475/575 Final Report

Group 35: Junhao Wang, Weiren Lai
GitHub repo: https://github.com/WrLai/475FinalProject

Abstract

In this project, we will complete the implementation of sequence similarity calculation algorithms. The specific algorithm is Levenshtein Distance Algorithm, which is encapsulated as a user-defined function, and compares it with adist hash and stringdist functions in R language. The RNA sequences of each country were extracted, and the similarity of RNA sequences in two consecutive days was calculated to find out the time when the virus RNA sequence occurred in larger pictures. China, the United States and Australia were taken as the main analysis objects.

Introduction

The covid-19 pandemic that broke out at the beginning of this year has had a serious impact on our lives and caused a global economic setback. In order to better control the epidemic, many scientists have done their best to understand the structure and composition of the virus in order to better formulate effective plans for it. One of the key proteins required by a virus to infect its host is called a spike protein. Its job is to bind to certain receptors in the host cell to access its machinery and achieve infection and reproduction. These spike proteins are considered to be one of the key goals of vaccine and antiviral drug development, so understanding their structure can provide valuable insights for fighting viruses.

Problem Definition

Visually show how the Covid-19 spike protein sequence will change during the pandemic.

1. At different times and locations, look at the specific mutations in the spike protein sequence and visualize them.

2. To understand the changes in the sequence of the spike protein by looking at the number of different changes in the specific mutation on the spike protein in different

time frames.

3. To understand the changes of the spike protein sequence by looking at the mutation sequence that is different from the known dominant sequence in a fixed time period.

Models/Algorithms/Measures

We will quantitatively calculate the similarity of two sequences, and then analyze whether they have obvious mutation in a certain period of time or in a certain region, that is, the change of similarity exceeds a certain threshold. We will quantitatively calculate the similarity in time and space for comparison and statistics, and finally analyze and draw a conclusion.

The similarity of sequences can be quantitative or qualitative. Similarity is a numerical value, reflecting the similarity of two sequences. As for the relationship between the two sequences, there are many nouns, such as identical, similar, homologous, homologous, direct homologous, symbiotic and homologous. The two concepts of "homology" and "similarity" are often used in sequence comparison, which are often confused. Homology of two sequences means that they have a common ancestor. In this sense, regardless of the degree of homology, the two sequences are either homologous or not homologous. For example, the similarity of the two sequences reaches 30% or 60%. Generally speaking, two sequences with high similarity often have homology. However, there are also exceptions, that is, the similarity between the two sequences is very high, but they may not be homologous sequences. The similarity between the two sequences may be caused by random factors, which is called "convergence" in evolution, and such a pair of sequences can be called homologous sequences. Orthologous sequences are from different species and genera, while paralogous sequences are from the same genus. They are generated by sequence replication in the process of evolution.

The basic operation of sequence comparison is alignment. The alignment of two sequences refers to a one-to-one correspondence between the characters in the two sequences, or the comparative arrangement of characters. Sequence alignment is a qualitative description of sequence similarity, which reflects where two sequences are similar and where there are differences. The optimal ratio pairs reveal the maximum similarity of the two sequences and point out the fundamental differences between the sequences.

Sequence comparison can be divided into four basic situations. The specific tasks and applications are as follows:

(1) Suppose there are two sequences of similar length from the same alphabet. They are very similar, but there are only some subtle differences, such as character insertion, character deletion and character replacement. It is required to find out the differences between the two sequences. For example, there are two laboratories simultaneously determining the DNA sequence of a gene, and the results may be different, so we need to compare the experimental results through sequence comparison.

(2) If there are two sequences, it is required to judge whether the prefix of one sequence is similar to the suffix of the other sequence. If so, the prefix and suffix are taken out respectively. This procedure is often used to assemble sequence fragments in large-scale DNA sequencing.

(3) If there are two sequences, it is required to judge whether one of them is a subsequence of the other. This operation is often used to search for specific sequence patterns.

(4) If there are two sequences, it is required to judge whether there are very similar subsequences in the two sequences. This operation can be used to analyze conservative sequences.

Of course, when comparing sequences, we often need to explain whether to adopt global comparison or local comparison. Global comparison is to compare two complete sequences, while local comparison is to find the most similar subsequence.

Two DNA sequences were observed: GCATGACGAATCAG and TATGACAAACAGC. At first glance, there is no similarity between the two sequences. However, if the second sequence is shifted by one bit and compared, the similarity can be found.

```
GCATGACGAATCAG
  ||||| ||
 TATGACAAACAGC
```

If we add a short line to the second sequence, we will find that the two sequences have more similarities.

```
GCATGACGAATCAG
  ||||| || |||
 TATGAC-AAACAGC
```

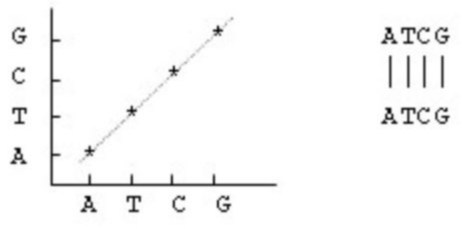
The above is a qualitative representation of the similarity between two sequences. In order to illustrate the similarity degree of the two sequences, quantitative calculation is also needed. There are two methods to quantify the similarity of two sequences: one is similarity, which is a function of two sequences. The larger the value is, the more similar the two sequences are; the other concept corresponding to similarity is the distance between two sequences, and the greater the distance, the smaller the similarity of two sequences. In most cases, similarity and distance can be used interactively, and the larger the distance, the smaller the similarity, and vice versa. But in general, similarity is used more frequently and is flexible.

The simplest distance is Hamming distance. For two sequences of equal length, Hamming distance is equal to the number of characters in corresponding positions. For example, the following figure shows the calculation results of Hamming distance for three groups of sequences.

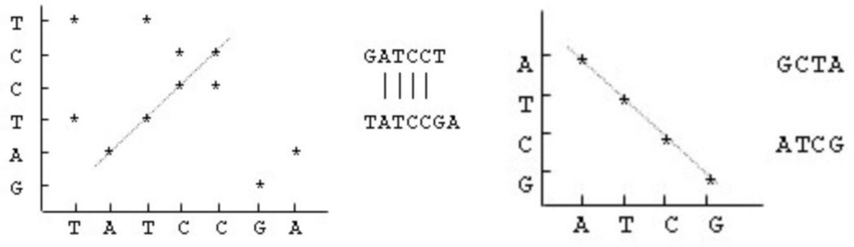
s =	AAT	AGCAA	AGCACACA
t =	TAA	ACATA	ACACACTA
Hamming Distance(s,t)=	2	3	6

It is not flexible to use the distance to calculate, because the sequence may have different length, and the characters in each position in two sequences are not necessarily the real correspondence. For example, in the process of DNA replication, errors such as deleting or inserting a base may occur. Although the other parts of the two sequences are the same, the Hamming distance is distorted due to the shift of position. In the case on the far right of the example in Fig. 3.1, Hamming distance is 6. From the perspective of Hamming distance, the two sequences are quite different (the length of the whole sequence is only 8 BP). However, if G is deleted from s and t is deleted from t, then both sequences become acaacaacaca, which indicates that there is only two characters difference between the two sequences. In fact, in many cases, it is unreasonable to use Hamming distance to measure the similarity of two sequences.

Another method for sequence comparison is matrix mapping or diagonal mapping, which was first proposed by Gibb. Place two sequences to be compared on the two axes of the matrix, one on the X axis, from left to right, and the other on the Y axis, from bottom to top, as shown in Figure 3.2. When the corresponding row and column sequence characters match, a "dot" mark is made at the corresponding position of the matrix. All character pairs are compared one by one to form a point matrix.



Obviously, if two sequences are identical, then there is a marker at the position of the main diagonal of the point matrix; if two sequences have the same substring, then for each same substring pair, there is a diagonal line composed of marked points parallel to the diagonal, for example, the diagonal line in figure a represents the same substring "ATCC"; for two sequences that are opposite to each other, they are in the anti diagonal There is a diagonal line composed of marked points in the direction, as shown in the figure below.



The non overlapping diagonal parallel lines in the matrix marker graph can be combined to form an alignment of two sequences. In the middle of two subsequences, the symbol "-" can be inserted to indicate the insertion of empty characters. In this comparison, the similarity of the two sequences is analyzed, as shown in the figure below. To find the best alignment of two sequences (the corresponding positions have the most equivalent characters) is to find the longest combination of non overlapping parallel slashes in the matrix marker graph.

Implementation/Analysis

For this project, a set of Covid-19 spike protein data collected between January 1st and August 7th of 2020. The data include the following attributes: accession ID: a code assigned to each published sequence by the popluar data repository Genbank, RNA sequence, collection date, and for some sequences collection location. These data all come from all the sequences in the data set we collected in the United States.
The algorithm we use to calculate the similarity of RNA sequences is Levenshtein Distance Algorithm, also known as edit distance algorithm. It refers to the minimum number of editing operations required to convert one string into another between two strings. Permitted editing operations include replacing one character with another, inserting a character, and deleting a character. Generally speaking, the smaller the editing distance, the greater the similarity between the two strings.
The algorithm needs the idea of dynamic programming, which is usually based on a recursive formula and one or more initial states. The solution of the current subproblem will be deduced from the solution of the previous subproblem. So our first goal is to find a state and a recursive formula. Suppose we can use d [x, y] steps

to convert the string x [1... i] to the string y [1…j] minimum number of steps required. In the simplest case, that is, when I = 0, that is to say, if the string x is empty, then the corresponding d [0, j] is x, adding J characters, i.e., it needs j steps to convert x to y; when j equals 0, that is, the string y is empty, then the corresponding d [i,0] is x, and i-step is needed to convert x to y. This is the minimum number of steps required.

Then we go further. If we want to convert x [1... i] to y [1... j] with the least number of operations, we can consider three cases:

(1) Suppose we can convert x [1... i] to y [1... j-1] in at least a steps, then we only need to add x [1... i] and Y [j] to complete the conversion of x [1... i] to y [1... j], so that x to y requires a + 1 step.

(2) Suppose that we can convert x [1... I-1] to y [1... J] in at least B steps, then we only need to delete x [i] to complete the conversion of x [1... i] to y [1... j], so that x to y requires B+1 step.

(3) Suppose that we can convert x [1... i-1] to y [1... j-1] in at least k steps, then we need to judge whether x [i] and y[j] are equal. If they are equal, we only need K steps to complete the conversion of x [1... i] to y [1... j]; if x [i] and Y [J] are not equal, we need to replace x [i] with y [J], so we need K + 1 steps to convert x [1... i] to y [1... j].

In these three cases, the operation of adding, deleting or replacing can be done at least in the previous state, so that the conversion from x [1.. i] to y [1.. j] can be completed only once or no operation is needed for strings X and y. Finally, in order to ensure that the minimum number of steps is required in the current state (x [1… i] to y [1…j]), we need to select the least step from the above three cases as the minimum number of steps required to convert x [1... I] to y [1... j]. If x [i] and y [j] are equal, then EQ = 0, otherwise EQ= 1.

The specific algorithm steps are as follows:

(1) Construct a two-dimensional array with m+1 rows and n+1 columns, which is used to save the minimum number of steps required to complete a string conversion, and convert the string x [1… m] to the string y [1 … n]. The minimum number of steps required is d [m][n]

(2) The 0th row of the initialization d matrix is 0 to n, and the 0th column of matrix d

is 0 to m.

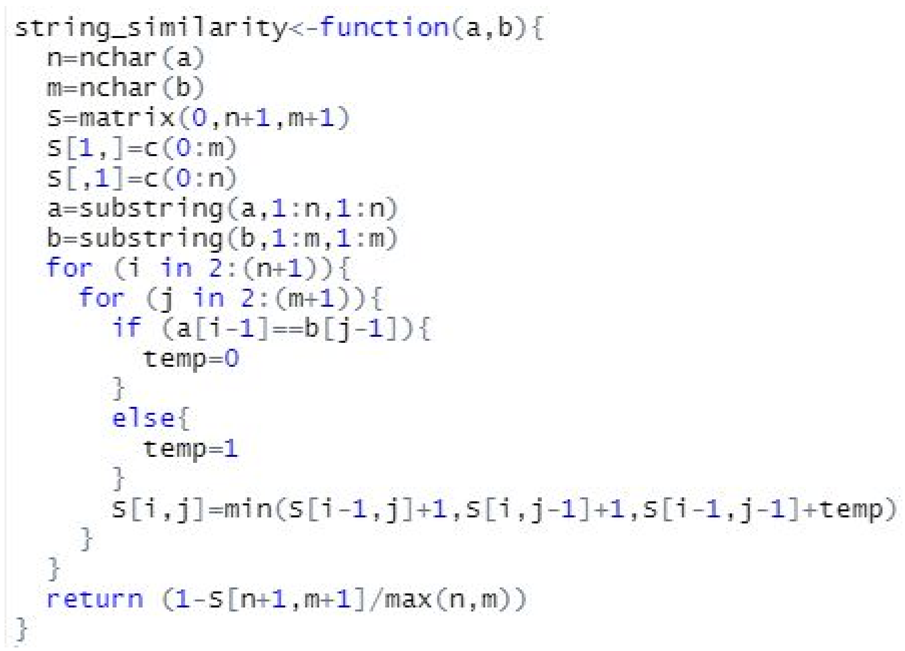
(3) If x [i] = y[j], then eq = 0, otherwise eq = 1. The recurrence formula is

d [i, j] = min (d [i-1, j], d [i, j-1], d [i-1, j-1] + EQ)

(4) D [m+1, n+1] is the distance between the string x and the string y

(5) The similarity calculation formula is s = 1-d/ max (m, n)

In the R program, we customize a name called string_ The input parameters are two strings. In the function, I implement Levenshtein Distance Algorithm. The final output parameter is the similarity of two input strings, and the value range is 0 to 1. If the two strings are exactly the same, it should output 1.



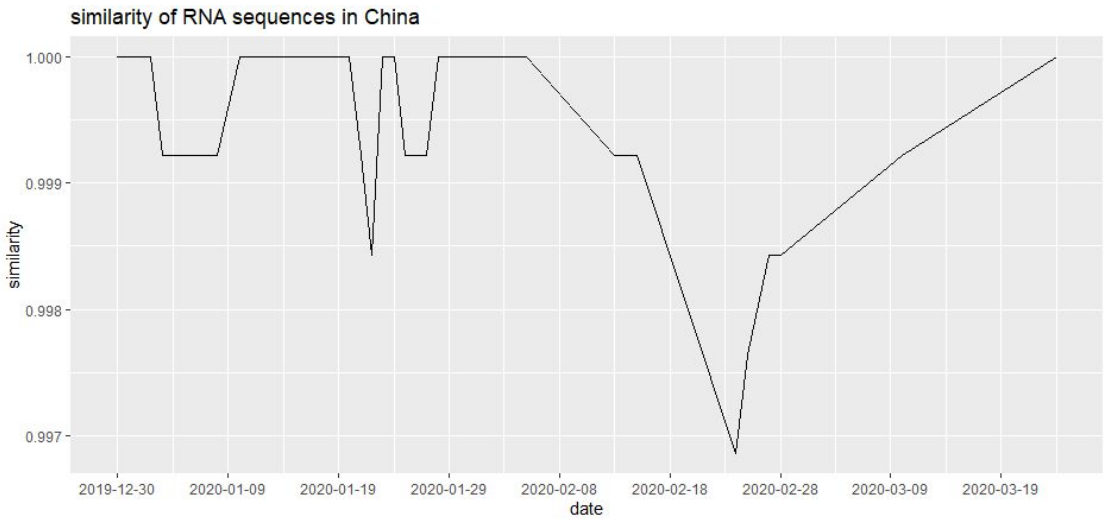
```r
string_similarity<-function(a,b){
  n=nchar(a)
  m=nchar(b)
  s=matrix(0,n+1,m+1)
  s[1,]=c(0:m)
  s[,1]=c(0:n)
  a=substring(a,1:n,1:n)
  b=substring(b,1:m,1:m)
  for (i in 2:(n+1)){
    for (j in 2:(m+1)){
      if (a[i-1]==b[j-1]){
        temp=0
      }
      else{
        temp=1
      }
      s[i,j]=min(s[i-1,j]+1,s[i,j-1]+1,s[i-1,j-1]+temp)
    }
  }
  return (1-s[n+1,m+1]/max(n,m))
}
```

There are other methods to calculate string similarity in R language, such as adist function in utils package and stringdist function in stringdist package. We will compare them with the results calculated by my own function to verify the correctness of the algorithm.

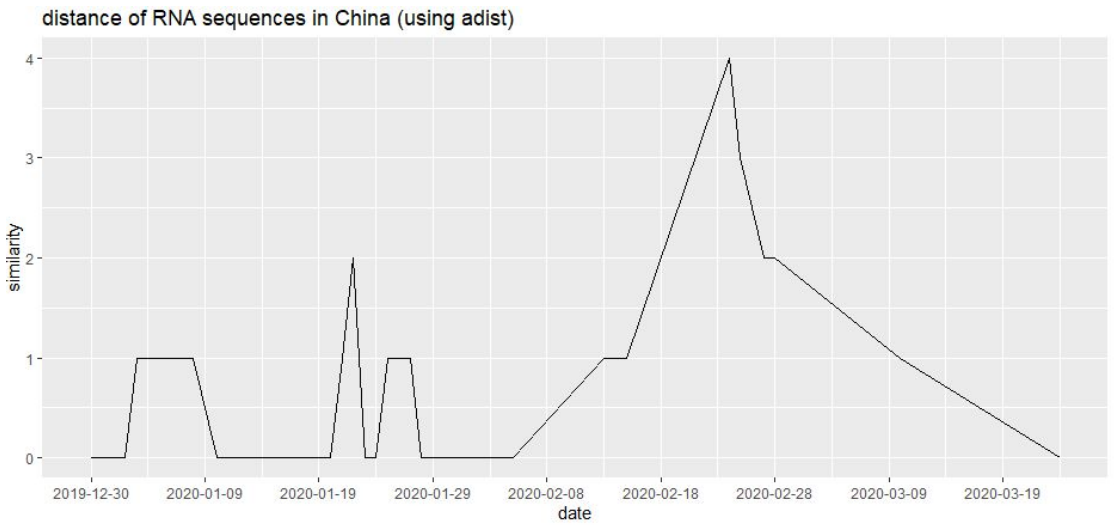
In addition, in R language, we will also use the functions in the dplyr package to clear the read data. For example, the filter function is used to filter out the sequence information of a specific country, the range function is used to sort the data by date, and slice (1) is used to keep one of the duplicate dates.

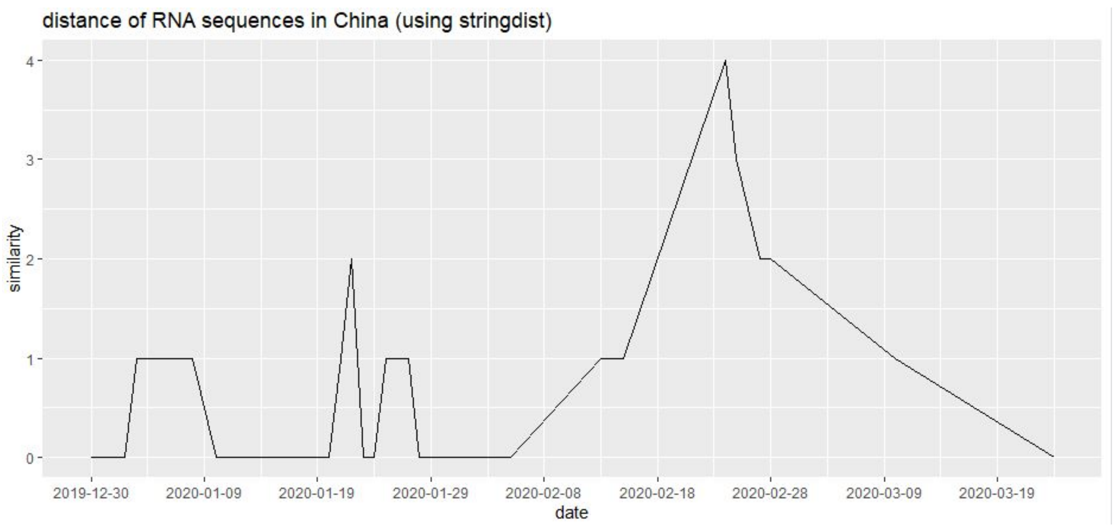
Results and Discussion

Using user-defined function to calculate the similarity between RNA sequence of each day and the RNA sequence of the previous day, we can find that between February 18, 2020 and February 28, 2020, the RNA sequence produces pictures, and the gene sequence of the virus changes obviously.

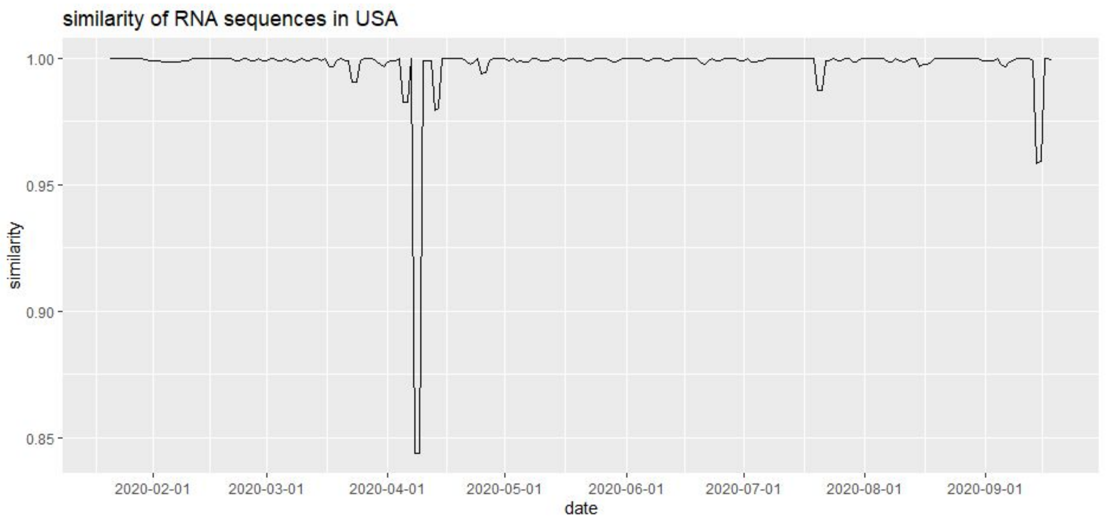


The following two pictures are the string distance calculated by adist function and sting dist function, which is consistent with the change trend of string similarity in the figure above, which shows that the implementation of the algorithm for calculating the similarity of user-defined strings is correct.

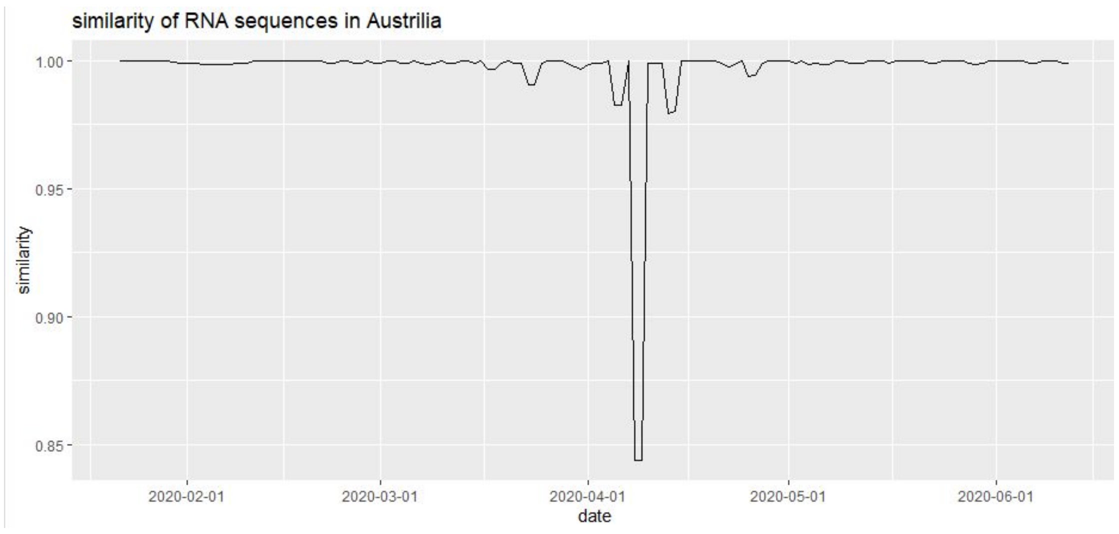




The same similarity calculation was carried out on the RNA sequences of the United States, and it was found that there were large mutations in the RNA sequences between April 1, 2020 and April 15, 2020. Around September 15, 2020, the second mutation will occur.



In Australia, the time of large mutation in RNA sequence is from April 1, 2020 to April 15, 2020, which is similar to that in the United States.



Related Work

A network is a common way to represent the relationships among a set of objects (also known as nodes or vertices). These relationships are represented by an edge, which connects these nodes to one another. Typically, edges are defined only between a pair of nodes. The set of these edges and nodes constitutes a graph (or network). Networks have become more prevalent with the rise or the internet and social media, both of which are frequently represented and analyzed as networks. Each of the following projects will interact with features of network construction, as it pertains to string data. This class of network is referred to as a sequence similarity network (SSN), which are a common tool in bioinformatics. They can be used to show how a set of protein sequences are related, in terms of their structural similarity. You can read more about the applications and details of SSNs in this paper. This type of network can also be applied to other types of string data, such as text. In order to build an SSN, you must choose some sequence similarity measure to determine which nodes are similar enough to be connected by an edge. You also need a model for including or excluding edges. The basic approach is to set some threshold on your similarity measure.

The implementation process of this method is more complex, and the amount of calculation is large. My method is more concise and easy to implement.

Conclusion

In this project, we deepen the understanding of sequence distance and sequence similarity, and successfully implement the algorithm of calculating distance and similarity, and verify its correctness after comparing with the function results of R language. In my future work, we will consider analyzing the change of RNA sequence similarity over time in more countries, and compare the similarity of RNA sequences between different countries, and analyze the relationship between RNA sequences among different countries.

Bibliography

[1] Pearson, W. R. (2013). An introduction to sequence similarity (“homology”) searching. Current protocols in bioinformatics, 42(1), 3-1.

[2] Atkinson, H. J., Morris, J. H., Ferrin, T. E., & Babbitt, P. C. (2009). Using sequence similarity networks for visualization of relationships across diverse protein superfamilies. PloS one, 4(2), e4345.

[3] Yujian, L., & Bo, L. (2007). A normalized Levenshtein distance metric. IEEE transactions on pattern analysis and machine intelligence, 29(6), 1091-1095.

[4] Heeringa, W. J. (2004). Measuring dialect pronunciation differences using Levenshtein distance (Doctoral dissertation, University Library Groningen][Host]).