Finding Transposable Elements in Genomic Data by Frequent Sequential pattern detection

Chia-Hung Yang
Northeastern University
Boston, Massachusetts, United States
yang.chi@husky.neu.edu

1 INTRODUCTION

Transposable elements (TEs) are genetic factors that do not have a constant location but occasionally move to a new position in the genomes. The transposition behavior leads to recurrently copying of the TE sequence and thus become prevalent. Transposable elements have gained recent attention from evolutionary biologists since it has been shown to cause mutations and chromosome rearrangement. Despite fruitful experimental identification of transposable elements in eukaryotic genomes, it will be influential to take advantage of modern sequencing techniques and detect TEs by mining frequent sequential patterns in nucleotide data.

The frequent sequential patterns finding problem is sketched as follow: Given sequences of characters as input, the objective is to reveal all the sequential patterns whose number of occurrences in the input data is not less than a pre-specified threshold. In the case of genomes the set of possible characters corresponds to the four nucleotides, and the outcome patterns reconcile to repeated sequences in DNAs with at least certain prevalence. Thanks to cheap high-throughput methods and consequently rich genome databases, the approaches of mining frequent patterns have a potential to provide an more efficient way uncovering TEs than identifying them from biological experiments.

Frequent sequential pattern detection however has faced a few challenges in genome data. While the alphabet set only consists of four different types of nucleotides, this task is intrinsically difficult because the length of the candidate patterns can be large. Therefore the brute-force counting approach will require enormous memory to store all the uncovered patterns and multiple passes scanning through the genome to enumerate subsequence instances of various length, which is computationally intractable. Furthermore, the revealed sequential patterns are sensitive to the pre-specified support threshold. A small threshold would likely lead to a remaining large fraction of all possible patterns, while a high offset can implicitly rule out long sequential patterns.

In this project I proposed a null-model technique to select support thresholds for patterns of different lengths such that the sequential patterns filtered from the data are naively significant. This threshold selection procedure serves as a pre-computation step of mining frequent sequential patterns, which not only makes the outcomes more reliable but also gives us a range of pattern lengths of interests. I implemented a position-based algorithms of finding frequent patterns and applied it to both genome data of *Arabdopsis thiliana* and its TE database. By comparing the results from the two datasets, the vaibility to mine transposable elements from genomes using frequent mining detection with threshold selection was assessed.

2 RELATED WORKS AND BACKGROUND INFORMATION

Finding frequent sequential patterns is a task of hard-counting subsequence instances in data. A standard model scans subsequences through the input sequences by a fixed-length window. With multiple scans of varying length, all subsequence instances can be enumerated and the supports of sequential patterns are obtained. Various approaches have been proposed to either reduce the number candidate patterns being tracked or to avoid passes through the data. For example, the popular *A-priori* algorithm in market basket model can be generalized to finding frequent sequential patterns due to the similar monotonicity: If a pattern is frequent than so are its prefix and suffix. Therefore as the subsequence instances are scanned in increasing order of length, candidate patterns either whose prefix or suffix is not frequent can be dropped to reduce memory costs.

A spanning tree data structure was further introduced by Kang et al. [5] to efficiently construct candidate patterns. In this tree data structure any node except the root is associated with a character and an integer. The tree satisfies the property that, let p be the string of characters that a path traverses from the root to a node v, the support of pattern p is stored as the integer assigned to node v. As a result, searching for existence of a sequential pattern or its support can be done in $O(l)$ computation, where l is the length of the target pattern. Constructing candidate patterns of length l is hence realized as follow: For each pattern p of length $l-1$ in the tree, search for its length-$(l-2)$ suffix. If the suffix node exists and one of its child represents another frequent pattern, then a candidate pattern is found by appending the character stored in the child to p.

Another class of approaches that utilize the contiguousness property of sequential patterns have been developed to avoid multiple passes scanning through the sequence data. Farzana Zerin and Jeong [4] proposed to additionally store position information of a sequential pattern in the spanning tree data structure, specifically a set of starting positions of the pattern instances. The idea was that if an instance of pattern p's prefix and suffix are adjacent to each other (i.e. off by one position), then it implies an instance of pattern p. For example and illustration, an instance of ATA starting at position 5 and another TAC instance at position 6 together indicate an ATAC instance starting at position 5. Consequently the algorithm suggested in [4] computes the support of candidate patterns by, instead of scanning through the sequence data, first searching for prefix-suffix pairs and then merging their adjacent instances. The instances of candidate patterns are simultaneously constructed so the algorithm can be applied iteratively. Such a position-based

method is able to obtain support of sequential patterns with a constant number of passes through the data. Tanvee et al. [6] improved the position-based approach by reversing steps of support computation. They found that, to build instances of length-l candidates, it is computationally more efficient to first sort all frequent length-$(l-1)$ patterns by its last occurring position and then join instances in position-order if they form a candidate pattern.

Despite a great diversity of studies on frequent sequential pattern detection approaches, practical leverage and selection of the pre-specified support threshold has received less attention. Intuitively applying different support thresholds to patterns of different lengths would result in a more ideal detection outcome. First, the number of possible sequential patterns grows exponentially with its length; second, that a short pattern is not frequent with respect to a small threshold implicitly adds a strong constraint on the data, and conversely it is hardly probable that any short pattern would be filtered out under any support threshold uniform to pattern lengths. We shall see a minimal way to select support thresholds for different pattern lengths in the next section.

3 PROPOSED APPROACH

Here I formally define the problem of finding frequent sequential patterns. Let genome sequence data $\mathbb{D}$ be a set of ordered arrays of characters which belong to the alphabet set Ω. Denote $\mathbb{S}$ as the space of all possible sequential patterns, $\lambda(p)$ as the length of pattern p and $s(p)$ as its support in $\mathbb{D}$. Let $\mathbb{L} = \{l_{min}, l_{min}+1, ..., l_{max}\}$ be the pattern lengths of interests and given the corresponding support thresholds $\{s_l : l \in \mathbb{L}\}$, the task of frequent sequential pattern detection is to find the set of frequent sequential patterns

$$\left\{p \in \mathbb{S} : \lambda(p) \in \mathbb{L} \text{ and } s(p) \geq s_{\lambda(p)}\right\} \tag{1}$$

3.1 Support Thresholds of Significance

In frequent pattern detection, the support thresholds $\{s_l : l \in \mathbb{L}\}$ directly affect the filtered sequential patterns, hence a devised threshold selection is more preferable than assuming an arbitrary constant. In order to ensure that the outcome patterns are "significant", a null model is assumed where I tune the support thresholds such that presence of a frequent pattern is deemed a rare event. Specifically, I consider a minimal null model in which a pattern instance of length l is assumed uniformly at random sampled from the space of all patterns with length l, $\mathbb{S}_l = \{p \in \mathbb{S} : \lambda(p) = l\}$. As a result for every $p \in \mathbb{S}_l$ the instance realizes p with probability $\mu_l = 1/|\mathbb{S}_l| = 4^{-l}$, and its support in $\mathbb{D}$ exactly follows a binomial distribution

$$\mathbb{P}\left(s(p) = k \mid \lambda(p) = l\right) = \binom{n-l}{k} \mu_l^k (1-\mu_l)^{n-l-k} \tag{2}$$

where n is the total number of characters (nucleotides) in $\mathbb{D}$. For all $l \in \mathbb{L}$, the the support threshold is selected as the minimum support such that its cumulative distribution exceeds a given confidence level α:

$$s_l = min\left\{m : \sum_{k=0}^{m} \mathbb{P}\left(s(p) = k \mid \lambda(p) = l\right) \geq \alpha\right\} \tag{3}$$

In practice the number of nucleotides n is large in genome sequence data and thus the support distribution is approximately Poisson,

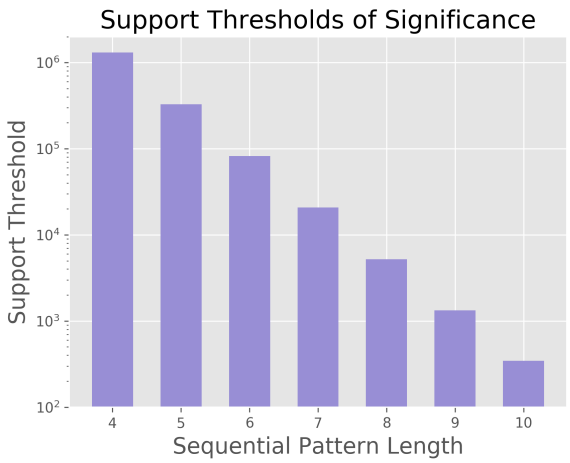

Figure 1: Support thresholds for patterns of various lengths in the *Arabdopsis thiliana* genome data. The thresholds and the range of pattern length were computed using (3) and (4) with minimum length $l_{min} = 4$, lower bound of support thresholds $s_{min} = 100$ and confidence level 0.9.

whose cumulative function has a closed form and enables efficient computation of the thresholds $\{s_l : l \in \mathbb{L}\}$.

The proposed threshold selection method also informs a range of pattern lengths of interests. The formalism of support thresholds in (3) entails that all the thresholds have a lower bound 0. It nevertheless reflects the fact that once the support threshold reaches the lower bound, all possible sequential patterns will be determined frequent and leads to a undesired scenario. Furthermore, as l increases such that the probability μ_l is small enough, the support thresholds would reach this zero lower bound. Therefore, the null-model threshold selection implies a reasonable maximum pattern length of interests above which prevalence of sequential patterns is undetectable. Alternatively, one can tune the support threshold lower bound s_{min} as a hyperparameter (along with the minimum pattern length l_{min} and the confidence level α) and obtain the maximum length of interests

$$l_{max} = max\left\{l : s_l \geq s_{min}\right\} \tag{4}$$

Figure 1 shows the support thresholds and the range of pattern lengths of interests for *Arabdopsis thiliana* genome data described in section 4. It shows that the support thresholds decrease exponentially as the pattern length increases. The decreasing trend can be explained by that, in the null model, the probability μ_l for a pattern to be sampled also decreases exponentially with l. Conversely, Figure 1 indicates a crucial property that the range of pattern lengths of interests scales logarithmically with the size of genome sequence data, especially $l_{max} \sim O(log(n))$.

The selected support thresholds can be incorporated into a frequent sequential patterns finding algorithm, yet only in an indirect fashion. Specifically, support thresholds decreasing with pattern lengths force it not effective to uncover candidate patterns using the monotonicity property of sequences. For instance, supposed

that for some length l we have $s_{l+1} < s_l$. Applying the monotonicity property and constructing candidate patterns of length $l+1$ leads to sequential patterns with support not less than s_l. However, since $s_{l+1} < s_l$, it is not sufficient to find all frequent patterns of length $l+1$ (i.e. whose supports are not less than s_{l+1}) by searching within those candidates. On the other hand, the desired frequent patterns will be included in the outcomes that a detection algorithm returns, where it is given a constant threshold s^* that is less than all the selected support $\{s_l : l \in \mathbb{L}\}$. Here I summarize the procedure to find frequent sequential patterns with respect to the selected support thresholds. I first selected the support thresholds and the range of pattern lengths of interests using (3) and (4). Then I ran a position-based method to obtain patterns with support not less than the lower bound s_{min}. Finally, I imposed the selected support thresholds on the revealed patterns and find the desired frequent sequential patterns.

3.2 Implementation of Position-based Method

I implemented a position-based method for frequent pattern detection followed from and inspired by related studies described in section 2. Let $\mathbb{X}$ be the space of position information of subsequence instances, any of whose elements x is a tuple of sequence ID in the input data and the starting position of the instance in that sequence. The implementation also involved indexing frequent patterns. Denote the index of a pattern p as $i(p)$ and $\mathbb{I}_l$ as the space of indices for frequent patterns[1] of length l. Let mappings $\psi_l : \mathbb{I}_l \times \mathbb{I}_l \to \{\text{True}, \text{False}\}$ indicate whether two patterns of length l jointly form a candidate pattern of length $l+1$ and $\phi_l : \mathbb{I}_l \times \mathbb{I}_l \to \mathbb{I}_{l+1}$ be the index of that candidate pattern. Last but not least, denote $\mathbb{Q}_l$ as an array of subsequence instances of length l, i.e. tuples of its position and the pattern index, that is ordered increasingly by position information x. An iteration of my position-based implementation is outlined below:

(1) With the tree data structure described in section 2 to store frequent patterns that have been found, construct the candidate patterns of length $l+1$, add them to the tree and simultaneously build mappings ψ_l and ϕ_l.
(2) For every neighbor instances q_1 and q_2 in $\mathbb{Q}_l$, say $q_1 = (x_1, i(p_1))$, $q_2 = (x_2, i(p_2))$ and $x_1 < x_2$, if p_1 and p_2 jointly form a candidate p where $\lambda(p) = l+1$ and

$$i(p) = \phi_l(i(p_1), i(p_2)) \tag{5}$$

, put an instance $q = (x_1, i(p))$ into the array $\mathbb{Q}_{l+1}$.
(3) As building up $\mathbb{Q}_{l+1}$ in step 2, update the supports stored in the tree.

Note that it is the ordered nature of $\mathbb{Q}_l$ that makes the above implementation efficient and only requires one scan through the array.

The position-based method has its memory limitation despite its capability to reduce the number of passes through the data. The number of frequent pattern instances can be numerous when the array $\mathbb{Q}_{l_{min}}$ is initially built, and storing it may not be feasible since it exceeds the allocated memory. Consequently, I further implemented a "hybrid" method, where for every iteration the algorithm used *A-priori* update as default. While the number of frequent pattern instances is small enough that can fit in the allocated memory, the algorithm initialized the instance array $\mathbb{Q}_l$ and switched to a position-based method.

[1]To clarify, for this subsection the term "frequent patterns" refers to the ones with support not less than a constant threshold, which was the lower bound s_{min} in the case.

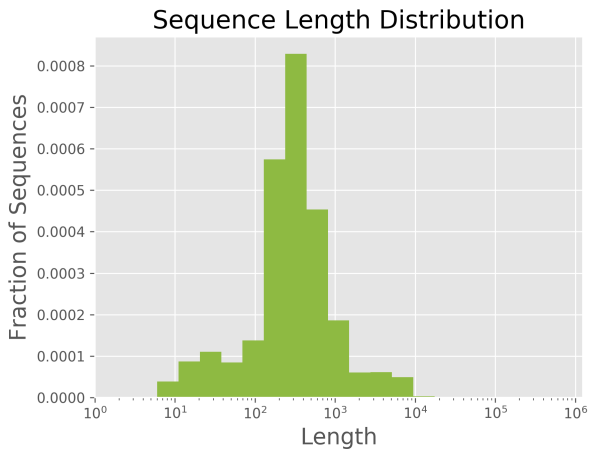

Figure 2: Histogram of sequence length distribution in genome data [3]. The size of nucleotide sequences ranges over orders of magnitude, while a great fraction of them consist of 100 1000 base pairs.

4 EXPERIMENTS

The main dataset I analyzed in this project is the *Arabdopsis thiliana* genome sequence data [3]. It contains 118554 nucleotide sequences assayed from the subjects, whose size ranges from 6 to 691915 base pairs and has 336092761 base pairs in total. Figure 2 provides a glance of the length distribution of this genome dataset. Besides the *Arabdopsis thiliana* genome data, I also looked into a transposable elements database — RepBase [2]. RepBase documents experimentally identified TEs in *Arabdopsis thiliana*, which are 513 long DNA sequences with total 1624389 base pairs.

4.1 Computational Costs

With the support threshold selection approach and the implementation of frequent pattern detection algorithms, I first would like to inspect the computational cost reduction of the position-based method. I generated a group of samples with various sizes from the genome data. Specifically, each nucleotide sequence in the genome data is retained with a sampling probability. By tuning this sampling probability, I obtained samples with total number of base pairs ranging over orders of magnitudes. Additionally, the range of pattern lengths of interests was inferred for each sample. I then applied an *A-priori* algorithm described in section 2 and the position-based implementation stated in section 3 to them and recorded the number of passes through the sampled data. Figure 3 demonstrates the computational costs of the two approaches. The position-based implementation displayed a constant number of passes. In contrast, the *A-priori* algorithm required passes through the sampled data, whose number increases with the sampling size. This deviation

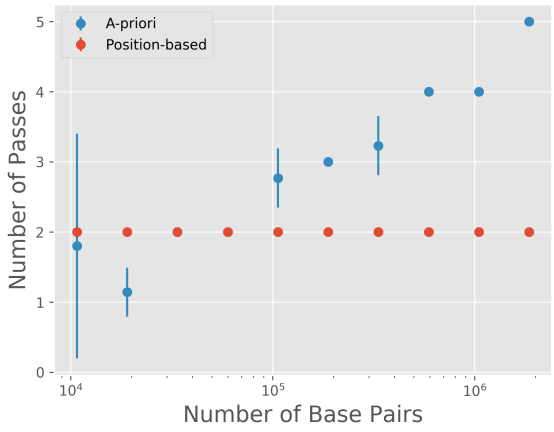

Figure 3: Computational costs of the *A-priori* algorithm and the position-based approach on sampled genome data. The values of sampling probability consist of 10 logarithmically spaced numbers from 5×10^{-5} to 5×10^{-3}, with each 10 samples were drawn. Results of the 100 experiments were collected into 10 logarithmic bins, where the average number of passes within a bin and its standard deviation are shown. The experiments indicate that the position-based implementation is less computationally costly than the *A-priori* algorithm for large data.

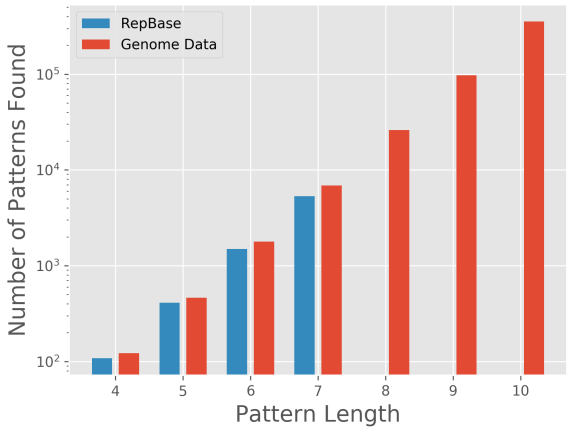

Figure 4: Number of frequent sequential found in the genome data and the TE database, for every pattern length in the range of interests. The results reflect a comparable amount of patterns were found in both datasets, and this number grew exponentially with the pattern length.

originated from that the *A-priori* needs the number of passes as many as the size of the pattern length range, and since the range expands when the total number of base pairs increases, it is computationally more costly. The position-based method, on the other hand, only requires a few passes to construct the initial instance array, and it is able to operate based on stored position information.

4.2 Ability of Mining TEs

I next utilized the developed implementation of frequent sequential pattern detection for a case study on how well one can find transposable elements from mining genome data. I applied the hybrid method described in section 3 to both the *Arabdopsis thiliana* genome data and the TEs extracted from RepBase. Comparison of revealed frequent sequential patterns from the two datasets is informative. If frequent pattern detection possesses high accuracy of discovering TEs, one would expect little divergence between the two sets of outcome patterns. Therefore the experiment is able to assess the performance of frequent sequential pattern detection on TEs' discovery.

After the frequent sequential patterns were obtained from the two datasets, I further categorized the frequent patterns by their length. In this way the influence of the proposed support threshold selection can be examined. Figure 4 shows the number of frequent sequential patterns found (within the range of pattern lengths of interests) from the genome data and the TE database respectively. Note that the range of pattern lengths differ between the two datasets because their total number of base pairs differ. Figure 4 entails two observations: First and surprisingly, the number of frequent patterns found displays a exponentially gain with its length. Although both the selected support thresholds decrease and the amount of possible sequential patterns grows exponentially with the pattern length, there was little anticipation that what were mined from a real-world dataset strongly associate with this according trend. Second, the number of patterns found in the genome data is comparable to the amount discovered in the TE database, for every overlapping length category. This seemingly signals a positive sign that frequent pattern detection has practical usage in finding transposable elements.

Nevertheless, frequent sequential pattern detection did not lead to desired similar outcomes in the genome data and the TE database. Figure 5 demonstrates the Jaccard similarity between sets of frequent patterns revealed in the two datasets. We see that for a longer pattern length, the sets of frequent patterns become less similar. I additionally looked into the fraction of relative complement in one set to the other, which is formally defined as $|A - B|/|A|$ for sets A and B. Figure 6 provides a peek of this metric over the common length range of interests. Observe that for both frequent patterns detected in the genome data and the TE database, the longer the patterns are the larger fraction of relative complement it possesses. To summarize and combine the results from Figure 4 to 6, frequent pattern detection along with support threshold selection uncovered more sequential patterns for longer length. However, those long patterns found in the genome data deviate from what were found in the TE database. The observed divergence did not originate from an one-sided effect: For both datasets, the proportion of frequent patterns found that were not revealed in the other increases with pattern length. The proposed frequent sequential pattern detection approach performs better on finding transposable elements for short-length patterns.

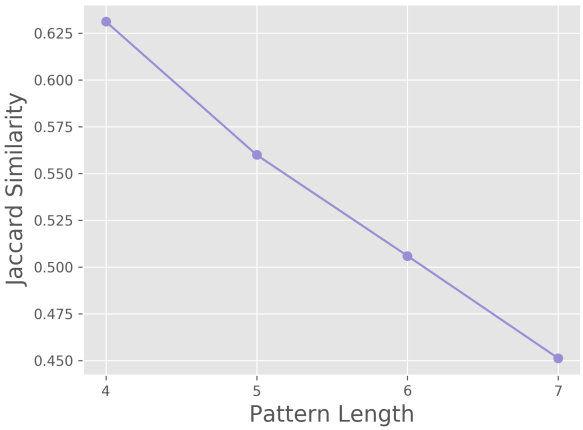

Figure 5: Jaccard similarity between sets of frequent patterns found in the genome data and the TE database for pattern length in the common range of interests. Longer patterns revealed were less similar in the two datasets.

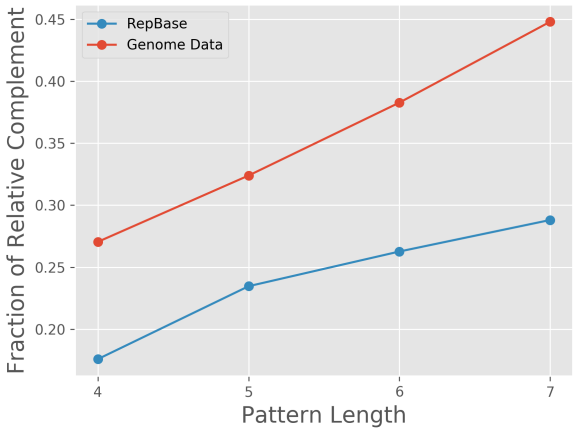

Figure 6: Fraction of relative complement in one frequent pattern set to the other, which were found in the genome data and the TE database respectively, and for pattern length in the common range of interests.

5 CONCLUSIONS AND FUTURE WORKS

In this work I have proposed a support threshold selection method to complement existing position-based approaches of frequent sequential pattern detection. I applied the developed algorithm to both genome data of *Arabdopsis thiliana* and its transposable element database. The results of experiments indicated that mining frequent subsequences in genomes can uncover short patterns in the transposable elements. The poor performance to reveal long sequential patterns has a few plausible sources. The proposed null-model threshold selection is indeed a minimal machinery, and it can definitely be improved by a refined null model, for example where every nucleotide in the sequences are randomly sampled. On the other hand, the poor performance could reflect that mining frequent patterns in genomes are intrinsically limited and insufficient to discover transposable elements. If TEs are not the dominated source to produce repeated segments of DNA sequences, the outcomes of the algorithm might be a mere dilution of TE patterns with others originated from an neglected mechanism. This project has also opened a question to myself about the leverage of different classes of methods in bioinformatics. Specifically in sequence motif mining, frequent pattern detection strategizes a hard-counting approach, which departs from popular probabilistic methods such as MEME+ [1]. It remains unclear that how much information we can gain from such a hard-counting approach and its potential and worthiness desires further exploration.

REFERENCES

[1] Timothy L Bailey, Charles Elkan, et al. 1994. Fitting a mixture model by expectation maximization to discover motifs in bipolymers. (1994).

[2] Weidong Bao, Kenji K Kojima, and Oleksiy Kohany. 2015. Repbase Update, a database of repetitive elements in eukaryotic genomes. *Mobile DNA* 6, 1 (2015), 11.

[3] Emilie Debladis, Christel Llauro, Marie-Christine Carpentier, Marie Mirouze, and Olivier Panaud. 2017. Detection of active transposable elements in Arabidopsis thaliana using Oxford Nanopore Sequencing technology. *BMC genomics* 18, 1 (2017), 537.

[4] Syeda Farzana Zerin and Byeong-Soo Jeong. 2011. A fast contiguous sequential pattern mining technique in DNA data sequences using position information. *IETE Technical Review* 28, 6 (2011), 511–519.

[5] Tae Ho Kang, Jae Soo Yoo, and Hak Yong Kim. 2007. Mining frequent contiguous sequence patterns in biological sequences. In *Bioinformatics and Bioengineering, 2007. BIBE 2007. Proceedings of the 7th IEEE International Conference on.* IEEE, 723–728.

[6] Moin Mahmud Tanvee, Shaikh Jeeshan Kabeer, Tareque Mohmud Chowdhury, Asif Ahmed Sarja, and Md Tayeb Hasan Shuvo. 2013. Mining Maximal Adjacent Frequent Patterns from DNA Sequences using Location Information. *International Journal of Computer Applications* 76, 15 (2013).