COMP 561 Final Project

Alignment Algorithm for Probabilistic Sequences

Angela Zhao & Xinxin Lu

1. Introduction

When analyzing the biological problem using a computational method, gene sequences are always needed such as gene finding using HMM. However, sometimes the specific nucleotide sequences are unknown or a more robust mode is needed for complex systems, then a probabilistic sequence model will be applied to the genome.(Ekisheva et al., 2006) The probabilistic sequence model can generate a genome sequence as a probabilistic matrix which contains the emission probability of each nucleotide (A/T/C/G). The probabilistic genome can be used when the nucleotides at specific positions are uncertain.

According to previous research on computational reconstruction of ancestral DNA (Blanchette et al., 2018), the ancestral genome sequence can be reconstructed using a set of extant orthologous DNA genomic sequences. The program eventually outputs a prediction of the ancestral genome, so the result can be described in a probabilistic genome sequence matrix. With the program, we are going to have a large ancestral genome database. To make use of the database, comparison and alignment of a short sequence with the database are required. If there is an algorithm that can properly align the query sequence with the database, then we can perform the comparison of the recent species genomes with ancestral species genome sequences.

To heuristically align and compare a short gene sequence to a large database, we can use BLAST (Pertsemlidis et al., 2001). BLAST, the basic local alignment search tool can search for local alignments with short perfect match, and then extend it to find the best alignment with the database (Lobo, I., 2008). However, BLAST only applies to the database and query sequence in regular form. Hence, we cannot approach the probabilistic ancestral genome sequence using BLAST alone. Therefore, we decided to develop an analogy to the BLAST, but it can search for the probabilistic sequence. To approach this problem, we are going to figure out how to find an optimal alignment position of a short sequence in the long probabilistic sequence of the ancestral genome. Using the algorithm, we can estimate the ancestor of the recent species or the common ancestor among a few species.

2. Methodology

We are implementing the algorithm with Java in four classes: *QueryGenerator, QueryEvaluation, QueryNWAlignment,* and *QueryTest*. To run the program, use the following commands:

*> javac *.java*

> java -cp QueryTest sequence_file.txt probability_file.txt

2.1 QueryGenerator

The first step is to randomly pick a start point of the query sequence in the probability genome. Then, based on the probability matrix, we randomly choose the nucleotide to finish the query sequence. Each randomly generated query has a length of 100 base pairs. In the query sequence, we randomly delete or insert nucleotides within a mutation rate (*variable errorRate in the code*) and return the start point and query sequence array.

2.2 QueryEvalutation

Preprocess and index the database

Based on the probability matrix, we generate a new database which is a regular gene sequence. We index every w-mer of the new database, store the indices in a map, and return the index map and new database.

Search for hits in the database

We obtain every seed with length of w in the query sequence and check if it is in the keys of the index map. If the w-mer of a query matches with the database, then we call the method to extend the hits to find the highest scoring pair (HSP). If the score from HSP is higher than the predetermined threshold value, then we put the score into the hspMap. This method returns the hspMap.

HSP

In this block, we tend to find the highest score pair by extending the match seeds in the query. First, we need to decide if we should extend to the left or extend to the right. (*checkDirection*). If the seed is on the left bound, then we only do left extension. If the seed is on the right bound, then we only do the right extension. If the seed is in the middle, then we extend to the side which does not drop more than our delta parameter. For each extension, we calculated the score as detailed: if the query matches with the database, increase the score by adding the probability of the nucleotide. If the query does not match with the database, decrease the score by ((1-P(position))/3)-P(position). Here, we assume that matching a nucleotide to the new database has score P(position), and mismatching the new database will result in another smaller probability. The subtraction (mismatching- matching) will be the decrease of probability of the nucleotide fit, so we set the score decreasing with (mismatching-matching).

Then, check if the decrease from highest score to current score is larger than delta. (*extendThreshold in code*). If yes, then we stop the extension on this side, and switch. If not, then the current score will be the new highest score. Finally, we combine the left score, right score, and seed score as our final score. We retrieve the index of the query and the index in the database based on the alignment and return the score.

Gapped Extension

For the gapped extension, we make a matrix to perform the NWalignment in each direction and return optimal (highest-scoring) alignment and its score.

For each alignment in hspMap, we let them go through the gapped extension which will return the score and best alignment in the format: [(query start, query end), (database start, database

end)]=score. We then find the alignment with the highest score, and return this alignment with score.

2.3 QueryNWAlignment:

This class implements the dynamic programming Needleman-Wunsch algorithm for sequence alignment. The goal of the algorithm is to fill a scoring matrix by matching two DNA sequences to optimize the score. While filling out the scoring matrix, we are simultaneously filling a traceback matrix that we use to obtain the alignment. Nucleotide matches increase the score and mismatches decrease the score according to the scoring matrix. We also use a linear gap penalty, g(L) = -1*L. After filling out the matrices, we obtain our score in the bottom right corner of the scoring matrix (position $M_{r, s}$ for an r * s matrix M) and the alignment from the traceback matrix.

2.4 QueryTest

Data Input

First, we read the sequence file and input it into an array. Then, we read the probability file and input it into an array. Next, we build a matrix to combine each position in sequence with corresponding probability. For the other three nucleotides, the probability is (1-P(position))/3. Finally, we return the sequence array and probability matrix.

Testing the accuracy

From the query generator, we get a randomly generated query sequence with a start point. From the query evaluation, we get the best alignment of the query sequence with the start index of alignment in the database. We are considering the alignment is accurate if the start point - 5 <= the start index of alignment <= start point + 5. We consider the accuracy in a range since nucleotide insertion/deletion might occur at the beginning of the query sequence. We repeat the program for n times and find accuracy = accurate alignments/n.

3. Results

3.1 Parameter settings

- query length=100
- w-mer=11
- delta=6.0
- threshold=10
- errorRate=0.01

3.2 How to read the output

The algorithm will output every alignment in the query and database and final accuracy after n repetitions of the program in the following format. An example of the output can be found in Figure 2.

Finding No. => original query sequence

start point - [(query start, query end), (database start, database end)]=score

Accuracy: 100 * accurate alignments/n %

3.3 Durability (Tests for different number of queries)

We tested the algorithm with 10, 20, 30, 50, and 70 queries and repeated each trial 3 times. There is no significant trend to our data and the accuracy tends to hover between 60% and 75% regardless of the number of queries (Figure 1). We see that when we increased the number of queries to 70, there was a significant drop in accuracy. However, this could be attributed to random error due to the probabilistic nature of our database.

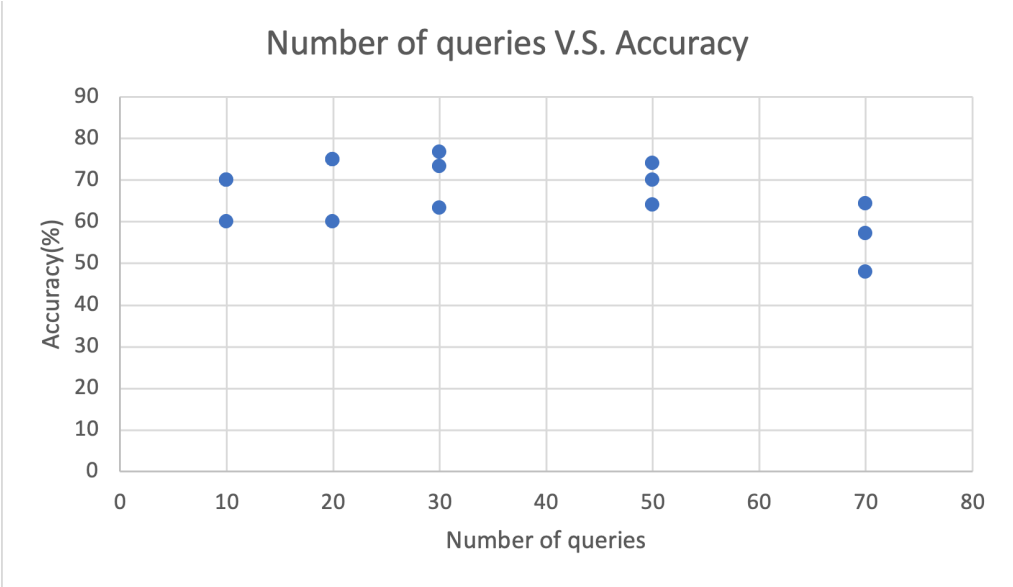

Figure 1. accuracy of our trails graphed as a funtion of the number of queries

The best accuracy score of 10 queries is 70.0% (Figure 2).

```
Finding 1 => CAAGACAAGCACAAAAGCACCACCCATGCCTCTGAAGAACATTGGACCATGCACCCTTGAAGAAAGCTTTGCCTCCTTCATCATGAGTAATCTTCCTCCAG
239646 - [(0,97),(239646,239743)]=59.88222222222227
Finding 2 => CCTTAGCTCACCCCCCCCCCCCACCGTGAGACACAGGAACTGCTTCCCCCATCGGCCCCGCGTCACAGCCCCTGTCCTCGCCTCCAGCAGGCACCAGCACA
529682 - [(1,100),(529682,529781)]=103.96888888888896
Finding 3 => ATCCCTGCTCCTAGGTTCAAGCAGTTCTTTTTCCATGGGAAGAATTAGTGGTTGTGGTTGTTTGAACATTTCTATTTTAGAGCCTCCTATTCTTGTTTCAT
286836 - [(0,97),(282542,282639)]=45.687777777777846
Finding 4 => AGGAGAAATCTGAGGCTCAGAATGGCCAAGCGACCCGTGCGAGGTCATACGGCTACTGCACATCCCAGAAAGCCACGCTGTGGGAAGCTTGGTCAGGTGTC
577419 - [(2,100),(577420,577518)]=154.43777777777768
Finding 5 => TGTCTCCACCTTCTGTGTATCCAGCTCAGGGCCTGCCCAGCGGAGATGCCCAAACCAGCACGCGGCAGCGATGCTTCTAGAGCCGGCTTTCTCAGGCACAA
191293 - [(6,100),(191298,191392)]=107.49444444444444
Finding 6 => TGACCAGAGGCCTCAGGATGGTAGAAAGAACCGGGGCTTTTGAAGTCAGACAAGACTAGGTTCAACTCTCGGCTCTGACACCTGCCAGCCACGGGCAACCC
498026 - [(4,100),(498029,498125)]=119.01555555555552
Finding 7 => AGGTCCATGGAAAGCCAAAGCGTTTCCTGAGCTGATGGACACCACCCGTGCATGTACCCTTCTAGAAGTGGTCTCGTCACCTCCCAGCTCCTAGCCTCCCG
571264 - [(2,100),(571265,571363)]=99.58888888888899
Finding 8 => TAAGTGTTCATTCTTGTTGTTGAGAAATTAGATCCATCCCCCTTAAATGAAGCATGCTTGGAAACTCTGGATTCTAGAGGGGGTTAATACTACAAAGGTAG
28548 - [(3,100),(6693,6790)]=37.995555555555605
Finding 9 => GGAGTCTACATGGTTTCGGGCGGAGAGGGGAATTACGTTGGGCTTTATCTAGTCCAATCGTGAACTGTAACAGAGCTAGATGAGGTCTTATAATGGGGAAC
7872 - [(12,100),(7883,7971)]=142.1144444444443
Finding 10 => ATTGTCAGAGAAAGTGGATTTATTGTGTCATGGGGGTTTCTGGGGCCCAGCTCTATCCTGAGAGTGGTGGGAAGATTGGGCATGGGACACCCCACTCTGCA
272077 - [(0,91),(272077,272168)]=104.33333333333347
Accuracy: 70.0%
```

Figure 2. Results and accuracy from the highest-accuracy trial with 10 queries.

The best accuracy score of 20 queries is 75.0% (Figure 3).

Figure 3. Results and accuracy from the highest-accuracy trial with 20 queries.

The best accuracy score of 30 queries is 76.7% (Figure 4).

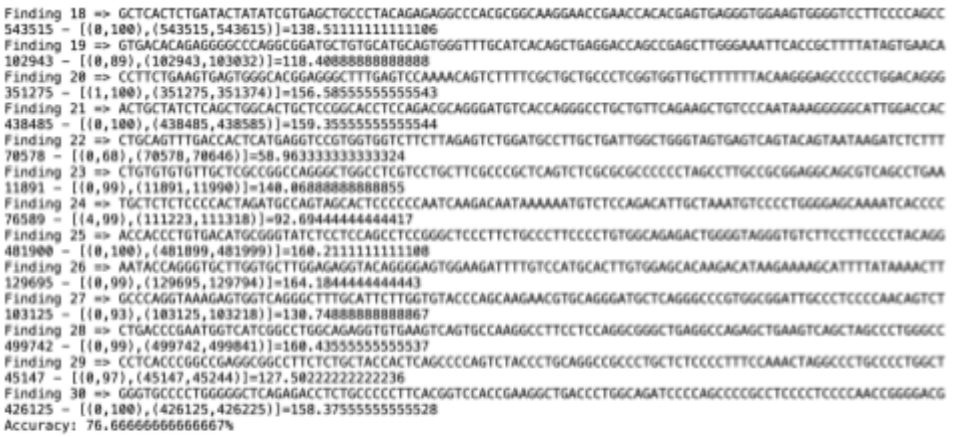

Figure 4. Results and accuracy from the highest-accuracy trial with 30 queries.

The best accuracy score of 50 queries is 74.0% (Figure 5).

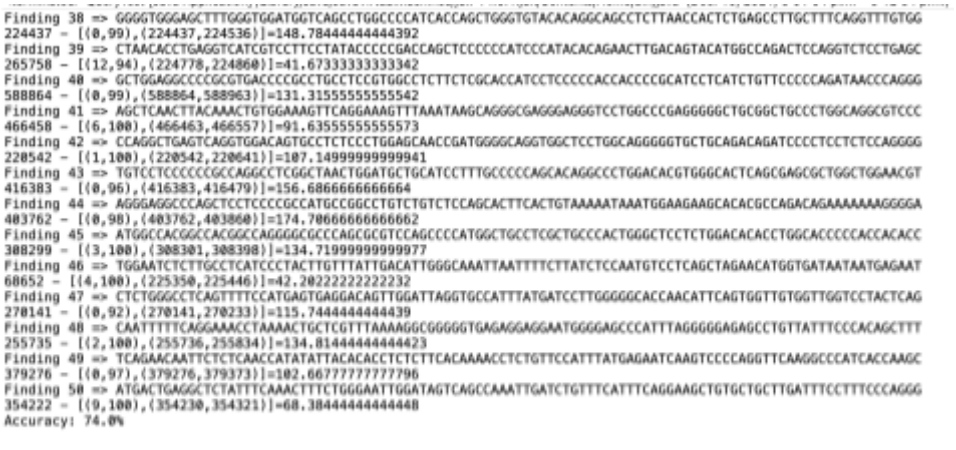

Figure 5. Results and accuracy from the highest-accuracy trial with 50 queries.

The best accuracy score of 70 queries is 64.3% (Figure 6).

```
:21370 - [(0,99),(221370,221469)]=152.61888888888873
inding 57 => TAGCACCCAGAGGCTTCCTTTTTCCACTCAGGAGCCACTGTCAACTATCAATCCTCTGGTGAGAGCATGGGGGTACGGCGGAAAGAACACTGGCCTAGCTG
:86720 - [(0,100),(386720,386820)]=132.70111111111072
inding 58 => AGCACGGTGGTCCGGAACTGTATACAAAACACGACCCCAAACTGCCTCTCGGTGTGTTTCCAATTAAACAACCTGCTCTAGTCTTCGATCTCTTTCCATCA
:44971 - [(6,100),(544976,545070)]=156.57999999999998
inding 59 => GGTCTGGTGCAAACAAGGTCTGTAAGGAAAGGCTGCACCAGCTCCGCCTCTACTGGTCCCCTTGGAGCCTTCCAGATATCCCCAGGCCTGGGGAGACTCCT
:58630 - [(0,100),(458629,458729)]=95.88111111111107
inding 60 => CCCAAAGCTGGGTGACCCACATGCAGGGGAGGACAGCATCGGAGTCCACACTTCTTTGGAGAGGGACCTTCAGCCACCAGGTCACTGCTGTGCTGGGGCCG
:3845 - [(1,100),(43845,43944)]=168.65666666666647
inding 61 => GTTTTGGGCCTGAGCAAGTGAGTGGAAAATGGAGGTGCCATTACCTGAGACGGGAAAGTGTTGGGGGTGGGGCAGGTTTGTGACCCTGCTGAGGCGGAGCT
:76309 - [(0,95),(476309,476404)]=137.48999999999975
inding 62 => AGGGGTCAGTCAGGTGCCCTGCAGTCCCCCAGGGCTGCTCCAGGAGGCCCTTCCTCGCGGGCTCAGCTGAGACGCCCATTCCTCTTTAGGTTCTGGGGGCT
:44161 - [(0,97),(144161,144258)]=137.8466666666663
inding 63 => ATTTACTACATGGATATAAGGGAAAAAAGACTTGTTCTTGGCATCAGTTTTATCCAGTGGCTGGGGTGGCATACTTTAGTTTGATCTCAGGTTCTTGGGGT
:51859 - [(0,100),(393389,393489)]=46.15000000000012
inding 64 => CAGTTCCGGAAGGACTTCTTCACCCGAATGTGTGCCCCACGCATCATGGGTGCTCCTCCAGGCTGGGGCCCCGTCTGGCTCAACCCCAGCCTCCAGCCCTG
:23083 - [(4,100),(423086,423182)]=98.12444444444452
inding 65 => TGTGTGTGTGTGTGTGTGTGAGAGAGAGGAGAGAGAGTGAGCGTGTGTGTGTGTGTGTGCGTGCGCGCCTGTGCAGGGGTGAGGTATTTTCACTGCCCT
:33890 - [(28,100),(233917,233989)]=115.7766666666667
inding 66 => GAGGGCAGTGCAGCTGAGCCGGAGCTAATGAGTGGCAAAGGGGGCGGGGGCTGGATCACAGCACGTGCAGAGGCCTGGAAGTGGGAGGTGAACAGGGAATC
:06880 - [(0,100),(106880,106980)]=147.9922222222221
inding 67 => TTGTAGGACCTTGTCATCACAGTGCCCTGCGTGACGCTCCTCTGTGGGCTGCTTCACATTCAGGCGTCTCCTCCCTAAGGCTGGCTGCTCCTCGTGGACAG
:5654 - [(1,100),(85654,85753)]=108.44111111111087
inding 68 => GCATCTTGGTGTTTGTCAAGATTATAAAATATTAACCAACATAAGGGGACTGTTGAGGGTTGGTAAGGCAGTGGCCCGCCTGGACCATTTGACTCCTCCCT
:40657 - [(0,98),(140657,140755)]=154.07999999999964
inding 69 => GGGGCGGACGCTCGGCGGTCCTCGCGCCGCTGCTCGCCGGGTGCTCCGGGGTTCCGCGCCTTTGCTGGTCTCCCGGCGGACCCTCGAGCCGAGCTGCACGC
:6550 - [(1,100),(96550,96649)]=153.69222222222191
inding 70 => TGCTCTGTGCCTTCACAGCCTTCTCTCACGGTCTGAAGCTATATTGGTGTGATTATTTGGTTAATCGCTGGCATTCCCATAACTTTAATCCTCATAAAGCT
:89052 - [(0,96),(589052,589148)]=42.407777777777916
ccuracy: 64.28571428571429%
```

Figure 6. Results and accuracy from the highest-accuracy trail with 7o trails

3.4 Runtime of 10, 20, 30, 50, and 70 queries

The Big-O complexity of our algorithm is $O(n^2)$. The runtimes for generating the results in Figures 1-5 are described by Figure 7. Runtime is increasing as we increase the number of queries as expected. However, it is not nearly as bad as our worst-case scenario represented by the Big-O complexity. In fact, the experimental runtime as a function of number of queries seems to fit a linear model with the equation $y = 9.4x - 42.4$ and an R^2 score of 0.7563, which is an acceptable statistic for the goodness of fit. This equation can be used to project the expected runtime, as opposed to the worst-case runtime represented by $O(n^2)$.

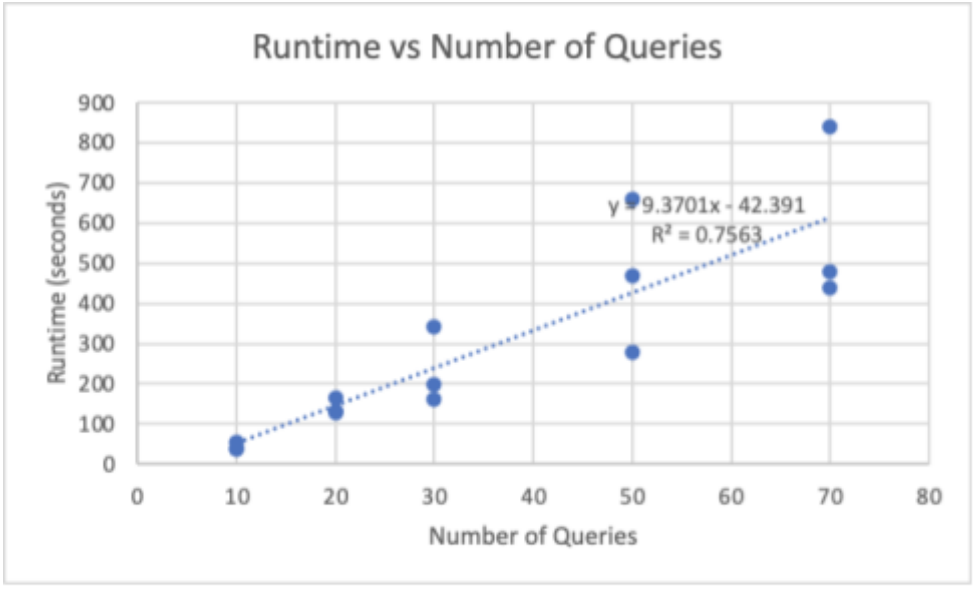

Figure 7. Number of queries per trial is on the x-axis. The y-axis represents the time in seconds it took for the program to successfully terminate. We added a trendline to characterize the experimental runtime fit to a linear model.

3.5 Overall Accuracy

We obtained the overall accuracy by taking the average for all accuracies from the 15 trials above.
Accuracy = 66.72%

4. Discussion and future work

Our program successfully shows alignments of the query matched to the newly generated probabilistic database with an accuracy of 66.72%. The output indicates the nucleotide positions of the alignment in the database as well as the corresponding positions in our query. It can select appropriate alignment scores and the whole query sequence can be aligned in most cases. The accuracy decreases with more queries being tested, and the program did not break out for testing large numbers of queries. The runtime has a high Big-O complexity, making it increasingly inefficient as the number of queries increases. However, with our sample of 10-70 queries, we were able to fit our data to a linear model, which suggests that at least within this order of magnitude (10^1), the expected runtime is much better than the Big-O runtime.

Since our method is based on BLAST, the pros and cons of the algorithm carry on to our program as well. For example, we have a set of parameters that we optimized to use for our data (see Results: Parameter Settings) to have the highest accuracy. Fortunately, this program can also be generalized to other databases with specific characteristics, which may require a different set of parameters. For example, by increasing the size of the w-mer, we are more likely to miss genes. However, if we decrease the size of the w-mer too much, we would end up with too many false positive hits, which will severely impede the runtime. Likewise, we used a linear gap extension, whereas affine gap entries allow the user to favour either abnormally long or short gaps in protein transcription. For now, we have hard-coded these parameters, but a simple improvement would be to take these parameters as user-specified inputs instead. Furthermore, our algorithm is heuristic and thus suboptimal as a means to make it possible to use it (ie. it will terminate within a reasonable amount of time) with real data, which may be at least thousands of base pairs. Finally, since our algorithm uses DNA alignment, of the nucleotides without regard for the reading frame. Perhaps a complex gap penalty can be used to favour deletions in groups of 3. However, the best method first and foremost would be to perform sequence alignment with amino acids instead of raw DNA when the amino acid data is available.

To some degree, we found the runtime of the program can be slow as the number of queries tested increases. For n=10, the runtime can be around 1 minutes, and for n=50 the running time is around 10 minutes. Some specific queries can take much longer time to align than the others. For better accuracy, we find hsp for every indexed w-mer in the database, and then take the alignments higher than certain threshold to go through the NW alignment and find the alignment with best score. With this method, we can get more accurate results but it will greatly increase the running time. For some queries, there might be many indexed w-mers that can have alignment score higher than the threshold so that NW alignment would be run for many times which is time consuming. To improve, we probably can find the w-mer with the highest alignment score first, and then do HSP. Also, we probably can set a threshold for gapped extension. Above a certain threshold, the gapped extension works instead of applying NW alignment for all possible alignment. However, we can not determine the value of the threshold that should be used for gapped extension, we can work further. Besides, the improvement of running time might decrease the accuracy. We should try to improve our program to

decrease the running time while maintaining accuracy.

Besides, the score calculation method in our program also needs some improvement. We used to consider calculating by multiple the probability of each nucleotide to the optimal alignment based on the principle of probability. However, multiplication can devastatingly decrease the score (0.1*0.9=0.09), so it will be hard to define the delta and find the alignment with the highest score. Therefore, we decided to calculate the score by summing up the probability, and decrease the score by the difference of probability of nucleotides. From the result, our method works properly, but we can study further to improve the accuracy by using better calculation methods.

Bibliography

Blanchette M., Diallo A.B., Green E.D., Miller W., Haussler D. (2008) Computational Reconstruction of Ancestral DNA Sequences. In: Murphy W.J. (eds) Phylogenomics. Methods in Molecular Biology™, vol 422. Humana Press. https://doi.org/10.1007/978-1-59745-581-7_11

Ekisheva, S., & Borodovsky, M. (2006). Probabilistic models for biological sequences: selection and Maximum Likelihood estimation. *International journal of bioinformatics research and applications*, *2*(3), 305–324. https://doi.org/10.1504/IJBRA.2006.010607

Lobo, I. (2008) Basic Local Alignment Search Tool (BLAST). Nature Education 1(1):215

Pertsemlidis, A., & Fondon, J. W., 3rd. (2001). Having a BLAST with bioinformatics (and avoiding BLASTphemy). *Genome Biology*, *2*(10), REVIEWS2002. https://doi.org/10.1186/gb-2001-2-10-reviews2002

Probabilistic sequence models - Pennsylvania State University. (n.d.). Retrieved December 3, 2021, from https://globin.bx.psu.edu/courses/fall2002/prob.pdf.