Discovering Motifs in DNA Sequences: A Suffix Tree Based Approach

Sanchi Prakash*
prakash.sanchi1998@gmail.com

Harshit Agarwal*
harshitgupta2696@gmail.com

Urvi Agarwal*
agarwalurvi22@gmail.com

Prantik Biswas*
prantik.biswas@jiit.ac.in

Suma Dawn*
suma.dawn@jiit.ac.in

* Dept. of Computer Science & Information Technology ,Jaypee Institute of Information, Technology Noida, India

***Abstract*—Motif discovery also known as motif finding is a challenging problem in the field of bioinformatics that deals with various computational and statistical techniques to identify short patterns, often referred to as motifs that corresponds to the binding sites in the DNA sequence for transcription factors. Owing to the recent growth of bioinformatics, a good number of algorithms have come into limelight. This paper proposes a competent algorithm that extracts binding sites in set of DNA sequences for transcription factors, using successive iterations on the sequences provided. The motif we work on are of unknown length, un-gapped and non-mutated. The algorithm uses suffix trie for finding such sites. In this approach the first sequence is used as base for constructing the suffix trie and is mapped with other sequences which results in extraction of the motif. Additionally, this algorithm can also be applied to related problems in the field of data mining, pattern detection, etc.**

Keywords—Motif finding problem, Suffix tree, DNA sequences, Trie, Transcription factors.

I. Introduction

In bioinformatics, a motif is an *l-mer* (a string of length l) that is widespread and is supposed to have a salient significance. Regulatory motifs are short DNA sequences that turn on immunity and are bounded by proteins known as transcription factors [1]. Subsequences that are required generally correspond to functionally or structurally vital elements in DNA or proteins sequences [2, 3]. Detecting these motifs or binding sites in huge DNA sequences without any prior knowledge of its length and position of occurrence is referred to as the non-mutated motif finding problem. Algorithms that can efficiently predict these motifs from DNA sequences would be of great help to the study mechanism regulating various gene expressions.

Deducing motifs in a fast and efficient way has proven to be very obliging in various critical problems in the domain of molecular and computational biology. Over the past few decades scientists have thoroughly studied various variants of the motif finding problem. Recent researches [11] clearly points to its application in the detection of transcription factor binding sites (TFBS) and antibody biomarkers [4]. Transcription factor binding sites helps us in learning the various complex mechanisms responsible for gene expressions [11] while different diseases can be efficiently diagnosed with the aid of antibody biomarkers [4]. An efficient algorithm which is capable of finding motifs in exceedingly long DNA sequences in minimal field of computer science.

In this paper we have developed an algorithm using suffix tree data structure. Given a set of DNA sequences, we intend to predict the non-mutated motif of unknown length that occurs perfectly in all the sequences. In this approach, the suffix trie[1] constitutes all the possible suffixes of the first sequence from the set of given sequences and by traversing the remaining sequences on this tree we find the maximum length *l-mer* that is present in all the DNA sequences. . By traversing the remaining sequences on this suffix tree we find the maximum length *l-mer* that is present in all the DNA sequences. This resultant *l-mer* is our required motif for the set of entered DNA sequences

II. Problem Definition

Given a set of 'm' DNA sequences, each of length 'n', the goal of non-mutated motif finding problem is to detect a hidden pattern of unknown length that occurs exactly once in each of the 'm' DNA sequences. This motif is the maximum length hidden pattern that necessarily occurs in every sequence of the set. Searching for the motif among lengthy DNA sequences becomes a tedious task as we have no prior knowledge about its length and site of occurrence. Each DNA sequence is often referred to as DNA strands and the motif can occur anywhere within a given sequence. Figure 1 depicts a set of four DNA sequences each of length 9. Figure 2 reflects the hidden motif occurring in each strand.

```
ACTGAATCG
TACTACTGG
TTACTGACG
GAAACTGCA
```

Fig. 1. Set of DNA sequences

There are numerous variants to the problem of motif finding; gapped motif finding, mutable motif finding, etc to name a few. People often cite analogy of the frequent word search problem with the non-mutated motif finding problem. Our algorithm finds the motif of unknown length implanted in each sequence in the given set of DNA sequences. The use of suffix tree along with trie nodes help greatly in minimizing the preprocessing time for the proposed algorithm.

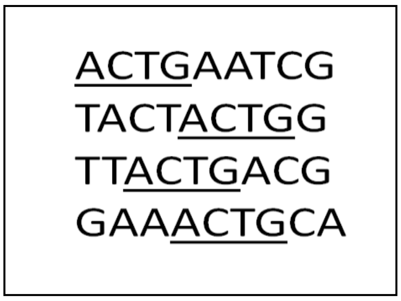


Fig. 2. Hidden motif ACTG in each DNA sequence.

III. Literature Survey

The recurrent goal among all the approaches developed till date for the motif finding problem has been to reduce the time complexity of the respective algorithm. Because of the intriguing nature of the problem, with time, there have been significant improvements in the already existing algorithms as well as the newer ones being proposed. Here, we have provided a brief survey of some of the notable algorithms for the motif finding problem.

The naïve approach [5] for solving the non-mutated motif finding problem incorporates a brute force search algorithm that scans through all possible l-mers in the first strand and then looks for its occurrence in all the trailing strands. The l-mer repeated in each of the strands serves as the motif for the given sequence. Considering the length of each strand to be m, and given there are n DNA strands, there are (n-l+1) l-mers possible per strand and the algorithm fails to deliver the result quickly for strands of higher lengths. The exhaustive search technique tends to be exponential.

Recent surge in the area of bioinformatics have witnessed a good number of algorithms coming into existence. Researchers have taken good interest in the problem and progress made in the field is very motivating. Most of these algorithms have deployed various advanced data structures and new programming methodology that intensify their speed of execution. The motif finding problem can be generally solved using approaches that are widely classified into two basic categories; word-based and probabilistic approach. The algorithms which are word-based can be dealt using suffix tree data structure. Position weight matrix also plays a crucial role in solving word-based problems like motif finding and often each position is represented as a stack of letters [6]. Probabilistic approaches are beneficial when there are small changes in the input data set [7]. Some algorithms modify existing time consuming algorithms to give the results faster [8].

Modern computational paradigms like machine learning have also been applied to solve the motif finding problem [9]. Ho-Kwok Dai and Modan K Das [10] have given a good survey of popular algorithms that are extensively used to solve the motif finding problem.

IV. Proposed Algorithm

This section narrates our proposed algorithm for solving the non-mutated motif finding problem. It take the set of DNA sequences as the input and outputs the motif that is repeated in all of the DNA strands provided in the input set. The use of suffix tree blended with trie data structure considerably reduces the preprocessing time of the algorithm. The suffix tree is initially constructed with all the suffixes of the first DNA strand and serves as the base for the algorithm. As the algorithm progress, the number of branches visited per DNA strand decreases, thus contributing to the reduced time complexity of the algorithm. Further details about the working process of the proposed algorithm are discussed in section VI.

```
check_motif_unknown_length (motif, temp)
   while(true)
      if (the next character of the
may_be_motif  matches the
         first letter of the new branch && visited=index-
1)
            if ( match further in the same node)
               match++;
            if  (no match further and not null)
               return false;
            if  (no match further and null)
            if (next->weight>match && visited=
            index-1)
                 weight=match
               if check_motif_unknown_length
               (motif, next)
               visited=index;
               return true;
               else
               return false;
if   (match further  &&  length=1 &&   visited=index-1)
if (next->weight>match)
                 weight=match
visited=index
else
            return false;

find_motif_unknown_length()
   for i(n-max, n-1)
      for j(0, i+1)
            may_be_motif=seq.substring(j ,n-1)
check_motif_unknown_length (motif, root)
```

V. Results and Discussions

The proposed algorithm, as mentioned before works on suffix tree and trie data structures. Using basic trie structure, all the possible suffixes of the given DNA sequence are obtained. The alphabet set of a DNA sequence consists of just 4 characters A, C, G and T. Each node consists of four parameters namely:- Value: it represents the string stored in the node, Visited: the index of the latest sequence that matched with the node value, Weight: the maximum number of characters of the node that matched the sequence and the Next [ ]: a pointer array that points to the next node in the suffix tree.

The construction of the tree begins by extracting all the suffixes from the first DNA sequence. For each suffix the algorithm checks whether it already exists in the tree or not. The whole suffix does not exist beforehand in the tree and hence a diversion is induced as soon as there is a mismatch between the sequence and the existing values of the tree.

The working of the algorithm is further elaborated with the three DNA sequences; ACTGTGCGCC, ACTGTGTAGA and TCTACTGATA. We append a $ at the end of each sequences to represent an end marker. By passing all the suffixes of the first sequence (ACTGTGCGCC$), we create the tree as shown below in figure 3.

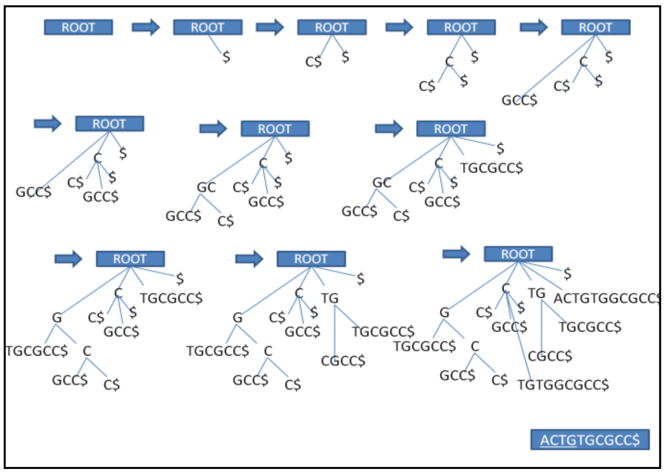


Fig. 3. Insertion of ACTGTGCGCC.

Now, when the second sequence of length n is encountered, the algorithm scans all the possible l-mers starting with length n-1 to 2, from each side of the string i.e. both the start and end of the respective string The function check_motif_unknown_length() checks the maximum match of that l-mer in the tree and updates the weight with the count of matched characters in the node only if the complete l-mer gets matched in the tree. The remaining l-mers are sent to match in a recursive manner and breaks as soon as no match is found ahead. The algorithm also updates the variable *visited* with the index of the current sequence. The l-mers of next sequence (with index x) are discarded if the index associated with node is less than x-1.

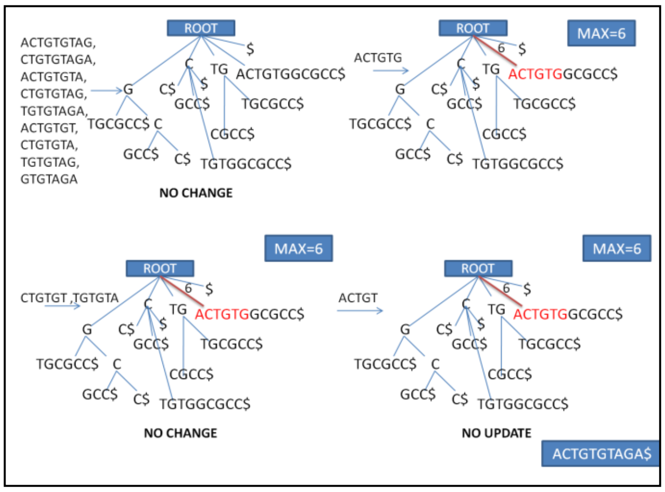

ACTGTGTAG, CTGTGTAGA, ACTGTGTA, CTGTGTAG, TGTGTAGA, ACTGTGT, CTGTGTA, TGTGTAG, GTGTAGA
NO CHANGE
ACTGTG
MAX=6
CTGTGT ,TGTGTA
NO CHANGE
ACTGT
NO UPDATE
ACTGTGTAGA$

Fig. 4.1. Insertion of ACTGTGTAGA.

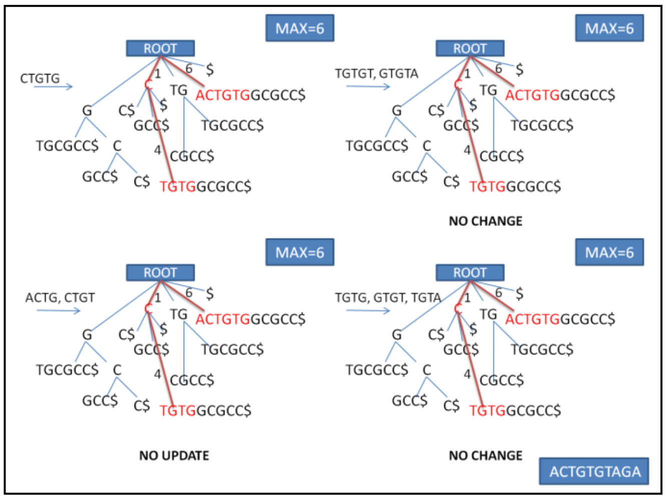


Fig. 4.2. Insertion of ACTGTGTAGA.

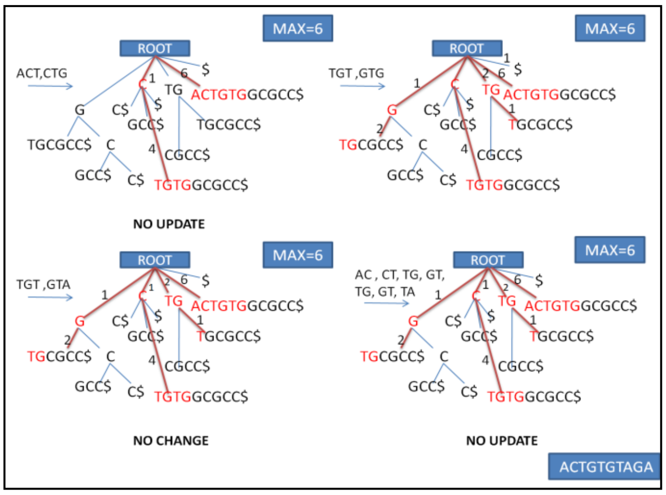
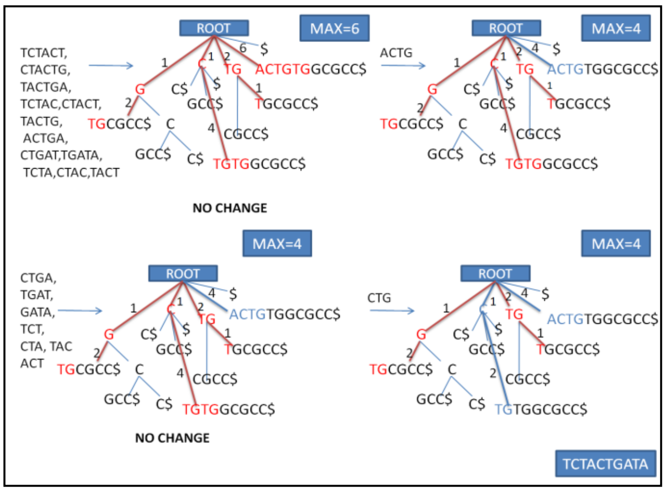


Fig. 4.3. Insertion of ACTGTGTAGA.

Figures 4.1, 4.2 and 4.3, depict the insertion of the second sequence ACTGTGTGTAGA into the tree. The nodes visited by the second sequence are depicted by red .The l-mers which did not match the tree, do not change the visited and weight variables which reduces the available paths for the next sequence. The variable *max1* keeps a track of the size of the l-mer of maximum length that matched in the tree. l-mers with length greater than *max1* need not enter because motif length cannot be greater than the l-mer length matched in already traversed sequences.



ROOT
MAX=6
TCTACT, CTACTG, TACTGA, TCTAC,CTACT, TACTG, ACTGA, CTGAT,TGATA, TCTA,CTAC,TACT
ACTGTGGCGCC$
TGCGCC$
TGCGCC$
CGCC$
GCC$
C$
TGTGGCGCC$
NO CHANGE
MAX=4
ACTG
ACTGTGGCGCC$
CTGA, TGAT, GATA, TCT, CTA, TAC, ACT
CTG
TGTGGCGCC$
TCTACTGATA

Fig. 5.1. Insertion of TCTACTGATA.

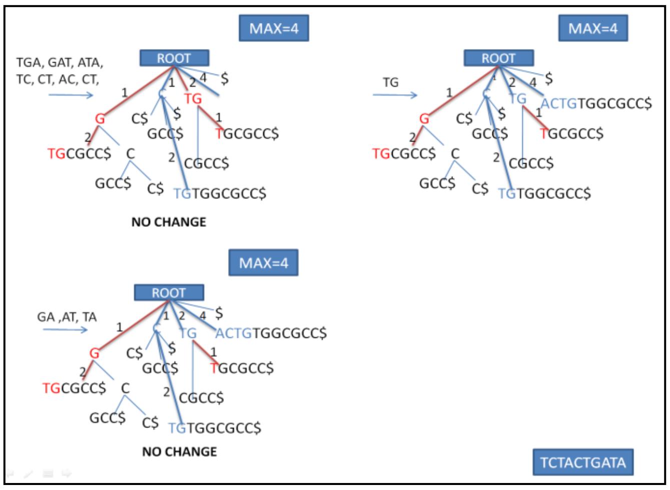


Fig. 5.2. Insertion of TCTACTGATA.

During the insertion of the third sequence: TCTACTGATA, the l-mers with length greater than *max1* are not compared. For those with length less than *max1*, the visited and weight variables are updated when match occurs as shown in Fig 5.1 & 5.2 respectively. The nodes visited by the third sequence are depicted in blue. Once all the sequences get traversed; the resultant tree contains all the paths that were traversed by all the sequences entered. The algorithm searches for the path having the maximum sum of weights and corresponding l-mer in that path. The l-mer of the resulting sequence (0 to max1) is our motif (i.e. ACTG in the given example).

We tested the algorithm for simulated data providing the DNA sequences as input. The PC had 8 GB DDR4 RAM running on Intel 6th generation core processor i5™ with 1 TB HDD. Table I and II summarize the results for searching the unknown motif for different test cases. The sequence length corresponds to the length of an individual DNA strand in the input set while the second column (No. of seq.) denotes the number of strands in each test case. The build time and the search time correspond to the preprocessing time, i.e. time taken to build the suffix tree and search time respectively. The motif length represents the length of the maximum length l-mer discovered by our algorithm. This maximum length l-mer represents the hidden non-mutated motif implanted in all the DNA sequences.

TABLE I. RESULTS WITH 10 SEQUENCES

Seq. Length	No. of seq.	Motif Length	Build Time(s)	Search Time(s)
20	10	5	0	0
30	10	8	0	0
40	10	10	0	0
50	10	12	0	0
60	10	15	0	0.00532
80	10	18	0	0.00975
100	10	20	0	0.01521
150	10	25	0	0.04560
200	10	50	0	0.08420
300	10	60	0	0.10458
400	10	80	0	0.27654
500	10	100	0	0.56247
600	10	150	0	1.64537
700	10	200	0	2.31626
800	10	250	0	3.76539
900	10	300	0	5.29991
1000	10	350	0.003509	10.24764

In Table I, a constant number of sequences (10) are used to analyze the change in time with respect to the Sequence length and the Motif length. We infer that the build time we need to create the suffix tree is optimum even if the sequence length is huge number.

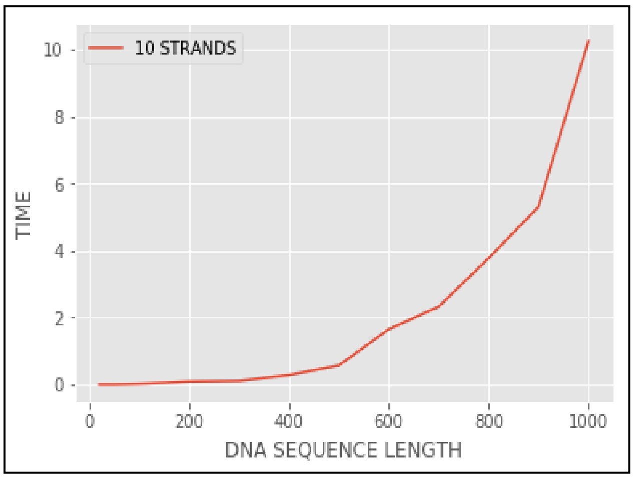


Fig. 6.1. Graph with 10 sequences

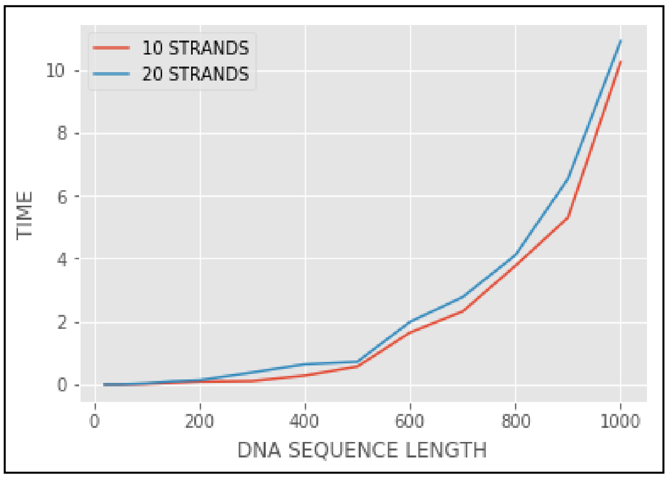


Fig. 6.2. Graph with 10 and 20 sequences

Fig 6.1 depicts the graph where X-axis→ time and Y-axis →Running time of the proposed algorithm with 10 sequences. The running time increases with the length of the sequences. Also the results the show that the build time which the algorithm takes is negligible even if the sequence length is a big number.

TABLE II. Results with 20 sequences

Seq. Length	No. of seq.	Motif Length	Build Time(s)	Search Time(s)
20	20	5	0	0
30	20	8	0	0
40	20	10	0	0
50	20	12	0	0
80	20	18	0	0.01935
100	20	20	0	0.03074
150	20	25	0	0.09765
200	20	50	0	0.13167
300	20	60	0	0.37465
400	20	80	0	0.63373
500	20	100	0	0.71780
600	20	150	0	1.98450
700	20	200	0	2.77468
800	20	250	0	4.10293
900	20	300	0	6.53169
1000	20	350	0	10.91208

After plotting the graph in Fig 6.2, we conclude that the running time does not increase much even after we increase the number of sequences twofold. For instance, if we take the former example, most of the branches are closed after the incoming of sequences.

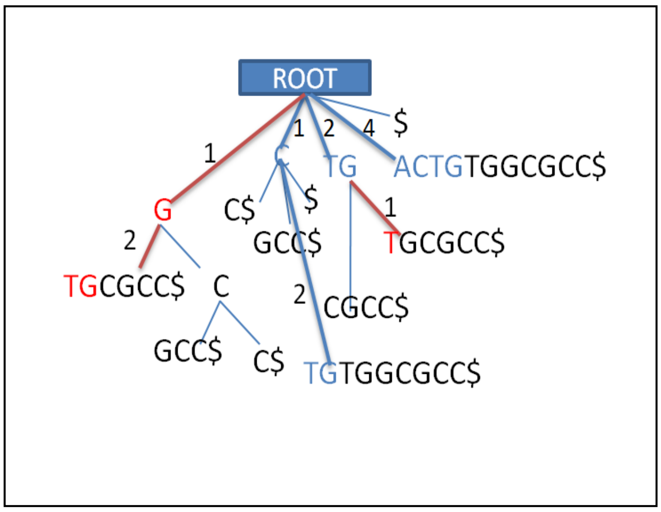


Fig. 7.1. Final tree after all traversals

Fig 7.1 shows the picture of the tree after traversal of all the sequences in the example. After traversal of the third sequence, only the branches colored blue are possible paths for the next incoming sequence. Now, if we consider increasing the number of DNA sequences, the misleading branches remain closed. The further sequences will only be updating the visited variable. Therefore, the running time of the algorithm reduces as the sequences have to traverse very limited branches of the tree. This proves to be beneficial for a large number of sequences.

VI. Conclusion

The algorithm proposed in this paper works perfectly for large length DNA sequences even on standard computing systems. The minimized build time supplemented with the reduction in the paths visited after each iteration makes the algorithm faster. The scope of this algorithm can be easily extended to various related problems in the field of data mining, pattern recognition, etc. Further streamlining of the execution time may be achieved with help of better computing paradigms and it still remains as an open scope for our future work. Despite being a well documented problem, only limited number algorithms exist in this domain and our algorithm fits aptly in the recent developments on this problem.

References

[1] S. Prado, “Motif Discovery using optimized Suffix Tries,” Master thesis, Faculty of Engineering and Architecture, Department of Information Technology – IBCN, Ghent University, Ghent, Belgium, 2012. Accessed on: Apr. 18, 2018. [Online]. https://upcommons.upc.edu/bitstream/handle/2099.1/16165/78135.pdf?sequence=1&isAllowed=y

[2] C. Angkawidjaja, A. Paul, Y. Koga, K. Takano, S. Kanaya,” Importance of a repetitive nine-residue sequence motif for intracellular stability and functional structure of a family I.3 lipase,” FEBS Letters, vol. 579, issue. 21, pp. 4707-4712, August 2005.

[3] J. S. Fink, M. Verhave, S. Kasper, T. Tsukada, G. Mandel, R. H. Goodman,”The CGTCA sequence motif is essential for biological activity of the vasoactive intestinal peptidegenecAMP-regulated enhancer,” *Proceedings of the National Academy of Sciences of the United States of America*. 1988;85(18):6662-6666.

[4] J. T. Ballew, J. A. Murray, P. Collin, M. M¨aki, M. F. Kagnoff, K. Kaukinen, and P. S. Daugherty, “Antibody biomarker discovery through in vitro directed evolution of consensus recognition epitopes,” Proceedings of the National Academy of Sciences, vol. 110, no. 48, pp. 19330-19335, 2013.

[5] P. Pevzner, N. C. Jones, An Introduction to Bioinformatics Algorithms, The MIT Press Cambridge, Massachusetts London, England, 2004.

[6] Y. Zhang, P. Wang, “A Fast Cluster Motif Finding Algorithm for ChIP-Seq Data Sets,” BioMed Research International, vol. 2015, Article ID 218068, 10 pages, 2015.

[7] A. Majumdar, “Finding DNA Motifs: A Probabilistic Suffix Tree Approach,” Ph.D dissertation, Computer Science and Engineering, Graduate College, University of Nebraska, 2016. Accessed on: Apr. 18,2018.[Online].http://digitalcommons.unl.edu/cgi/viewcontent.cgi?article=1143&context=computerscidiss.

[8] H. Liu, F. Han, H. Zhou, X. Yan and K. S. Kosik, "Fast motif discovery in short sequences," *2016 IEEE 32nd International Conference on Data Engineering (ICDE)*, Helsinki, 2016, pp. 1158-1169. doi: 10.1109/ICDE.2016.7498321.

[9] H. R. Hassanzadeh, P. Kolhe, C. L. Isbell and M. D. Wang, "MotifMark: Finding regulatory motifs in DNA sequences," *2017 39th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, Seogwipo, 2017, pp. 3890-3893. doi: 10.1109/EMBC.2017.8037706.

[10] H. K. Dai, M. K. Das,” A survey of DNA motif finding algorithms,” *Proceedings of the Fourth Annual MCBIOS Conference. Computational Frontiers in Biomedicine*, 2007.

[11] F. Zambelli, G. Pesole, and G. Pavesi, “Motif discovery and transcription factor binding sites before and after the next-generation sequencing era,” Briefings in bioinformatics, vol. 14, no. 2, pp. 225–237, 2013.