A Time Efficient Approach for Quick Splitting and Finding Maximum Matches Sequences in Bio-sequence Analysis

Mohammad Hasan
Department of Computer Science and Engineering
Bangladesh Army University of Science and Technology

Muhammad Ibrahim Khan
Department of Computer Science and Engineering
Chittagong University of Engineering and Technology

Fatema Tuz Zohra,
Abu Saleh Musa Miah,
Ashrafun Zannat,
Md. Al Hasan,
Md. Mamunur Rashid
Department of Computer Science and Engineering
Bangladesh Army University of Science and Technology

ABSTRACT

DNA sequence analysis & comparison computation is a vital task in terms of memory & time which is used huge size of data set for biological research. Perfectly aligned sequence find out the matching point or mismatches between two sequences. Our proposed algorithm is composed of two major part. The first part is Fast Splitting(FS), a "Recursive technique" based algorithm which divides the source sequence in appropriate and exact length according to the preference of target sequence. Second part is Fast_Maximum Matches Subsequence Finder(Fast_MMSS). It builds the specialized successor table according to the identical characters of two strings (TLSS & Target Sequence). Then using some special pruning condition, we get the final MMSS. In previous work, dynamic programming and some sorts of Brute force techniques are applied which are faster in terms of time but requires huge memory, while our proposed algorithm maintains the 'Time and Space' tradeoff.

Keywords

Fast Splitting (FS), Maximum Matches Subsequences (MMSS), Target Length Splitted Sequence(TLSS), Successor Table.

1. INTRODUCTION

Bioinformatics is a cross field that retrieves information from the bio-sequences. Bio-sequence analysis leads to identify the similarities, matches or mismatches, alignments between various sequences. Sequence alignment or finding similarities between two sequences leads to identify the bio-logical identity of various species, drug design, and disease identification etc. For finding similarities, there are two types of alignment: Local and Global[13]. Local alignment firstly, takes a portion (substring) of the source sequence and then tries to match with the target sequence. But in Global alignment we take the whole source sequence and try to make an end to end comparison with the target sequence[14]. The sequences which are locally optimal, these are must be globally optimal. There are a lot of evolutionary methods used in various genome projects which can handle a large scale of biological data. Alignment of sequence is the initial state of every genome project. The common matching or similarities indicates the evidence of common genome similarity and homologous relationship. So the alignment of the sequence or finding maximum matched sequence is the vital part for genome projects. There are a lot of algorithms for sequence alignment which are used dynamic programming[16], pruning[17], parallel implementation techniques[18] and recursive backtracking techniques [12]. But using these techniques it is difficult to maintain the 'Time and Space trade-off'. Because dynamic programming reduces the time but takes huge memory for memorization technique.

First algorithm is the BLAST [1], They proposed a shared memory version which can easily compute the local optimality. But the main limitation is that it can only search for a single query. Needleman–Wunsch[2] proposed a dynamic programming based algorithm that is used to align the protein or DNA sequences. But the main limitation is that it can only compute the global optimality. Besides, it consumes considerable amount of memory and exponential time complexity is O(nm) where, n and m is the length of two sequence[10][11].

We also examined the Smith-Waterman [3] algorithm which can compute local alignment perfectly. But it can't perform global alignment. Time & space complexity same as Needleman-Wunsch. T-coffee[4] proposed a new dynamic algorithm that is used to align the sequence. But the time complexity of T-coffee is exponential[9] such as $O(N^3L)$, where N is the number of sequences and L is the average sequence length. ProbCons[5] proposed a posterior probability based scoring method for all sub-optimal alignment. It also takes exponential time such as $O(k^2n^2 + k^3)$time. Newberg [6] proposed a memory efficient method which is used dynamic programming technique for calculating the index of the optimal check point. But It also takes exponential time complexity of O(MN^3) or $O(N^4)$ and memory complexity of O(N^2M). where M is the no. of states and N is the no query sequences. RSAM [7] and another algorithm [15] are also proposed a dynamic programming based algorithm which can solve the local sequence alignment problem. It also takes a considerable amount of time.

Our proposed algorithm deals with some specific types of problem. These types of problem scenarios are discussed according to following way: Firstly, In bio-sequence analysis, In many cases, the input or source sequence length is very long which contains billions of base pairs. On the other hand, the target sequence length is so short where the number of base pair is not greater than 50. For this type of situation, we badly needed to divide or break the source or input sequence into the preference of the target sequence for further analysis such as aligning, matching or finding maximum matched sequence. So the first part of our algorithm is Fast Splitting(FS). FS breakes the huge sized input sequence into a feasible length according to the preference of the target. FS produces the Target Length Splitted Sequence (TLSS).

The second part of our proposed algorithm is Fast_MMSS Finder. In this part we take the TLSS and target sequence as input and find out the maximum matches sequence between each TLSS and target sequence. Then store the all maximum matched-sequence between every TLSS and target sequence. Sequence of alignment whole work we have various subsection such as system architecture, analytical representation, Complexity Analysis, Experimental set up, Experimental Evaluation.

2. METHODOLOGY

2.1 System Architecture

The system architecture of our proposed algorithm are consist with four layer. The integrated architecture started from input to the systems and stopped at finding Desired subsequence. Input Sequences may be any DNA, RNA and proteins. We have imposed our system for DNA data shows in Fig.1: 'Fast_Splitter' is used to split the input sequence into desired size.'Splitted Sequence store' stores the output of Fast_Splitter which is Target Length Splitted Sequence(TLSS).'Fast_MMSS Finder' is used to detect the matching point between TLSS & T. 'Aligned sequence store' is finally stored the desired result. Our initial input is two sequences. These are: Input or Source Sequence (S) and Target or Reference Sequence(T).

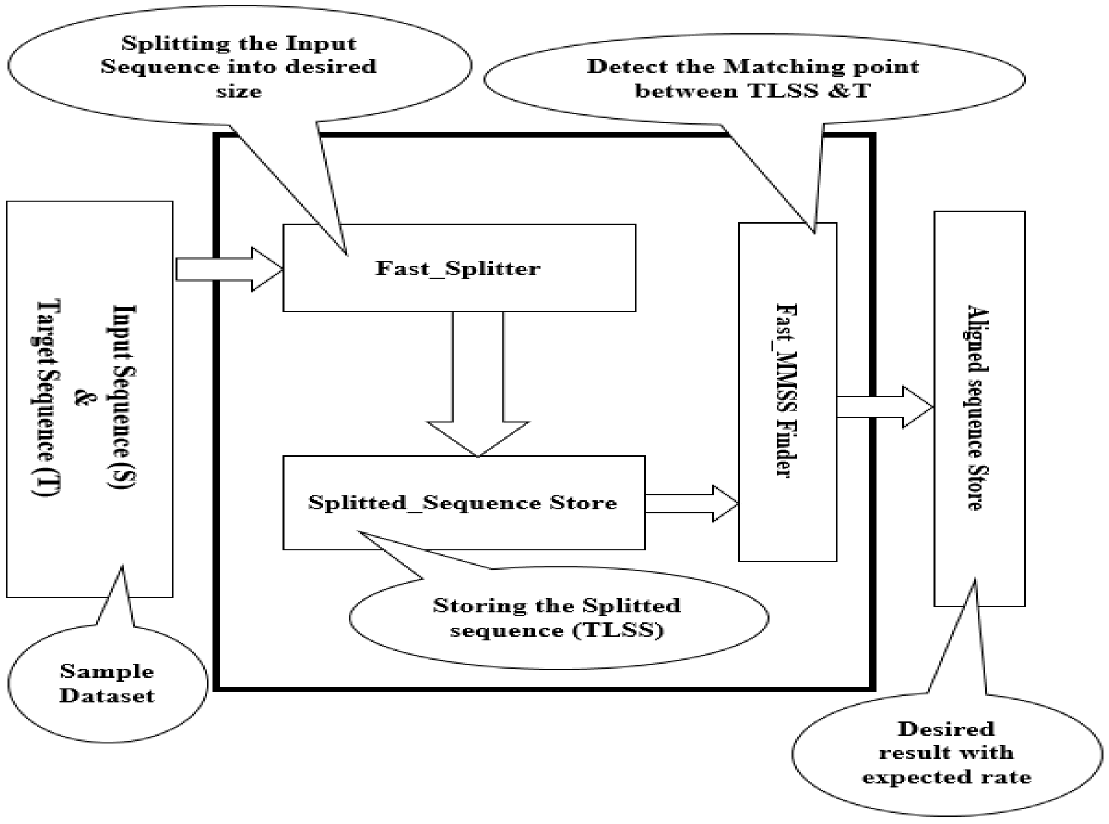
Splitting the Input Sequence into desired size
Detect the Matching point between TLSS &T
Input Sequence (S) & Target Sequence (T)
Fast_Splitter
Fast_MMSS Finder
Aligned sequence Store
Splitted_Sequence Store
Storing the Splitted sequence (TLSS)
Sample Dataset
Desired result with expected rate

Fig. 1. : Complete Flowchart of Fast Splitting

2.2 Fast Splitting(FS) Procedure

Algorithm 1: Fast Splitting(FS)

Input: Input Sequence(S) and Target Sequence(T).

Output: Target TLSS.

2.2.1 Fast_Splitting(Sequence_array,Start,End,Mid,Target_TLSS)

```
1: Sequence_array ← AAA...TTT
2: Last ← length of (Sequence_array)
3: Start ← Sequence_array[0]
4: End ← Sequence_array[Last]
5: Mid ← int((Start+End)/2)
6: Tar_len ← Length of (Target_Sequence)
7: **While** Start<End **do**
8: Left_Recursor(Sequence_array,Start,Mid)
9: Right_Recursor(Sequence_array,Mid+1,End)
10:If(Total_dist(|Start-Mid|)>Tar_len or Total_dist(|End-Mid|) > Tar_len)  then
rget_TLSS←TLSS with desired length.
11: **End While**.
```

Due to recursive subdivision, we reduce the data size. The complete flow chart is:

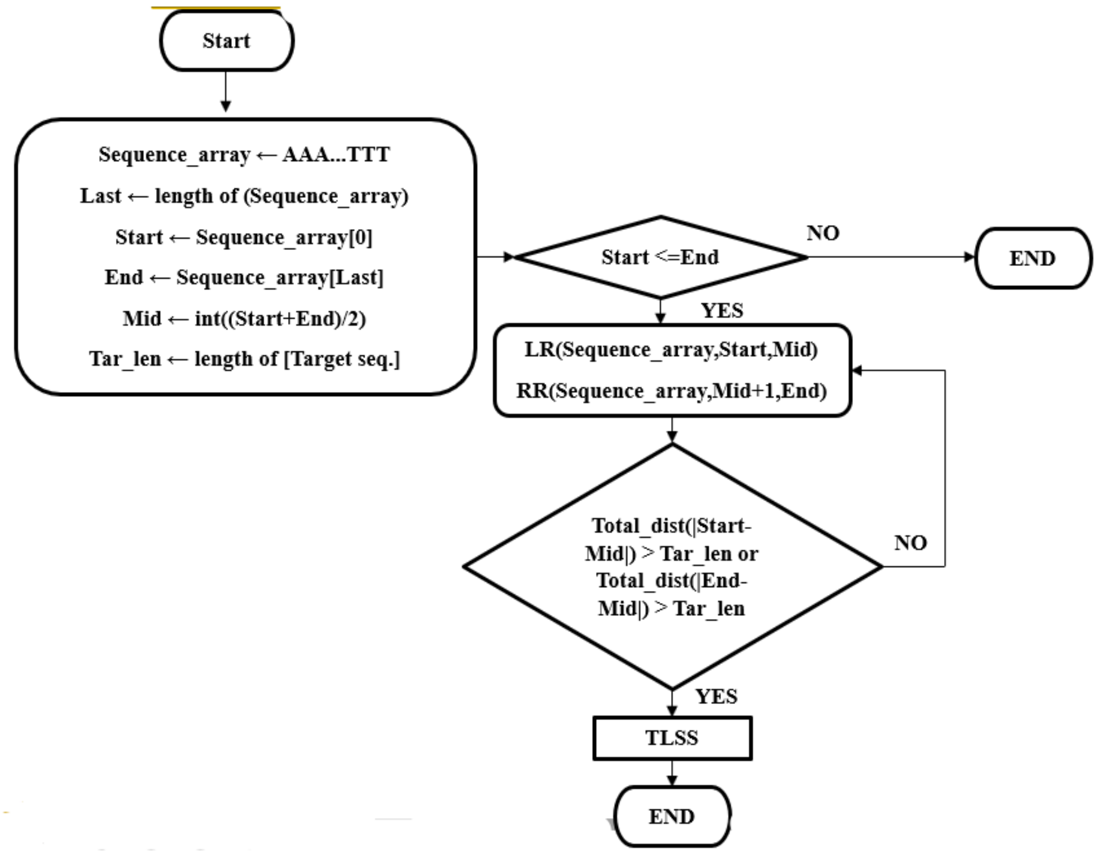
Start
Sequence_array ← AAA...TTT
Last ← length of (Sequence_array)
Start ← Sequence_array[0]
End ← Sequence_array[Last]
Mid ← int((Start+End)/2)
Tar_len ← length of [Target seq.]
Start <=End
NO
END
YES
LR(Sequence_array,Start,Mid)
RR(Sequence_array,Mid+1,End)
Total_dist(|Start-Mid|) > Tar_len or Total_dist(|End-Mid|) > Tar_len
NO
YES
TLSS
END

Fig 2: Complete Flowchart of Fast Splitting.

Let's see the successive subdivision for a sample sequence.

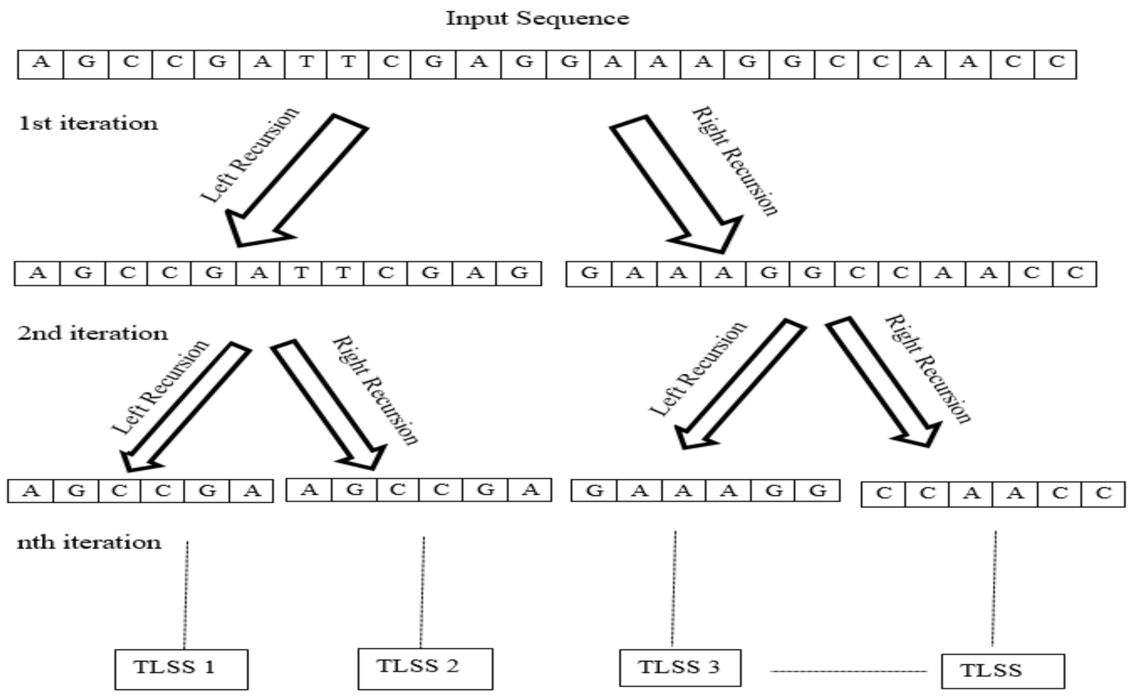

Fig 3: Graphical representation of FS

2.3 Fast_MMSS

Suppose X ($x_1, x_2, \dots, x_n$), $Y = (y_1, y_2, \dots , y_m)$ is the two bio-sequences as well as two TLSS from the Quick splitter, where x_i , y_i $\{A,C,G,T\}$. We can denote an array CA of the four characters so that *CA*(1)="*A*", *CA*(2)="*C*", *CA*(3)="*G* " and *CA*(4)="*T* ". To find Maximum Matches Sub-sequence between TLSS and T, we firstly construct the tables of successors of the uniform characters for the two strings. The successor tables of X and Y are defined as GX and GY.

Definition 1: [8]
For the sequence X = (x_1, x_2… x_n), its successor table SX of identical character is defined as:

$$GX(i,j) = \begin{cases} \min\{k \mid k \in SX(i,j)\} & SX(I,j) \neq \emptyset \\ - & Otherwise \end{cases}$$

Here, SX (i, j)={k| xk = CA(i), k>j}, i = 1,2,3,4, j = 0,1,…n. If GX (i, j) is not "-", it denotes that the index of the next character similar to CA (i) after the jth index in sequence X. If GX(i, j) is equal to "-", it denotes that there is no character CA(i) after the jth index.

Example 1: [8]

Let X ="G C T A T A ". The corresponding successor tables of *GX* is shown

Table 1. Successor table of GX

i	CA(i)	0 1 2 3 4 5 6
1	A	4 4 4 4 6 6 -
2	C	2 2 2 - - - -
3	G	1 - - - - - -
4	T	3 3 3 5 5 - -

Lemma 1[8]. We denoted the length of the maximum matches subsequence of *X*, *Y* as |MMSS(X, Y)|, then |MMSS(X, Y)|=max {level (i, j) $|(i, j)^{\in} S$ (X, Y)}.

Algorithm Fast_MMSS

Input: An array of string which contains the TLSS from Fast_Splitter (algo. 1) in each index and the Target Sequence.

Output: Perfectly aligned sequence with desired accuracy.

```
1:Array[N]← $\{TLSS_1, TLSS_2, \dots\dots\dots\dots\dots\dots. TLSS_n\}$
Where, N is the no. of TLSS from Fast Splitter.
Store[N] is an empty array which contain the final
output.
2:$Store[N] \ \leftarrow \emptyset$
3: for init = 0 to N do
For each Array[i] Call Procedure 1;
4: Store[i] ←Output of Procedure 1 for Array[i]}
5: End for.
```

- **Procedure 1[8]:**

Input: Target Sequence and Array[i] where i means ith TLSS.

Output: Perfectly aligned sequence for Array[i].

```
Begin:
1: Firstly make the tables GX and GY;
2: Search for all the initial uniform character pairs:
(GX (k, 0), GY (k, 0)), k=1, 2, 3, 4;
3: Then put all the initial uniform pairs (k, GX(k, 0),
GY(k, 0), 0, ф, action), k=1,2,3,4 to the table pairs.
Put this value, For all the initial uniform pairs,
pos=1, prev=ф and state=action
4: Repeat
5: for all action uniform pairs (k, i, j, pos, prev,
action) in pairs parallel-
Do Generate all the successors of (k, i, j, pos,
prev, action).
Shift the state of (k, i, j, pos, prev, action) into
inaction.
End for
6: Until any record found in action state in table
pairs.
7: Calculate z = the maximal level in the table pairs.
8: For all uniform pairs (k, i, j, z, l, inaction) with
pos = z in pairs
Parallel-do
Pradi = l; Res(z) = $x_i$.
While Pradi ≠ф do
8.1.1: take the Pradi-th record (prev, g, h,
z', l', inactive) from table pairs.
8.1.2: Pradi = l'; Res(z') = $x_g$.
8.3: End While.
End.
```

3. EXPERIMENTAL RESULT & COMPLEXITY ANALYSIS

3.1 Efficiency of the System

Let time required by random data set as Random time is = T_r

Time required by quick splitting process is = T_s

Total integrated time for all data set time is = T_t

Overhead time or non-useful time is = $T_t - T_r$ (Due to random timing)

Total time spent for all DNA segments are $T_t = n * T_s$

Here,

n = Number of segments of DNA.

So we can define speedup is the ratio between random time dataset processing to Bounding box bounded time

$$Speed\ Up = \frac{Random\ \ Time}{Quick\ \ Splitting\ \ Time} = \frac{T_r}{T_s}$$

By using the mathematical equations, we can define as

$T_r = \theta(n^n)$ due to its randomness.

$T_s = \theta(nlogn)$

So the speed up is

$$Speed\ up = \frac{\theta(n)}{\theta(nlogn\ )}$$

Example:

For the data length n =4

The serial time for linear is 256 ns seconds.

The quick splitting time is 2.41 ns.

The speedup would appear = 256/2.41 = 106.22ns

Efficiency is a measure of the fraction of time for which a processing element is usefully employed. Mathematically, it is given by

$$E = \frac{S}{P}$$

Following the bounds on speedup, efficiency can be as low as 0 and as high as 1.

$$Efficiency = \frac{Speed\ Up}{Complete\ \ segments\ \ volume} = \frac{S}{P}$$

3.2 Implementation tools

Firstly, we needed to set Cygwin environment on our system. On that purpose, we preferred the latest version of Windows 10 Enterprise. Finishing the installation of the environment it is required to install the proper libraries for running the experiment. Then C++11, GNU GCC, OpenMPI libraries are installed with proper manuals and instructions. Code::Blocks IDE and Sublime Text Editor is being installed for code generation, modifications and experimental purposes.

3.3 Experimental Result

CPU utilization time & Execution time comparison with different algorithm is (see Fig. 4 & Fig. 5).

Fig. 4 & Fig. 5 illustrates the execution time & CPU utilization time for sequence alignment for different data length. The bar chart indicates the CPU utilization time for sequence alignment. The skyblue bar indicates CPU utilization for QS+Fast_MMSS. Orange and ass bars are indicates the CPU time for RSRM and BLAST respectively. Less DNA sequence need less CPU time. Our QS+Fast_MMSS measures less CPU. QS+Fast_MMSS used divide and conquer approach for data splitting and reduced the data size. Though, MMSS finding approach performs in linearly. But MMSS finding process has less effect in CPU utilization. RSRM measures less CPU time than BLAST and maximum than QS+Fast_MMSS. RSRM used bounding box approach for data splitting. Bounding box approach selects the data from large data sets in linearly. It also randomly selects the data pattern that need more time than our divide and conquer approach.

Table 2. CPU time comparisons among three algorithms: BLAST, RSRM and QS+Fast_MMSS.

Data Length	QS+Fast_ MMSS(s)	RSRM(s)	BLAST(s)
1000000 bp	0.35	0.39	0.47
1500000 bp	0.37	0.41	0.49
2000000 bp	0.40	0.43	0.52
2500000 bp	0.42	0.45	0.53
3000000 bp	0.43	0.46	0.55
3500000 bp	0.45	0.48	0.56
4000000 bp	0.46	0.50	0.58
4500000 bp	0.50	0.53	0.60
5000000 bp	0.51	0.55	0.63
5500000 bp	0.53	0.56	0.66
6000000 bp	0.55	0.57	0.69
6500000 bp	0.57	0.59	0.70
7000000 bp	0.59	0.60	0.71
7500000 bp	0.60	0.62	0.71
8000000 bp	0.61	0.64	0.73

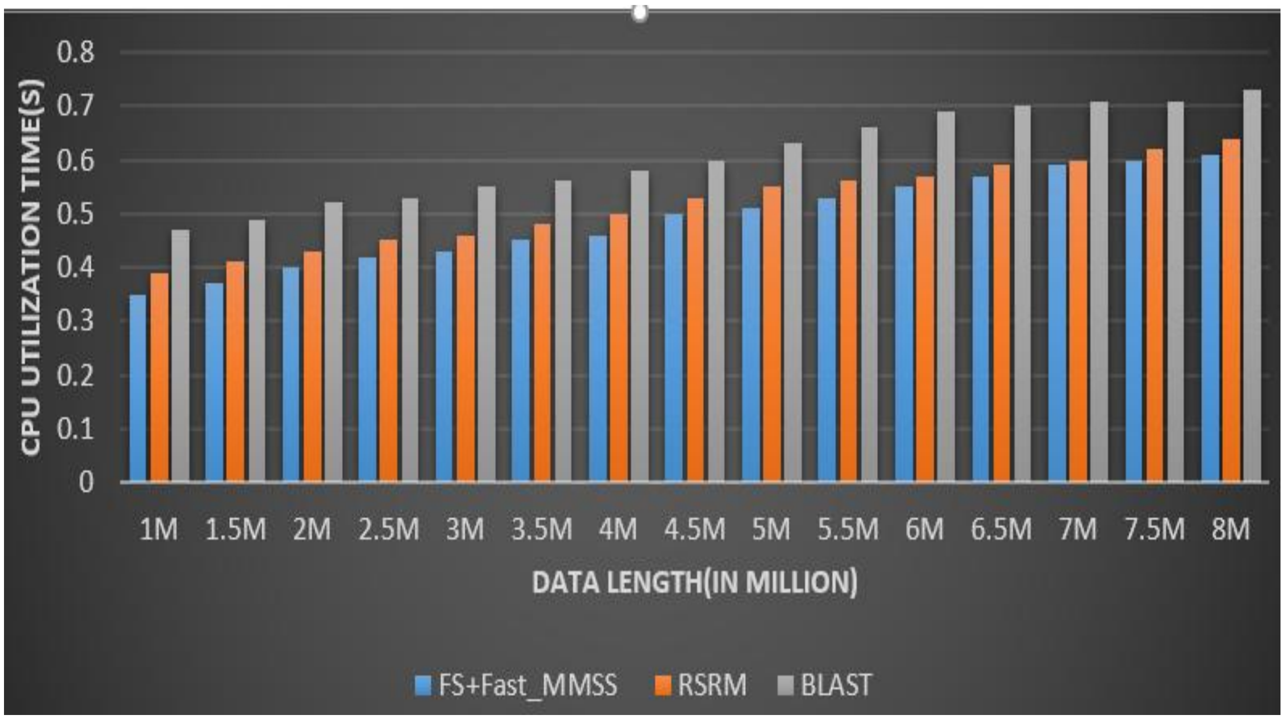

Fig 6: Visualization CPU utilization time comparison with different algorithm

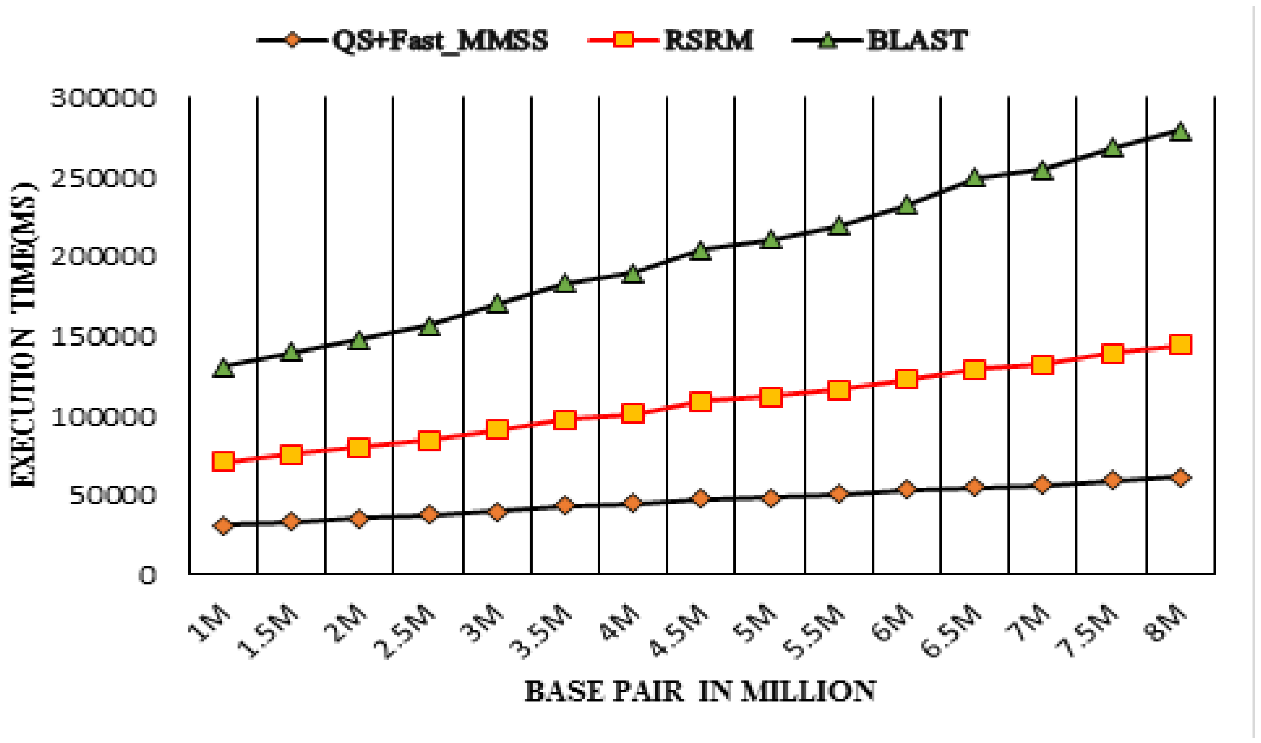

Fig 7: Execution time measurement of RSRM, BLAST and QS+Fast_MMSS.

3.4 Complexity

Our main focus is on time complexity. For computing the time complexity, we should consider the following parameter:

Total number of dataset = N, Length of the given data set = n,

Number of segments = t,

Random Search Time = T_r,

Splitting Time = T_s

- **Complexity of FS:**

In all Cases: $T(n) = 2T\left(\frac{n}{2}\right) + \alpha n$

$= 2\left(2T\left(\frac{n}{4}\right) + \frac{\alpha n}{2}\right) + \alpha n$

$= 2^2\left(2T\left(\frac{n}{8}\right) + \frac{\alpha n}{4}\right) + 2\alpha n$

$= 2^3 T\left(\frac{n}{8}\right) + 3\alpha n$

$= 2^k T(\frac{n}{2^k}) + \text{k} \propto n$

$T(n) = nT(1) + \propto nlogn, which\ is\ O(nlogn).$

- **Complexity of Fast_MMSS**

If we use the parallel implementation technique, then the time complexity of Fast MMSS is O (n*|MMSS(X,Y)|) in all cases[8], here |MMSS(X,Y)| is the length of the MMSS of TLSS & Target sequence. Here, n is the total no. of TLSS. So the Overall Complexity is = $O(nlogn)$ +n |MMSS(X,Y)|).

- **Space Complexity**

Though our algorithm only focused about time efficiency, but our algorithm space-efficient also. We compare the memory requirement between Needleman-Wunsch [2], BLAST[1] and RSAM[7].

Table 3. Comparison of Memory Complexity and Memory requirement between Needleman-Wunsch, BLAST and our proposed algorithm

Algorithm	Complexity	Memory requirement
Needleman-Wunsch	O(M*N)	(M*N)
BLAST	O(w * (m+n))	(m+n+(w*(m+n)))
FS+Fast_MMSS	O(m+n+4*(n+1)+4*(m+1))	m+n+4*(n+m+2)

4. CONCLUSION

In the paper, we have proposed a time efficient DNA sequence analysis algorithm, called FS+Fast_MMSS that can manage the huge dataset dynamically. FS+Fast_MMSS require very less time than that of BLAST or other previous algorithm. Experiment results prove the effectiveness of our proposed algorithm. We have worked with experimental dataset. Further, this concept will be implemented for real world dataset, RNA & Protein sequences and also use this concept to handle the dislocated or broken sequences

5. ACKNOWLEDGMENTS

Our thanks to the experts who have contributed towards development of the template.

6. REFERENCES

[1] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, D. J. Lipman. "Basic local alignment search tool," J. Mol. Biol., vol. 215, pp. 403-410, 1990.

[2] S. B. Needleman, C. D. Wunsch. "A general method applicable to the search for similarities in the amino acid sequence of two proteins," J.Mol Biol., vol

.48, pp. 443-453, 1970.

[3] T. F. Smith, M. S. Waterman. "Comparison of bio-sequences," Adv. Appl. Math.2, PP. 482-489, 19981.

[4] C. Notredame, D. G. Higgins, J. Heringa. "T-Coffee: A novel method for that and accurate multiple sequence alignment," Journal of molecular biology, vol. 302, no.1, pp. 205-217, 2000.

[5] A. B. Do, M. S. Mahabhashyam, M. Brudno, S. Batzoglou. "ProbCons: Probabil-istic consistency-based multiple sequence alignment," Genome research, vol. 15. no. 2, pp. 330-340, 2005.

[6] L. A. Newberg. "Memory-efficient dynamic programming backtrace and pair-wise local sequence alignment," Bioinformatics, vol. 24. no. 16, pp- 1772-1778, 2008.

[7] M.I. Khan, M.S. Kamal. ”RSAM: an integrated algorithm for local sequence alignment.” Arch Sci vol. 5 pp. 395–412, 2013.

[8] W. Liu , L. Chen ,” A fast longest common subsequences algorithm for biosequence alignment.” Computer And Computing Technologies In Agriculture, volume I. CCTA 2007, The International Federation for Information Processing, vol 258. Springer, Boston, MA

[9] D. J. Lipman, W. R. Pearson. "Improved tools for biological sequence comparison," Proc. Natl Acad. Sci., vol. 85, pp. 2444-2448, 1998.

[10] T. Watanabe, A. Takeda, K. Mise, T. Okuno, T. Suzuki, N. Minami, H. Imai. "Stage-specific expression of microRNAs during Xenopus ,development," FEBS Left., vol. 579, no. 318, 2005.

[11] S. Griffiths, A. Bateman, M. Marshall, A. Khanna, and S. R. Eddy. "Rfam: An RNA Family Database," Nucleic Acids Research, vol. 31, no. 1, pp. 439-441, 2003.

[12] D. Lee, K. Han. "Prediction of RNA Pseudoknots: Comparative Study of Genetic Algorithms," Genome Informatics, vol. 13, pp. 414-415, 2003.

[13] F. Pais, P. Ruy, G. Oliveira and R. S. Coimbra. "Assessing the efficiency of multiple sequence alignment programs," Algorithms for Molecular Biology, vol. 9, no. 4, pp. 1-8, 2014.

[14] D. Kletiogiannisl, P. Kalnisl and V. B. Bajic2. "Comparing Memory-Efficient Gcnomc Assemblers on Stand-Alone and Cloud Infrastructures," PLoS ONE, vol. 8, no. 9, pp. 1-11, 2013.

[15] S. Kamal, & M.I. Khan, “An integrated algorithm for local sequence alignment”, Netw Model Anal Health Inform Bioinforma (2014) 3: 68. https://doi.org/10.1007/s13721-014-0068-8.A. Khedher, I.Jraidi, & C. Frasson, “Local Sequence

[16] Alignment for Scan Path Similarity Assessment”, International Journal of Information and Education Technology, vol. 8, no. 7, July 2018.

[17] D. Nath, J. Kurmi & D. N. Shukla,” A Revised Algorithm to find Longest Common Subsequence”, International Journal for Research in Applied Science & Engineering Technology (IJRASET), vol. 6, Issue IV, April, 2018.

[18] Z. Yang, R. Zhu and L. Zhang, "The improvement and implementation on the algorithm for local alignment of pairs of DNA sequences," IEEE IMCEC, Xi'an, 2016, pp. 1316-1320. doi: 10.1109/IMCEC.2016.7867426.