APPROXIMATE MATCHING USING INTERVAL JUMPING SEARCHING ALGORITHMS FOR DNA SEQUENCES[1]

JIA-HAN CHU, WEI-YUAN CHANG, TUN-WEN PAI
Department of Computer Science, National Taiwan Ocean University, No. 2, Pei-Ning Rd. Keelung ,Taiwan 20224, Republic of China
twp@mail.ntou.edu.tw

MARGARET DAH-TSYR CHANG, HSIU-LING TAI, HAO-TENG CHANG
Department of Life Science, National Tsing Hua University, No. 101, Sec. 2, Kuang Fu Rd. Hsinchu ,Taiwan 30013, Republic of China
dtchang@life.nthu.edu.tw

Abstract - *In this paper, a novel algorithm for approximate pattern matching from multiple DNA or amino acid sequences is designed. To improve efficiencies in approximate matching, interval jumping searching algorithms and voting mechanism are combined and applied in this system. Since k-mismatch problem of DNA or amino acid sequences from the giant genomic database is usually time consuming, we developed a parametric encoding methodology to search tolerant sub-strings and reduce the time complexity. In this paper, we have shown that the system can rapidly identify potential binding motifs for the transcription factor, such as GAL4 in the promoter regions of gal4 and gal6, which are neither predicted nor demonstrated to be regulated by GAL4 previously.*

1. Background and Motive

Approximate matching is an important research topic with application to a variety of problems including DNA sequence matching, degraded signals detection, and pattern matching for typos in texts. There are numerous approximate matching techniques developed for various applications, and they can be classified into two main categories: indexed searching techniques and online searching algorithms [1,2,3,4,5]. Indexed approximate searching algorithms build a persistent data structure prior to the searching processes and there are several reasons that prevent keeping the indices on the text including extra space requirements, volatility of text (indices building is quite costly and needs to be amortized over multiple searches), and satisfied adequate performance in real applications of huge texts. Online searching algorithms process the pattern matching without preprocessing the text and building an index at the beginning. Their practical problems of statistical behavior, history, development, and relative computational complexities were well described and discussed by Navarro[1]. These online approximate searching algorithms can be mainly grouped into four categories: Dynamic Programming Matrices, Finite Automata, Filters, and Bit-Parallelism. There are also several combinations of different categories such as Bit-parallel techniques based on automata or dynamic programming matrices. All these different algorithms face various weakness properties in either speed or space considerations [1]. In this paper, the proposed approximate matching methodology is based on interval jumping searching algorithms[6], which focused on the transformed number sets, and employed voting mechanism to efficiently match tolerant patterns from multiple sequences. More technical details are described in the following sections, and its main application will focus on bioinformatics areas. In the post-genomic era, genomic sequences of many species have been completely determined. Important subsequent studies are gene annotation and functional analysis [7], in order to gain the insights in features, structures, and functions of the regulatory segments of genes. Generally, high similarities are found among nucleotide sequences and the deduced amino acid sequences of conserved protein families. Employing the tools provided by existing DNA or protein databases, matching of such characteristics can be found quickly. However, vast areas on chromosomes still contain segments with unknown genetic functions, which require new bioinformatic tools to perform analysis. Chromosomes often contain special patterns for protein recognition and binding. Such DNA and protein interaction can affect gene expression or protein functions. For example, the upstream promoter region of an eukaryotic gene possesses so called “transcription element” for regulatory protein to bind, and its corresponding protein is named as transcription factor. Though features of known short DNA sequences can be identified by several programs[8,9,10], they are often obtained by one-to-multiple string matching, which means sending a DNA string sequence into a database and retrieving segments with known features. However, unknown or novel k-mismatch segments are difficult to be identified using the conventional methods. Thus, we

[1] This work is supported by NSC-93-3112-B-007-005

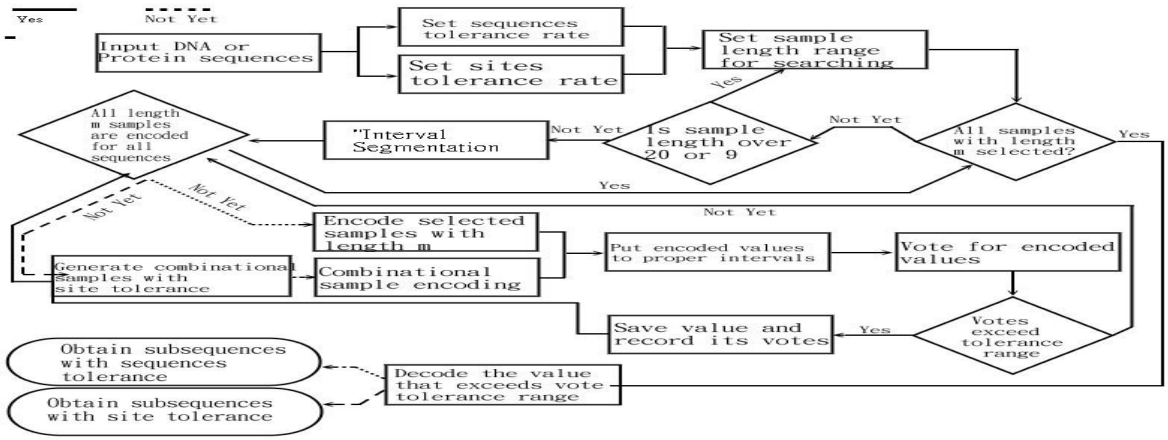

Figure 1: System Architecture

have developed a complete program as a new solution and we can search for special consensus DNA sequences that are not well aligned in a set of gene family efficiently by employing the so called approximate matching methodologies based on Ladderlike Stepping and Interval Jumping Searching Algorithms [6].

2. System Description

The architecture of our system is shown in Fig. 1. Through encoding, interval segmentation, and voting mechanism, we can search all approximate patterns meeting the occurrence threshold values. The detailed algorithms are described in the following sections

2.1. Subsequence representation feature of occurrence

The concepts of coding theory and interval segmentation, as well as the listed data structure in[6] are applied for exact or approximate matching among multiple sequences. Both matching algorithms require finding existed common substrings at the first step. Here we propose an intuitive statistic model which requires common substrings to appear only in part of family sequences with respect to user-defined occurrence rate, which is also known as representation characteristic of occurrence. The common substrings in certain family with occurrence rate means that they only occur in K sequences among all N input sequences, where $K \le N$. For example, searching for common subsequences with at least 80% representation percentage in 10 sequences means that the selected common substrings occur in at least 8 sequences from the family set.

2.2. Interval Segmentation

The searching pattern and text are encoded into number set by our previously developed algorithm, LIJSA (Ladderlike Stepping and Interval Jumping Searching Algorithms), based on Karp-Rabin(Ricardo, B.Y. 1992). To improve computational efficiency, the algorithm applies the bitwise and uniform interval segmentation from [6]. With uniform segmentation as $\lfloor A/sbase \rfloor$, the interval location of encoded value can be calculated, where A is the encoded value and *sbase* is the fundamental segmentation base with respect to the pattern length. Based on the process of interval segmentation, the computational complexity can be effectively reduced by appropriately allocating the encoded data by comparison.

2.3. Cumulative Voting Algorithm

We apply voting theorem algorithm (VTA) on N encoded sequences for its statistical features. Each different encoded value is a representative as a subsequence candidate, and if such a value occurs in one sequence, one vote is obtained. Thus each candidate subsequence representative can obtain at most N votes. The parameter K is the threshold to be assigned. If the user-defined possibility parameter is P%, K can be calculated from $K = \lceil N \times P\% \rceil$. The sample with P% representation percentage means that they appear at least in K sequences among N sequences. Fig. 2 describes the modules for searching substrings with occurrence rate. In the first module, the number of intervals with sample length m is calculated through numeric interval segmentation. The second module encodes the sequences and the third module obtains the interval allocation and vote counting. By rapid searching method and the interpolation nature of list structure, we could locate values and place nodes for cumulative voting. The data structure is described as follows:

Figure 2: Flowchart of String Searching Algorithm with Substring occurrence rate

In the third module for list searching and interpolation we use a three-level data structure, *DS*[||], where,, and refer to the first, second and third level, respectively. Data structure of level is an array, in which each element indicates an interval: [*Interval* 0|*interval* 1…| *interval* I_{total}] ; level is a list structure, where each node contains data structure {*pn*; *tn*; *tfes*}, in which *pn*, *tn*, *tfes* refers to *pattern number*, *ticket number*, and *ticket flag for each sequences*, respectively. *pn is* defined as unique numbers for each encoded values. For distinguishing different encoded values, list searching algorithm is applied. *tn* is the total votes for each encoded value obtained. *tfes* is the flag annotation to record whether an encoded sample value is obtained from sequence *N*, and should be a set with *N* elements, in which set the 1st element refers to occurrence flag of encoded pattern from the 1st sequence. "1" implies at least one votes, "0" implies no vote for encoded sample. For example, while N=5，*tfes*={1,1,0,0,1}, the encoded pattern has votes from sequences 1,2, and 5. Level is an array structure, which is represented as {{pis_1()};{pis_2 ()};…;{pis_N()}} for the pattern allocation in N sequences. For example, while N=5, *DS*(6;24, 8;2; {1,0,0,0,1},{7,,,,56;158}) means that the 8th encoded pattern is located in the 6th interval, position 24, and totally 2 votes obtained from sequence 1 and 5. These common patterns are located at the 7th and 56th site position in sequence 1, and 158th site position in sequence 5. If $tn \geq K$, the sample is selected, otherwise , it will be ignored.

2.4. String Searching Algorithms with Site-Fixed/Site-Opened Tolerance

In previous sections, the algorithm searches for common substrings in *N* nucleotide or amino acid sequences. And the symbols in each site in common substrings are fixed. However in practical biological evolution, degeneracy in each site is possible while the chemical properties remain similar. In the process of mRNA to protein transcription, three base pairs are transcribed to 20 proteins through the codon table with 64 combinations. Obviously, site tolerance is important while searching for common patterns, especially while searching for meaningful samples such as TATA-box, CG-box, or TT-box. Thus in this section we introduce the site tolerance searching system we developed.

The searching algorithms with tolerant characteristics are classified into site-fixed and site-opened tolerance algorithms, respectively. In determined or randomized approximate searching, named as site-fixed algorithm, user decides which site position of pattern are allowed with tolerance in the user interface. Each site has a corresponding checking box for option True (default) or False. False setting means that this site is with tolerance, so the base pair can be represented with that symbol N. In the second site-opened algorithm, user-defined parameters are pattern length and number of tolerance sites. The system will search all combination of possible mismatch patterns according to these settings.

For the first algorithm, site-fixed tolerant pattern matching algorithm is based on a modification from the ladder-like interval jumping searching algorithms[6]. To approach the function, the tolerance sites are encoded as 0 during the encoding phase. Then the ladder-like interval jumping searching algorithm rules can be rapidly performed to search common pattern with specified site tolerance. In the second algorithm, site-opened tolerance algorithm, users only need to input the number of tolerance sites and searching pattern length as parameters. Suppose there are *N* sequences and the searching pattern length is *m*, when the length of longest sequence in *N* sequences is L_{max} and there are $P_{\max}$ different substrings with pattern length *m*, where $P_{\max} \leq L_{\max} - m + 1$, user-defined parameter *k* is the number of tolerance sites in a sample, where $k \leq m$, there are *k* variant tolerance sites in a pattern. Thus each pattern has $C_k^m \times 4^{m-k}$ possible combinations. Users could also search for all possible common substrings with 0~*k* tolerance sites. In this case, the *k* mismatch pattern combinations become $\sum_{i=0}^{k} C_i^m \times 4^{m-i}$. In the worst case, only after the system has compared with all combinations of a certain pattern in all other sequences for all possible combination of this pattern, the system can verify if a certain combination is the final common substring. Thus the time complexity becomes $O((\sum_{i=0}^{k} C_i^m \times 4^{m-i}) \times N \times P_{\max}^2 \times (N-1))$. If we fix the sample length *m*, $\sum_{i=0}^{k} C_i^m \times 4^{m-i}$ can be viewed as a constant *C*. If $P_{\max}$ is replaced by $L_{\max} - m + 1$, and $L_{\max} >> m$, the time complexity is approximate to $O(CN^2 L_{\max}^2)$. To search all common subsequences for all possible tolerance sites rapidly, we propose such an algorithm which lowers searching time complexity to $O(L_{\max} \times n_{\max} \times (N + C))$, where $n_{\max}$ represents the quantity of values in intervals containing the most quantity of values after uniform interval segmentation, $n_{\max} \leq N(L_{\max} + m - 1)$,the procedure as shown in Fig. 3. According to Fig. 3, in the first module users input the data for sequence searching. In the second module, the system runs the cumulative VTA, for classifying all the patterns of input sequences into common subsequences and non-common substrings. All the common substrings are satisfied for final solutions, while non-common substrings require further analysis and comparison. Once non-common substrings are extracted, they are viewed as input data for further analysis and voting module, as non-common substrings module in Fig. 3. There are slight differences between Fig. 3 and Cumulative Voting Theorem Algorithm in their purpose, data structure, and algorithm. After site-opened tolerance voting algorithm, the system outputs all possible substrings.

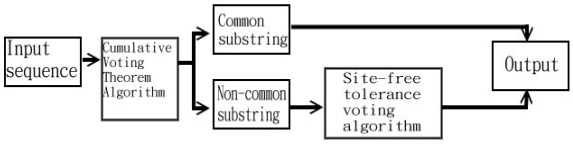

Figure 3: Site Tolerance Searching Algorithm

There are two steps required for the Site-Opened Tolerance Voting Algorithm, and depicted in Fig. 4. The system produces all possible combinations of substrings from non-common substring set with tolerance sites number equal or less than *k*, given by users. The method to produce combinations, for example, given pattern length= 4 ，*k*=2, as in Fig. 5, 'x' refers to the positions of tolerance sites, and the positions without 'x' refers to non-tolerance sites positions. Produced tolerance positions in the first time is marked as 'x_1' ,and 'x_2' in the second time. Dotted lined 'x_2' means that the position before the continuous line'x_2' is placed on. The arrow between 'x' and 'x' represents the moving direction of 'x'. The leftmost column represents the results of the first round.

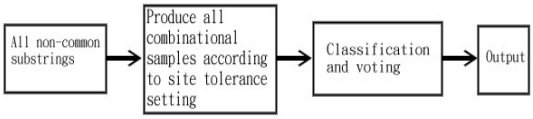

Figure 4: Site-opened Tolerance Voting Algorithm

When all the possible combination is produced, the third procedure, classification and voting module as shown in Fig. 4 is applied. The encoded values of all combinations are placed in intervals and were counted for votes while placing patterns. Placing methods are list searching and interpolation algorithms. If the same combination occurs in one sequence, one vote is added. To be elected, the pattern must occur in *N* sequences and obtain combination pattern with *N* votes.

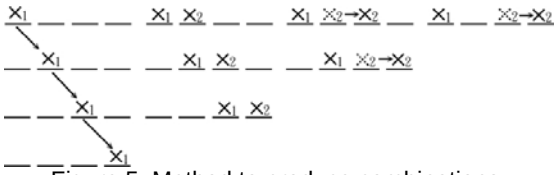

Figure 5: Method to produce combinations

3. Results and Discussions

In the worst case of the original searching algorithms, each pattern requires comparing once with patterns from other sequences to ensure it is the final common subsequence. Suppose $L_{\max}$ represents the length of the longest sequence among *N* sequences, with searching pattern length *m*, the worst case time complexity is $O(N\times(L_{\max}-m+1)^2\times(N-1)\times m)$. If $L_{\max} >> m$, time complexity can be reduced to $O(N^2{L_{\max}}^2 m)$. If we apply the encoding technique in this paper to the searching algorithm, time complexity can be reduced to $O(N^2{L_{\max}}^2)$. As for the voting algorithm in this section, time complexity for finding common substrings is $O(N\times(L_{\max}-m+1)\times n_{\max})$, $n_{\max}$ represents the number of elements in the interval with most elements after bitwise segmentation, where $n_{\max}\le N(L_{\max}-m+1)$, while $L_{\max} >> m$,in the worst case, time complexity becomes $O(N^2{L_{\max}}^2)$. In practical application, $n_{\max} << NL_{\max}$, searching time is reduced significantly. However in practical example, $n_{\max} << NL_{\max}$, as in Fig.6, we apply the proposed algorithms on RNase, primates and non-primates, yeast GAL promoter, and yeast MATalpha2 promoter. The RNase family contains 8 nucleotide sequences, 3000 bps each, primates and non-primates families contain 8 nucleotide sequences with 500 bps each, yeast GAL promoter family includes 6 sequences, 350 bps each, and yeast MATalpha2 promoter family includes 7 sequences, 1300 bps each. We compared the speed of VTA with other similar traditional searching algorithms: KR(Karp-Rabin), and TKR(Tuned Karp-Rabin). With the same hardware environments and pattern length settings, VTA with interval jumping techniques performs obviously superior to other algorithms.

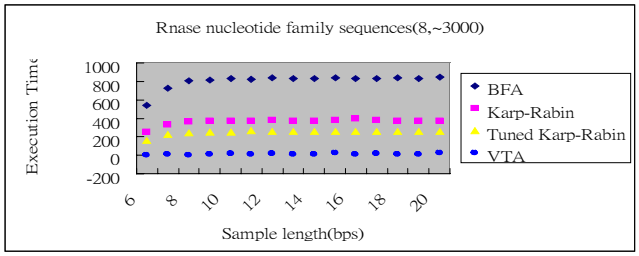

Figure 6(a): Execution time comparison between VTA,BFA,KR, TKR, with RNase Family nucleotide family sequences

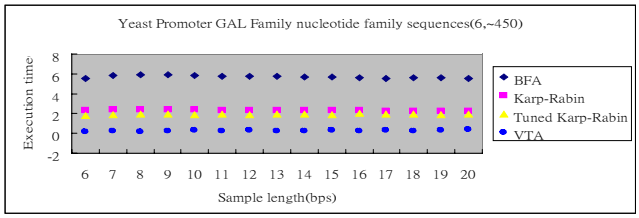

Figure 6(c): Execution time comparison between VTA,BFA,KR, TKR, with Yeast Promoter GAL Family nucleotide family sequences

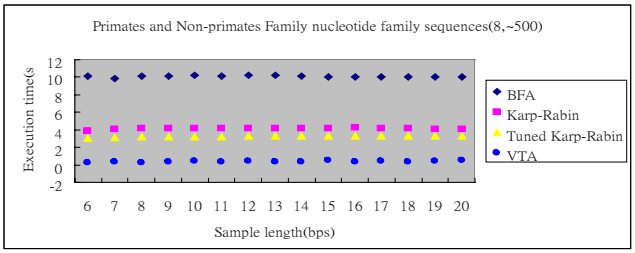

Figure 6(b): Execution time comparison between VTA,BFA,KR, TKR, with primates and non-primates Family nucleotide family sequences

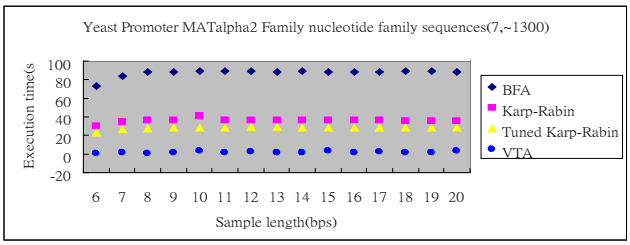

Figure 6(d): Execution time comparison between VTA,BFA,KR, TKR, with Yeast Promoter MATalpha2 Family nucleotide family sequences

For the applications of k-mismatch pattern searching, we utilize the GAL protein family for demonstrating the effectiveness of the proposed algorithms against other existing searching tools. The GAL protein family consists of twelve members, GAL1, GAL2, GAL3, GAL4, GAL5, GAL6, GAL7, GAL10, GAL11, GAL22, GAL80 and GAL83. It is one of the most well-studied systems in budding yeast *Saccharomyces cerevisiae*. The GAL genes are strongly regulated at transcription level via carbon source and some of them were underlying the genetic relationships [11]. It has been reported that transcription of *gal1*, *gal2*, *gal7*, *gal10* and *gal80* were regulated by a transcription factor GAL4 [12] and the consensus sequence of its binding site was defined as CGGNNNNNNNNNNCCG according to the Promoter Database of *Saccharomyces cerevisiae* (http://cgsigma.cshl.org/jian/index.html). In order to investigate whether the other genes of the GAL family were also regulated by GAL4, we retrieved the upstream 1,000 nucleotides of the twelve GAL genes and analyzed these sequences via TRANSPOLER® 1.2 downloaded from Biobase (http://www.gene-regulation.com) and our proposed algorithms. The results revealed that among twelve query sequences, TRANSPOLER® 1.2 could only identify 6 GAL4 binding sites, one for each of the 6 genes, *gal1*, *gal2*, *gal3*, *gal7*, *gal10* and *gal80*. Utilizing our program to search for a sequence motif of 17 bases containing 3 consensus nucleotides at 5' and 3' ends with 11 degenerate nucleotides in between, we found that different representation percentages from sequence family, 100%, 80%, 70%, and 65%, resulting in 0, 7, 24, 60 motifs, respectively (Table1). When tolerance of 65% was used, the specific binding motif for GAL4 was located in the upstream sequences of each of the 8 genes, *gal1*, *gal2*, *gal3*,*gal4*, *gal6*, *gal7*, *gal10* and *gal80*. Of which the GAL4 binding sites of *gal4* and *gal6* were not predicted by TRANSPOLER® 1.2. Our results indicated that *gal3*, *gal4*, and *gal6* were potential genes co-regulated by GAL4.

Table 1. The different representation percentages from GAL family, 100%, 80%, 70%, and 65%, resulted in 0, 7, 24, 60 motifs, respectively

k-mismatch motifs found with 100% representation	k-mismatch motifs found with 80% representation	
None	TTT**********GTT(17) TTT**********AAT(17) TTT**********AAA(17) GTT**********TTT(17)	TAA**********TTT(17) AAG**********CTT(17) GAA**********AAA(17)
k-mismatch motifs found with 70% representation	k-mismatch motifs found with 65% representation	
TTT**********GTT(17) GAA**********AAA(17) GTT**********TTT(17) TAA**********TTT(17) AAG**********CTT(17) TTT**********AAA(17) TTT**********AAT(17) AAT**********ATT(17) TTA**********TTT(17) AAA**********CTT(17) TAA**********ATA(17) TAA**********AAA(17) TGT**********TTT(17) ATT**********CTT(17) ATA**********ATA(17) AAA**********AAA(17) TTT**********CTT(17) TCA**********AAT(17) AAA**********TTT(17) AAA**********AAT(17) ATG**********TTT(17) TTT**********TTC(17) AAG**********AAA(17)	GAA**********AAG(17) GAA**********AAA(17) GTT**********TTT(17) GCA**********TTT(17) AAG**********AAA(17) AAG**********CTT(17) AAA**********GAA(17) AAA**********AGA(17) AAA**********AAG(17) AAA**********AAA(17) AAA**********AAT(17) AAA**********AAC(17) AAA**********ATT(17) AAA**********TTA(17) AAA**********TTT(17) AAA**********CTT(17) AAT**********AAA(17) AAT**********AAT(17) AAT**********ATT(17) AAT**********TGA(17) AAT**********TAC(17) ATG**********ATT(17) ATG**********TTT(17)	TAG**********TTT(17) TAA**********AAA(17) TAA**********ATA(17) TAA**********TTT(17) TAT**********GGA(17) TAT**********TTT(17) TTG**********TGC(17) TTA**********TTT(17) TTT**********GTT(17) TTT**********AGA(17) TTT**********AAA(17) TTT**********AAT(17) TTT**********ATA(17) TTT**********ATC(17) TTT**********ACT(17) TTT**********TAA(17) TTT**********TAT(17) TTT**********TTG(17) TTT**********TTA(17) TTT**********TTT(17) TTT**********TTC(17) TTT**********CAA(17) TTT**********CTT(17)

TTT**********AGA(17)	ATA**********ATA(17) ATA**********TTT(17) ATT**********AAT(17) ATT**********CTT(17) ATC**********CAA(17) TGA**********AAA(17) TGT**********TTT(17)	TTC**********AAA(17) TTC**********TTT(17) TCA**********AGC(17) TCA**********AAT(17) CGG**********CCG(17) CAA**********TAA(17) CAA**********TTG(17)

4. Conclusions

A novel algorithm for approximate matching tolerant segments based on interval jumping searching algorithm for multiple DNA or amino acid sequence analysis is designed. To improve efficiencies and effectiveness in approximate matching, ladderlike interval jumping searching algorithms (LIJSA) and voting mechanism are combined and applied in this system. The k-mismatch problem of DNA or amino acid sequences from the giant genomic database is usually time-consuming, we developd a parametric encoding methodology to search tolerant substrings so that the time complexity in the comparison can be reduced. For the proposed interval jumping and voting theorem algorithms, the computational complexities are $O(NL_{\max}{}^2 \times (N+C))$ in the worst case and k-approximate mismatch in $O((NL_{\max}{}^2 \times (N+C))/2)$ on average, where N is the number of sequences, L_{max} is the maximal length of the set of sequences, C is the total possible k-mismatch combinational number. In our experiment, with different sequence occurrence rate and k-mismatch settings, result shows our system can locate degenerated transcription factors, binding sites, in multiple sequences, for example, GAL4 in the promoter regions of gal4 and gal6, which are neither predicted nor demonstrated to be regulated by GAL4 previously by other tools.

References

1. G. Navarro, A guided tour of approximate string matching. *ACM Computing Surveys*, No. 1:31-88, 2001.
2. P. Jokinen and E. Ukkonen, Two algorithms for approximate string matching in static texts, In Proceedings of the 2nd Mathematical Foundations of Compter Science(MFCS'91). Springer-Verlag, Berlin, Vol. 16:240-248, 1991.
3. G. Myers 1994a, A fast bit-vector algorithm for approximate string matching based on dynamic programming. J. ACM 46,3:3995-415, 1999.
4. E. Sutinen and J. Tarhio, Filteration with q-samples in approximate string matching. In Proceedcings of the 7th Annual Symposium on Combinatorial Pattern Matching(CPM'96). LNCS, Springer-Verlag, Berlin, Vol. 1075:50-61, 1996
5. R. Baeza-Yates, and G. Navarro, Block addressing indices for appromimate text retrieval. J.Am. Soc. Inf. Sci.(JASIS)51,1:69-82, 2000.
6. T. Pai, D. Chang, J. Chu, W. Chang, and H. Tai, Ladderlike stepping and interval jumping searching algorithm for DNA sequences, APBC2004, Vol.29:93-98, 2004.
7. L. Stein, Genome annotation : from sequence to biology. Nature reviews genetics, 2:493 – 503, 2001.
8. P. Baldi and P. Baisnee, Sequence analysis by additive scales : DNA structure for sequences and repeats of all lengths, Bioinformatics, 16: 865-889, 2000.
9. Z. Ning, Cox AJ, and JC. Mullikin, SSAHA:a fast search method for large DNA databases, *Genome Research*, 11:1725-1729, 2001.
10. N. Volfovsky, B.J. Haas, and S.L. Salberg:A clustering method for repeat analysis in DNA sequences, *Genome Biology*, 2001.
11. D. Lohr, *et al.* Transcriptional regulation in the yeast GAL gene family: a complex genetic network. *FASEB J* 9(9):777-87, 1995.
12. E. Giniger, S.M. Varnum, and M. Ptashne, Specific DNA binding of GAL4, a positive regulatory protein of yeast. Cell 40:767-774, 1985.