CS 5390 Project: Implementation of different algorithms in computational biology

Govinda KC*

Computational Science Program, The University of Texas at El Paso, Texas 79968, USA.

E-mail: gbkc@miners.utep.edu

Introduction

Sequence comparison has become one of the most fundamental problems of computational biology.[1] As databases of sequences (such as protein sequences) increase in size, it is most likely to depend on previously classified proteins to determine the structure and function of new protein. Biologists, if they find unknown sequence, compare this query sequence with the known database sequences, which helps to find evolutionary relationship among them with the help of their similarity scores. This technique is known as sequence alignment. The optimal alignment of query sequence and known database is the way to determine similarity. The scoring of the alignment is obtained on the basis of match, mismatch, insertion, and deletion. A optimal alignment contain minimum mismatches and gaps between the sequences. There are many algorithms (some older such as Needleman-Wunsch,[2] Smith-waterman[3] and some new such as FASTA,[4] BLAST[5]) to search and compare the given sequence in entire database. Older algorithms are computationally costly but still find the optimal alignment. Newly developed algorithms are faster; however, they lack in their accuracies. Three sequence alignments that are commonly used are: global, local, and semi-global. Out of them, the global alignment (Needleman Wunsch), and local alignment (Smith-waterman) were implemented in this work. These two algorithms based on dynamic programming generally subjected to 3 steps: (1) Initializing the dynamic programming (DP) matrix (2) Filling the DP matrix with scores of optimal partial alignments. (3) Trackback, that is extracting one or more alignments from the dynamic programming matrix, via a trackback pointer. Needleman and Wunsch[2] proposed a DP method in 1970, which solves the global alignment problem between two sequences. It is a global alignment technique, and can not used to find the local regions of high similarity. The alignment with the largest score must be the optimal alignment. The Smith–Waterman algorithm[3] (also abbreviated to SW-algorithm) is a well-known algorithm also based on DP performs the local sequence alignment. It was developed by Temple F. Smith and Michael S. Waterman in 1981. It compares all the segments of both strings and determines the optimal alignment between them. The Smith-Waterman algorithm is potentially more biologically relevant due to the fact that the ends of proteins tend to be less highly conserved than the middle portions, leading to higher mutation, deletion, and insertion rates at the ends of the protein. It allows us to align proteins more accurately without having to align the ends of related protein which may be highly different.

Edit distance is a way of quantifying how dissimilar two strings (e.g., words) are to one another by counting the minimum number of operations required to transform one string into the other string. This concept was given by Levenshtein in 1966[6] and now widely used in various field of computational linguistics and computer sciences, for example, in natural language processing to automatic spelling correction of misspelled word , where it selects the words from a dictionary that have a low distance to the word in question.

The repetitive substrings are important in biological sequence as well in other field of studies such as data mining , data compression, and computer assisted music analysis. These repeated substrings have important meaning and functions, for example, motifs or short strings common to protein sequences are assumed to represent a specific property of the sequences. A longest repeated substring is obtained using the suffix array. Suffix array is more compact so that it is memory efficient than suffix tree. It enables lookup of any substring of a text and identification of repeated substrings. A useful auxiliary data structure is an' LCP array', an array of lengths of the longest common prefix between each substring and its predecessor in the suffix array. The longest common substring problem are important in biology such as comparing two or more coding DNA sequences, regions that are well conserved are often of particular biological interest. The longest repeated substring is at the suffix array index with the largest LCP. The Burrows-Wheeler transform[7] (also abbreviated to BWT) was first developed by Michael Burrows and David Wheeler in 1994. It is found an efficient algorithm (O(n)) as it can be implemented using suffix arrays. The transformation is reversible because the original string of any transform string can easily be obtained without any other information. The BWT is basically a permutation of the input string and thus no compression on its own. However, it clusters the similar contexts in the output.

Among many algorithms that are widely used in computational biology, five aforementioned algorithms were implemented, tested, and also evaluated their time and space complexities.

Methods

The brute force approach to compute all the possible alignments and score them, would take exponential time to obtain the optimal global alignment for Needleman-Wunsch algorithm. Similarly, for Smith-Waterman, align all substrings of S with all substrings of T and get the optimal local alignemnt would take $O(n^3m^3)$. Five different algorithms mainly based on dynamic programming were implemented in this project.

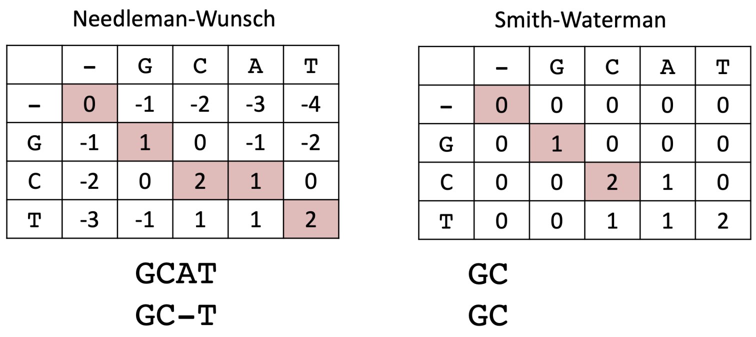
To find either global alignment (Needle-Wunsch) or local alignment (Smith-Waterman), a scoring matrix was first generated based on match reward, mismatch penalty, and insertion/deletion penalty. The second step was back tracking from the calculated matrix to obtain the alignment score.

Needleman-Wunsch:

$$V[i,j] = max \begin{cases} V[i-1,j-1] + \delta(S[i],T[i]) & \text{match/mismatch} \\ V[i-1,j] + \delta(S[i],-) & \text{delete} \\ V[i,j-1] + \delta(-,T[j]) & \text{insert} \end{cases} \tag{1}$$

$$\begin{array}{rcr} V(0,0) = 0 & & \delta(-,x) = -1, x \in \sum \\ V(0,j) = V(0,j-1) + \delta(-,T[j]) & with & \delta(x,-) = -1, x \in \sum \\ V(i,0) = V(i-1,0) + \delta(S[i],-) & & \delta(x,y) = 1, y = x \\ & & \delta(x,y) = -1, y \neq x \end{array} \tag{2}$$

The scoring function for Needleman-Wunsch algorithm is as shown in equation (1). Two sequences of S and T with their respective length n and m were used and, matrix of V of size n x m initialized as shown in equation (2). Each cell V(i,j) calculated the score of the best sub-alignments of S[1...i] and T[1...j] and the optimal alignment score was given by V(n,m).


Needleman-Wunsch

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

```
GCAT
GC-T
```

Smith-Waterman

	-	G	C	A	T
-	0	0	0	0	0
G	0	1	0	0	0
C	0	0	2	1	0
T	0	0	1	1	2

```
GC
GC
```

Figure 1: Example showing (a) optimal global alignment (b) optimal local alignment

Smith-Waterman:

$$V[i,j] = max \begin{cases} 0 & \text{align empty strings} \\ V[i-1,j-1] + \delta(S[i],T[i]) & \text{match/mismatch} \\ V[i-1,j] + \delta(S[i],-) & \text{delete} \\ V[i,j-1] + \delta(-,T[j]) & \text{insert} \end{cases} \tag{3}$$

$$V(0,0) = 0 \qquad and \qquad \begin{aligned} \delta(-,x) &= -1, x \in \sum \\ \delta(x,-) &= -1, x \in \sum \\ \delta(x,y) &= 1, y = x \\ \delta(x,y) &= -1, y \neq x \end{aligned} \tag{4}$$

The scoring function for Smith-Waterman is as shown in equation (3) with initialization V(0,0) = 0. It is a modification of Needlman-Wunsch algorithm. The implementation of algorithm used the same sized n x m matrix, but each index (i,j) was used to hold the maximum global alignment score for all substrings S[k....i] and T[h...j] where, $1 \leq k \leq i$ and $1 \leq h \leq j$. The score of the best local alignment was obtained as max(V(i,j)).

Burrows Wheeler Transform: A string S of n characters is transformed by the Burrows-Wheeler transformation by forming n cyclic shifts of S. These n permutations of S are then sorted in lexicographical order. An extra character ($), not in the alphabet of S, is added to keep track of the end of the original string. The BWT of S is then the concatenation

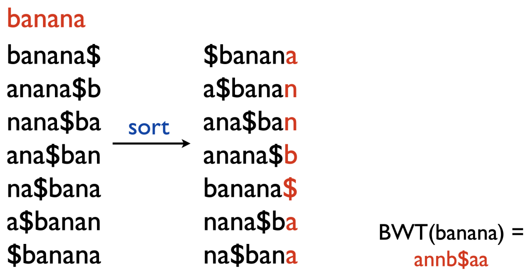


Figure 2: Example of BWT

of the last character of each permutation in sorted order, excluding $. In Figure 2 shows an example transformation of the string "banana". The list to the left in Figure 2 is the cyclically shifted permutations of S and the list to the right contains the same permutations, but in lexicographically sorted order. The result of the Burrows-Wheeler transformation is then the characters at the last index in each column, highlighted in bold in Figure 2. The Burrows-Wheeler transformation of a string S="banana" becomes BWT(S)= annb$aa. The original string is identified by having a $ at the end. Looking at BWT(S), we can see that equal characters are now grouped together.

Longest repeated substring: For a given string, the longest repeated substring was obtained obtained using suffix array and longest common prefix (LCP) array. At first, all the suffixes of the string are obtained and suffix array contains the starting position of the suffixes of a string when listed in lexicographic order. The suffix array index with the largest LCP is the index of the longest repeated substring as shown in figure 6.

Edit distance:

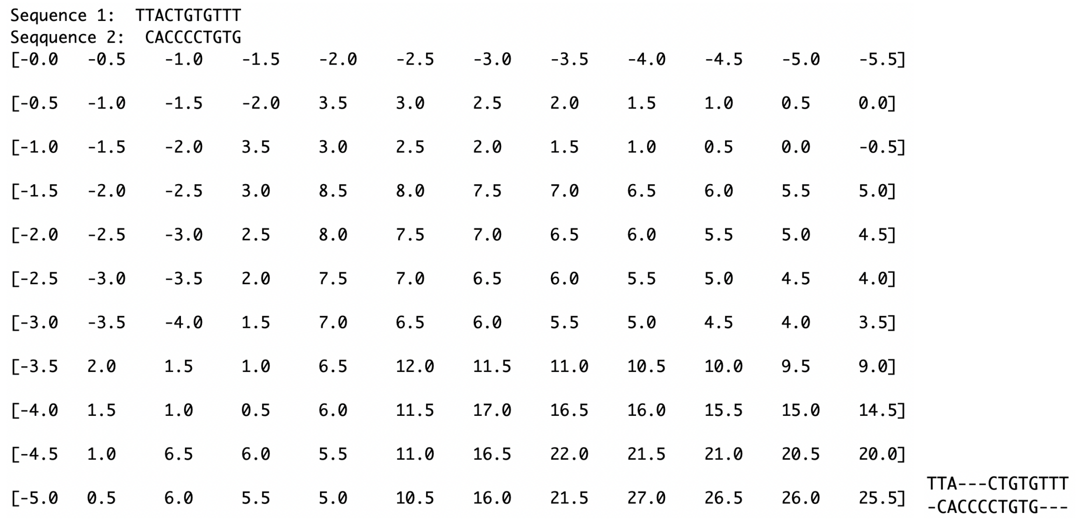
The recurrence relation for edit distance is:

$$V[i,j] = min \begin{cases} V[i-1,j-1] & \text{if } s_1[i] = s_2[j] \\ V[i-1,j-1]+1 & \text{if } s_1[i] \neq s_2[j] \\ V[i,j-1]+1 \\ V[i-1,j]+1 \end{cases} \tag{5}$$

This algorithm was also based on dynamic programming approach. A matrix was initialized measuring in the (m, n) cell the edit distance between the m-character prefix of one with the n-prefix of the other word. The matrix can be filled from the upper left to the lower right corner. Each jump horizontally or vertically corresponds to an insert or a delete, respectively. The cost is normally set to 1 for each of the operations. The diagonal jump can cost either one, if the two characters in the row and column do not match else 0, if they match. Each cell always minimizes the cost locally. This way the number in the lower right corner is the edit distance between both words. The intuition behind this algorithm is simple. If the last characters of both strings are the same, then the edit distance is equal to the edit distance of the same two strings, up to their second to last character, and if the last character is different, then the edit distance is equal to the minimum of the cost of inserting, deleting, or replacing the last character of the first string.

Results and Discussions

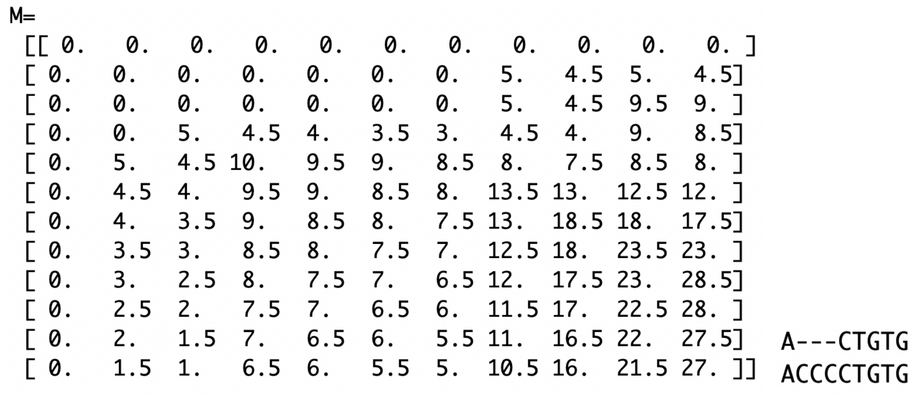
The results of five different algorithms is as shown in below. For an example, I have chosen two sequences (a) S = TTACTGTTT and (b) T = CACCCCTGTG as a pair of sequences to see the optimal global alignment, optimal local alignment, and minimum edit distance. Similarly, MISSISSIPPI and TCGATCGA were used for the implementation of BWT and longest repeated substring respectively. The time and space complexities of this method were



```
Sequence 1:  TTACTGTGTTT
Seqquence 2:  CACCCCTGTG
[-0.0  -0.5   -1.0   -1.5   -2.0   -2.5   -3.0   -3.5   -4.0   -4.5   -5.0   -5.5]

[-0.5  -1.0   -1.5   -2.0   3.5    3.0    2.5    2.0    1.5    1.0    0.5    0.0]

[-1.0  -1.5   -2.0   3.5    3.0    2.5    2.0    1.5    1.0    0.5    0.0    -0.5]

[-1.5  -2.0   -2.5   3.0    8.5    8.0    7.5    7.0    6.5    6.0    5.5    5.0]

[-2.0  -2.5   -3.0   2.5    8.0    7.5    7.0    6.5    6.0    5.5    5.0    4.5]

[-2.5  -3.0   -3.5   2.0    7.5    7.0    6.5    6.0    5.5    5.0    4.5    4.0]

[-3.0  -3.5   -4.0   1.5    7.0    6.5    6.0    5.5    5.0    4.5    4.0    3.5]

[-3.5  2.0    1.5    1.0    6.5    12.0   11.5   11.0   10.5   10.0   9.5    9.0]

[-4.0  1.5    1.0    0.5    6.0    11.5   17.0   16.5   16.0   15.5   15.0   14.5]

[-4.5  1.0    6.5    6.0    5.5    11.0   16.5   22.0   21.5   21.0   20.5   20.0]
                                                                                   TTA---CTGTGTTT
[-5.0  0.5    6.0    5.5    5.0    10.5   16.0   21.5   27.0   26.5   26.0   25.5]  -CACCCCTGTG---
```

Figure 3: (a) Scoring matrix obtained from Needleman-Wunsch implementation along with ptimal global alignment

$O(nm)$ for both. The running time complexity and space efficiency for Smith-Waterman was



```
M=
 [[ 0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0. ]
 [ 0.   0.   0.   0.   0.   0.   0.   5.   4.5  5.   4.5]
 [ 0.   0.   0.   0.   0.   0.   0.   5.   4.5  9.5  9. ]
 [ 0.   0.   5.   4.5  4.   3.5  3.   4.5  4.   9.   8.5]
 [ 0.   5.   4.5 10.   9.5  9.   8.5  8.   7.5  8.5  8. ]
 [ 0.   4.5  4.   9.5  9.   8.5  8.  13.5 13.  12.5 12. ]
 [ 0.   4.   3.5  9.   8.5  8.   7.5 13.  18.5 18.  17.5]
 [ 0.   3.5  3.   8.5  8.   7.5  7.  12.5 18.  23.5 23. ]
 [ 0.   3.   2.5  8.   7.5  7.   6.5 12.  17.5 23.  28.5]
 [ 0.   2.5  2.   7.5  7.   6.5  6.  11.5 17.  22.5 28. ]
 [ 0.   2.   1.5  7.   6.5  6.   5.5 11.  16.5 22.  27.5]  A---CTGTG
 [ 0.   1.5  1.   6.5  6.   5.5  5.  10.5 16.  21.5 27. ]] ACCCCTGTG
```

Figure 4: (a) Scoring matrix obtained from Smith-Waterman implementation along with ptimal local alignment

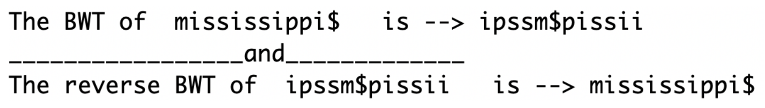
found similar to Needleman-Wunsch.

The table 1 is an example illustration of edit distance between two strings S and T. Both

Table 1: Showing the results of edit distance between two strings

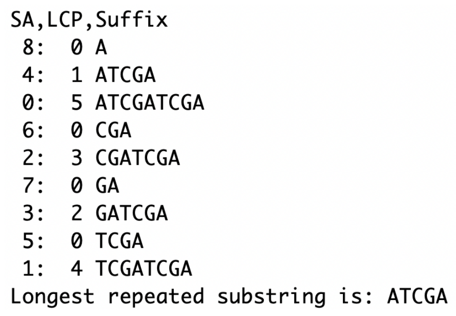
		C	A	C	C	C	C	T	G	T	G
	0	1	2	3	4	5	6	7	8	9	10
T	1	1	2	3	4	5	6	6	7	8	9
T	2	2	2	3	4	5	6	6	7	7	8
A	3	3	2	3	4	5	6	7	7	8	8
C	4	3	3	2	3	4	5	6	7	8	9
T	5	4	4	3	3	4	5	5	6	7	8
G	6	5	5	4	4	4	5	6	5	6	7
T	7	6	6	5	5	5	5	5	6	5	6
G	8	7	7	6	6	6	6	6	5	6	5
T	9	8	8	7	7	7	7	6	6	5	6
T	10	9	9	8	8	8	8	7	7	6	6
T	11	10	10	9	9	9	9	8	8	7	7

time and space complexities of edit distance algorithm were $O(nm)$. The recursive approach to calculate edit distance was also implemented in order to compare the running time with the DP approach. For same two strings, recursive approach took 10 seconds, where as DP took less than 1 second.



```
The BWT of  mississippi$   is --> ipssm$pissii
___________________and_____________
The reverse BWT of  ipssm$pissii   is --> mississippi$
```

Figure 5: Example of BWT and inverse BWT



```
SA,LCP,Suffix
 8:  0 A
 4:  1 ATCGA
 0:  5 ATCGATCGA
 6:  0 CGA
 2:  3 CGATCGA
 7:  0 GA
 3:  2 GATCGA
 5:  0 TCGA
 1:  4 TCGATCGA
Longest repeated substring is: ATCGA
```

Figure 6: Longest repeated substring of "ATCGATCGA"

Conclusion

Five different algorithms mainly based on dynamic programming were implemented and evaluated their time and space complexities. It is interesting to note that many of them had similar performances. Needleman-Wunsch was used to find the best alignment on entire inputs, where as Smith-Waterman computed the best alignments locally, i.e. between the substrings of the strings. Smith-Waterman was implemented after modification on Needleman-Wunsch and it could be bit more useful for tasks like determining the difference between DNA sequences. Edit distance, Burrows Wheeler transform, and longest repeated substring were also implemented. Suffix arrays were found memory efficient in comparison to suffix trees and could be used to compute the largest common prefix, which was eventually used to obtain the longest repeated substring. Edit distance was computed using both recursive method and DP in order to compare the running time. The recursive approach was found computational costly in comparison to dynamic programming.

References

(1) Liu, Y.; Hong, Y.; Lin, C.-Y.; Hung, C.-L. Accelerating Smith-Waterman Alignment for Protein Database Search Using Frequency Distance Filtration Scheme Based on CPU-GPU Collaborative System. *International journal of genomics* **2015**, *2015*, 761063–761063, 26568953[pmid].

(2) Needleman, S. B.; Wunsch, C. D. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology* **1970**, *48*, 443 – 453.

(3) Smith, T.; Waterman, M. Identification of common molecular subsequences. *Journal of Molecular Biology* **1981**, *147*, 195 – 197.

(4) Lipman, D.; Pearson, W. Rapid and sensitive protein similarity searches. *Science* **1985**, *227*, 1435–1441.

(5) Altschul, S. F.; Gish, W.; Miller, W.; Myers, E. W.; Lipman, D. J. Basic local alignment search tool. *Journal of Molecular Biology* **1990**, *215*, 403 – 410.

(6) Levenshtein, V. I. Binary Codes Capable of Correcting Deletions, Insertions and Reversals. *Soviet Physics Doklady* **1966**, *10*, 707.

(7) Burrows, M.; Wheeler, D. J. *A block-sorting lossless data compression algorithm*; Technical report 124, 1994.