
Department of Electrical Engineering
National Taiwan University
Taipei, Taiwan
mschen@cc.ee.ntu.edu.tw

***Abstract*—Mining biological data can provide insight into various realms of biology, such as finding co-occurring biosequences, which is essential for biological analyses and data mining. Sequential pattern mining reveals all-length implicit motifs, which have specific structures and are of functional significance in biological sequences. Traditional sequential pattern mining algorithms are inefficient for small alphabets and long sequences, such as DNA and protein sequences; therefore, it is necessary to move away from these algorithms. An approach called the *Depth-First Spelling* algorithm for mining sequential patterns (motifs) with *Gap* constraints in biological sequences (referred to as *DFSG*) is proposed in this work. In biological sequences, *DFSG* runtime is substantially shorter than that of *GenPrefixSpan*, where *GenPrefixSpan* is a method based on *PrefixSpan* (*PrefixSpan* is one of the fastest algorithms in traditional sequential pattern mining algorithms).**

Keywords-sequential patterns; data mining

I. INTRODUCTION

Finding co-occurring biological sequences, effective classification of biological sequences, and cluster analysis of biological sequences [5] [9] [10] are critical research issues in biological data analysis and biological data mining. Mining sequential patterns (motifs) from unaligned biological sequences in molecular biology represents that the patterns have specific structures and are functional significance. Sequential pattern (motif) mining algorithms facilitate the identification of co-occurring biological sequences and the discovery of relationships in DNA or protein sequences [13] [14]. Sequential pattern mining has been shown to be useful in the field of biology, such as classifying biological sequences, predicting transcription factor binding sites, recognizing protein folds, and identifying hot regions in protein-protein interactions.

This work proposes a new approach called the *Depth-First Spelling* algorithm for sequential pattern (motif) mining of biological sequences with *Gap* constraints (referred to as *DFSG*). *DFSG* is a generalization algorithm that contains gap constraints. A gap constraint represents the limit in distance between two items in one sequence. It is suited for data character, which has less items and longer sequences, such as biological sequences. Gap constraints skip the irrelevant regions in the evolution process of biological sequences. The user can assign a maximum value of the gap constraint to limit the distance of the separate items.

The traditional problem of sequential pattern mining is that it mines numerous items and short sequences. However, DNA and protein sequences have two distinct features. First, DNA sequences are formed by combinations of four letters, and protein sequences are formed by combinations of 20 letters. Second, the sequence lengths are usually in the hundreds or thousands. Therefore, small alphabets and long sequences in biological data are difficult to process with traditional sequential pattern mining algorithms [1] [4] [11] [19]. Furthermore, numerous items and short sequences, such as transaction sequences, are suited to traditional methods; therefore, biological sequences are processed inefficiently by them.

The *DFSG* is proposed as a method that moves away from traditional sequential pattern mining algorithms, and copes with the problems in execution time. The *DFSG* has a shorter execution time for mining biological sequences, compared to traditional sequential pattern mining methods. A series of experiments was conducted to evaluate the *DFSG* method. First, we used *DFSG* and *GenPrefixSpan* [3] to process real and simulated DNA sequences. We then compared the performance of the two methods in processing gap constraints, protein sequences, and several parameters in simulated biological sequences. As indicated by the results of these experiments, *DFSG* mines biological sequences faster than *GenPrefixSpan*, and it has more scalability.

The remainder of this paper is organized as follows: Section 2 introduces the related works of *DFSG*. Section 3 presents the *DFSG* method; Section 4 provides experimental results, and lastly, Section 5 offers a conclusion.

II. RELATED WORKS

Pattern mining can be used to mine Web log patterns, biological patterns, crime patterns, and purchasing patterns. Consequently, the academic community has proposed many approaches for pattern mining, e.g., traditional sequential pattern mining [1] [4] [11] [19], maximal sequential pattern mining [17], incremental sequential pattern mining [8], sequential pattern mining of data streams [18], progressive sequential pattern mining [15], and closed sequential pattern mining [20]. Some mining research has been conducted in the field of bioinformatics.

978-1-4673-2560-8/12/$31.00 ©2012 IEEE

For example, [6] mined predictive and non-spurious patterns using temporal pattern mining techniques. With the increasing use of biological sequences in bioinformatics, developing methods for fast and timely data analysis is becoming increasingly important.

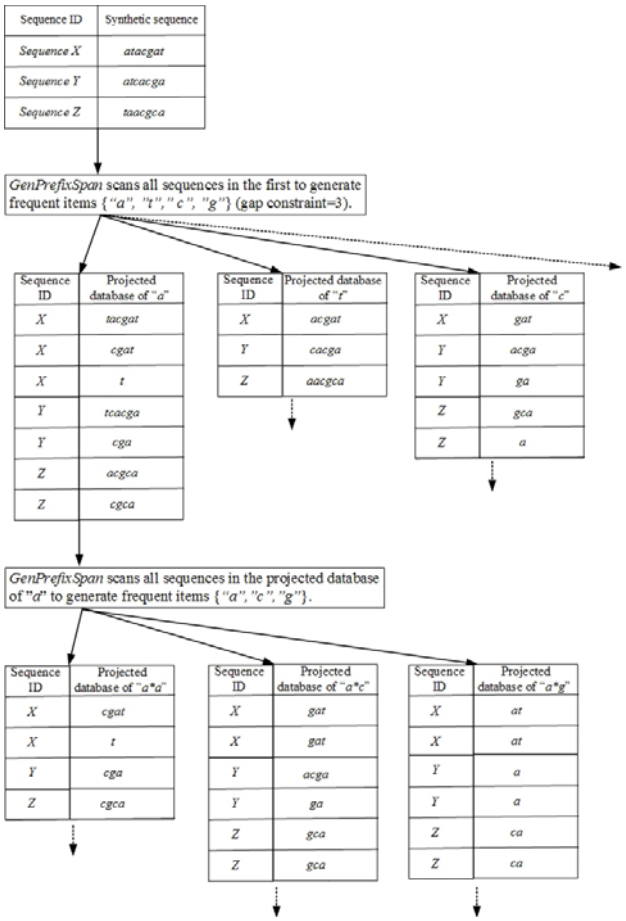

Fig. 1. A partial example of the *GenPrefixSpan* process

Sequential pattern mining problems are related to some traditional biological computing problems [2] [7] [12] [16], such as motif finding problems and sequence alignment problems. Sequential patterns of biological sequences may be useful functions in biology, which represent to have been conserved in biological sequences throughout evolution. The first sequential pattern mining algorithm designed specifically for biological sequences was the 2PDF method [21], but it mined too many patterns and did not handle gap constraints.

The fast processing speed and scalability of DFSG result from several factors. DFSG and the projection-based pattern growth algorithm, GenPrefixSpan, are compared. Unlike traditional projection-based pattern growth algorithms, DFSG does not need to construct corresponding projected databases; therefore, the algorithm does not need to scan these databases. Thus, recursive projecting and scanning time are saved. A partial example of the GenPrefixSpan process is shown in Fig. 1. The databases are projected recursively by GenPrefixSpan until no other frequent items can be generated.

III. DFSG Algorithm and Its Example

This section introduces the Depth-First Spelling algorithm for Gapped sequential pattern mining of biological sequences (referred to as DFSG). Section A describes DFSG, and Section B provides an example of DFSG.

A. DFSG Algorithm

Two steps are executed in the DFSG algorithm. First, DFSG scans the given biological database once to construct three-dimensional indices. Then, gapped sequential patterns are generated by the DFSG-Generation algorithm. DFSG-Generation contains candidate-gapped sequential patterns spellings and gapped patterns verification.

The verification process includes direct access and binary search with the three-dimensional indices. The position of the next occurrence of each letter depends on the letter's prefix in each pattern-generating process. Hence, the three-dimensional indices and the counting matrix for DFSG-Generation were designed. The position of the next occurrence cannot be stored in advance for the counting matrix because the total number of possible positions for the next occurrence is too large and unknown.

Definition 1. Let $E = \{e_1, e_2, ..., e_A\}$ be the set of all letters. In addition, let A=20 be a simulated protein sequence and A=4 be a simulated DNA sequence. An ordered list of letters is a *sequence*. A sequence $\boldsymbol{s}$ is denoted by $\{s_1s_2s_3...s_n\}$, where s_i is a letter. A letter can appear several times in one sequence, and biological sequences are generally long.

Definition 2. A sequence u is denoted by $\{u_1u_2u_3...u_q\}$, where u_i is a letter. If $\{u_1u_2u_3...u_q\}$ can be mapped to $\{s_1s_2s_3...s_n\}$ $(q \leqq n)$ sequentially, u is contained in the sequence s. In this situation, u is called a *subsequence* of s.

Definition 3. The number of tuples containing α in the biological data set is the *support count* of a sequence α. The sequence α is called a gapped sequential pattern if the minimum support count is less than the support count.

Basically, any kind of biological sequence is not limited with the gapped sequential pattern mining problem. An example of a biological sequence is the following: {*X: atacgat*, *Y: atcacga*, *Z: taacgca*}.

Definition 4. A sequential pattern p is denoted by $\{p_1p_2p_3...p_m\}$, where p_m is a letter. A pattern p is a subsequence of s, if s contributes the support count to p. A pattern p satisfies *the gap constraint* G, if the position of s_i plus the value of the gap constraint and one is larger than the position of s_k, where p_j is mapped to s_i and p_{j+1} is mapped to s_k.

Definition 5. The numbers of letter A_i, the numbers of sequence E_j, and the numbers of the position W_k are three dimensions of the three-dimensional indices. The letter A_i of the first dimension, which occurs in the sequences of the second dimension E_j is the position W_k.

Definition 6. The multiple W_k in each E_j for the spelling gapped sequential pattern process is represented by *the counting matrix* C_l. Let $\beta=<t_1t_2...t_{n-1}t_n>$ be a sequence with the prefix α that $\alpha=<t_1t_2...t_{n-1}>$ is a gapped sequential pattern in a biological data set. The counting matrix C_l of β is represented by W_k. (In each E_j, W_k can be null.) The letter $<t_n>$ in each E_j in the three-dimensional indices determines the value of W_k. The counting matrix C_l of α must be less than the value of W_k for β.

The counting matrix records the positions of the latest item for the gapped pattern. The latest positions pass the gap constraint in each sequence.

Definition 7. The number of rows that have value in the *counting matrix* C_l is the *support count* of β. A pattern β is regarded as a sequential pattern only if the support count γ is larger than the minimum support count.

B. Example: DFSG Algorithm

The execution of the DFSG algorithm is illustrated in the following example. DFSG mines a set of letters {*a, t, c, g*} in the sequence of database *D* {*X: atacgat*, *Y: atcacga*, *Z: taacgca*} with a minimum support $\lambda = 3$ in this example. The following are the steps. First, DFSG scans *D* once to build the three-dimensional indices. The *sequence S* is scanned, and the scanned letter A_i is inputted into the three-dimensional indices with the value W_k by DFSG. Second, sequential patterns are found by using DFSG-Generation, as shown in Fig. 2. To enumerate candidate-gapped sequential patterns α, DFSG spells letter *I* depth-first. The three-dimensional indices and the counting matrix C_l are used to verify the support of candidate-gapped pattern α. The pattern is a real sequential pattern, and the process continues to run recursively if the support of a candidate pattern α is larger than the minimum support.

The depth-first process spells the candidate pattern "*C*A*" in Fig. 2. To find a minimum value that is larger than the value in the *Cl* position, the values in dimension "*A*" of the three-dimensional indices are searched. The previous *Cl* {(4),(3,5),(4,6)} is less than the new *Cl* {(6),(4,7),(7)}. The candidate-gapped pattern "*C*A*" is a gapped sequential pattern because the *support* is greater than the *minimum support* 3. Spelling and verifying candidate sequential patterns in a depth-first manner are continued by the process. Then, another candidate pattern "*A*T*" is observed. DFSG searches the values in dimension "*T*" of the three-dimensional indices. The new *Cl* is {(2,7),(2),(-)} and the *support* is 2. Thus, the candidate-gapped pattern "*A*T*" fails to be a sequential pattern. The process does not continue to spell subsequent candidate gapped patterns after this candidate's gapped pattern because this spelling failed.

The DFSG-Generation Algorithm

Input: ***The gap constraint G, the support threshold λ,*** **and** ***the three-dimensional indices P.***
Output: The sequential patterns with the gap constraint

```
1.  for S = 1 to N do  /* N is the number of sequences.*/
2.  Initialize count_flag = false;
3.    for I = 1 to I_N do
4.     /* I_N is the number of indices in each sequence.*/
5.       Initialize search_flag = false;
6.       Initialize first = 1;
7.       Initialize last = the length of P(W, S);
8.        /* W is the letter. */
9.       Initialize count = 0;
10.      if C(S,I) has a value then
11.       /* C is the counting matrix. */
12.         while first ≦ last
13.            m = (first + last) / 2;
14.            if C(S,I) less than P(W, S, m) then
15.                 near = P(W, S, m);
16.                 last = m - 1;
17.                 search_flag = true;
18.            else
19.                 first = m + 1;
20.             end if
21.          end while
22.       end if
23.  if search_flag equals true and near < C(S,I)+G then
24.        C(S,I) = near;
25.         count_flag = true;
26.  else
27.        C(S,I) = null;
28.  end if
29.    end for
30.    if count_flag equals true then
31.       count = count + 1;
32.    end if
33.    if count ≧ λ then
34.       break;
35.    end if
36.  end for
37.  if count ≧ λ then
38.       for each letter W do,
39.          call DFSG-Generation (W, C, P);
40.       end for
41.  end if
```

IV. PERFORMANCE EVALUATION

A number of experiments were conducted to evaluate the performance of DFSG. DFSG performance and that of GenPrefixSpan in mining real and simulated DNA sequences are compared in Section A. Gap constraints, simulated protein sequences, and several parameters were tested for the two algorithms in Section B. We performed all the experiments on a 3.20 GHz Pentium(R) 4 PC with 1 GB of RAM. The operating system was Microsoft Windows XP Professional (2002), and we wrote the programs with Microsoft Visual C++ 6.0.

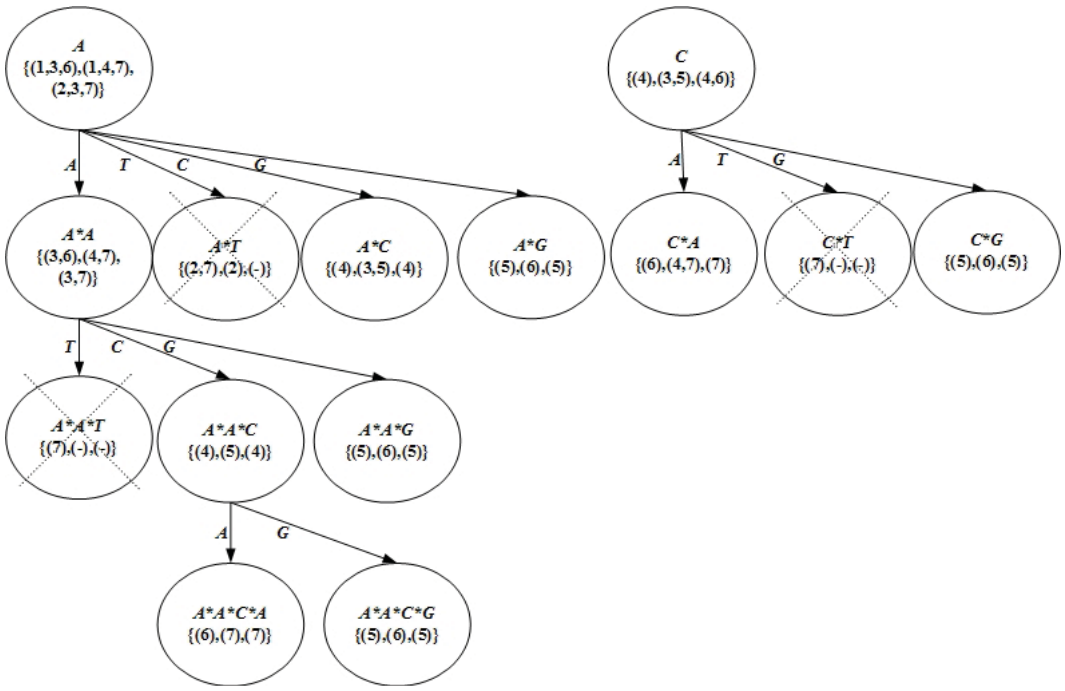

Fig. 2. An example of partial *DFSG-Generation* with *the counting matrix*.

A. Real and Simulated DNA Sequences

Real DNA sequences obtained from the National Center for Biotechnology Information (NCBI) were used to evaluate the algorithms. In the following discussion, L represents the length of a sequence, A is the number of letters, G is the value of gap constraint, S is the minimum support, and N is the number of sequences. In the experiments for DNA sequences, L is increased from 25 to 35 when A=4 (for real and simulated DNA sequences). DFSG outperformed GenPrefixSpan in real DNA sequences, as shown in Figs. 3-5. Here, the number of letters for real DNA sequences is 4; the values of gap constraint are 10, 7, and 5; the lengths of the sequences are 25, 30, and 35; and the number of sequences is 1000. The vertical axis represents the runtime (in seconds) and the horizontal axis represents the minimum support in each figure. GenPrefixSpan's runtime divided by that of DFSG is the runtime rate. In Fig. 5, the runtime rates for real DNA sequences are 8.68, 11.27, 14.77, 20.94, and 30.18. As the minimum support becomes larger, the rate increases.

We used the synthetic DNA sequence data in the experiments followed [1]. According to Figs. 3-5, each figure shows that DFSG outperforms GenPrefixSpan in simulated DNA sequences. The letters are simulated for DNA sequences in this experiment. The lengths of the sequences are 25, 30, and 35; the number of letters is 4; the value of the gap constraint is 5; and the number of sequences is 3000. In Fig. 5, the runtime rates for simulated DNA sequences are 14.04, 22.88, 28.31, 41.41, and 68.93. As the minimum support increases, the rate increases invariably. DFSG performance for the synthetic DNA sequences is the same as those for the real ones in the experiment.

B. Gap Constraints, Simulated Protein Sequences, and Several Parameters

We used gap constraints, simulated protein sequences, and several parameters to compare the performance of DFSG with that of GenPrefixSpan. In the experiments, the value of the gap constraint G is increased when A=4 (for simulated DNA sequences). As shown in Figs. 6-7, DFSG outperforms GenPrefixSpan with variable gap constraints. The number of letters is 4; the lengths of the sequences are 40 and 50; and the numbers of sequences are 1000 and 3000.

As shown in Fig. 8, DFSG outperforms GenPrefixSpan when N is increased. The numbers of sequences are 4000, 5000, 6000, 7000, 8000, and 9000; the number of simulated DNA sequences is 4; the length of the sequences is 30; the minimum support is 0.9; and the value of the gap constraint is 3. Moreover, despite GenPrefixSpan being scalable [3], DFSG is more scalable. Because of the proportional scale, DFSG runtimes appear to form a straight line (Fig. 8). The runtimes are 2.781 s (4000 sequences), 2.890 s (5000 sequences), 2.937 s (6000 sequences), 3.015 s (7000 sequences), 3.125 s (8000 sequences), and 3.187 s (9000 sequences). In Fig. 8, the variations are not as obvious. The runtime rates are 37.12, 55.38, 77.24, 102.32, 128.77, and 160.72. The runtime rate increases with the number of sequences.

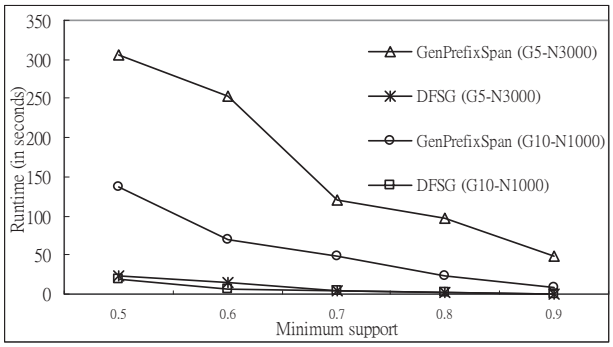

Fig. 3. Execution time for real DNA sequences (L=25, A=4, G=10, and N=1000) and simulated DNA sequences (L=25, A=4, G=5, and N=3000).

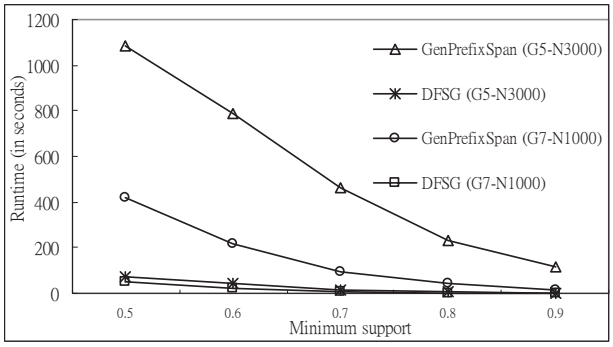

Fig. 4. Execution time for real DNA sequences (L=30, A=4, G=7, and N=1000) and simulated DNA sequences (L=30, A=4, G=5, and N=3000).

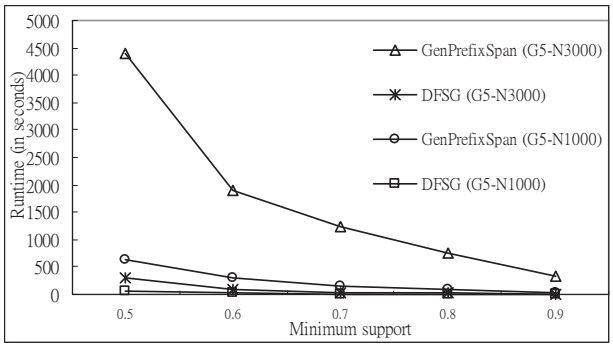

Fig. 5. Execution time for real DNA sequences (L=35, A=4, G=5, and N=1000) and simulated DNA sequences (L=35, A=4, G=5, and N=3000).

The DFSG algorithm mines much faster than GenPrefixSpan when L is increased (Fig. 9). The lengths of the sequences are 45, 46, 47, 48, 49, and 50; the number of letters is 4; the minimum support is 0.9; the value of the gap constraint is 5; and the number of sequences is 1000. Compared to GenPrefixSpan, the runtime of DFSG shows a steady rise as L increases. If the maximum sequence length is larger than 50, the DFSG runtime appears more as a straight line in the figure because of the proportional scale. Moreover, we have already shown that DFSG is significantly faster than GenPrefixSpan in this experiment when L is increased.

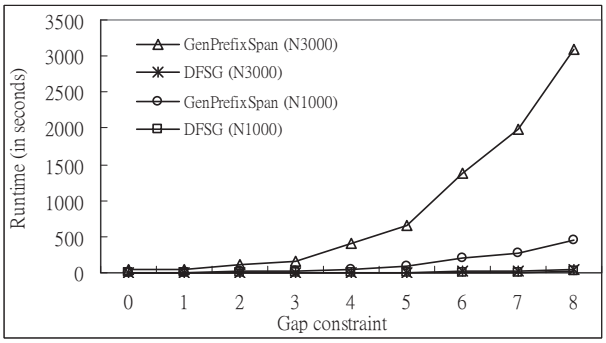

Fig. 6. Execution time for the increased values of the gap constraint (the number of letters is 4; the length of the sequences is 40; and the numbers of sequences are 1000 and 3000).

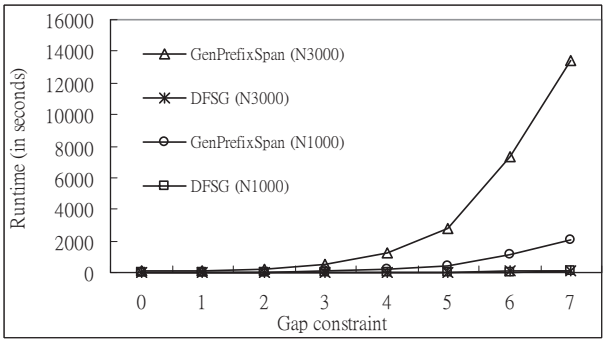

Fig. 7. Execution time for the increased values of the gap constraint (the number of letters is 4; the length of the sequences is 50; and the numbers of sequences are 1000 and 3000).

As shown in Fig. 10, DFSG outperforms GenPrefixSpan when A=20 in simulated protein sequences. In this experiment, the letters are simulated protein sequences. The number of letters is 20; the length of the sequences is 100; the number of sequences is 500; and the value of the gap constraint is 20.

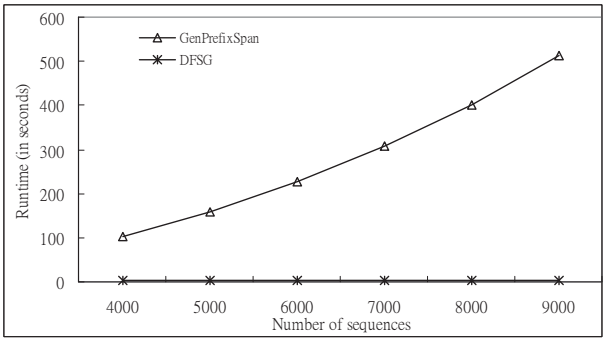

Fig. 8. Execution time for the increased numbers of sequences (L=30, A=4, G=3, and S=0.9).

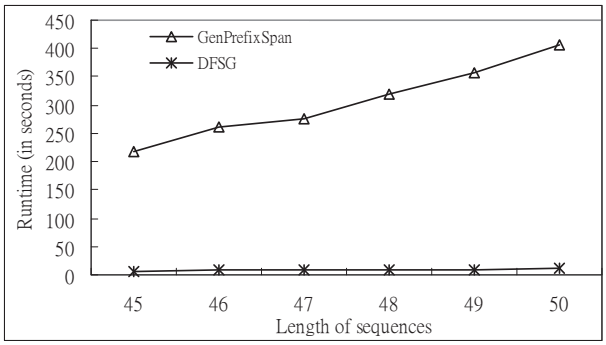

Fig. 9. Execution time for the increased lengths of sequences (A=4, N=1000, G=5, and S=0.9).

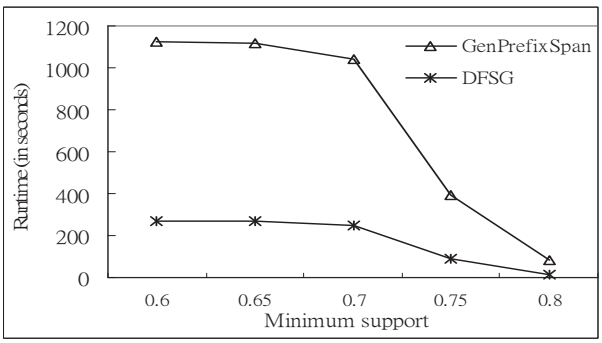

Fig. 10. Execution time for simulated protein sequences (A=20, L=100, G=20, and N=500).

To assess the scalability of DFSG, N is increased from 100k to 500k. DFSG's runtimes are 36.218, 73.640, 108.390, 152.109, and 206.640 s, and the numbers of sequences are 100k, 200k, 300k, 400k, and 500k, respectively. When the data sets are larger, DFSG is scalable in the experimental results. It is steady and small for the growth rate of execution time. Here, the number of letters in the simulated DNA sequences is 4; the minimum support is 0.9; the value of the gap constraint is 3; and the length of the sequences is 30. Total experimental results show that DFSG outperforms GenPrefixSpan in various parameters, including scalability, simulated DNA/protein biological sequences, real biological sequences, and gap constraints.

V. Conclusion

DFSG is faster than GenPrefixSpan in mining biological (DNA and protein) sequences, which have few letters and long sequence lengths. DFSG can be used to discover the interactions and relationships in genes or proteins; therefore, it can be useful for biologists. Faster algorithms related to the DFSG and other algorithms that help mine biological data will be designed in our future research.

References

[1] R. Agrawal and R. Srikant, Mining Sequential Patterns, *Proceedings of the 11th International Conference on Data Engineering*, pages 3-14, Feb. 1995.

[2] S.F. Altschul, W. Gish, W. Miller, E.W. Myers, D.J. Lipman, Basic local alignment search tool, *Journal of Molecular Biology*, 215(3):403-410, 1990.

[3] C. Antunes and A. Oliveira , Generalization of Pattern-growth Methods for Sequential Pattern Mining with Gap Constraints, *Proceedings of the International Conference on Machine Learning and Data Mining in Pattern Recognition (MLDM'03), Lecture Notes in Computer Science*, Leipzig, Germany, pages 239–251, 2003.

[4] J. Ayres, J. Gehrke, T. Yiu, and J. Flannick, Sequential pattern mining using a bitmap representation, *Proceedings of 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 429 – 435, Jul. 2002.

[5] P. Bajcsy, J. Han, L. Liu and J. Young, Survey of Biodata Analysis from a Data Mining Perspective, Chapter 2 of Jason T. L. Wang, Mohammed J. Zaki, Hannu T. T. Toivonen, and Dennis Shasha (eds.), *Data Mining in Bioinformatics*, Springer Verlag, pages 9-39, 2004.

[6] I. Batal, H. Valizadegan, G. F. Cooper and M. Hauskrecht, A Pattern Mining Approach for Classifying Multivariate Temporal Data, *Proceedings of the IEEE International Conference on Bioinformatics and Biomedicine*, 2011.

[7] BLAST, http://www.ncbi.nlm.nih.gov/Education/BLASTinfo/

[8] H. Cheng, X. Yan, and J. Han. Incspan: Incremental mining of sequential patterns in large database. *Proceedings of 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 527 – 532, 2004.

[9] J. Han, How can data mining help bio-data analysis? Proceedings of the Workshop on Data Mining in Bioinformatics (BIOKDD'02 with SIGKDD'02 Conference), Edmonton, Canada, pages 1-4, 2002.

[10] J. Han and M. Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, 2000.

[11] J. Han, J. Pei, and X. Yan, Sequential Pattern Mining by Pattern-Growth: Principles and Extensions, *Recent Advances in Data Mining and Granular Computing*, W.W. Chu and T.Y. Lin, eds. Springer, 2005.

[12] M. Hirosawa, Y. Totoki, M. Hoshida, M. Ishikawa, Comprehensive study on iterative algorithms of multiple sequence alignment, *Bioinformatics*, Vol. 11, pages 13-18, 1995.

[13] C.-M. Hsu, C.-Y. Chen, and B.-J. Liu. MAGIIC-PRO: detecting functional signatures by efficient discovery of long patterns in protein sequences. *Nucleic Acids Res.*, W356-W361, 2006.

[14] C.-.M. Hsu, C.Y. Chen, B.J. Liu, C.C. Huang, M.H. Laio, C.C. Lin, and T.L. Wu. Identification of hot regions in protein-protein interactions by sequential pattern mining. *BMC Bioinformatics*, 8(Suppl. 5), S8. 2007.

[15] J.-W. Huang, C.-Y. Tseng, J.-C. Ou, M.-S. Chen, A General Model for Sequential Pattern Mining with a Progressive Database, *IEEE Transactions on Knowledge and Data Engineering*, 11 Feb 2008.

[16] N. Jones, and P. Pevzner, *An introduction to Bioinformatics Algorithms*. MIT Press, Cambridge, MA, 2004.

[17] C. Luo and S. M. Chung. Efficient mining of maximal sequential patterns using multiple samples. *Proceedings of the 5th SIAM International Conference on Data Mining (SDM'05)*, 2005.

[18] A. Marascu and F. Masseglia. Mining sequential patterns from data streams: a centroid approach. *Journal of Intelligent Information Systems (JIIS)*, 27(3):291 – 307, Nov. 2006.

[19] J. Pei, J. Han, B. Asl, J. Wang, H. Pinto, Q. Chen, U. Dayal, and M. Hsu, Mining Sequential Patterns by Pattern-Growth: The PrefixSpan Approach, *IEEE Transactions on Knowledge and Data Engineering*, Oct 2004.

[20] J. Wang and J. Han. Bide: Efficient mining of frequent closed sequences. *Proceedings of the 20th International Conference on Data Engineering,* pages 79 – 91, 2004.

[21] K. Wang, Y. Xu, J. Yu, Scalable Sequential Pattern Mining for Biological Sequences, *Proceedings of the ACM International Conference on Information and Knowledge Management*, 2004.