Journal of Engineering Design and Analysis
Volume 3, Issue 2 - 2020, Pg. No. 1-7
Peer Reviewed Journal

Research Article

A String-Matching Operation using Finite Automata and Online Interface for Bioinformatics Algorithms

Sambit Pattanaik

Gandhi Institute for Technological Advencement, Bhubaneswar, Odisha, India.

I N F O

E-mail Id:
sambit_cse@gita.edu.in
Orcid Id:
https://orcid.org/0000-0002-4651-7633

How to cite this article:
Pattnaik S. A String-Matching Operation using Finite Automata and Online Interface for Bioinformatics Algorithms. *J Engr Desg Anal* 2020*;* 3(2): 1-7.

Date of Submission: 2020-11-04
Date of Acceptance: 2020-12-02

A B S T R A C T

In this study, I present a new web interface for major bioinformatics algorithms and introduce a novel approximate string-matching algorithm. My web interface executes major algorithms on the field for the use of computational biologists, students or any other interested researchers. In the web interface, algorithms come under three sections: Sequence alignment, pattern matching and motif finding. In each section, I introduce algorithms in order to find best fitting one for specific dataset and problem. The interface introduces execution time, memory usage and context specific results of algorithms such as alignment score. The interface utilizes emerging open source languages and tools. In order to develop light and user-friendly interface, all parts of the interface coded with Python language. On the other hand, Django is used for web interface. Second contribution of the study is novel A-BOM algorithm, which is designed for approximate pattern matching problem. The algorithm is approximate matching variation of Backward Oracle Matching. I compare my algorithm with popular approximate string-matching algorithms. Results denote that A-BOM introduces 30% to 80% short runtime improvement when compared to current approximate pattern matching algorithms on long patterns.

Keywords: Bioinformatics, A-BOM, Interface, Approximate Pattern Matching

Introduction

Recent development of the technology has introduced big amount of data in scientific fields. For instance, biologists extract has DNA sequences of organisms, where a human genome consists of nearly 3 billion nucleotides. In order to the DNA store and extract its features, new computational methods and tools are needed. As a result of this fact, a new discipline, Bioinformatics, has been emerged. Bioinformatics is an interdisciplinary study field which tries to understand biological information. For this goal, researchers of the computer science and biology introduces various tools and software that can collect, store and process biological data. Particularly, main motivation of computer scientists presenting new algorithms and software tools. One sub field of Bioinformatics is fast and accurate sequence matching among long nucleotide sequences. The sequence matching studies are important since DNA strand of living organisms are very long. For instance, human genome consist of nearly 3 billion nucleotides and sequence alignment among the genome sequences are computationally expensive. Therefore, efficient algorithms and software tools are highly demanded.

In terms of computational aspects of biology, there exist three major sequence alignment problems. Literature denotes these problems as sequence alignment, pattern matching, motif finding.

The sequence alignment process is finding relationships between the sequences to identify similarity of species. The problem is mutations can occur in DNA sequences and a single mutated nucleotide on middle of long sequence corrupts all alignment. This problem handled by dynamic programming and its variations like Smith Waterman and Needleman Wunsch.[1]

Pattern matching is the second challenging problem in bioinformatics.[2] The process in the basic is detecting the exactly same of pattern presences of a given pattern in a long sequence. Since the single one sequence consist of about 3 billion nucleotides, in case of programming 3 billion characters, brute force approaches can't handle the problem in reasonable time. To solve this problem, searching approach should detect the positions which have no chance to match and skip these points for reduce volume of searching points.

Motif finding is the third and still under development problem in bioinformatics. The main idea for motif finding is detect the most repetitive sub sequences.[3] There are many problems for the process like how many is motif length should be or how can group 'k' length patterns current approaches usually offer divide and conquer technique. In the interface, an algorithm had presented for motif finding with the technique.

Sequence alignment is commonly is used by biologists to compare nucleotide sequences and to find functions of the genomes. There exist various software utilities that contain tools to do string matching methods such as sequence alignment, pattern matching and motif finding. On the other hand, advancements in web framework technologies and programming languages enables to design better software tools. Also, novel string- matching algorithms give rise to new interfaces and tools.

Performance of the string-matching algorithms depends on the data set and problem[4] Even further performance of and approximate string matching depends on the data. Most commonly used techniques are based on Dynamic Programming.[5] However, the techniques require high memory consumption.

Finally, some of the most efficient genomic analysis tools require licenses. Also, the tools may have access limitations. In contrast, developing an open source tool with easy access property contributes to the educational demands. So that native language support can be introduced as well. This study aims to present free, complete, user friendly interface for whole bioinformatics field. The tool supports both English and Turkish languages. Therefore, it can be useful for biology students who cannot read English.

The tool also introduces a novel approximate string matching algorithm. The algorithm speeds up string matching time. Due to its automata-based technique, it also reduces memory consumption.

Overall the study has two contributions to the literature. First, it presents a novel approximate string-matching algorithm. Second it introduces a new bioinformatics interface, which is coded with open source languages. The bioinformatics interface presents a simple and efficient interface. Together with its native language support it supports academic improvement.

Definitions and Literature

Sequence alignment, pattern matching and motif finding problems have several solution approaches. In this section each major problem and their fundamental solutions will be mentioned in separated subheadings. Only de facto algorithms have explained in detail, other approaches in literature are variations of these major algorithms.

Sequence Alignment

Sequence alignment aims to find similarity of two squences. Let's suppose that we have two sequences defined as:

$$T_1 = t_0, t_1 ..., t_{n-1} \quad and \quad T_2 = t_0, t_1 ..., t_{n-1} \quad \{t_i \in A, C, G, T\}$$

The sequences are not exactly the same but they are very similar to each other. In example, there are two text that are T1 and T2 and all characters of the texts are same except i-th character. The i-th character of T1 is not same with i-th character of T2. So, the equation can be defined as:

$$T_1(0) = T_2(0), T_1(1) = T_2(1) ..., T_1(i) \neq T_2(i), ..., T_1(n-1) = T_2(n-1)$$

To find optimum relativity between the sequences, sequences need to be realigned with gaps. Sequence alignment is an essential problem because in real world, sequences not always remain in their original form of being created due to mutations. On the other hand, corruptions may arise during sequencing. To solve these kinds of problems, there are two major approaches in literature; Smith-Waterman and Needleman Wunsch. Both algorithms are variation of dynamic programming.[6]

Smith-Waterman Algorithm is a variation of dynamic programming. Dynamic programming approaches for sequence alignment have common variables like match score, mismatch score and gap score to calculate similarity score. Dynamic programming using for creating a relativity matrix from the sequences in Smith-Waterman algorithm. Each node of matrix value is maximum value of transitions from left, top and left top diagonal nodes. There are one more value which is zero for calculation maximum value additionally. Zero value gives a guarantee to there are no negative value in matrix. This particular precaution increases alignment success efficiency. The diagonal transition represent match, other transitions means gaps. Once matrix have crated, trackback on matrix from last node to first node for calculating alignment score. Algorithm and explanations can be found on.[7]

Pattern Matching

In terms of exact string search, pattern matching can be defined as detecting occurrences of the pattern on a long sequence.

Let suppose we have a sequence *T* as defined in sequence alignment. On the other hand, we have another short sequence which entitled pattern, *P*, as:

$P = p_0, p_1 \ldots, p_m$

Pattern matching aims to locate the sub sequences which is same with P exactly or tolerance contrast in a range. In general, there are two approach to pattern matching: exact and approximate matching. Exact pattern matching aims to find presences of exactly the same of P in sequence as follows:

$p_0 = t_i, p_1 = t_i{+}1, p_2 = t_i{+}2, \ldots, p_m\, t_i{+}m{+}1$

Current approaches using several skip algorithms to do this process efficiently. Skip algorithms boost matching process because many position skips and that means far less operations while matching. Essentially there are two main idea behind the skip algorithms, bad character and good suffix. Bad character means if there is any mismatch while matching, shift the pattern until the bad character is not in current sub sequence. The good suffix means if there is any prefix which same with suffix on mismatch point, shift pattern to align prefix with suffix. All major algorithms developed with these two approaches like KMP, Boyer Moore, BOM etc.

Knuth Morris Pratt algorithm is an exact pattern matching algorithm which searches for presences of P within a subsequence T by using bad character approach. Before the matching process, pre-process should be done on P for calculating skip count for every position of P. Detailed explanation can be found on.[8]

The Boyer-Moore algorithm is another exact matching algorithm. As a distinct from KMP, Boyer-Moore algorithm combining good suffix and bad character approaches. While matching, if there is a mismatch between current part of T and P, looking bad character and good suffix tables respectively for decide shift count on current position. Details can be found on.[9]

Another exact pattern matching algorithm is Backward Oracle Matching, BOM. BOM is an automat-based algorithm which is variation of BNDM algorithm. The details of BNDM algorithm can be found on. The BOM algorithm based on the Boyer-Moore strategy. Thereupon try to match prefix with suffix of the pattern on mismatch position. On the other hand, matching progress performs right as a necessity of good suffix approach. The algorithm using automat instead of tables unlike other Boyer-Moore approaches.

The first step of generating BOM automat is taking reverse of pattern and generates states for each character in reversed pattern and character transitions are added between the states respectively. After produced all factors of *P*, transitions for factors appends to the automat. Search algorithm and details can be found on.

The second approach for pattern matching is approximates pattern matching. Approximate pattern matching differs from exact matching with mismatch tolerance. That means matching process tolerance to mismatches as long as number of mismatches is under threshold. Formula:

$[P/t_{i \ldots i+m+1}] < k \ (0 \leq k < m : m = pattern\ length)$

The approach makes possible to find out mutated presences but this gain also cause computational weight to the matching process. For reducing this weight, approximate matching algorithms should have very efficient skip algorithms. On the other hand, producing skip algorithm for approximate pattern matching algorithms harder than exact algorithms because skipped part could contain possible matches unlike exact approaches. To solve this problem usually skip algorithms does pre-processing on pattern, text or both of them.

Approximate pattern matching approaches compare pattern and text characters one by one until mismatch counter reach to the threshold or overall characters of the pattern has been compared. If mismatch counter exceeds the threshold the text shifts one character. On the other hand, if does not exceed the threshold after all characters have been matched, that means there is a match on current position. In other words, naive search using hamming distance to decide matching occurred on current position or not. If distance is under threshold there is a match or exceeding threshold is not. There is no skip mechanism in naïve search that means naïve search is a linear brute force matching algorithm but still useful small patterns and sequences due to no needs pre- process on neither text nor pattern.

An efficient approximate matching algorithm which is Burrows Wheeler transform firstly developed for data compression but nowadays there are many usage areas like pattern matching and sequence alignment. The basic idea behind BWT is produce the permutations of the characters of text and positioning closely to similar contexts. That means in approximate matching, k mismatched contexts can be found in k neighborhoods. This process increases efficiency of approximate matching but on the other hand, pre-process on long patterns takes long execution times. Exhaustive explanations and detailed example could be found on.

Approximate BOM

In this study, we present the approximate version of Backwards Oracle Matching algorithm. Recall that approximate pattern matching enables to find very similar presences of pattern on text. The flexible matching approach extends scope of matching but the profit comes with computation weight because of permutations of the pattern. In general, approximate search algorithms are slower by nature. Especially matching takes huge execution times on long patterns. To overcome the problem, any pre-process should be done on pattern before matching.

BOM algorithm is an automat based exact pattern matching algorithm as mentioned above. The algorithm offers an automat for permutation problem. The automat provides how many shifts performs any location on mismatch. The automat accelerates the matching process because shift counts for all permutations have already calculated. From this idea, the automat-based approach could be applying on approximate pattern matching. The novel A-BOM algorithm is approximate variation of Backward Oracle Matching algorithm. BOM algorithm is best fit when long pattern searching case because all suffix combinations (factors) are calculated before search process and factor automaton prepared for search process. That means when any mismatch occurs on any position, search already know to how many shifts are necessary. Therefore, like BOM algorithm, approximate BOM algorithm is supposed to be powerful on long pattern search.

Approximate BOM algorithm using same automata logic and matching function with BOM algorithm can be found on. Approximation feature provided on calculating match score of current sub sequences. Unlike BOM, the algorithm doesn't skip current position on mismatch until error counter is under threshold. When any mismatch occurs as long as error counter under threshold, matching branch out sub matching process by all transitions of current state.

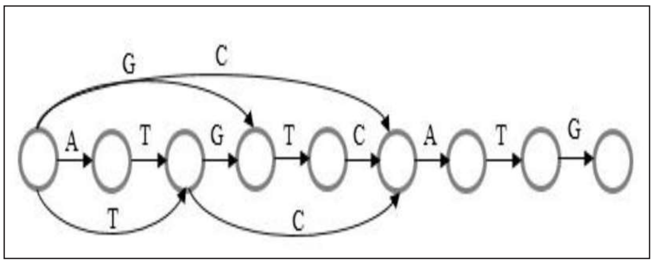


Figure 1.Factor Oracle Automat of the Pattern P=GTAACTGTA

Let's suppose there is a pattern like P=GTACTGTA. The automata of reversed pattern shown in Figure 1.

On the other hand, let assume that also there is a sequence T= GTACTTTA. Let's suppose that the threshold is 3. The score function performs matching from end to begin due to Boyer Moore characteristics. When the score function come at third letter, the letter T is not match with the third character of pattern G. The approximation mechanism step in and branching starts at position 3. The root process branches out four sub matching process because of alphabet consist of four letter which A, T, G and C. The branching shown in Figure 2.

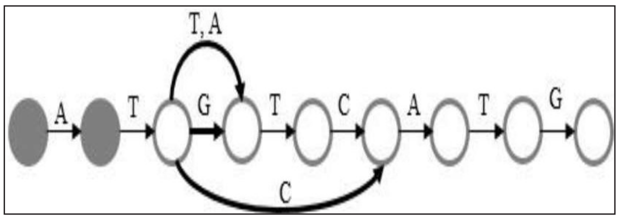


Figure 2.Branching on the Mismatch Third Character

The sub processes perform matching after mismatch location and they can branch out as long as error doesm't reach up to threshold. Therefore, matching score function designed as recursive. Branches go on matching with related transition of current state. There is a significant detail on the transitions. If there are no transition or the transition offers to jump over left error tolerance, branch go on matching with state of next expected character on the pattern. After all branches done of any parent process, largest matching score of branches adds parent's score and this adding process continues until the root matching process.

After all branches of root process's done, function returns the matching score to matching function. The matching function announces there is a match on current position when the matching score equals to pattern length. On the other hand, if they are not equal, skips the matching location as much as subtraction of pattern length and matching score. Pseudo code of match score function explained in Algorithm 1.

Experimental Result

In this section we introduce experimental performance comparison results of our approximate matching algorithm against Barrows Wheeler and Naive hamming distance based approximate matching algorithms. All the experiments we perform on a computer, with an Intel i3, 2.30 GHz CPU with 8 GB of RAM and running Ubuntu 19.10, 64-Bit. The code was written in C and compiled with Geaney IDE.

#Score Function

```
WHILE index > 0 and current_state != length of automat
IF sequence[index] in current_state move to next state
```

```
Else Break End While
IF index > 0 and error_counter < threshold FOR EACH transition in alphabet
IF (next_state_transition current_state + error_counter) <= threshold and next_state_transition is not null
error counter += next_state current state
append Score(next_state_transition) to temp_indexes
Else
append Score(next_expected_state) to temp_indexes
index =min(temp_index) RETURN index
```

Algorithm 1: Match score function algorithm

Test Sequence has 50K nucleotide and pattern lengths are variable. All experiments are repeated 100 times and averaged results are collected. In Table 1, execution times of algorithms are represented. Short names in Table 1, used as, A-BOM for Approximate Backward Oracle Matching, BWT for Burrows Wheeler Transform and NAIVE for Hamming Distance based approximate string matching.

Table 1.Average Running Time for 50K Length DNA Sequence

Pattern Length	Error Rate	A-BOM	BWT	NAIVE
5	1	0. 084729	3.896761	0. 052436
10	1	0. 037810	3.867000	0. 046877
15	1	0. 031255	3.838748	0. 046883
25	1	0. 015626	3.932374	0. 053443
50	1	0. 002548	3.983852	0. 062507
5	2	0. 226232	3.821203	0. 062556
10	2	0. 084656	4.098129	0. 069030
15	2	0. 037353	3.824674	0. 068765
25	2	0. 022149	4.027949	0. 062505
50	2	0. 015628	3.898494	0. 069025
10	5	0. 268755	4.067687	0. 099815
25	5	0. 115907	3.974571	0. 099862
50	5	0. 052977	4.229776	0. 099817
75	5	0. 046839	4.139655	0.100351
100	5	0. 031255	4.007760	0. 100320

The results on the Table 1, presents execution time of algorithms in seconds. The observations donate A-BOM algorithm performance increasing with pattern length progressively unlike naïve matching algorithm. On the other hand, performance of BWT algorithm doesn't show significant variance on different patterns lengths.

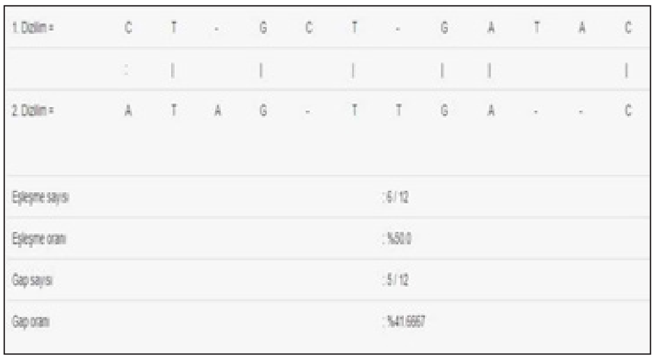
The result denote that, A-BOM algorithm yields performance result on long patterns. Results of Table 1, denotes that highest performance improvement occurs when the pattern length is 50 or 100. In general, observations donate the algorithm has from 30% to 80% better performance when pattern lengths over 10. Table 1, concludes that A-BOM is slower than naïve algorithm on short patterns, but still donates reasonable execution time. Increasing error rate influence unfavourably all algorithms. Our algorithm is affected than high error rates because of the branching characteristic.

In summary, A-BOM yields efficient approximate string matching for long patterns. Since pattern search on long DNA sequences is common, our algorithm can make sense for DNA sequences that contain mutations.

Web Interface

The web interface can accessible on https://github.com/burakkoca/BioLab address. In the interface, sequence alignment, pattern matching and motif finding can be easily done with user friendly graphic interface. Interface supports big amount of data. That means the interface can be used for academic and research projects. Students can use learning major solutions and try on own datasets also compare with several algorithms for the best fit solution. All algorithms which mentioned in proposal are presented in the interface. There are Smith-Waterman and Needleman Wunsch algorithms for sequence alignment. For exact pattern matching KMP, Boyer-Moore and BOM algorithms are available and BWT, naïve search and A-BOM presented to perform approximate pattern matching. Motif finding can be done with greedy algorithm. How to use the interface introduced in separated subheadings for all solutions.

Sequence Alignment. The Sequence alignment algorithms can reachable sequence alignment collapsible item on left menu. Both KMP and Boyer-Moore algorithms have same interface for alignment. There are five field that has been labelled for sequences, gap, match score and mismatch score. After all fields press align button for alignment. Note that, sequence length must be under ten thousand and must be consist of nucleotide letters.



1. Dizilim =	C	T	-	G	C	T	-	G	A	T	A	C
	:	|		|		|		|	|			|
2. Dizilim =	A	T	A	G	-	T	T	G	A	-	-	C

Eşleşme sayısı	:6 / 12
Eşleşme oranı	:%50.0
Gap sayısı	:5 / 12
Gap oranı	:%41.6667

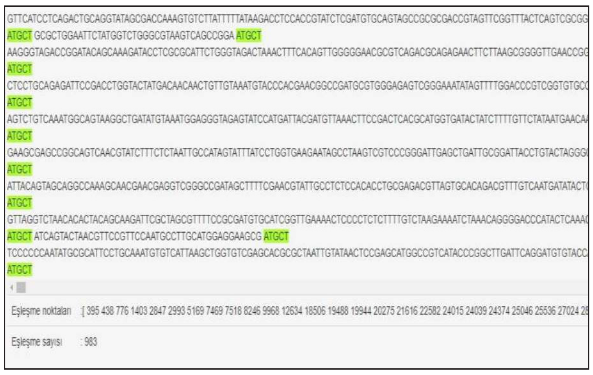
Figure 3.Sequence Alignment Results Page

When a query is given on the interface, results are returned in another page. Aligned string, gap score, gap ratio, match score, match ratio introduced in that page. In Figure 3 first row presents aligned sequence. In Aligned sequence string, dashes stand for gaps and rods represent matches. Second row coloured and presents match, mismatch and gap statics.

Pattern Matching

The Pattern search algorithms can reachable "Exact Pattern Search" and "Approximate Pattern Search" collapsible items on left menu. Both exact and approximate matching pages have same interface for all algorithms but approximate pattern matching page has threshold field additively. The fields introduce with related parameter of matchin. After all fields filled press "Match" button for pattern matching. Note that, for only pattern matching sequence length could reach up to one million.

Pattern matching results are presented in consecutive web page. In other words, sequence, match points, and match count presented in result page. In Figure 4, First row on the page presents treated sequence with coloured presences of pattern. Second row represents matched locations and last row demonstrate match count.

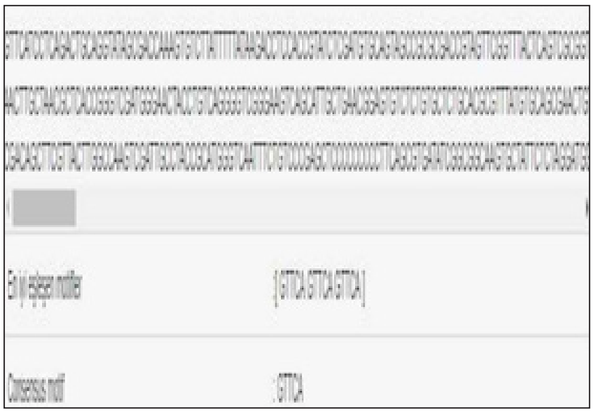


```
GTTCATCCTCAGACTGCAGGTATAGCGACCAAAGTGTCTTATTTTTATAAGACCTCCACCGTATCTCGATGTGCAGTAGCCGCGCGACCGTAGTTCGGTTTACTCAGTCGCGG
ATGCT GCGCTGGAATTCTATGGTCTGGGCGTAAGTCAGCCGGA ATGCT
AAGGGTAGACCGGATACAGCAAAGATACCTCGCGCATTCTGGGTAGACTAAACTTTCACAGTTGGGGGAACGCGTCAGACGCAGAGAACTTCTTAAGCGGGGTTGAACCGG
ATGCT
CTCCTGCAGAGATTCCGACCTGGTACTATGACAACAACTGTTGTAAATGTACCCACGAACGGCCGATGCGTGGGAGAGTCGGGAAATATAGTTTTGGACCCGTCGGTGTGCC
ATGCT
AGTCTGTCAAATGGCAGTAAGGCTGATATGTAAATGGAGGGTAGAGTATCCATGATTACGATGTTAAACTTCCGACTCACGCATGGTGATACTATCTTTTGTTCTATAATGAACAA
ATGCT
GAAGCGAGCCGGCAGTCAACGTATCTTTCTCTAATTGCCATAGTATTTATCCTGGTGAAGAATAGCCTAAGTCGTCCCGGGATTGAGCTGATTGCGGATTACCTGTACTAGGGC
ATGCT
ATTACAGTAGCAGGCCAAAGCAACGAACGAGGTCGGGCCGATAGCTTTTCGAACGTATTGCCTCTCCACACCTGCGAGACGTTAGTGCACAGACGTTTGTCAATGATATACTC
ATGCT
GTTAGGTCTAACACACTACAGCAAGATTCGCTAGCGTTTTCCGCGATGTGCATCGGTTGAAAACTCCCCTCTCTTTTGTCTAAGAAAATCTAAACAGGGGACCCATACTCAAAC
ATGCT ATCAGTACTAACGTTCCGTTCCAATGCCTTGCATGGAGGAAGCG ATGCT
TCCCCCCAATATGCGCATTCCTGCAAATGTGTCATTAAGCTGGTGTCGAGCACGCGCTAATTGTATAACTCCGAGCATGGCCGTCATACCCGGCTTGATTCAGGATGTGTACCA
ATGCT

Eşleşme noktaları   :[ 395 438 776 1403 2847 2993 5169 7469 7518 8246 9968 12634 18506 19488 19944 20275 21616 22582 24015 24039 24374 25046 25536 27024 28

Eşleşme sayısı      : 983
```

Figure 4.Pattern Matching Results Page



```
[illegible]
[illegible]
[illegible]

En iyi eşleşen motifler        [ GTTCA GTTCA GTTCA ]

Consensus motif                : GTTCA
```

Figure 5.Motif Finding Results Page

Motif Finding

Motif finding algorithm can reachable "Motif Finding" collapsible item on left menu. There are two input for motif finding on the web page. The first input takes sequence and second one for motif length. Multi sequences could be using for motif finding by separating sequences with comma.

After all fields filled, press " Find Motif" button for motif finding.

Founded motifs and consensus motif presented in Results page that shown in Figure 5. Treated sequence stay on the first row of results. Second row presents founded motifs and last row show us the consensus motif.

Comparison

Each solution group have own "Compare" tab in collapsible menus. Comparison pages have same interface with related solution page. Comparison results pages are same for all solutions. The results introduced in a table which each comparison parameter heading for each algorithm.

Conclusion

In this study I introduced a useful interface for all major bioinformatics problems solution algorithms. The interface differs from variations with wide scope. From educational to scientific purpose, any people who interested in bioinformatics can take the advantage of the interface because of the extensive content's opportunity. Also, the interface offers to execute algorithms with large amount of data opportunity in free form. On the other hand, to the best of our knowledge there is no national interface that provides pattern matching, sequence alignment and motif finding for bioinformatics field and this study fulfill the need. We believe that the study nourishes bioinformatics studies in our country and worldwide.

The second contribution of this study is a novel approximate string-matching algorithm which presents best performance for long patterns. Experimental results show that approximate approach of BOM speeds up approximate matching on long patterns. My solution yields up to 80% better performance compared to Burrows Wheeler and Hamming Distance approach if pattern length is longer than 10. The results may contribute to the recent bioinformatics researches. For example, A-BOM may fit better for approximate matching problems like error correction or merging read data from new generation DNA sequencing methods like Nanopores. In summary, the algorithm can be used for tolerant pattern matching with long patterns.

References

1. Pevsner J. Bioformatics and fuctional genomics.
2. Smith TF, Waterman MS. Identification of comman molecular subsequences. *Journal of molecular biology.* Academic Press Incorporated, London, 40-48. doi: 10. 1016/00222836(81) 90087-5.
3. Needleman SB, Wunsch CD. A general method applicable to the search for similarities in the amino acid sequence of two protiens. *Journal of molecular biology.* Academic Press Incorporated, London. 1970; 40-48. doi: 10. 1016/00222836(81) 90087-5.

4. Bishop CM. Machine learning and pattern recognition. Information science and statistic springer, Heidelberg.
5. Dhaeseleer P. How does DNA sequence motif discovery work? *Nature biotechnology* 2006; 24(8): 959-961.
6. Ozcan G,Unsal OS. Fast bitwise pattern matching algorithm for DNA sequences on modern hardware. *Turkish Journal of Electrical Engineering & Computer Sciences* 2015; 23(5): 1405-1417.
7. Langmead B, Salzberg SL. Fast gapped read alignment Bowtie 2. *Nature Methods* 2012; 9(4): 357.
8. Knuth DE, Morris JH, Pratt WR. Fast pattern matching in Strings. *Journal of Molecular Biology*, *SIAM Journal on Computing* 1977; 323-350. DOI: 10.1137/0206024
9. Boyer RS, Moore JS, Pratt WR. A fast string searching algorithm. J*ournal of Molecular Biology* 1977; 762-772.