http://ansinet.com/itj

ITJ

ISSN 1812-5638

INFORMATION TECHNOLOGY JOURNAL

ANSInet
Asian Network for Scientific Information
308 Lasani Town, Sargodha Road, Faisalabad - Pakistan

Information Technology Journal 6 (2): 189-195, 2007
ISSN 1812-5638

ANT: A Novel Heuristic Algorithm for Finding Motif

Saifuddin Md. Tareeq, Sumanta Saha, Tohidul Islam and Rumana Quazi
Department of Computer Science and Engineering, University of Dhaka, Dhaka-1000, Bangladesh

Abstract: Motif detection in unaligned DNA sequences is a fundamental problem in molecular biology. In particular, algorithms must be developed for annotating biologically meaningful features in multimegabase DNA sequences either by observing their similarity to known genes, regulatory sites and other features or to conserved copies of the same features in an equally long sequence from another organism. Annotation on such a large scale must be both computationally efficient and sensitive enough to recover subtle but significant features that would otherwise be lost in a mass of unannotated and hence undifferentiated sequence. This study presents an algorithm, named ANT, for biosequence annotation that uses a heuristic function which allows the algorithm to run faster and also allows analysis of very long biological sequences. The result achieved by ANT are better than most of the present motif finding approaches when tested on simulated sample sets with various parameters. The comparison shows the proposed algorithm performs better in finding motif even in twilight zone.

Key words: Motif, gene, DNA sequence

INTRODUCTION

A motif is a conserved DNA sequence pattern recognized by a transcription factor or by other cellular machinery. In computer science terminology the problem of finding motif is as follows: given a set of strings, find a set of non-overlapping, approximately matching sub-strings. This problem is farther complicated by the natural phenomenon of mutations that occur in any DNA sequence. That is, the motif that we have to find is not only unknown and planted in unknown positions in the sequences but also they can differ in many places from the actual pattern that we have to find. Sometimes this mutation makes the original pattern very subtle in the planted positions. For this kind of mutated motifs a standard notation is used: an (l, d) motif (or pattern) is a pattern of length that is implanted in sequences from the sample with at most d mutations. The implants are called instances.

The challenge problem was first introduced by Pevzner and Sze (2000). The problem is to find a motif of length 15 with at most 4 mutations from a set of sequences where it is planted in each of them. This problem may look rather naïve and the reason why it is algorithmically so complex may not be visible at the first look. But the fact is that the consecutive instances can differ with each other in at most 2d places, i.e., in the (15, 4) problem the instances can be mutated form each other in at most 8 positions, which is rather high. This obscures the original pattern and makes the signal finding problem so much difficult to handle. Because of this, many algorithms fail and tend to converge to a local optima rather than the original pattern. Figure 1 shows the problem with an example.

Over the years, many algorithms have been developed to address the (l, d) problem. Most of them are based on probability and also on greedy methods. The most naive approach to find such motif is to enumerate all the substrings that can be generated with 4 DNA letters of length l. And then check if the pattern appears on all the sequences with at most d mutations. But this approach is exponential in time and takes much more time than it is pragmatic. Enumerative algorithms include methods by Blanchette *et al.* (2000), Brazma *et al.* (1998); Galas *et al.* (1985), Neuwald *et al.* (1994) and Tompa (1999). On the contrary, statistical and greedy methods tend to produce the same result more often than not with considerably low amount of time. While using these

```
seq 1 ..acttgcAGTTACGAGATGTAGacgtacgtcagtcgtacg..
seq 2 ..gtgcatgcagtcaagtcagacAACTAACACTTAGAGgct..
      ..........................................
seq n ..tgATCTCAGAGATATACatgcgtacgtcagtcacgtgcg..

  seq 1 motif instance 1        AGTTACGAGATGTAG
  seq 2 motif instance 2        AACTAACACTTAGAG
  ..... ................         . . . . . . . .
  seq n motif instance n        ATCTCAGAGATATAC

  final  motif in question      AGCTAAGACATATAG
```

Fig. 1: A planted (15, 4))motif and its solution. The motif in each sequence is shown in uppercase

Corresponding Author: Saifuddin Md. Tareeq, Department of Computer Science and Engineering, University of Dhaka, Dhaka-1000, Bangladesh Tel: 8802 9670734/319

methods it is possible that the correct result may not be produced or all the instances may not be found but the tradeoff is towards the very low time limit compared to the brute force process. Motif finders implementing one of these techniques include methods by Bailey and Elkan (1995), Herts and Stormo (1999), Lawrence *et al.* (1993), Lawrence and Reilly (1990), Rocke and Tompa (1998) and Buhler and Tompa (2002).

These approaches use rather complex statistical method and try to guess the motif implanted in the sequences. But in the twilight zone as the sequence length grows or the number of mutations gets larger the success rate of these algorithms tend to deteriorate. We present our algorithm that is rather simple in construct but have the potential of acting quickly as the greedy statistical methods while having the power of finding all the embedded motifs in the sequences. This approach does not use any complex statistical construct to find the motif rather takes a heuristic approach that works on the assumption that if a (15, 4) pattern resides in all the sequences then it is possible to predict about its presence from the very beginning. Each time a new level or a new sequence is added to the total consensus, the probability of getting a final motif is revised according to a predefined plan.

The algorithms presented by Pevzner and Sze (2000) named SP-STAR and WINNOER have shown great promise to solve the planted (15, 4) problem introduced by the same. Our algorithm solves the same problem efficiently with a different approach. It eliminates the tracks of instances that cannot probably go on to make a complete (l, d) motif as fast as possible. Moreover, as the algorithm does not work on the probability or expectancy of motifs occurring on some positions but rather on all the possible positions, so it finds all the instances of the motif that is hidden in a sequence. For this, the algorithm presented here produce a better result than the contemporary motif finding algorithms.

The study describe the algorithm and evaluate its performance as defined by Pevzner and Sze (2000), (n = 600 and t = 20). Although the (15, 4) problem was the main target of the algorithm, it has been tested in a wide range of parameters. Its performance is then compared with some of the existing algorithms. At last, some scopes of possible future improvements have been listed.

ANT: The Algorithm: A new algorithm, named ANT, for finding motifs from biological sequence is proposed. This algorithm performs a rigorous search over the search space like an Ant seeking its food and uses an efficient heuristic to speed up the search by throwing out the spurious paths in advance. The implementation detail and the construction of the heuristic function will be discussed here.

This algorithm basically acts on two stages. The first one is the initialization where all the possible patterns are enumerated from a few first sequences. And the second one is a heuristic iteration stage. In this stage the algorithm selects the best possible instances that have the highest probability of reaching the final result. It iterates over the sequences one after another and total number of iteration is equal to the number of sequences. In each iteration, every possible pattern is checked and the majority pattern is updated. While iterating, the algorithm maintains some specialized data structure to keep the information needed and also defines some customized operations on the data structure. At the last step it produces the final result.

Construction of the heuristic function: As we know, in an (l, d) problem each instance of the motif planted in a sequence has at most d number of mutations from the original motif. The heuristic function works on this property of the problem. This would best be described using an example. Let's say, we have a total of 20 sequences and have discovered 5 instances of a likely-to-be-motif after scanning 5 sequences. Now if the likely-to-be-motif is to be one of the real motifs planted then the instances can not be more than d mutations away from the likely-to-be-motif. So, it must be possible to compute an l length motif from the 5 instances because if they are to be a part of the 20 sequences then they must have been mutated only at most d positions from the original motif. So it is possible to derive the original motif from them. On the other hand, if we can not get a motif for those 5 instances then it would be impossible for them to be a part of the complete array of instances. So what the heuristic function does is this, whenever it finds that from a number of discovered instances we can not derive a motif from which all the instances mutate only at most d positions then we completely throw away that line of search and start another search. This tremendously lowers the amount of time needed without hampering the credibility or the rigidity of the search.

Definition 1: A window with respect to our algorithm is a series of length l patterns that has the possibility of being a part of a full series of instances that has at most d mutations from a motif.

Definition 2: The algorithm is like an Expanding Window because the size of the window is gradually increased

until the window is either invalidated or successfully completed. The basic idea is, each window starts with at least three patterns and patterns are gradually added to the window and after each addition the validity of the window is checked. If found invalid, the window is destroyed and another window is checked and expanded from the first. On the other hand, if the window is expanded to the final length, i.e., the total number of sequences, then the motif is reported and the total mutation is also reported. The whole scenario may seem a little hazy but it should be clear in a while when we present some examples.

Data structures: Define the distance $d(s_i, s_j)$ be the difference between two patterns s_i and s_j. As we know from the definition of the problem, the maximum distance between two planted instances of the same motif can be at most 2d. So it is practical to consider only those patterns in the sequences that have at most 2d difference among themselves. In our algorithm, for each pattern s_i in the first sequence where $1 = i \leq n-l+1$, we make a tree where nodes are the patterns s_j of the remaining sequences having $d(s_i, s_j) = 2d$, where $1 < j \leq t$. The tree has a depth of t and has s_i as its root. Each distinct level of the tree consists of the patterns found as above from corresponding sequence. For space efficiency the tree is kept as a linked list without repetition and the list is traversed in a clever way which makes it act like a complete tree. We have introduced a new data structure named probable pattern that keeps all the information about the intermediate patterns generated in between the sequence traversal. This probable pattern keeps record of the trail of patterns that has been traversed to create it. A majority pattern is generated from the trail and is used as the main generator of the probable pattern. Using this information the heuristic cycle filters and selects the best patterns-so-far to use in next step. We would also like to introduce a new operator, which takes a previous probable pattern as argument to use the constituent patterns of it and also a pattern of current level to create a new probable pattern, i.e., if we have four probable patterns and a pattern from current level we will get four probable patterns with track list modified by current level.

Construction of the tree: We kept the tree as a linked list where redundant nodes at the same level are stored only once. The list is traversed in such a clever way that dynamically transforms all possible combination of the element of the list into distinct tree. For each s_i of length l of the first sequence, a separate list is created which contains a list of substring s_j of the first pattern of each level that has $d(s_i, s_j) = 2d$.

The head node of the list of each level has a link to the next level list. Thus each tree is of depth t and each row of the list is of variable length depending on the number of substrings in a sequence having difference at most 2d from the root.

Assumption: The assumption behind the algorithm is simple. From several l length patterns a majority pattern is created using the majority function. Majority function works by selecting the residue that occurs most frequently in a particular position among the patterns. If a trail of l length pattern produces a majority pattern then we call that majority pattern valid only if with every l length pattern in the trail, the majority pattern has a distance less than or equal to d. The algorithm assumes that if a trail of l length patterns lead to a valid (l, d) pattern then, while traversing, for each previous level, we should get a valid (l, d) major pattern. Based on this idea the algorithm proceeds and each time when a new level is considered those patterns are spliced out which do not conform to the assumption, i.e., is not a valid (l, d) major pattern of its respective trail. In this way, the spurious probable patterns are discarded very quickly. This early detection of the spurious patterns leads to a quite good runtime of the algorithm.

Roadmap: The algorithm proceeds in two major steps. At the first stage it constructs a large array of all probable patterns which have the slightest possibility of getting to a successful result. Then in the second stage the array is refined over and over again to get to a real result. In the refining process any probable patterns may be discarded if it does not conform to the assumption and if so, that specific trail is not considered anymore. The refining process is a heuristic step that takes the best results of the previous level and after refining it using the current level passes the result to the next. The major two steps of the algorithm are described as follows.

Initialization: It is obvious that the majority function must be working with at least three patterns otherwise it may not have enough residue information for making majority choice. So at the initialization step we take the first three levels of the tree and traverse it in all combinations and compute a probable pattern using the majority function for each combination. So after the initialization step using first three levels, we have an array of probable patterns for refinement.

//**Algorithm 1**
// Initialization:
//Tree construction

read all the sequences from the input file
for i = 1 to totalSeq-support+1 do
make i'th sequence the pivot
for each l length string str in pivot
a. create a tree with str as the root
b. for each sequence having index greater than i do
1. create a new level in the tree.
2. insert into the level all valid substrings with respect to str in current sequence.
end for
end for
end for
//Initialization of probable patterns
for each tree constructed in the previous step do
a. traverse the first three levels in all possible way
b. for each path from the root to the third level do
1. compute the major pattern using majority consensus
2. create probablePattern object and store major pattern along with the traversing trail
end for
end for

Heuristic cycle: In heuristic cycle the probable patterns are refined again and again to get the original result. At each step, we add a new level to our partial result and recompute the result. To ensure all possible combinations, each pattern in the new level of the tree is combined with each probable pattern that has been generated upto this level. When combining a new pattern to a probable pattern, it is added to the trail of patterns and the major pattern is updated by recomputing it using majority function over the new pattern trail. After the computation, the new majority pattern is checked against the pattern trail to see if it is a valid (l, d) pattern. If not, then the probable pattern is ignored and next combination is checked. In this way, at each level the set of probable patterns are refined and updated so that only the feasible probable patterns stay. As can be seen from Fig. 2, for example, after third level there may be four probable patterns. Then the algorithm merges the fourth level with the probable patterns and refines it. So, the four probable patterns combining with three level-four patterns create a total of twelve new probable patterns. But after testing all the probable patterns for validity, say, only two can pass the test and stays, others are eliminated. In this way in each level the number of probable patterns is quickly reduced, leaving us with only a few probable patterns in the leaf level. If we get one or more probable patterns after the leaf level and if the patterns can pass the validity test then we keep that result as a valid (l, d) pattern. In this way, all the valid patterns are discovered and stored.

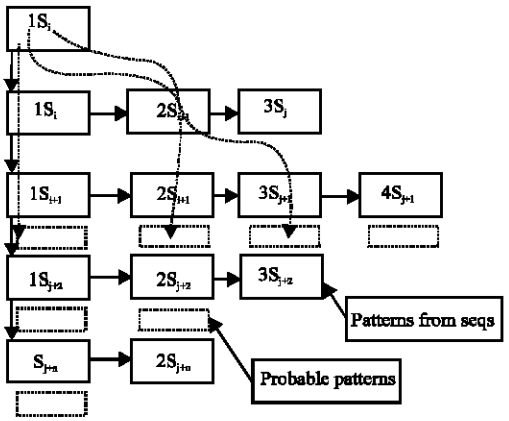

Fig. 2: A schematic diagram of an intermediate state of the algorithm

//**Algorithm 2:**
//Heuristic cycle:
for each tree created do
for each level from four to leaf
a. take as input the previous level's probable patterns
b. for each pattern at current level do
1. combine with each probable pattern of upper level
2. calculate all the majority patterns resulting
3. check for its validity
4. if current level is leaf level then store the results
else input them to next levels
end for
end for

To get the best one compute the total number of difference places between the majority pattern and the trail patterns.

Report the majority pattern with lowest difference count as the best motif.

After getting all the valid patterns in hand, we now check for the best valid pattern. This is done by the total distance count. Each valid pattern v is checked with its trail (t_0, t_1, $t_2 \ldots t_n$) of patterns and distance of each pattern with the major pattern $dist_i = d(v, t_i)$ is computed. Each major pattern is given the weight of total distance calculated, i.e., weight = $\sum dist_i$. Then the major pattern with the lowest weight, i.e., having the lowest distance count is declared as the best pattern and is reported. In case of a tie, all the patterns having the minimum distance count is reported.

Example: Now, it would be better for understanding if an example is given to show the making of probable pattern. Let's consider a simple example where the number of

sequences is 5, length of motif l is 5 and maximum number of mutation d is 1. The sequences shown in Table 1.

In each sequence the signal is implanted at the specified positions. The original signal is TGTTT where the mutation frequency is 1. That is the original motif is changed at least in one position before being implanted in the sequence. One noticeable point is that the sequence length, pattern length, mutation frequency and supports are taken very small for the sake of simplicity and ease of understanding. The algorithm would work similarly whatever those parameters are.

In our example, the support is full and so the offsets available in the first sequence to be operated on are 1st, 2nd and 3rd (as the sequence length is 7 and pattern length is 5). Let's concentrate on each substring separately.

Substring started from position 1: For this position the first sequence's substring is ACGTT. The third sequence contains no substring of length 5 which differs from this sequence by 2 mismatches and hence any further operation for this position is discarded.

Substring started from position 2: At this position the substring of length 5 is ACGTT in the first sequence. Then for each of the other sequences, each possible substring are checked for mismatches with ACGTT by at most 2 (2d = 2X1 = 2) positions. The substrings obtained from each of the sequences are then arranged in a tree as shown in Fig. 3.

After the construction of the tree is complete, the initialization step of the algorithm gives us only a single probable pattern after the 3rd level because there is only one combination of the patterns found (Fig. 3). The probable pattern is TGTTT. The majority information of this probable pattern is presented.

Now this probable pattern combined with the 4th levels substring TGTTT gives us new majority information (Table 3). From the Table 3 we obtain our new probable pattern to be TGTTT. Also this probable pattern satisfies its track that is all substrings within its track contain at most one (d = 1) mismatch. Similarly at level 5 we obtain new probable major TGTTT which also satisfies its track. As we reached the leaf level with a certifiable probable pattern we store it for future processing.

Substring started from position 3: Four probable patterns are obtained at the 3rd level. They are TTTTT, TTTTT, GTTTA and GTTAT. These are found by calculating majority from each combination of the patterns of first three levels. So at the 4th level we have to check a total of 8 (4*2) patterns. Among the eight patterns only TTTTT

Table 1: Example sequences to make probable pattern

Sequence	Motif implanted at
ACGTTTT	1
TTGTTAA	1
TGTTTAT	0
TGTTTAG	0
TGTTAGC	0

Table 2: Matrix for majority calculation after 3rd level

	Pos 1	Pos2	Pos3	Pos4	Pos5
A	0	0	0	0	1
C	1	0	0	0	0
G	0	3	0	0	0
T	2	0	3	3	2

Table 3: Majority matrix after 4th level

	Pos 1	Pos 2	Pos3	Pos4	Pos5
A	0	0	0	0	1
C	1	0	0	0	0
G	0	4	0	0	0
T	3	0	4	4	3

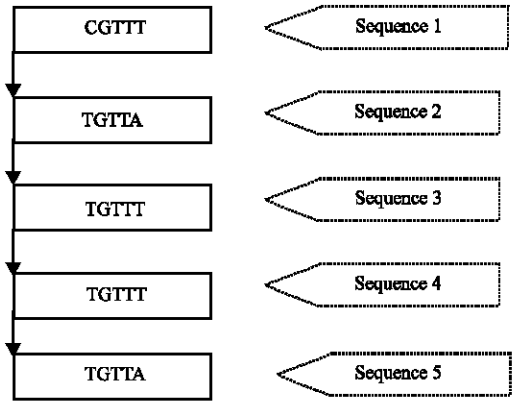

Fig. 3: A schematic diagram of the substring from each

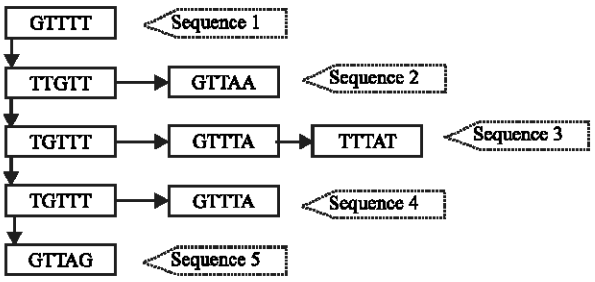

Fig. 4: A schematic diagram of the levels of recent tree

and GTTTA are satisfied by their track so we continue to the next level with these patterns and others are dropped. Now at the final leaf level we check for two new probable patterns but none of them would satisfy their track and hence we get no reportable pattern finally. For this position the tree obtained is shown in Fig. 4.

In this way, after considering all the trees we take the best of the reportable pattern based on some predetermined criterion and report it finally to the user.

Table 4: Performance comparison among different algorithms for fixed l, d and varying n. ANT showed tremendous improvement over others in finding motifs from a large sample. Even at a length 2000 its performance stays over 0.60

Sequence length	200	400	600	800	1000	1200	1400
Consensus	0.94	0.31	0.07	0.09	0.04	0.00	0.00
Gibbsdna	0.96	0.46	0.12	0.34	0.12	0.04	0.06
Meme	0.78	0.37	0.10	0.03	0.00	0.00	0.00
Winnower (k = 2)	0.98	0.95	0.92	0.02	0.02	0.02	0.00
Sp-star	0.98	0.96	0.84	0.69	0.23	0.24	0.16
ANT	1.00	1.00	0.90	0.90	0.90	0.81	0.81

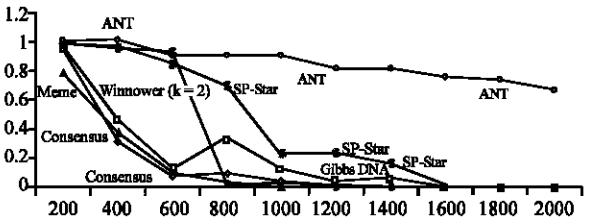

Fig. 5: Comparative performance of related algorithms (data from Pevzner and Sze (2000) with permission) and ANT

Performance evaluation: To evaluate the performance of our algorithm we generated some random test sample according to Workman and Stormo (2000). That is in the data set (t = 20) we have N = 600, 800, 1000, 1200, 1500, 1800, 2000 where each of the sequence of length 'N' is random. Then we implant the signal pattern (l, d) randomly in the sequences where each signal pattern is mutated randomly.

For performance calculation, let K be the set of known signal positions in a sample set and let P be the set of positions predicted by our algorithm. We have used performance coefficient |KnP|/|KUP| as defined by Pevzner and Sze (2000). The result is very good as we can see from Fig. 5 that for length 600 the performance is excellent and even for larger N the performance coefficient is very good.

To compare the performance of our algorithm with other motif searching algorithms we take a look at the charts provided by Pevzner and Sze (2000) and here we provide our own performance with respect to percentage of mismatch (number of mutation) for various pattern length (Table 4). The performance clearly shows the difference between the algorithms mentioned above and our algorithm. The range of percentage of tolerable mismatches is far improved here.

DISCUSSION

In this study a new algorithm for finding motifs based on heuristic refinement is presented. Experimental results have shown that our algorithm is much more effective at recovering planted (l, d) motifs in simulated data than some previously existing algorithms. Important feature in our algorithm is that it can accommodate more than one planted instances per sequence which many of the existing algorithm can not handle. The enhanced performance achieved by our algorithm stems from the utilization of the heuristic refinement step; instead of refining all or some percent of the seed patterns the heuristic refinement discard most of the unrealistic pattern very early of the refinement so that they cannot bother the algorithm any more. We should mention here that in its current implementation the algorithm cannot handle flexible gaps and/or insertion and deletion. We are currently extending our algorithm so that detection of gaps becomes the integral part of the algorithm.

The algorithm can be further improved by reducing the time complexity. This can be done by combining some statistical calculation with the rigidity of the proposed algorithm. It can also be extended to find motifs with unknown or variable length which would be possible by merging small preliminary motifs to build a larger optimal motif.

REFERENCES

Bailey, T.L. and C. Elkan, 1995. Unsupervised learning of multiple motifs in biopolymers using expectation maximization. Machine Learn., 21: 51-80.

Blanchette, M., B. Schwikowski and M. Tompa, 2000. An exact algorithm to identify motifs in orthologous sequences from multiple species. In Proceedings of the 8th International Conference on Intelligent Systems for Molecular Biology, San Diego, CA., pp: 37-45.

Brazma, A., I. Jonassen, J. Vilo and E. Ukkonen, 1998. Predicting Gene Regulatory Element In-silico on Genomic Scale. Genome Res., 15: 1202-1215.

Buhler, J. And M. Tompa, 2002. Finding Motif Using Random Projection. J. Comput. Biol., 9/2: 225-242.

Galas, D.J. M. Eggert and M. Watermann, 1985. Rigorous Pattern-recognition Methods for Dna Sequences: Analysis of Promoter Sequences from Escherichia Coli. J. Mol. Biol., 186/1: 117-128.

Herts, G.Z. and G.D. Stormo, 1999. Identifying DNA and protein patterns with statistically significant alignments of multiple sequences. Bioinformatics, 15: 563-577.

Lawrence, C.E. and A.A. Reilly, 1990. An Expectation Maximization (EM) algorithm for the identification and characterization of common sites in unaligned biopolymer sequences. Protein: Struct. Func. Gene., 7: 41-51.

Lawrence, C.E., S.F. Altschul, M.S. Boguski, J.S. Liu, A.F. Newald and J.C. Wootton, 1993. Detecting subtle sequence signals: A Gibbs sampling strategy for multiple alignments. Science, 262: 208-214.

Neuwald, A.F. and P. Green, 1994. Detecting Patterns in Protein Sequences. J. Mol. Biol., 239/5: 698-712.

Pevzner, P. and S.J. Sze, 2000. Combinatorial approaches August 21, 2006to finding subtle signals in DNA sequences. In Proceedings of the 8th International Conference on Intelligent Systems for Molecular Biology, San Diego, CA., pp: 268-278.

Rocke, E. and M. Tompa, 1998. An algorithm for finding novel gapped motif in DNA sequences. Proceedings of the 2nd Annual International Conference on Computational Molecular Biology, New York, pp: 228-233.

Tompa, M., 1999. An exact method for finding short motifs in sequences, with application to the ribosome binding sites. Seventh International Conference on Intelligent Systems for Molecular Biology, Heidelberg, Germany, pp: 262-271.

Workman, C. and G. Stormo, 2000. ANN-SPEC: A method for discovering transcription factor binding sites with improved specificity. In: Pacific Symposium of Biocomputing, 6: 464-476.