Project Report for BLAST algorithms with Python
3180110985 3180111439

Abstract:

BLAST algorithms in this project aim to find regions of local similarity and distinguish the differences among amino acid sequences. The sequences are compared to the indexed databases including six species and undergo phases of seeding, hit calling and extending. Main algorithms include binary search, Needle Wunsch algorithms and Breadth-First Search (BFS) that are based on dynamic programming (DP). BLAST algorithms in our project provide accurate alignment results compared with online NCBI BLAST program (https://blast.ncbi.nlm.nih.gov/Blast.cgi) and display visualized information for the protein sequences with statistics values like E-values. We conclude that BLAST algorithm is an efficient tool for sequence analysis. Future improvements include filtering low complexity sequences and redundant protein alignment, dealing with genomes by using an extendable HDF5 dataset, and applying more scoring matrices for proteins with biased composition. Further, BLAST algorithms can be combined with machine learning in the near future.

Introduction to the question:

As the nucleotide and amino acid sequence databases keep growing and store more information at present, sequence analysis becomes more essential. Sequence alignment is the basis of sequence analysis as well as the core of bioinformatics. Search for sequence homology and sequence similarity against various species can provide the structural, functional, and evolutionary details of genes and proteins, which can be further used in biomedical research (Altschul et al., 1990).

Basic Local Alignment Search Tool (BLAST) finds regions of local similarity between reference and query sequences and distinguishes their differences. In our project, BLAST algorithms compare query protein sequences to the indexed databases, and then calculates the statistical significance of matches.

The source of proteome dataset in this project is from UniProt, with FASTA files of the listed six species (website link: https://www.uniprot.org/proteomes).

Species:	Proteome ID:
Caenorhabditis elegans	UP000001940
Drosophila melanogaster	UP000000803
Danio rerio	UP000000437
Gorilla gorilla	UP000001519
Mus musculus	UP000000589
Homo sapiens	UP000005640

Algorithm implementation:

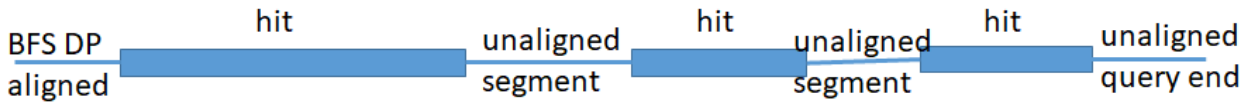
The input are amino acid sequences in FASTA format, which has a definition line and the sequence information (Fig1).

```
>sp|O95825|QORL1_HUMAN Quinone oxidoreductase-like protein 1 OS=Homo sapiens OX=9606 GN=CRYZL1 PE=1 SV=2
MKGLYFQQSSTDEEITFVFQEKEDLPVTEDNFVKLQVKACALSQINTKLLAEMKMKKDLF
PVGREIAGIVLDVGSKVSFFQPDDEVVGILPLDSEDPGLCEVVRVHEHYLVHKPEKVTWT
EAAGSIRDGVRAYTALHYLSHLSPGKSVLIMDGASAFGTIAIQLAHHRGAKVISTACSLE
DKQCLERFRPPIARVIDVSNGKVHVAESCLEETGGLGVDIVLDAGVRLYSKDDEPAVKLQ
LLPHKHDIITLLGVGGHWVTTEENLQLDPPDSHCLFLKGATLAFLNDEVWNLSNVQQGKY
LCILKDVMEKLSTGVFRPQLDEPIPLYEAKVSMEAVQKNQGRKKQVVQF
```

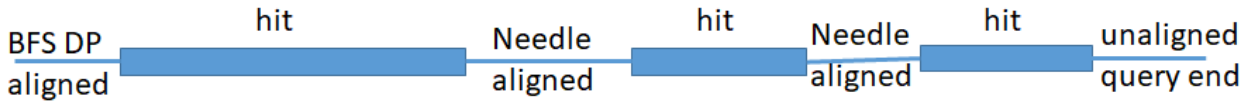
Fig1. Amino acid sequence in FASTA format. GN=GeneName. PE=ProteinExistence

Overall workflow:

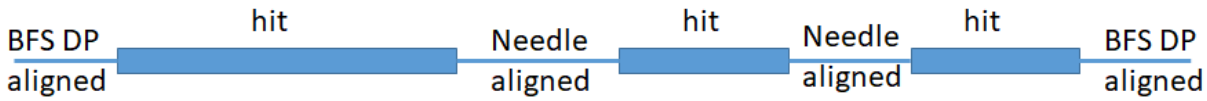
1. Identify direct hits. Hits occur within the same gene are combined.
2. Find local optimal alignment between unaligned query head and reference sequence.



3. Extract sequence pairs in the hit. Attach the sequence to previous obtained alignment.
4. Find global optimal alignment for unaligned segments between two direct hits. Attach the sequence to previous obtained alignment.
5. Repeat last two steps until no hits are left.



6. Find local optimal alignment between unaligned query head and reference sequence. Attach the sequence to previous obtained alignment. Calculate E-value for the obtained alignment. If E-value is smaller than the limit, append the result sequence pairs to result list.



7. Repeat step 2-6 for the remaining hits.
8. Sort the alignments in result list by similarity scores. Display alignments. Save results to a text file.

There are 4 major phases in BLAST algorithms: Indexing, seeding, hit calling and linking, and extending. The results are eventually evaluated and saved into a txt file.

The first two phases involve simple traversal. In the indexing phase, each amino acid in the database is matched with a number from 0 to 24 (Fig2). The protein sequences are split into 4-mers, which can be treated as a Base 25 number and be converted into a decimal number as the index of occurrence list (Fig3). The position of 4-mer is added to occurrence list entry with the corresponding index. Since the occurrence list of the seeds gets longer when reading the FASTA files, the indexed database is saved in a H5 file that the seeds are served as keys. Other files that will be used in the following steps are also generated (Table 1).

A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V	B	J	Z	X	U
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

Fig2. Each AA is matched with a number from 0 to 24.

```
def word_to_num(letters):
    word_num = {}
    for i,l in enumerate(letters):
        word_num[l] = i
    return word_num # dictionary, match AA to number (0~24)

def word_to_idx(word_num,word,word_len): #convert seeds to occurrence list index
    word_index = 0
    for i,l in enumerate(word,1):
        word_index += (len(word_num))**(word_len-i)*word_num[l]
    return word_index
```

Fig3. 4-mers are converted to decimal numbers.

	Format	Data
Input	Fasta	Proteome of 6 organism
Output	h5	Occurrence list of 4-mers
Output	txt	_seq.txt (combined database sequence)
Output	numpy array	_seq_idx.npy (start point of each sequence in _seq.txt)
Output	numpy array	_seq_name.npy (gene name of each sequence)
Output	numpy array	_species_idx.npy (start point of each organism in _seq.txt)
Output	numpy array	_seq_name.npy (name of each organism)

Table 1. The format of input and output data.

In the seeding phase, we split query sequence to 4-mers list and obtain the seed occurrence for each 4-mer (Fig4). By viewing the hit map we can find some seeds occur in a cluster. Continuous seeds on a diagonal line are considered as a fully matched hit (Fig5A).

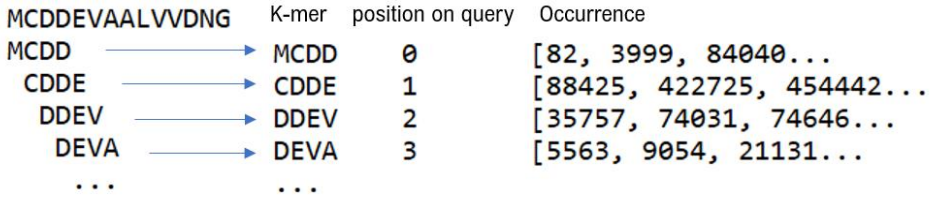


Fig 4A. Query sequence is spilt to 4-mer list and the seed occurrence is obtained.

```
def get_seed_position(self,seeds):
    positions = []
    for seed in seeds:
        try:
            pos = self.indexed_db[seed][:] #get occurence list with seed as key
        except KeyError:
            pos = [] #the seed did not occur in database sequence
        positions.append(pos)
    return positions
```

Fig 4B. Codes for obtaining the seed occurrence

Algorithms based on DP are implemented in the next two phases to allow gaps in the high scoring pairs. Compared with exhaustive search and local optimization, DP is exact and efficient. It is also willing to trade space for time as computed results are stored to avoid re-computation.

In the hit calling phase, the denoising step counts seeds on each diagonal line. If the seed number is smaller than the given number H which is set 5, the seeds are discarded. If the number of continuous seeds is larger than the given number N on the same diagonal line, the residues can be considered as one hit (Fig 5A). The hit length and the start point of query and reference sequences are stored. Binary search algorithms are used when finding adjacent hits in the same gene and when defining gene names of the matched sequences. With the low time complexity of O (log n), it repeatedly divides the search intervals into half until find the targeted values quickly.

During hit linking, if the distance of two hits is smaller than given number A and the hits occur within the same gene, they are combined as one hit. Unmatched sequence pairs between the two hits are extracted and aligned according to Needleman-Wunsch algorithm (Needleman and Wunsch, 1970) (Fig 5B).

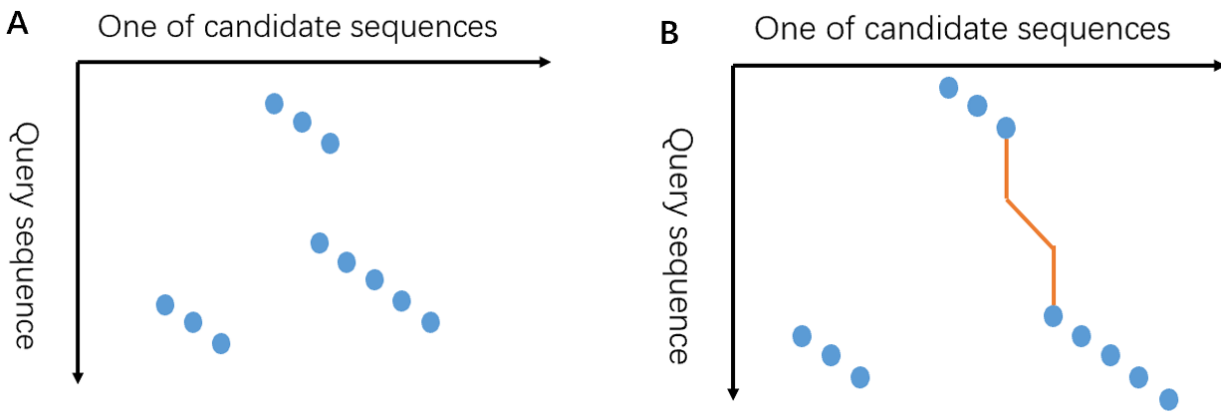


Fig5. A: Hit map of seeds. B: Two close hits are combined.

Needleman-Wunsch algorithm is a well-known algorithm for global alignment with the time complexity of O(nm). We set the scoring matrix BLOSUM62, gap open penalty -10 and gap extend penalty -0.5 as parameters. Initially, different length of the gaps should be considered when computing optimal scores for each cell. To reduce the time complexity, we used three matrices M, X and Y. Matrix M represents score obtained by substitution, X and Y represents deletion and insertion. All three matrices have a length of len(Q) (query sequence) and a width of len(S) (reference sequence). The values of first column of Matrix X and M and the first row are calculated (Fig6). Value of the rest cells are calculated based on the following formulas:

$$M(i,j) = \max\left(M(i-1,j-1), X(i-1,j-1), X(i-1,j-1)\right) + \text{Substitution}(\text{query}(i-1), \text{refseq}(j-1))$$

$$X(i,j) = \max\left((Open\ gap), \max\left(M(i,j-1), Y(i,j-1)\right) - \text{gap penalty},\right.$$
$$\left.(\text{Extend gap})\ X(i,j-1) - \text{gap extend penalty}\right)$$

$$Y(i,j) = \max\left((Open\ gap), \max\left(M(i-1,j), X(i-1,j)\right) - \text{gap penalty},\right.$$
$$\left.(\text{Extend gap})\ Y(i-1,j) - \text{gap extend penalty}\right)$$

During traceback, we create a new matrix to traceback the same size of scoring matrices. For traceback(i,j), we compare between M(i,j), X(i,j) and Y(i,j). The value of traceback(i,j) is mostly decided by the max value of M(i,j), X(i,j) and Y(i,j). But if previous executed cell is traceback (i, j+1) X(i,j+1) is equal to X(i,j) – gap extend penalty, traceback(i,j) is set as “gap in query” preferentially. If previous executed cell is traceback (i+1, j) Y (i+1, j) equal to Y(i,j) – gap extend penalty, traceback(i,j) is set as “gap in refseq” (Fig7).

Matrix X

j	0	1	2	3
0	g	g+e	g+2e	g+3e
g	g+g			
g+e	g+g+e			
g+2e	g+d+2e			
g+3e	g+g+3e			

Matrix M

j	0	1	2	3
0	g	g+e	g+2e	g+3e
g	sub(0,0)	sub(1,0)+g	sub(2,0)+g+e	sub(3,0)+g+2e
g+e	sub(0,1)+g			
g+2e	sub(0,2)+g+e			
g+3e	sub(0,3)+g+2e			

Matrix Y

j	0	1	2	3
0	g	g+e	g+2e	g+3e
g	g+g	g+g+e	g+d+2e	g+g+3e
g+e				
g+2e				
g+3e				

g=gap penalty
e=entend penalty
sub=substituion score (Qi+Sj)

Fig6A. Initial value of Matrices M, X and Y.

```
    j = 1
    while j < n:
        i = 1
        S_base = S[j]

        p_pointer = (0,j-1)
        pointer = (0,j)
        j += 1
        while i < m:
            match = self.matrix[Q[i]+S_base]
            i += 1
            pointer = (pointer[0]+1,pointer[1])

            # for matrix mt, p_pointer = upper left
            mt[pointer] = max(mt[p_pointer],x[p_pointer],y[p_pointer]) + match

            # for matrix y, p_pointer = upper
            p_pointer = (p_pointer[0], p_pointer[1]+1)

            open_gap = max(x[p_pointer],mt[p_pointer]) + self.gap_penalty
            extend_gap = y[p_pointer] + self.extend_penalty

            if open_gap > extend_gap:
                y[pointer] = open_gap
            else:
                y[pointer] = extend_gap

            # for matrix x, p_pointer = left
            p_pointer = (p_pointer[0]+1, p_pointer[1]-1)

            open_gap = max(mt[p_pointer],y[p_pointer]) + self.gap_penalty
            extend_gap = x[p_pointer] + self.extend_penalty

            if open_gap > extend_gap:
                x[pointer] = open_gap
            else:
                x[pointer] = extend_gap
```

Fig6B. Codes for Needleman-Wunsch algorithm (Needleman and Wunsch, 1970).

```
p_pointer = 0 # direction of previous step

i = m - 1
j = n - 1

trace = np.empty((m,n), dtype=int) # vertical:Length of s1(query)  horizontal:Length of s2(Refs

while i>=0 and j>=0:
    pointer = (i,j)
    mp = mt[pointer]
    # previous step is 1 and the right value - this value in x matrix is extend penalty
    if p_pointer == 1 and self.extend_penalty == x[pointer[0],pointer[1]+1]-x[pointer]:
        trace[pointer] = 1 #Left, gap(s) in query sequence
        j -= 1
    # previous step is 2 and the lower value - this value in y matrix is extend penalty
    elif p_pointer == 2 and self.extend_penalty == y[pointer[0]+1,pointer[1]]-y[pointer]:
        trace[pointer] = 2 #Up, gap(s) in reference sequence
        i -= 1
    elif mp >= x[pointer] and mp >= y[pointer]: #mp is max
        trace[pointer] = 0; # Diagonal, matched
        i -= 1
        j -= 1

    elif x[pointer]>=y[pointer] and j > -1: #x[pointer] is max
        trace[pointer] = 1
        j -= 1
    elif i > -1: #y[pointer] is max
        trace[pointer] = 2
        i -= 1
    else:
        print("Warning: Error encountered in Needleman-Wunsch traceback algorithm")
        return -1
    p_pointer = trace[pointer]
```

Fig7. Codes for traceback algorithms.

If there are unaligned query ends, we need to find their optimal local alignment in the reference database. The length of reference sequence is undefined and can be much longer than the final aligned sequence. In the hit extending phase, we loop through the database in BFS order. While it is unable to prevent endless searching on amino acid sequences in Depth-First Search (DFS) order which loops through matrix column by column, BFS is a preferred choice to establish state transition equations and find the stop point of extending hits in our project. Based on BFS, we first complete cells closest to the start point, and scoring matrices expand with each loop (Fig8).

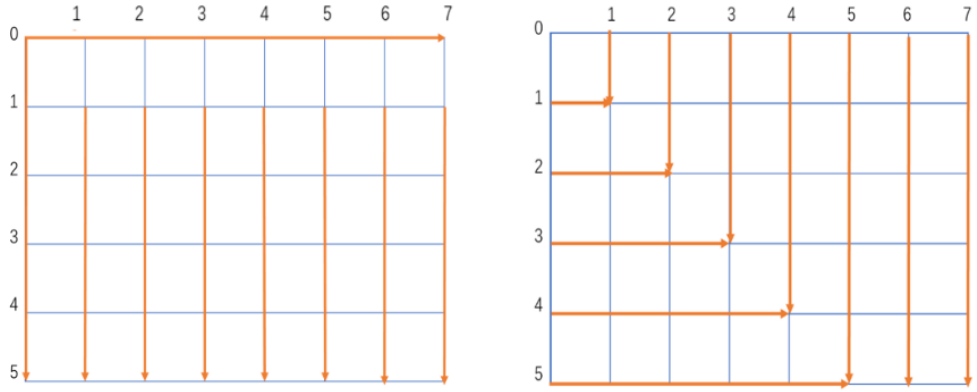


Figure 8A. Comparison between DFS (left) and BFS (right) in hit extending phase.

```
set mt, x, y as 2d lists (unfixed size)
pointer = 1
While optimal score > threshold:
    if pointer == length of reference sequence:
        retrive more sequence from database
        if reference sequence before end of protein sequence have all been extracted and pointer >= query length:
            break loop
    for 0 in 1 to pointer-1:
        if pointer < query length:
            compute value for mt[pointer][i], x[pointer][i], y[pointer][i]
        if pointer < reference seq length:
            compute value for mt[i][pointer], x[i][pointer], y[pointer][i]
    if pointer < reference seq length and pointer < query length:
        compute value for mt[pointer][pointer], x[pointer][pointer], y[pointer][pointer]
    update max score
    pointer += 1
```

Fig 8B. BFS algorithm pseudocodes in hit extending

During the extending phase, the value and position of the max scoring cell keeps updating. When the optimal alignment score drops below max score minus X, the extension stops and returns the value and position of the max scoring cell (Fig9). The alignment pairs are generated by tracing back the scoring matrices and then attached to the corresponding end of hits.

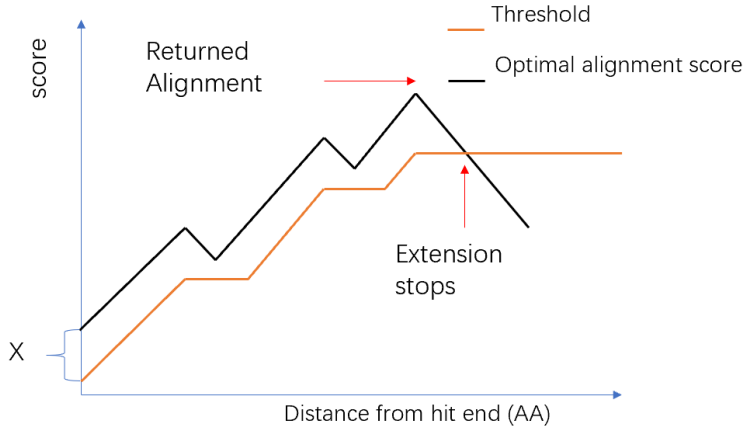


Fig9. Changes of optimal local alignment and the threshold.

Results:

The data type belongs to FASTA format representing amino acid sequence information. The sequence data are indexed and saved initially.

Species	Genes	K-mers	Time (seconds)	Memory usage (MB)
All	271574	168007	259.9	5142.6
Drosophila	22040	160888	53.2	735.2
Danio rerio	46848	162262	70.9	1124.0
C.elegans	26702	159464	48.9	627.4
Gorilla	44726	160748	68.7	1072.8
Mus musculus	55486	162888	69.6	1080.0
Homo sapiens	75777	164082	75.0	1187.2

Table 2. The results of indexing databases. Hardware information: CPU: CPU i7-10875 RAM 16GB

The results are displayed by order of similarity scores from the highest with identity percentages and E-values. Matches with E-values that are larger than e^{-10} will not be displayed in the results. Below these evaluation values in the header are visualized sequence alignments, including gaps (-), same matches (|), similar amino acids (:) and amino acids with relatively big differences (.). The results should be set in consolas font, otherwise the visualization would be in wrong format. The running time of BLAST algorithm is about 1 to 5 seconds for amino acids in various length.

We select an unnamed protein sequence in *Mus musculus* found in NCBI.

The Information of the unnamed protein:

>CAO79487.1 unnamed protein product [Mus musculus]

MMEGLSPASSLPLLLLLLSPAPEAALPLPSSTSCCTQLYRQPLPSRLLRRIVHMELQEADGDCHLQAVVLHLA
RRSVCVHPQNRSLARWLERQGKRLQGTVPSLNLVLQKKMYSNPQQQN

And the results in txt file is shown below. Detailed information is in the supplementary files.

```
Query = MMEGLSPASSLPLLLLLLSPAPEAALPLPSSTSCCTQLYRQPLPSRLLRRIVHMELQEADGDCHLQAVVLHLARRSVCVHPQNRSLARWLERQGKRLQGTVPSLNLVLQKKMYSNPQQQN
Similarity score matrix: BLOSUM62
Use settings: gap penalty=-10 gap extend penalty=-0.5 X=-15 A=70 H=4 N=2
Display max 20 matches.
_____

Mus_musculus Ccl27a Score:618 Query cover:100% Identity:(120/120)1.00 Gap:(0/120)0.00 Expect:9.50e-126
_____

      0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17
Query  M  M  E  G  L  S  P  A  S  S  L  P  L  L  L  L  L  L
       |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
Sbjct  M  M  E  G  L  S  P  A  S  S  L  P  L  L  L  L  L  L

      18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35
Query  S  P  A  P  E  A  A  L  P  L  P  S  S  T  S  C  C  T
       |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
Sbjct  S  P  A  P  E  A  A  L  P  L  P  S  S  T  S  C  C  T

      36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53
Query  Q  L  Y  R  Q  P  L  P  S  R  L  L  R  R  I  V  H  M
       |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
Sbjct  Q  L  Y  R  Q  P  L  P  S  R  L  L  R  R  I  V  H  M

      54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71
Query  E  L  Q  E  A  D  G  D  C  H  L  Q  A  V  V  L  H  L
       |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
Sbjct  E  L  Q  E  A  D  G  D  C  H  L  Q  A  V  V  L  H  L

      72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89
Query  A  R  R  S  V  C  V  H  P  Q  N  R  S  L  A  R  W  L
       |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
Sbjct  A  R  R  S  V  C  V  H  P  Q  N  R  S  L  A  R  W  L

      90  91  92  93  94  95  96  97  98  99  100 101 102 103 104 105 106 107
Query   E   R   Q   G   K   R   L   Q   G   T   V   P   S   L   N   L   V   L
        |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
Sbjct   E   R   Q   G   K   R   L   Q   G   T   V   P   S   L   N   L   V   L

      108 109 110 111 112 113 114 115 116 117 118 119
Query   Q   K   K   M   Y   S   N   P   Q   Q   Q   N
        |   |   |   |   |   |   |   |   |   |   |   |
Sbjct   Q   K   K   M   Y   S   N   P   Q   Q   Q   N
```

Discussion:

There are several highlights in our project. First, H5 format we used can read data rapidly with the low time complexity of O(1) and prevent reading the whole file into RAM. Second, we consider the borders of amino acid sequences when computing BLAST algorithms because amino acid sequences are smaller than nucleotide sequences, otherwise alignments may occur in the head of one sequence and the tail or other sequences. In addition, BLAST algorithms involve many parameters that can be adjusted in “other options” (Fig10). The lower the values of H and N, the higher the sensitivity. While lower H and N values are more likely to produce false positive results, we set E-value threshold to weep out these noisy data.

```
-g      gap penalty
-e      extend penalty
-l      max number of displayed alignment
-x      during extending, lowest score accepted below best score
-a      max distance between two hits when they can be considered as one hit
-h      during denoising, min number of seeds on a diagonal line that can be collected for furthur analysis
-n      during hit calling, min continuous seeds that can be considered a hit
-m      substitution matrix name
-d      database path(without suffix)
```

Fig 10. Parameters in BLAST algorithms which are not case sensitive.

E-value equal to $Kmne^{-\lambda S}$ is associated with two parameters, K and λ. While some articles and websites give clues that they are normalizing constants related to the search space size and the scoring system respectively, little evidence is found to determine the exact K and λ values. By comparing the E-values of the same query sequence input using NCBI BLAST online program, we decide to let K be 0.1 and λ be 0.5 in our project. When coming to biological interpretation, we do not worry too much about the underlying mathematics because they should be used to filter redundant data and have little use beyond this.

Speaking of redundant data, we should make improvement in data cleaning. Before seeding and extending, low complexity sequences that may generate false positive results are not filtered in our project. A filter step should be added to our project when indexing the database. On the other hand, the results may contain proteins with the same name in one organism, though their positions in the database are different and may be distant. The replication should be excluded and remain only one amino acid sequence (Fig11).

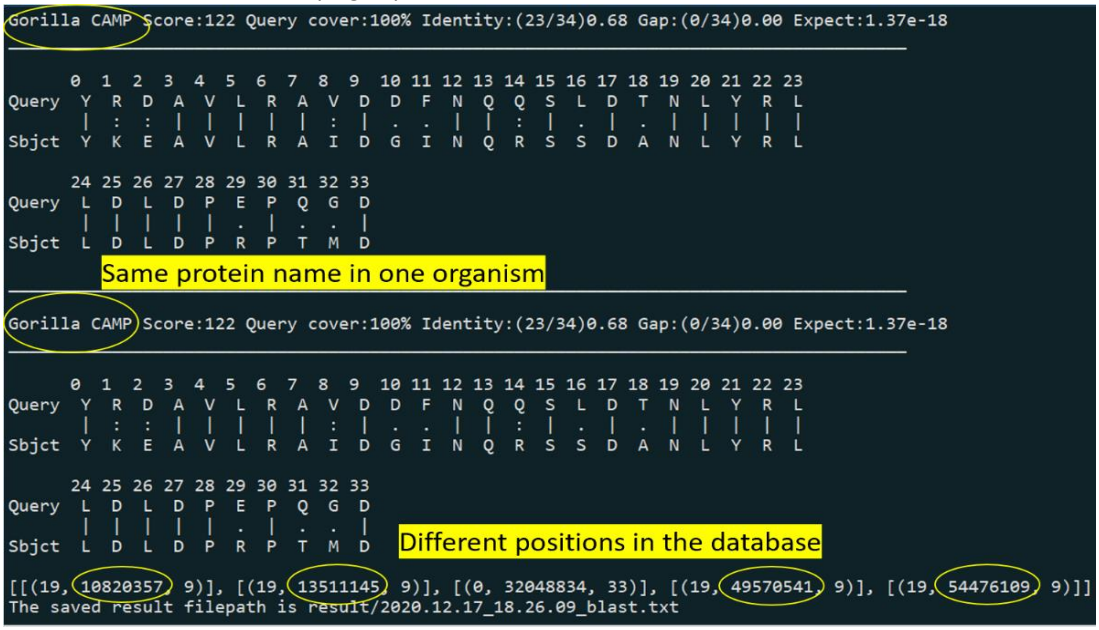


Fig11. Redundant data should be removed during indexing phase.

To broad the field of BLAST algorithms, our project can be improved to deal with both nucleotide sequences and amino acid sequences in the future. The primary challenge is running out of memory as genome sequence is much longer (~2.9GB) compared to proteome sequence (~30MB). Currently we store occurrence list in the RAM. Since an occurrence list converted from 100MB FASTA files occupies about 500MB RAM, we assume that memory overflow might be encountered when generating indexed database from multiple genome files. The solution is to use HDF5 dataset with an extendable size (Fig12).

```
with h5py.File("test.h5", "w") as f:
    position = 0
    for each k-mer:
        try:
            d = f[k-mer]
            d.resize(current length + 1,)
            d[current length:current length + 1,] = position
        except KeyError:
            d = f.create_dataset(k-mer, (1,), maxshape=(None,))
        position += 1
```

Fig12. Algorithms pseudocode that reduce the RAM used during indexing

The alignment accuracy and search sensitivity in BLAST algorithms are also closely associated with the type of scoring matrices for proteins. While most ordered proteins have standard amino acid compositions, some kinds of proteins such as transmembrane proteins contain enriched amino acids with certain types. Therefore, the specific scoring matrices should be chosen based on the compositions of proteins and their different residue substitution patterns (Trivedi and Nagarajaram, 2020). Common scoring matrices include “deep” scoring matrices like BLOSUM62 and shallower scoring matrices like PAM250 (Pearson, 2013). BLOSUM matrices are the primary choice of scoring matrixes when aligning proteins, because it introduces less misaligned residues than the extrapolated PAM matrices. In addition, specialized scoring matrices are being developed to identify classes of proteins with biased amino acid compositions, like transmembrane/cysteine-rich proteins substitution matrices (Trivedi and Nagarajaram, 2020). Thus, our future improvement can apply not only general matrices for proteins with unbiased amino acid compositions like BLOSUM and PAM matrix series, but also these new matrices for specific categorical proteins in database search and sequence alignments.

A more advanced improvement is to combine BLAST algorithms with natural language processing (NLP). Many NLP systems consist of character sets with each language having its own sets of words and each word having respective meanings. An idea is to refer to the strategy of computing text similarity and consider proteome as a kind of language with sets of biologically functional “words”. While experiments have shown that genome has functional site segments such as transcription factor binding sites that connect the gene “words”, proteome may lack these specific segments. The feasibility of this combined approach requires more studies in the future.

Source code:

Language version: Python 3.8

Packages and the versions:

Numpy v1.19.2

Pandas v1.1.3

h5py v2.10.0

Reference:

Altschul, S., Gish, W., Miller, W., Myers, E. and Lipman, D., 1990. Basic local alignment search tool. *Journal of Molecular Biology*, 215(3), pp.403-410.

Ncbi.nlm.nih.gov. 2020. *The Statistics Of Sequence Similarity Scores*. [online] Available at: <https://www.ncbi.nlm.nih.gov/BLAST/tutorial/Altschul-1.html#ref14> [Accessed 16 December 2020].

Needleman, S. and Wunsch, C., 1970. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48(3), pp.443-453.

Pearson, W., 2013. Selecting the Right Similarity‐Scoring Matrix. *Current Protocols in Bioinformatics*, 43(1).

Smith, T. and Waterman, M., 1981. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147(1), pp.195-197.

Trivedi, R. and Nagarajaram, H., 2020. Substitution scoring matrices for proteins‐An overview. *Protein Science*, 29(11), pp.2150-2163.