An Efficient Fast Pruned Parallel Algorithm for finding Longest Common Subsequences in BioSequences

Sumathy Eswaran
Dr.M.G.R. Educational & Research Institute, Chennai, India
sumathyeswaran@drmgrdu.ac.in

S. P. RajaGopalan
Dr.M.G.R. Educational & Research Institute, Chennai , India
sasirekaraj@yahoo.co.in

ABSTRACT: This paper presents an Efficient and fast approach to identify the Longest Common Subsequence between Biosequences. Identifying Longest Common Subsequence between two or more biosequences is an important problem in computer science due to its complexity and applicability to the field of biology. This Algorithm achieves its efficiency in using computational resources by doing specific pruning in the process of identifying the Longest Common Subsequence. The proposed approach has various steps like Identifying Initial Identical Character Pairs between the Sequences, its Successors, Pruning to retain potential successors and backtracking. The execution results indicate that with the proposed algorithm Memory Efficiency and Fast Execution are achieved over the prominent FAST_LCS Algorithm while retaining the precision of FAST_LCS.

Introduction

Multiple Sequence Alignment (MSA) of biological components like DNA, RNA, Protein or genome are an important problem in Bioinformatics and Genomics. This problem is of interest to Biologists because the input query sequence, which is a sample, is assumed to have an evolutionary

relationship by which it shares a lineage from a common ancestor. Although MSA is a small part of a larger solution to biologists, this is the first step to the larger solution and has a very high computational complexity. Hence the computational Researchers are interested to identify improved solutions to MSA problems.

MSAs display and summarize relationship among sets of sequences, which would be useful in the areas like protein modeling, protein structure prediction, molecular evolution, detection and quantification of sequence motifs. Also important are the production of more accurate alignment and providing a range of permissible variations between the sequences. It is in this aspect that identifying Longest Common Subsequence (LCS) of bio-sequences is important. LCS detects similarity between two or more sequences.

MSA finding is a two part sequential activity:

- Finding a local alignment between a set of given sequences
- Finding a global alignment using the local alignment result to assist the biologists in their interpretation.

Thus to identify and interpret global alignment, the basis or first step is to identify local alignment i.e finding LCS. There are many global alignment methods like Dynamic Programming [FASTA, BLAST family], progressive alignment methods like [ClustalW], iterative methods [MUSCLE], and probabilistic models [HMM] etc.

Longest Common Subsequence (LCS) is to find a substring or substrings that are common to two or more given strings and is the longest such string(s). There are many sequential algorithms available in this area and a few parallel algorithms like CREW PRAm model, Systolic Arrays, Run length encoding basis etc., making use of the technology to arrive at a quick solution. FAST_LCS algorithm was proposed by Wan, Liu & Chen [CWL06] and Quick DP MLCS by Wang, Korkin and Shang [WKS10].

FAST_LCS based on Initial Identical Pairs (IIDP) and Successors claimed to have memory complexity as max[4*(n+1)+4*(m+1),L] and time complexity of O|LCS(X,Y)| for parallel implementation [CWL06]. The strength of FAST_LCS is in its precision to identify the LCS [CWL06]. In fact it identifies all the possible LCS. QuickDP MLCS works with whole dominant point set between two strings; then divide and Conquer technique is applied for parallel implementation. QuickDP claims to work for longer strings on linear time complexity while silent on memory requirement [WKS10].

The performance of an LCS algorithm depends on:

- The length of the query string and reference data string

- The number of LCS between the two strings (the string composition)
- The Algorithm

Note: The absolute numbers on timings or memory requirement will depend on the program implementation efficiency. However abstract comparative information on performance can be obtained from graphs. The quantitative comparison is meaningful only in terms of percentage figures.

Our work, An Efficient Fast Pruned Parallel Algorithm for finding LCS in BioSequences (EFP_LCS) provides improvement over FAST_LCS. EFP_LCS gives 40% to 60% improvement in time complexity and 40 to 70% improvement in memory requirement over FAST_LCS. EFP_LCS obtains the improvement by identifying redundant pairs and pruning them.

1. Successor Table

Let $X = (x_1,x_2,\ldots\ldots,x_n)$ and $Y= (y_1,y_2,\ldots\ldots,y_m)$ and Character set $CH = \sum =(A,C,G,T)$ where $x_i,y_i \in \sum$.

The Successor Table of X and Y are denoted as TX and TY. TX and TY are two dimensional array with elements $\sum$ x X and $\sum$ x Y. The size of TX is $|\sum|$ x ($|X|$ +1) and the size of TY is $|\sum|$ x ($|Y|$ +1). The entries into successor table are defined as below:

$$T(i,j) = \begin{cases} \min\{k \mid k \in S(i,j)\} & \text{when } S(i,j) \neq \emptyset \\ - & \text{otherwise} \end{cases}$$

$$S(i,j) = \{k \mid x_k = CH(i), k>j\}$$

where i= 1,2,3,…,$|\sum|$ and j = 0,1,2,….,n i.e. length of sequence

When $T(i,j) \neq -$, the values indicate the forthcoming position of CH(i) in the sequence; when = "-" i.e. no value, indicates that no more occurrence of CH(i) after j^{th} position.

Example:

Let X = " TCGTAC" and Y = "ATGCTAA". $|X| = n = 6$; $|Y| = m = 7$.

Their successor Tables TX and TY are shown in Table 1.

Table 1. Successor Tables TX and TY

TX:

i	j = CH(i)	0	1	2	3	4	5	6
1	A	5	5	5	5	5	-	-
2	C	2	2	6	6	6	6	-
3	G	3	3	3	-	-	-	-
4	T	1	4	4	4	-	-	-

TY:

i	j = CH(i)	0	1	2	3	4	5	6	7
1	A	1	6	6	6	6	6	7	-
2	C	4	4	4	4	-	-	-	-
3	G	3	3	3	-	-	-	-	-
4	T	2	2	5	5	5	-	-	-

2. Identical Character Pair (IDP)

For sequences X and Y , if $x_i = y_j$ = CH(k), then (i,j) is an identical pair of CH(k). The set of all identical character pairs of X and Y is denoted as S(X,Y).

(TX(k,0), TY(k,0)) , where k = 1,2,3,4 are known by special name Initial Identical Character Pair(IIDP). The IIDPs of X and Y. In this case these are (5,1), (2,4), (3,3), (1,2). The IIDPs are eligible to get entry into Pair Table at level 1. All other IDPs get their level entry as the case may be.

3. Efficient Pruning Techniques

Identifying non redundant IIDPs: (Pruning A)

In the IIDP, eliminate redundant pairs which are unlikely to generate LCS.

Example:

(5,1), (2,4), (3,3), (1,2) are IIDPs. Since (2,4) >(1,2) , (2,4) is unlikely to produce a longer LCS than (1,2). Hence (2, 4) can be pruned. Similarly (3, 3) >(1,2). Hence (3, 3) can also be pruned. Thus (2, 4) and (3, 3) are identified to be redundant IIDPs and the useful IIDPs are only (5,1) and (1,2). In this case, 50% reduction in IIDPs at initial level, reduces the redundant successor generation and hence computation time.

Of course the final number of useful IIDPs depends on the two sequences getting compared.

Pruning B:

Any successor pair having a no value member i.e ., "-" is redundant.

Example:

(-, 6), (6, 4), (-, 3), (-, 2) are successors at level 1 for IIDP (5, 1) corresponding to character A. Except for (6, 4) all are having no value members. Hence these are redundant and can be pruned.
In another case (5,6), (6,-), (3,-) and (4,5) are successors of IIDP (2,4) corresponding to Character C. Here (6, -) and (3, -) are no value redundant members and can be pruned. Only useful successors are (5,6)and (4,5)

Pruning C:
Among the successors of the same parent, a successor is said to be redundant if it satisfies the condition as below:
Let $(i_1,j),(i_2,j),\ldots.(i_m,j)$, (k,l) are all successors in the same level of a parent, such that $i_1<i_2<\ldots i_m<k$ and $j<l$
Then (i_1,j) alone is the non redundant successor. Therefore if $(i_1<i_2)$ .and. $(j\leq l)$ then prune (i_2,l).

Pruning D:
While identifying successor for a pair (i,j), if either the members of column i of the TX table or members of column j of TY table are all having no entry then the pair (i,j) is redundant in the next level. Hence does not qualify to get entry into pair table; prune in this level itself. This is a lemma to Prune A.

Pruning E:
This pruning is useful in Sequential implementation of EFP_LCS algorithm.

Take the 1st eligible IIDP. Identify the LCS starting with this IIDP character. Store the LCS and |LCS| arrived at. Do the same with next IIDP. Compare the length of 2nd LCS arrived with the previous one. Discard the lesser length one and retain only the longer one. Continue the iteration until all the IIDPs are over. This will let us hold only the longest LCS arrived so far and hence operate with less memory.

4. EFP_LCS Algorithm

Method: Given two sequences over an alphabet $\sum$, generate successor tables and create Initial Identical Pairs(IIDP). Starting from early non redundant IIDPs, keep identifying their successors at successive levels until no more successors. This is the point of longest common subsequence(s). The level number also means the length of LCS i.e. length of LCS,

|LCS(X,Y)| = maximum level. The number of LCS identified between the given strings is given by the number of pairs in the last level. It is possible to have more than one or none string as LCS. Now backtrack to identify the LCS. Backtracking can be carried out in parallel if there are more LCS and hence all can be obtained concurrently. Also parallel implementation is possible in identifying successors at all level for a particular Initial Identical Character Pair.

Algorithm EFP_LCS(X,Y): (Parellel version)

Input: Query data sequence X of length m
Reference Data Sequence Y of length n
Let X,Y be contained in alphabet set |∑| = k.
Output: string LCS of length |LCS| which is /are Longest Common Subsequence(s) between X and Y.

Procedure:
Step 1: Build Successor Table TX and TY for the X and Y sequences over the alphabet ∑.
Step2: IIDP entry into Pair Table
Step 2a.: Find all the Initial Identical Pairs(IIDP) of X and Y sequences over ∑ such that {TX(k,0),TY(k,0)| k=1,….,k}

Step 2b: Apply Prune-A on IIDPs.
Step 2c: Add the potentially useful IIDPs to the Pair Table.
Pair Table Data Structure: (i, j, level, Predecessor, Record number)
i = TX(k,0)
j = TY(k,0)
level =1 for IIDP, otherwise level = parentlevel +1.
Predecessor = Ø | IIDP ,
= parent level |otherwise }

Record number = last entry in pair Table +1

Step 3: Producing Successors level wise
Repeat steps 3.1 thru 3.2 until no more successors are found
Step 3.1:
For all the current level identical pairs in pair table parallel do
3.1.1. identify all direct successors
3.1.2. Apply Prune B, C and D.

Step 3.2:
Add the potentially useful successors to pair table.

Step 4: Backtracking to collect LCS
For each of the pair in the max level, parallel do
Pred = predecessor; LCS(r) = X_i ;
While pred ≠ Ø do
Read the pair table entry of Recordnumber
Set pred = predecessor; LCS(r') =X_i ;
LCS of X,Y is stored in the array LCS . |LCS| = max level.

Algorithm EFP_LCS(X,Y): (Sequential version)

Input: Query data sequence X of length m
Reference Data Sequence Y of length n
Let X,Y be contained in alphabet set |∑| = k.
Output: string LCS of length |LCS| which is /are Longest Common Subsequence(s) between X and Y.

Procedure:
Step 1: Build Successor Table TX and TY for the X and Y sequences over the alphabet ∑.
Step2: IIDP entry into Pair Table
Step 2a.: Find all the Initial Identical Pairs(IIDP) of X and Y sequences over ∑ such that {TX(k,0),TY(k,0)| k=1,….,k}

Step 2b: Apply Prune-A on IIDPs.
Step 2c: Add the potentially useful IIDPs to the Pair Table.
Pair Table Data Structure: (i, j, level, Predecessor, Record number)
i = TX(k,0)
j = TY(k,0)
level =1 for IIDP, otherwise level = parentlevel +1.
Predecessor = Ø | IIDP ,
= parent level |otherwise
Record number = last entry in pair Table +1

Step 3: Producing Successors level wise for each IIDP
Take the 1st IIDP do Step 3 and 4
Repeat steps 3.1 thru 3.2 until no more successors are found

Step 3.1:

For the current IIDP in pair table

3.1.3. identify all direct successors

3.1.4. Apply Prune B, C and D.

Step 3.2:

Add the potentially useful successors to pair table.

Step 4: Backtracking to collect LCS

For each of the pair in the max level, parallel do

Pred = predecessor; LCS(r) = X_i ;

While pred ≠ Ø do

Read the pair table entry of Recordnumber

Set pred = predecessor; LCS(r') =X_i ;

LCS of X, Y is stored in the array LCS-current . |LCS-current| = max level.

Take one by one each IIDP in the pair table and do Steps 3 to 5

When no more IIDP in pair table exit to step 6

Step 5: Apply Prune E|

Go back to step 3

Step 6: The LCS is content of LCS and exit.

5. Results

Results on Sequential Computation on two sequences

The execution time and memory requirement are proportional to number of IIDP at level 1 and the |LCS|. The lengthier the sequence X and Y, the chances of |LCS| being large is high and hence the resource requirement for finding solution also is high.

The genomic database "*aedb*" subset from EBI [fas**] and the protein sequences "*pdb*" from EBI ftp server [pdb**] were used. The algorithm EFP_LCS and FAST_LCS were executed on Intel Dualcore/1.6GHz/1GB Desktop Computer system. Experiment was done with 3 each datasets of Genome sequences and Protein sequences on both Algorithms. Since test on one pair sequence would take very less time, the datasets were iterated 50 times. The algorithm is compared with FAST_LCS. The resultant LCS sequences were compared for both length and number and EFP_LCS is found to produce identical result with FAST_LCS. EFP_LCS is found to work consistently well for larger sequences than FAST_LCS as the memory usage is less by 40% to 70%.

EFP_LCS Computational efficiency (CPU Time) i.e. speedup in finding solution is faster by 40% to 70% than FAST_LCS. The precision of EFP_LCS is the same as that of FAST_LCS

Genome Dataset1 had a query genome sequence of |50|. The Y sequence was varied from |40| to |400| and duplicated for iterative purpose. As seen from table.2, EFP_LCS is able to identify LCS by generating only one 3rd or less records and hence the memory used is also that much less. This had been possible by the additional pruning taken up by EFP_LCS. This is also the reason why EFP_LCS is faster than FAST_LCS in spite of the overheads due to additional pruning explorations. It is to be noted that FAST_LCS has failed to converge with some of the Y sequences, where as EFP_LCS has succeeded to identify LCS over those critical ranges too. The average speedup achieved by EFP_LCS over FAST_LCS on this dataset-1 was 67% and memory efficiency achieved was 70%. The performance is plotted in Fig. 1 and Fig. 2.

Table 2 Results with DATASET-1

Y sequence length	FAST_LCS			EFP_LCS			Speedup in CPU time	Memory Efficiency
	Max Records generated	Memory used in KB	CPU time in secs	Max Records generated	Memory used in KB	CPU time in secs		
40	105264	1233	0.531	32451	380	0.203	61.77	69.17
50	502000	5882	2.516	121470	1423	0.828	67.09	75.80
60	2755866	32295	12.656	573656	6722	3.609	71.48	79.18
70	8033993	94148	33.953	2037663	23878	10.765	68.29	74.64
80	14438168	169197	60.562	3490658	40906	18.406	69.61	75.82
90	28879413	338430	118.540	7693702	90160	38.328	67.67	73.36
100	Failed			9800183	114845	50.641	NA	NA
110	Failed			12754824	149470	62.531	NA	NA
120	56614872	663455	223.040	15126615	177265	71.484	67.95	73.28
130	Failed			15744843	184509	75.063	NA	NA
140	Failed			16507549	193447	77.547	NA	NA
146	61859249	724913	244.100	16518431	193575	77.859	68.10	73.30

NA- Not Applicable

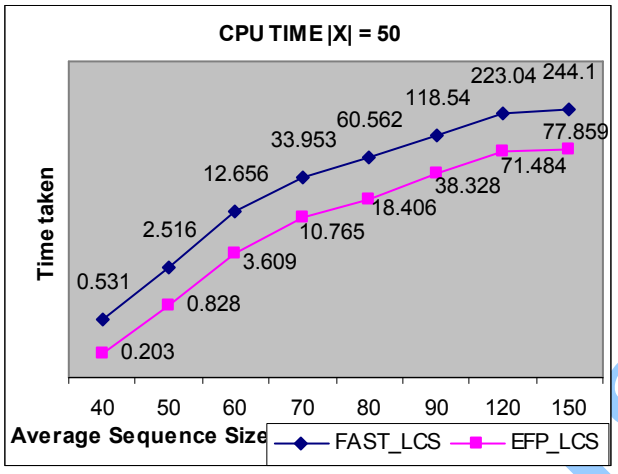

Figure 1. CPU Time with DATASET-1

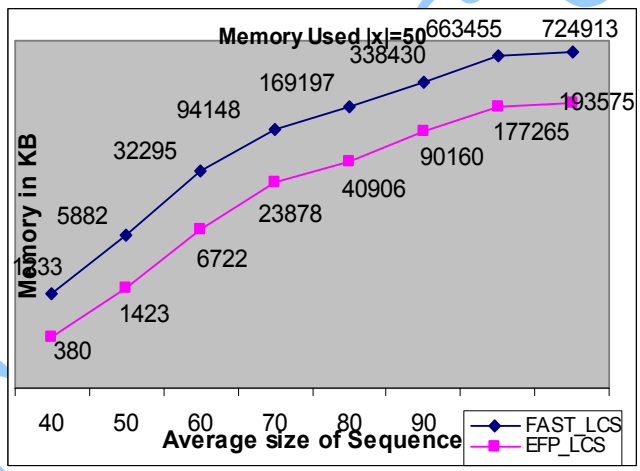

Figure 2. Memory used with DATASET-1

Genome Dataset-2 is an experiment with *aedb* genome sequences of variable length. 100 such sequences were taken as Y sequences. The |Y| was ranging from 3 to 54. The |X| was varied from 17 to 54 and iterated. On this *aedb* genome sequences, EFP_LCS has yielded 35% speedup of CPU activity and 42% memory efficiency. On |X|=24, in spite of the CPU time being indifferent, memory efficiency is achieved. The results with Dataset-2 are displayed in Table 3 and Figure 4 and Figure 5.

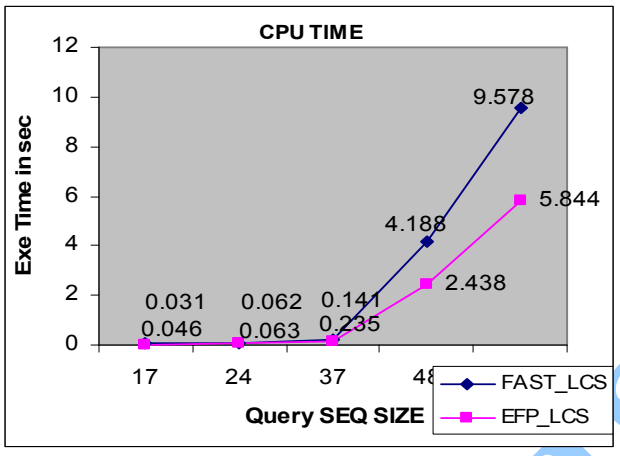

Figure 3. CPU Time with Dataset-2

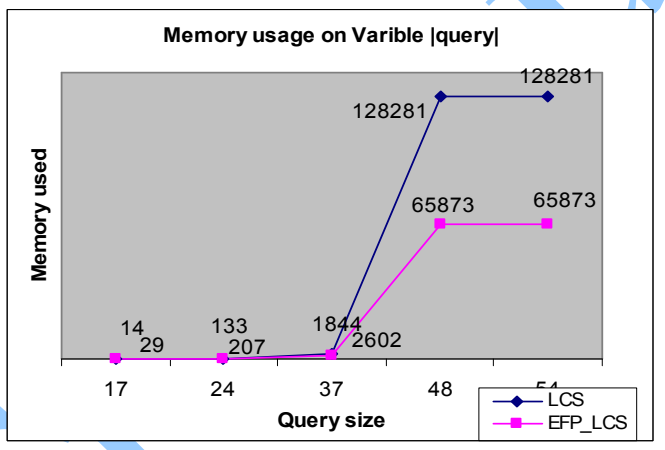

Figure 4. Memory usage with Dataset-2

Table 3. Results with Dataset-2

Query sequence length X	FASTLCS		EFP_LCS		Speedup in CPU time %	Memory Efficiency %
	Memory used in KB	CPU time in secs	Memory used in KB	CPU time in secs		
17	29	0.046	14	0.031	32.61	50.00
24	207	0.063	133	0.062	1.59	35.98
37	2602	0.235	1844	0.141	40.00	29.12
48	128281	4.188	65873	2.438	41.79	48.65
54	128281	9.578	65873	5.844	38.99	48.65

Genome Dataset3 was experimented to bring out the data dependency behavior between the X and Y sequences in identifying LCS. Hence the X sequence alone was changed in the dataset-1. Here again EFP_LCS continued to perform better with CPU speedup of 35% and memory efficiency of 42%. The details are shown in table.4. On higher lengths when FAST_LCS fails EFP_LCS continues to perform. In these cases the speedup and memory efficiency become "NA" meaning not applicable as the data for FAST_LCS was not available.

Table 4. Results on Genome Dataset=3

Sequence length Y	FASTLCS			EFP_LCS			Speedup in CPU time %	Memory Efficiency %
	Max Records generated	Memory used in KB	CPU time in secs	Max Records generated	Memory used in KB	CPU time in secs		
100	14493326	169843	64.922	7635007	89472	42.547	34.46	47.32
110	25582052	299789	108.047	14754224	172901	74.922	30.66	42.33
120	34280097	401719	144.375	19400963	227355	99.516	31.07	43.40
130	41080502	481412	175.641	23090835	270595	124.2	29.29	43.79
150	Failed			40747821	477513	203.48	NA	NA

Similar experiment was done with protein sequence *pdb* from EBI database [pdb**]. It was observed that EFP_LCS provides almost the same efficiency as that with genome data. However EFP_LCS was able to converge with |X| =80 while FAST-LCS was able to converge only with |X| =50. The |Y| limit for EFP_LCS was up to 258 on *pdb* data while that for FAST_LCS was 146.

On Applying Prune-E the EFP_LCS could converge with genome data length of 900 whereas FAST_LCS fail to converge for want of resource requirements. The details as below:

XLen: 49 No.of YStr: 10 YLen Max: 900 Avg: 900
TotET: 97.437
MaxRecs: 27152506 Mem used: 318193KB (318MB).

This shows it is possible to have long sequences compared within the available resources with EFP_LCS than with FAST_LCS.

Conclusion

Since EFP_LCS produces all the possible LCS, the global alignment that can be obtained thru models like HMM [KC09] and EFP_LCS is expected

to give more relevant results. On a Sequential implementation EFP_LCS basis has been proven to give efficiency in the use of computational resources and memory usage. It is expected to give better memory efficiency in MSA implementation for which experiment is in progress. If EFP_LCS were run on powerful computer, it can certainly cross the limits of most leading LCS algorithms. The numeric limits on |X| and |Y| are computer system resource dependent and hence only comparative performance should be used to understand the EFP_LCS ability to perform.

References

[AHJ76] A. Aho, D. Hirschberg and B. Jullman - *Bounds on the Complexity of the Longest Common Subsequence Problem*, J. Assoc. Comput. Mach., Vol. 23, No. 1, 1976

[Ber**] Bryan Bergeron - *Bioinformatics computing*, Pearson Education publication

[BHR00] L. Bergroth, H. Hakonen and T. Raita - *A survey of longest common subsequence algorithms*, Seventh International Symposium on string Processing information Retrieval, 2000

[CWL06] Yixi Chen, Andrew Wan and Wei Liu - *A fast Parallel Algorithm for finding the Longest Common Subsequence of multiple biosequences*, BMC Bioinformatics 2006, 7 (suppl 4): 54, ©2006 Chen et al; licensee BioMed Central Ltd.

[fas**] ftp://ftp.ebi.ac.uk/pub/databases/fastafiles/asd/

[FB04] V. Freschi and A. Bogliolo - *Longest Common Subsequences between run length encode strings: a new algorithm with improved parallelism*, Information Processing Letters, 2004

[Gro68] Bob Gross - Multiple Sequences alignments, Bio68

[KC09] Anoop Kumar and Lenore Cowen - *Augmented Training of Hidden Markov Models to recognize remote homologs via simulated evolution*, bioinformatics/btp265, vol25, no.13/2009

[pdb**] ftp://ftp.ebi.ac.uk/pub/databases/pdb_seq/

[SW90] T. F. Smith and M. S. Waterman - *Identification of common molecular subsequence*, Journal of Molecular Biology, Vol. 215, 1990

[WKS10] Qingguo Wang, Dmitry Korkin and Yi Shang - *Efficient Dominant Point Algorithms for the Multiple Longest Common Subsequence (MLCS) problem*

[***] *** - Multiple sequence Alignment, http://en.wikipedia.org

ftp://ftp.ebi.ac.uk/pub/databases/pdb_seq/

http://www.pdb.org

www.**ebi**.ac.uk