ijecs
open access www.ijecs.in
International Journal Of Engineering And Computer Science ISSN:2319-7242
Volume 2 Issue 7 (July 2013), Page No. 2139-2144

Devise, Extend and Upgraded Algorithm of Gene Sequence for Local Alignment

Abu Sadat Mohammaed Yasin[1], Md. Majharul Haque[2], Kollol Naha[3], Md. Shakil Ahamed Shohag[4]

[1]Shajalal University of Science & Technology, Department of Computer Science & Engineering,
Sylhet, Bangladesh
abusadatyasin@gmail.com

[2]University of Dhaka, Department of Computer Science & Engineering,
Dhaka, Bangladesh
mazharul_13@yahoo.com

[3]Shajalal University of Science & Technology, Department of Computer Science & Engineering,
Sylhet, Bangladesh
kallolnaha@gmail.com

[4]University of Development Alternative, Department of Computer Science & Engineering,
Dhaka, Bangladesh
shakilshohag@email.com

Abstract: ***Finding similar regions from two strings or nucleotides or protein sequences is very much desirable for determining the functional similarity between them. In the ground of bioinformatics, for determining analogous constituency between two sequences, sequence-alignment can be used, which is the way of arranging sequences. This can also be helpful for non-biological field such as natural language processing or financial data. Finding out the larger sequence from the dissimilar sequences those are suspected to contain regions of similarity motifs within their long sequence context, local-alignment (maximum length exact matching) is very useful which mostly works on identifying the best pair of regions. In this research work an innovative method is proposed for searching bio-sequences/gene-sequences for the local alignment. This paper also provides an evaluation of the proposed algorithm and turns a black and white comparison with one of the popular existing methods and a modified version of the existing method. The evaluation result shows that the existing method is very time and space/memory consuming where the promising approach of projected technique is to seek out the identical sequences by taking less computational time and using less memory space. Therefore, we have faith that the new era of searching similar region from sequences is going to be raised for this proposed innovative method.***

Keywords: Functional similarity, analogous constituency, sequence-alignment, local-alignment, gene-sequence.

1. Introduction

Local-alignment [7], [8] discovers the exact matching between two Gene-sequences [1]-[4]. Generally Smith-Waterman algorithm [5], [9] is used for local-alignment [7], [8] searching. Usually Gene-sequences are very large and when the sequences are over sized they take much long time and memory to find out the result/search or result/matching part [4]. The ultimate goal of our paper is to reduce both the time and space/memory complexity [5], [6] of the Smith-Waterman Algorithm [5], [9].
Md. Shihabuddin Sadi et al [12] in 2009 worked on Smith-Waterman algorithm [5], [9] to reduce time complexity, where they proposed a new algorithm and they claimed successes in terms of time but not in space/memory. In this paper, main target is to reduce time (make much faster than the modified algorithm [12]) and space/memory both so that the searching technique can be more efficient and fast.

2. Smith-Waterman Algorithm [5], [9]

This algorithm works for local-alignment [7], [8]. For each cell of the matrix computation it has to check the upper position, left position, and upper-left diagonal position. So the computation depends on previous three cells. It has to compute whole matrix of dimension (M, N) for all time and its best, average, and worst case time complexity [5], [6] are O (MN). So, it is very time consuming algorithm. Its space/memory complexity [5], [6] is also very high, O (MN). Here is the description of Smith-Waterman Algorithm [5], [9] with examples depicted in table 1:

(i) If two sequences AACCTATAGCT & GCGATATA with length of M & N, declare a matrix with (M+1 x N+1) dimensions and assigns 0 to the first row and column.

(ii) Start searching from cell (2, 2) position of the matrix for char. by char. matching.

(iii) If a match found than add a bonus point (+1) with its upper left diagonal positions' value.

(iv) If no match is found than find out the max value from its' upper, left and upper left diagonal position and subtract a penalty (1).

(v) In this way compute the whole matrix and store, then update the max value and from the max value find out the maximum matches found or obtained. Keep track of the direction from which a score was derived.

(vi) The result of this example is "TATA".

Table 1: The Matrix calculations for Smith-Waterman Algorithm

		A	A	C	C	T	A	T	A	G	C	T
	0	0	0	0	0	0	0	0	0	0	0	0
G	0	0	0	0	0	0	0	0	0	1	0	0
C	0	0	0	1	1	0	0	0	0	0	2	1
G	0	0	0	0	0	0	0	0	0	1	0	1
A	0	1	1	0	0	0	1	0	1	0	0	0
T	0	0	0	0	0	1	0	2	1	0	0	1
A	0	1	1	0	0	0	2	1	3	2	1	0
T	0	0	0	0	0	1	1	3	2	2	1	2
A	0	1	1	0	0	0	2	2	4	3	2	1

3. Background

In the year 2009 Md. Shihabuddin Sadi et al [12] worked on the Smith-Waterman algorithm [5], [9] and they proposed an upgraded Smith-Waterman algorithm with less time complexity [5], [6].

According to the previous example here the procedure of their upgraded algorithm [12] is described below:

(i) If a mismatch in any position of the table is found, put a zero in that position and proceed to the next horizontal position of the table. If there is no horizontal position remains, proceed to the first position of the next row. No comparison will be done if a position is filled previously.

(ii) If a match in any position of the table is found, put a +1 value in that position and define it as a root and proceed to the next diagonal position of the table. If another match is discovered, the score will be added with the last diagonal position's score. By this way, the process will be continued until getting a mismatch.

(iii) When a mismatch is observed or if the table ended while continuing diagonal comparison, set the last non-zero value as maximum number and go back to the root.

(iv) If the maximum number is L, select L-1 rows and L-1 columns from the both end. No comparison will be started from these selected areas. Only the diagonal comparison value will be performed in these areas.

(v) After going back to the root, the next horizontal position of root will be the new start position.

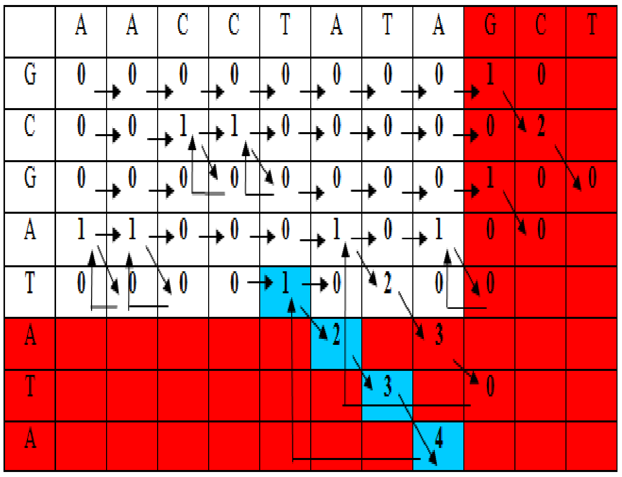
(vi) If another maximum number is found out by this way, the greater maximum number will be taken.

(vii) After finishing traversing by this way, the last maximum value and its' diagonal chain are picked to the root. This will be the best locally aligned sequence between two sequences.

(viii) The Result of this example is "TATA".

The full process is portrayed in the table 2.

Table 2: The matrix calculations for the upgraded algorithm



	A	A	C	C	T	A	T	A	G	C	T
G	0	0	0	0	0	0	0	0	1	0	
C	0	0	1	1	0	0	0	0	0	2	
G	0	0	0	0	0	0	0	0	1	0	0
A	1	1	0	0	0	1	0	1	0	0	
T	0	0	0	0	1	0	2	0	0		
A						2		3			
T							3		0		
A								4			

According to this algorithm its' best case complexity [5], [6] is **Ω (N),** average case complexity is **O(N(M+1)/2),** and worst case complexity is **O(MN).** Its space or memory complexity is **O(MN)**.

4. Proposed Model

An innovative algorithm is proposed here which is much faster in practically from both the Smith-Waterman algorithm [5], [9] and the upgraded algorithm [12]. The space complexity [5], [6] is also reduced from O(MN) to O(max(M,N)). Here is our proposed technique with the same example:

a) At first the matrix starts for matches from the first diagonal starting from position (1, 1), than advances towards to its diagonal way (the colored marked diagonal is shown in table 3).

Table 3: Working diagonal path of the matrix

	A	A	C	C	T	A	T	A	G	C	T
G	🟦										
C		🟦									
G			🟦								
A				🟦							
T					🟦						
A						🟦					
T							🟦				
A								🟦			

b) If a mismatch in any diagonal position of the table is noticed, put a zero in that position and proceed to the next position of the diagonal of the table. If there is no diagonal position remains, than starts for next one from its upper side diagonal way and second from its lower side diagonal way and again third from its upper side diagonal way and forth from its lower side diagonal way and so on. In the table 4, it is exposed that only colored marked numbers indicate the sequences of diagonals need to be checked.

Table 4: The corresponding numbers indicate the way to check diagonal

1	2	4	6	8	10	12	14	16	18	20	22	24
3		A	A	C	C	T	A	T	A	G	C	T
5	G											
7	C											
9	G											
11	A											
13	T											
15	A											
17	T											
19	A											

c. If a match in any position of the table is found, put +1 value in that position and proceed to the next diagonal position of the table. If another match is occurred, the score will be added with the previous diagonal position's score. By this way, the process will be continued until getting the end of the diagonal as in table 5. If a mismatch is raised after a score than put zero (0) in this position and start counting from 1 for finding next match.

Table 5: Scoring or counting matches

	A	A	C	C	T	A	T	A	G	C	T
G	0										
C		↘0									
G			↘0								
A				↘0							
T					↘1						
A						↘2					
T							↘3				
A								↘4			

d) Last non-zero maximum value is traced as the maximum value of the matches' string.

e) If the maximum number is L, it will not access the last L-1 length indexes of diagonal and L-1 indexes of columns. No comparison will be started from these selected areas. Only the diagonal comparison value will enter in these areas if its immediate previous position has a non-zero value (colored shown in table 6).

Table 6: Reducing computation

	A	A	C	C	T	A	T	A	G	C	T
G	0	0									
C	0	↘0	↘1								
G		↘0	↘0	↘0							
A			↘0	↘0	↘0						
T				↘0	↘1	↘0					
A					🟦	↘2	🟦				
T						🟦	↘3	🟦			
A							🟦	↘4	🟦		

f.If another maximum number is exists by this way, the greater maximum number will be taken.

g. After traversing by this way, the last maximum value and its diagonal chain are picked. This will be the best locally aligned sequence between two sequences.

h. Keep track of the direction from which a score was derived.

i. From this table it is determined that the maximum length of 4 and the substring is "TATA". Here it works for local-alignment [7], [8]. Table 7 shows the complete computations. The colored cells those are indicated in table 7 do not need any comparison or computation.

Table 7: Complete table after full computation

	A	A	C	C	T	A	T	A	G	C	T
G	0	0	0	0	0	0	0	0	🟦	🟦	🟦
C	↘0	↘0	↘1	↘1	↘0	↘0	↘0	↘0	🟦	🟦	🟦
G	↘0	↘0	↘0	↘0	↘0	↘0	↘0	↘0	🟦	🟦	🟦
A	↘1	↘1	↘0	↘0	↘0	↘1	↘0	↘1	🟦	🟦	🟦
T	↘0	↘0	↘0	↘0	↘1	↘0	↘2	↘0	↘0	🟦	🟦
A	🟦	🟦	🟦	🟦	🟦	↘2	🟦	↘3	🟦	🟦	🟦
T	🟦	🟦	🟦	🟦	🟦	🟦	↘3	🟦	🟦	🟦	🟦
A	🟦	🟦	🟦	🟦	🟦	🟦	🟦	↘4	🟦	🟦	🟦

5. Computational Complexity [5], [6] of Proposed Algorithm, Smith Waterman Algorithm [5], [9] and Modified Algorithm [12]

If the length of the two sequences are M and N, (M >=N)

Table 8: Complexity analysis in theoretically or mathematically

Computational Time Complexity	Our Proposed Modified Smith-waterman Algorithm	Modified Smith-waterman Algorithm [12]	Smith Waterman Algorithm [5] [9]
Best Case:	Ω(N)	Ω(N)	O(MN)
Worst Case:	O(MN)	O(MN)	O(MN)
Average Case:	(N (M+1)/2)	(N (M+1)/2)	O(MN)

5.1 Calculation of Average Case

Average case =N*1/(MN-N+1)+(N+1)*1/(MN-N+1)+(N+2)*1/(MN-N+1)+(N+3)*1/(MN-N+1)+…………+(MN)*1/(MN-N+1)

$=1/(MN-N+1)\ (N+(N+1)+(N+2)+(N+3)+\ldots+MN)$
$=1/(MN-N+1) * (MN-N+1) * (N(M+1)/2)$
$= (N(M+1)/2)$

5.2 Space Complexity

In our method, about a matrix is thought but in practical code/program it is not needed to create matrix or two-dimensional array. Always compute value from the diagonal of the matrix and each cell value depend on the current diagonal' previous cell and current cell doesn't need any other diagonals' cell value. So, here only requires enough space to store the diagonal it is currently working on and which can be done using a single array. Assuming the longer sequence has length N, the space complexity [5], [6] of the proposed algorithm is O(N).
Thus, space complexity is reduced [5], [6] from O (MN) to O (N).

Table 9: Space or Memory complexity table

	Our Proposed Modified Smith Waterman Algorithm	Modified Smith-waterman Algorithm [12]	Original Smith Waterman Algorithm [5] [9]
Space complexity	O(N)	O(MN)	O(MN)

6. Result Analysis

In the following table 10 shows how many computations take these three algorithms. This table shows only three small examples' computations data.

Table 10: No of computations needed for some small examples

Examples	Our Modified Algorithm	Modified Algorithm[12]	Smith-Waterman Algorithm[5][9]
AACCTATAGCT and GCGATATA	45	55	88
ACGTGCATT and CTGTGCCAT	41	44	81
ATGCACTGC and ATGC	10	10	36

6.1 Practically Time Complexity Analysis and Graph

For practical calculation some raw data is taken (example: agttagtggagatcacacgacgtgctaggacttacatctgccctaggctgcagactacca ttagagagacgctactgccaacattataggcactgatgtaactcatggtacatccgtcgct gagcgccattttgttacgtcacctggctggaacgtcgtccacaggaaacatcggcccacg ccggtactccatacgcttgaccacatacctgcaatcgccgagggccgggtcatggaaaa aacacgagttgtaattgcttatatagtaccgcagcgaggtactgtcatccagggcatttagg gcggcgcctgagagagggctgcagcgtccggtgggtcgcgctagtggcgctatctctc cgttggtgctttgatgacattgaattaggcctggcctggagcagttgaattgcatgaattgg catgtcgttacaggacacaaggccgtcagaagtcccttacttggtgtagattaccgcatta actgtatggatttctgacgttcgtttctccatatcgacgcggctcagtgtctgggtcgattctc cctccaaagccgtagcctttaaaccaaccacctggatctgcgttcctgcgtgc).
As like gene-sequence [4], we apply those data on the proposed modified algorithms, upgraded algorithm [12] and the original Smith-Waterman algorithm [5], [9] Visual C++ is used for implementation of these three algorithms. After applying, it is found that in average case and best case proposed algorithm gives better performance. But for worst-case (where no match of any character is found) proposed algorithm give a little lower performance than upgraded algorithm [12]. From the implementation of these three algorithms, collected data is presented below with tabular-form and graph.

6.2 Best Case Time Analysis (Practically)

Table 11: Best case time data analysis

Input type (1000 inputs with length (N) character)	Smith Waterman Algorithm [5] [9] (Time ms)	Modified Smith-waterman Algorithm[12] (Time ms)	Our Proposed Modified Smith-waterman Algorithm (Time ms)
N=100	250	31	31
N=200	812	62	62
N=300	1953	109	109
N=400	3406	140	140
N=500	5265	156	156
N=600	7564	203	203
N=700	10312	234	218
N=800	13218	265	265
N=900	16562	312	281

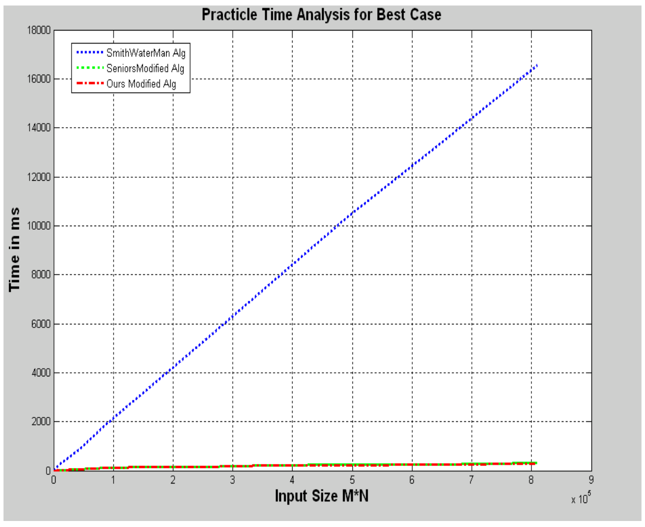


Figure 1: Best case time curve

6.3 Average Case Time Analysis (Practically)

Table 12: Average case time data analysis

Input type (1000 inputs with n length character)	Smith Waterman Algorithm [5] [9] (Time ms)	Modified Smith-waterman Algorithm[12] (Time ms)	Our Proposed Algorithm (Time ms)
100	750	265	187
200	3062	953	671
300	6718	2156	1375

400	11968	4000	2421
500	18484	6156	3828
600	26406	8921	5468
700	35609	12093	7453
800	46500	15703	9765
900	58641	19890	12390

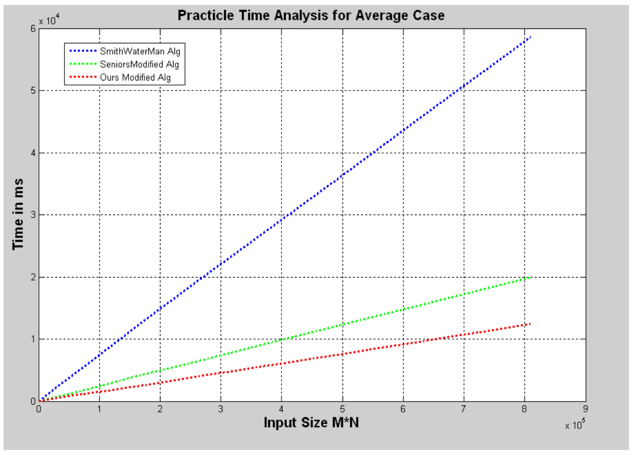


Figure 2: Average case time curve

6.4 Worst Case Time Analysis (Practically)

Table 13: Worst case time data analysis

Input type (1000 inputs with n length character)	Smith Waterman Algorithm [5] [9] (Time ms)	Modified Smith-waterman Algorithm[12] (Time ms)	Our Proposed Algorithm (Time ms)
100	765	93	140
200	3078	406	453
300	7093	793	937
400	12812	1359	1593
500	19765	2058	2406
600	28343	2937	3453
700	38421	3859	4718
800	50125	5156	6031
900	63656	6515	7500

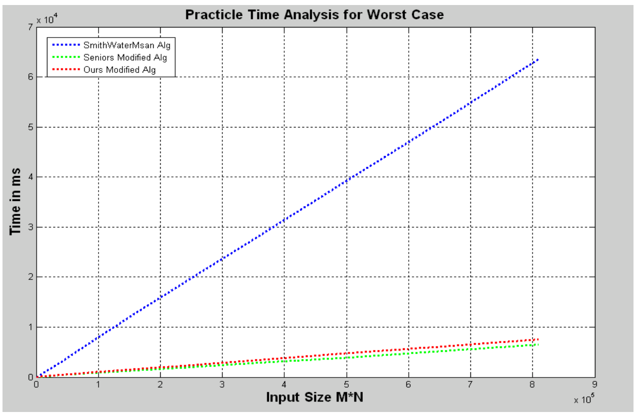


Figure 3: Worst case time curve

According to these collected data and analysis, it is apparently observed that the proposed algorithm performs much better and much faster than both Smith-Waterman algorithm [5], [9] and upgraded algorithm [12] to find out or comparing a gene-sequence from a genome database.

6.5 Practically Space Complexity Analysis and Graph

For practical calculation the space/memory complexity is done by the topic "complexity of algorithm" [5], [6]. It is clear that in average case, worst-case (that means number of similarities is so small) and best case proposed algorithm gives better performance for memory.

Table 14: Memory taken by these three algorithms

Input type (1000 inputs with n length character)	Smith Waterman Algorithm [5] [9] (Space in Mb)	Modified Smith-waterman Algorithm[12] (Space in Mb)	Our Proposed Algorithm (Space in Mb)
100	38.204193	38.204193	0.476456
200	152.626038	152.626038	0.952911
300	343.265533	343.265533	1.429367
400	610.122681	610.122681	1.905823
500	953.197479	953.197479	2.382278
600	1372.489929	1372.489929	2.858734
700	1868.000031	1868.000031	3.335190
800	2439.727783	2439.727783	3.811646
900	3087.673187	3087.673187	4.288101

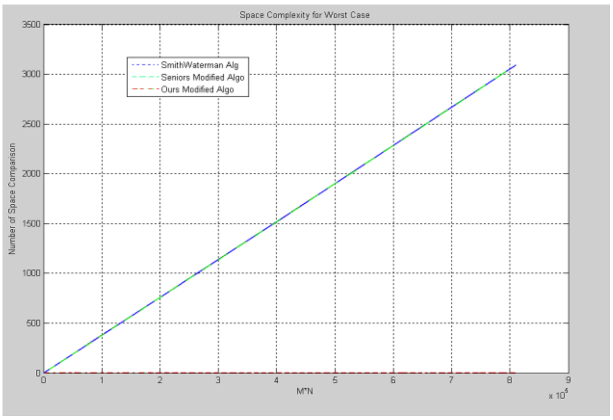


Figure 4: Full curve of space or memory complexity of these three algorithms

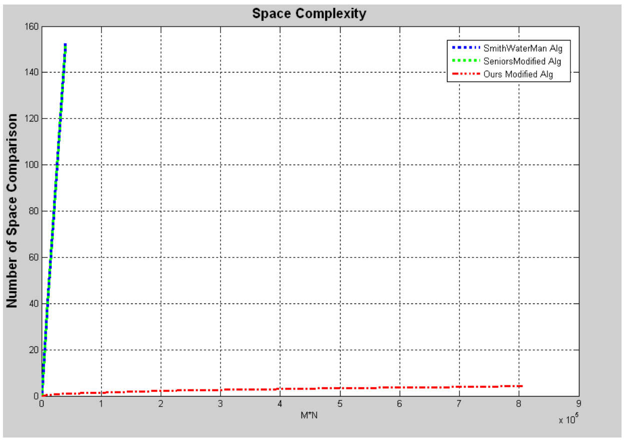


Figure 5: Partial curve of space or memory complexity of these three algorithms (for better understanding)

7. Conclusion

In bio-informatics there are a huge number of papers for decreasing the time and space complexity but increasing the accuracy of sequences matching and reducing computational complexity is still a great challenge; because existing algorithms also give better performance, such as heuristic tool like FASTA (a protein sequence alignment software package) partially uses the Smith-Waterman Algorithm. The computational complexity of Smith-Waterman Algorithm for local-alignment sequence matching is relatively high as (MN). But our proposed algorithm reduces that computational complexity to (N (M+1)/2). The space complexity of Smith-Waterman Algorithm is O (MN) but our proposed algorithm reduces that space complexity to (N). The major analysis of this research based on primary structure. But we believe that anyone can also get help for further research on secondary and tertiary structure for deriving new procedure besides using this proposed algorithm in real life.

References

[1] S. C. Rastogi, Namita Mendiratta, Parag Rastogi, S.C. Rastogi, Jeremy Strong, “Bioinformatics-methods and Applications”, Paperback, Prentice-Hall Of India Pvt.Ltd, ISBN-10: 81-203-2582-6 / ISBN-13: 978-81-203-2582-1, 2004.

[2] David W. Mount, “Bioinformatics: Sequence & Genome analysis”, 2nd Ed. Cold Spring Harbor Laboratory Press,U.S, 2004.

[3] Andreas D. Baxevanis and B. F. Francis Ouellette, “Bioinformatics: A Practical Guide to the Analysis of Genes and Proteins”, 2nd Ed. John Wiley & Sons Inc, 2001.

[4] Dan E. Krane, Michael L. Raymer, “Fundamental Concepts of Bioinformatics”, Publisher: Benjamin Cummings, Bookseller: HPB-Ohio, Columbus, OH, U.S.A., ISBN: 0-8053-4633-3, 2002.

[5] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein, Introduction to Algorithms, 2nd Ed. MIT Press, 2001.

[6] Seymour Lipschutz, “Schaum's Outline of Theory and Problems of Data Structures”, Mcgraw-Hill, 1986.

[7] Jelena Prokić, Martijn Wieling and John Nerbonne, “Multiple sequence alignments in linguistics,” Proceedings of the EACL 2009 Workshop on Language Technology and Resources for Cultural Heritage, Social Sciences, Humanities, and Education, pp. 18-25, 2009.

[8] S.F. Altschul, W. Gish, W. Miller, E.W. Myers, and D.J. Lipman, “Basic local alignment search tool,” Journal of Molecular Biology, vol. 215, no. 3, pp. 403-410, 1990.

[9] T. F. Smith and M. S. Waterman, "Identification of Common Molecular Subsequences," Journal of Molecular Biology, vol. 147, pp. 195-197, 1981.

[10] Hong Zhou, Yufang Wang and Xiao Zeng, “Fast and Complete Search of siRNA Off-Target Sequences,” Saint Joseph College, West Hartford, CT, USA. University of Southern Mississippi, Hattiesburg, MS, USA. Superarray Bioscience Corporation, Frederick, MD, USA, 2006.

[11] Kevin Greenan, “Method-Level Code Clone Detection on Transformed Abstract Syntax Trees Using Sequence Matching Algorithms,” Department of Computer Science, University of California Santa Cruz, 2005. [Online] Available: http://users.soe.ucsc.edu/~ejw/courses/290gw05/greenan-report.pdf (Accessed: May 31, 2013)

[12] Md. Shihabuddin Sadi, Abu Zafar Mohammad Sami, Imtiaz Uddin Ahmed, A. B. M. Ruhunnabi and Nirmalendu Das, “Bioinformatics: Implementation of a proposed upgraded Smith-Waterman Algorithm for local-alignment”, Proceedings of the 6th Annual IEEE conference on Computational Intelligence in Bioinformatics and Computational Biology, pp. 87-91, 2009.