Coarse-to-Fine Circular Sequence Alignment

Senior Thesis:

Coarse-to-fine Sequence Alignment Project

Emma K. A. Rogge

Harvard University School of Engineering & Applied Science

Department of Computer Science

Advisor: Dr. Stuart M. Shieber

Introduction

When I began working on my thesis in fall of 2019, I had no idea I'd be finishing this project not in the Lowell House library or the hallway outside my advisor's office in Maxwell Dworkin but in the basement of my family home in Cincinnati, Ohio. All year, I have tried to explain why my thesis topic, a novel algorithm for the faster approximate alignment of small, circular sequences, is interesting to friends, family and each person whose invitation to do something fun I've declined in favor of working on this project. Suddenly, people all over the world are talking about viruses, the very organisms whose genomes have occupied my attention since September. Specifically, this project aims to speed up the process of pairwise sequence alignment for small (20,000 base pairs), circular genomes.

Pairwise sequence alignment, a central process in the field of bioinformatics, is the comparison of two biological sequences to identify regions of similarity that may indicate functional, structural, or evolutionary relationships between them (Fernandes et al., 2009). While effective algorithms exist for the pairwise alignment of linear biological sequences, an additional challenge is posed by the alignment of circular biological sequences, though such sequences occur frequently in nature. Prokaryotes' primary genomic information is circular in structure; human mitochondria and plant chloroplasts possess circular genomes that enable metabolism and thus survival. Yet despite the importance and prevalence of cyclic sequences throughout nature, most algorithms for pairwise alignment require the manual rotation of the circular information to the 'correct' start index based on known genomic features before any alignment can occur, an additional step which requires preexisting knowledge about the genomes to be aligned, the most accurate of which has a time-complexity for two sequences a and b with lengths m and n of $O(mn)$ (Grossi et al., 2016). While some algorithms reliant upon heuristics exist for aligning cyclic sequences specifically, the search for a sub-$O(n^3)$ time-complexity solution to the problem of approximate circular sequence alignment has not been exhausted, particularly for approximate alignment (Panagiotis, Charalampopoulos et al., 2019). The following report will detail existing algorithms for optimal and approximate circular sequence alignment and propose two new algorithms for this problem. I will discuss the proposed algorithms' theoretical time-complexity, implementations of the algorithms in Python and experimental results thus far.

An Upper Bound for Pairwise Circular Sequence Alignment

To establish an upper bound on the time-complexity of an algorithm for the approximate pairwise alignment of circular biological sequences, let us discuss the naive solution. The pseudocode for the naive algorithm is listed below.

1. For a sequence of length n, there exist n possible linearizations of the original sequence. This step has a time-complexity of $O(n)$.
2. Compute, for each linearization, a global alignment. This step has a time-complexity of *n* times the complexity of the alignment. We will use the Needleman-Wunsch algorithm to find the optimal, global alignment, which for two sequences of length n and m, respectively, has complexity $O(nm)$. Thus this step has time-complexity of $O(n^2m)$.

Therefore, the total complexity of the naive solution to the problem of pairwise circular sequence alignment is $O(n^2m)$.

Existing Algorithms for Pairwise Circular Sequence Alignment

For many years, the Maes algorithm reigned as the best known optimal circular sequence alignment algorithm, with time-complexity O(nm log n) and O(nm) space required [Maes 1990]. However, it produces an optimal rather than approximate alignment.

Improving Upon Naive Circular Sequence Alignment

To improve on the step requiring n linearizations of a circular sequence, let us consider whether an optimal global alignment is most appropriate in the specific case of circular sequences. While the Needleman-Wunsch algorithm guarantees the optimal global alignment for a given scoring system, it also assumes that the optimal alignment is a global one versus a local one. Could a reasonable yet quick global alignment be achieved by first identifying the areas of greatest local similarity, then refining this coarse alignment using any number of the myriad edit distance techniques available? The two algorithms proposed in this project take advantage of segmentation and hashing, techniques of great utility in natural language processing applications, to produce an approximate global alignment which can then be refined via a traditional edit distance mechanism, such as Levenshtein edit distance, the Needleman-Wunsch algorithm, et cetera (Needleman and Wunsch, "A General Method Applicable to the Search for Similarities in the Amino Acid Sequence of Two Proteins."). Pseudocode for the two proposed algorithms follows.

Circular Alignment via Rolling Hash

This approach utilizes the Rabin-Karp rolling hash algorithm, which reduces the complexity of hashing individual snippets of text by taking advantage of the fact that they overlap in the context of sequence alignment (*Rec06.pdf*, n.d.). Specifically, we utilize the rolling hash algorithm to discover common substrings of length k between our sequences S and T, which we assume here have length n. The algorithm proceeds as described below.

1. Hash the first length k substring of S. The complexity of this step is . $O(k)$

2. Using the rolling hash technique, compute the hashes of each subsequent substring in S, of which there are n. Add each substring and its hash into a hashtable. This step has the time-complexity $O(n)$due to the frugality of the rolling hash technique.
3. Hash the first length k substring of T. This step has time-complexity *O(k)*.
4. As in Step (2), compute the hashes of each subsequent length-k substring in T, of which there are n, adding each substring and hash to the hashtable as before. The time-complexity of this step is *O(k)*.
5. Perform for each of S and T: In a sliding window of size w, selecting the minimum hash value. As these have already been computed, this step is *O(1).*
    a. If the minimum of the current window is the same as that of the previous window, continue.
    b. Else a new minimum has been found; record the index and value of that minimum to a table.

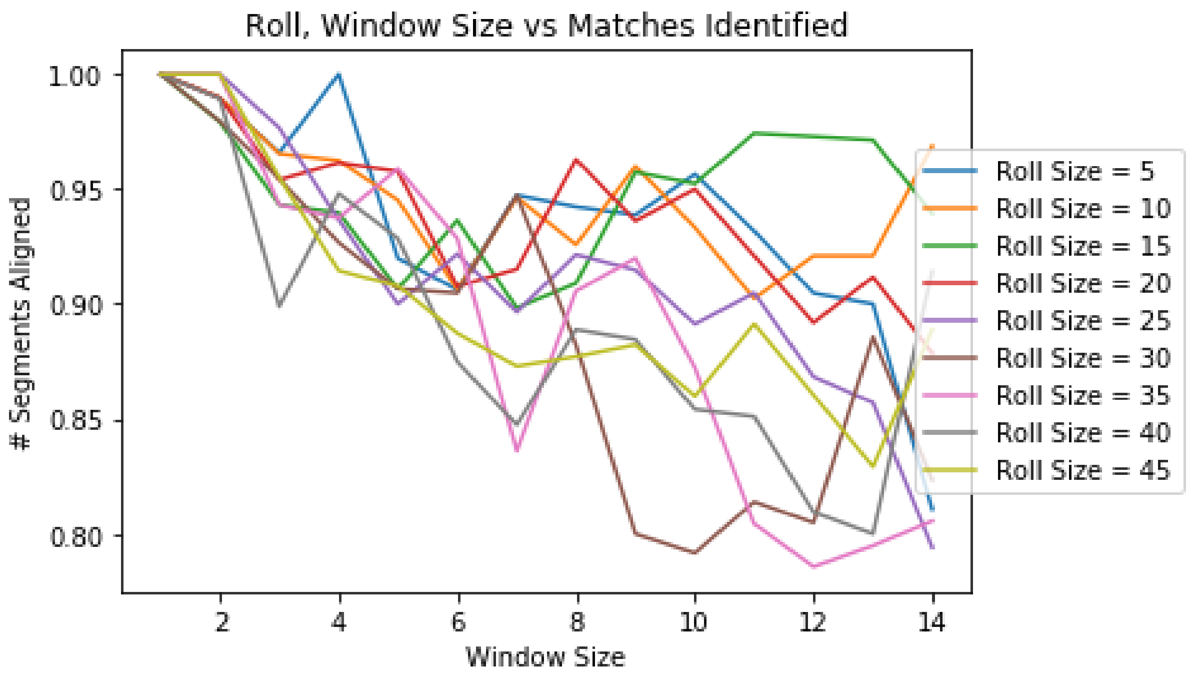
   At the completion of this step, we have a pair of hash tables H_T and H_S .
   of the form *{minimum hash value: (start_index, stop_index)}* that record the intervals for which the minimum hash value is a particular number for each sequence.
6. Sort the contents of $H_S$with respect to the indices. This has a worst-case complexity of $O(nlog(n))$.
7. Using the $O(1)$lookup time of our hash table H_T, align all chunks with matching local widow minima from S to the corresponding intervals in sorted H_S. This concludes the coarse step of the alignment. In the worst-case scenario, in which the minimum changes in every window, this step has time-complexity of $O(n-w) \ = O(n)$. Experimentally, I have found that this is much larger than the usual value for 4-base-pair DNA sequences.

One challenge of this stage of algorithm development is that the nature of the biological sequences being aligned will most likely require differently tuned parameters for different kinds of sequences. For instance, mitochondrial DNA is highly conserved such that even a single deletion may be of great significance, whereas bacteria and viruses evolve extremely rapidly, suggesting that insertions and deletions are likely more common and should be less costly in that context. Additionally, the algorithm will need to handle situations where no match is found for a given segment. One way of handling this would be to introduce a gap of equal length to the unmatched segment at a location adjacent to another segment whose indices neighbor those of the unmatched segment; another way would be to divide such unmatched segments from both sequences into smaller pieces and attempt an alignment of those pieces. The latter idea would be feasible if it could be guaranteed that only a small fraction of the segments were unmatched; otherwise, it could result in huge time-complexity. For the purposes of this research project, therefore, I will leave the biology to the biologists by utilizing the widely-accepted EMBOSS Needle pairwise alignment algorithm, with its defined default parameters, as the gold truth of the success of an alignment (F et al., 2019).

Experimental Results of Parameter Search

I performed a parameter search for the appropriate roll and window sizes for the coarse step of the algorithm, utilizing samples of human and chimpanzee mitochondrial DNA (hence 'MtDNA'). The MtDNA sequences are between 16,000-17,000 base pairs in length and represent the entire mitochondrial genome. For ground truth, I performed a sequence alignment using the 'Pairwise Align DNA' tool on the website *Sequence Manipulation*, which utilizes the naive rotation-based algorithm that derives from the Needleman-Wunsch algorithm (Needleman & Wunsch, 1970). This achieved a result of 94% similarity between the genomes and served as the maximum possible similiarity, for an operation with overall complexity of *O(mn)* where *m, n* are the lengths of sequence *a* and *b*.

The parameter search reflected the expected outcome that if we take each base pair and match it with another, we should approach 100% segment alignment, but as we increase roll and window size, we reduce the granularity of information available for mapping as a tradeoff for the increased efficiency of reduced numbers of comparison. The parameter search found that setting *w* between 50 and 250 is optimal for sequences longer than approximately one thousand base pairs, especially as mitochondrial DNA is slower-changing and exhibits less frequent mutation than viral genomes (A. Mosig, IL Hofacker, PF Stadler, 2006). The figure below shows the results of one parameter search performed for shorter viroid sequence alignment.



The performance most reflective of the baseline standard for shorter sequences occurred when both the roll and window size were 5 base pairs wide.

Circular Alignment via Locality-Sensitive Hashing

This approach first hashes the two sequences *S* and *T* using a Rabin-Karp rolling hash on overlapping shingles of size *k*. Next, the hash sequences that result are reduced to a 'fingerprint' for each sequence by selecting the minimum hash value in a given window of size *w*. This produces fingerprints *F_S* and *F_T*. We then compute the Levenshtein edit distance for each rotation of *F_T* with *F_S* and select the rotation for which the Levenshtein edit distance from *F_T* is minimum. This concludes the coarse step.

For the refining step, we linearize sequence *T* at the index for which the fingerprints' Levenshtein distance is minimum and search within a window of size 2*w* for the best alignment of the original sequences.

Complexity Analysis

1. Computing a rolling hash over each subsequence of size *k* for two sequences of lengths *m* and *n* occurs in roughly linear time, $O(m\ +n)$.
2. The upper bound of the length of the fingerprints $F_S$and $F_T$is produced when the minimum hash in each window changes for every single window of size *q*, producing fingerprints of length *n/q* and *m/q*. We select *q* such that $m, n\ >>\ q$ e.g. $\sqrt{m}, \sqrt{n}\ > q > 3$.Therefore, the worst-case complexity for calculating the Levenshtein edit distance of each rotation F_{TR}, which has length at most $\frac{m}{\sqrt{m}}$, with F_S, which has length at most $\frac{n}{\sqrt{n}}$, is therefore$O(\sqrt{mn})$. There are at most $\frac{m}{\sqrt{m}}$ such rotations possible, so the overall complexity of this step is $O(m\sqrt{n})$.
3. Once the best rotation is found, refining the alignment of the original sequence *T* split at the point identified in step (2) within a window of $2w$to the original sequence *S* using Levenshtein distance takes $O(4w^2)$. Since $m, n\ >>\ w$, this term is negligible.

Therefore, we have achieved an approximately best alignment of two sequences using locality-sensitive hashing with overall complexity of $O(m\sqrt{n})$.

Implementation of caLSH

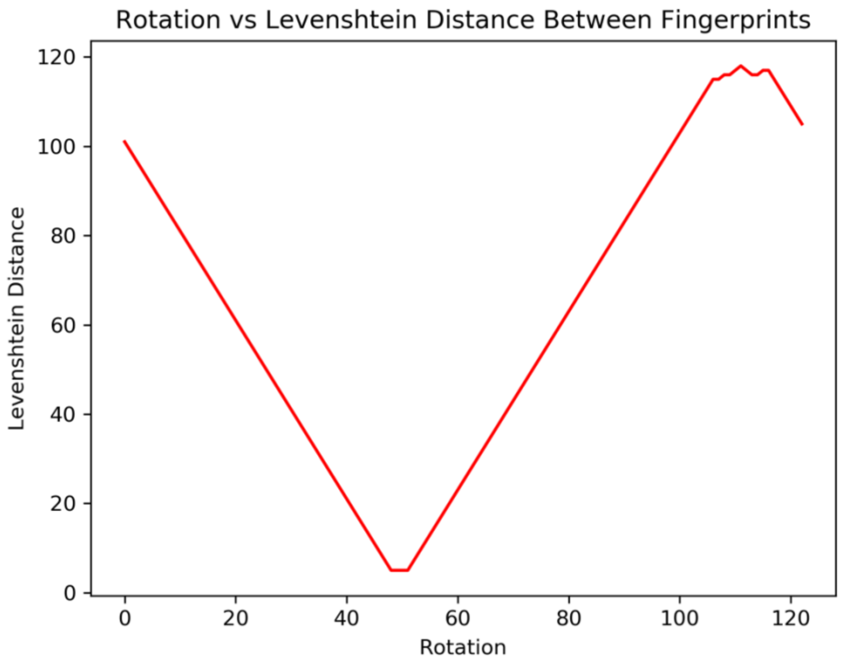
I implemented the locality-sensitive hashing algorithm described previously in Python, leveraging existing Python libraries for the reading and writing of biological sequences to commonly used formats, such as FASTA. I implemented the Rabin-Karp rolling hash algorithm in addition to helper methods to make experimental application of the alignment algorithm, henceforth referred to as **caLSH** (circular alignment via Locality Sensitive Hashing).

Experimental Results

Choosing Sequences for Evaluating the caLSH Algorithm

For comparison, I looked to the literature to find the genomes used to test recently published algorithms for the alignment of circular biological sequences. Since the Grossi lab published both simulated and real DNA and RNA sequences used in their 2017 paper on GitHub, I chose to utilize those in order to achieve data that could be directly compared to their results (Grossi et al., 2016). Specifically, I used the human, chimpanzee and gorilla mitochondrial DNA sequences (henceforth referenced as, respectively, NC_001807, NC_001643 and NC_011120) and eighteen related viroid RNA sequences curated in the RefSeq database, henceforth referenced by their RefSeq identifier (Pruitt et al., 2007).

The graph below shows the result of sanity-checking the algorithm on viroid sequence NC_001464, with default linearization from the BLAST database, against a rotation at that same sequence with split point at the 154th nucleotide. This sequence is 371 base pairs long; with parameters k = 5 and q = 5, caLSH produces a fingerprint of length 123. Intuitively, the algorithm should identify the local minimum at which the original sequence's fingerprint and its rotation's fingerprint are perfectly aligned; and indeed, this is shown in the graph of the results,



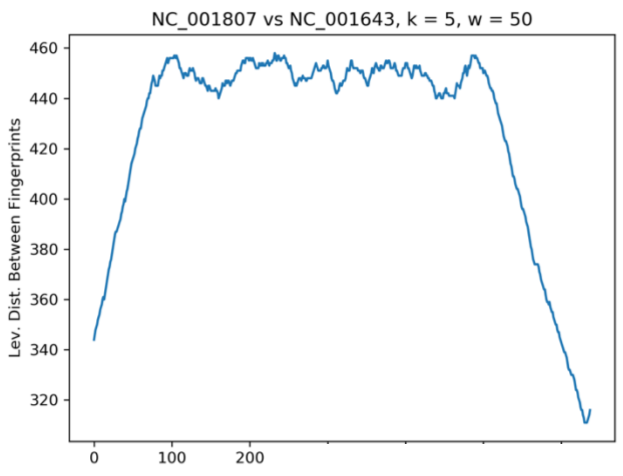
below.

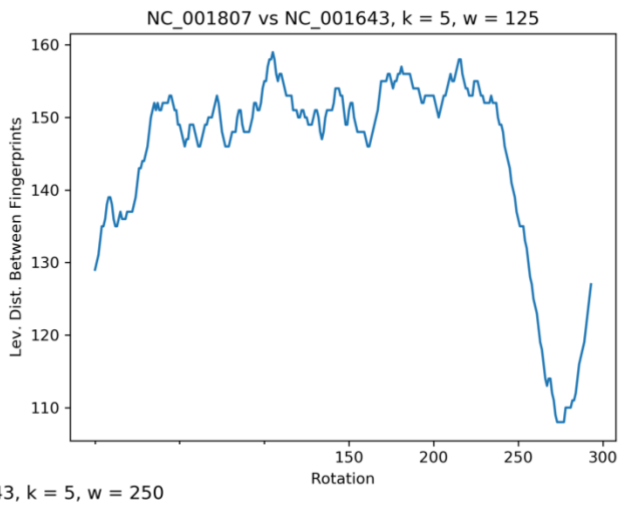
In order to have a meaningful comparison of the efficacy of this new method of identifying the optimal split point for linearizing the sequence to the methods used by Grossi, I ran the algorithm on the same mammalian mitochondrial DNA sequences used by that group, namely, human, chimpanzee and gorilla mitochondrial genomes approximately 16,000-17,000 base pairs in length.

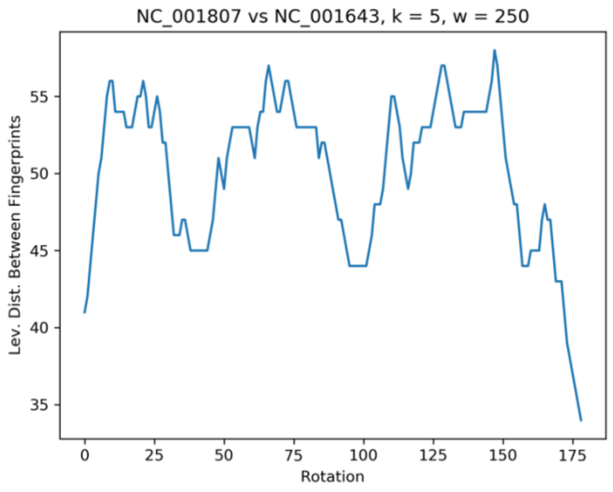
Experimental Results of Coarse Step Only

The following figures display the results of the coarse step of the algorithm for finding the approximately best split point for alignment of a sequence given another sequence. I used the human, chimpanzee and gorilla MtDNA sequences from GenBank, also utilized by Grossi et al., in order to facilitate the most direct comparison possible between the algorithm described in their article, caCSC, and the novel algorithm described in this document, caLSH (Benson et al., 2000; Grossi et al., 2016).

For the coarse step of the human-chimpanzee sequence alignment, the results for different window sizes *w* are shown below.







The split points for human-chimpanzee and human-gorilla alignments identified by the coarse step of the novel algorithm are detailed in Table 1. I computed the Levenshtein minimum edit distance between the first sequence and the rotated second sequence for each method and parameter value listed, and utilized the EMBOSS Needle API to compute the biologically-relevant EMBOSS Needle similarity of the sequences, which is widely used for pairwise sequence alignment in the biological science (Madeira et al., 2019).

Table 1: Experimental alignment of primate mitochondrial DNA sequences

Sequences Aligned	**Algorithm for Determining Split Point**	**Index of Split Point**	**Sequence Similarity (EMBOSS Needle)**	**Levenshtein Edit Distance**
NC_001807 (human), NC_001643 (chimpanzee)	None	None	85.1%	2509
	caCSC (Grossi et al.)	578	91%	3591
	caLSH (coarse step only, w = 50)	15995	89.6%	1495
	caLSH (coarse step only, w = 125)	15855	89.7%	1717
	caLSH (coarse step only, w = 250)	15805	89.1%	1719
	caLSH (w = 50)	16513	90.7%	1529
	caLSH (w = 125)	15855	89.7%	1719
	caLSH (w = 250)	15805	89.1%	1719
NC_001807 (human), NC_011120 (gorilla)	None	None	83.5 %	2692
	caCSC (Grossi et al.)	578	88.4%	
	caLSH (coarse step only, w = 50)	15832	90.9%	2055
	caLSH (coarse step only, w = 125)	15832	87.5%	2055
	caLSH (coarse step only, w = 250)	192	89.1%	3031
	caLSH (w = 50)	16401	87.3%	2041
	caLSH (w = 125)	15839	87.5%	2041
	caLSH (w = 250)	16407	81.5%	3022

In the human-chimpanzee MtDNA alignment, the Grossi algorithm found a rotation for alignment resulting in 91% similarity, just a hair better than that found by the than the caLSH algorithm, which found, given a window size of 50, a rotation with 90.7% similarity. Using the coarse step of the caLSH algorithm alone, a respectable rotation resulting in 89.6% similarity was identified, improving upon the default alignment of the original sequences, which had similarity 85.1%. For all parameter values, the caLSH algorithm produced a rotation that resulted in greater similarity after pairwise sequence alignment using EMBOSS Needle than did the original GenBank linearizations, although caLSH did not perform as well as the caCSC algorithm on this alignment.

In the human-gorilla MtDNA alignment, the coarse step alone of the caLSH algorithm outperformed the caCSC algorithm for all values of *w*. Interestingly, for this pair of sequences, application of only the coarse step of caLSH to find the rotation point resulted in a better EMBOSS Needle alignment for all parameters than did application of the full caLSH algorithm in terms of producing rotations with greatest similarity. Upon inspection of the human-gorilla alignment FASTA files, I suspect this is because there is no true alignment at the beginning of this pairing in their original form, so any shifting of the overall alignment to reflect a common subsequence within the first window is not effective. Thus, a future investigation into how better to refine the coarse algorithm while keeping the overall complexity of that step low could further improve the performance of caLSH.

Viroid Sequence Outcomes

In order to evaluate the efficacy of my algorithm for identifying the approximately best rotation of a sequence for pairwise alignment with another sequence, I chose to utilize the workflow presented in Grossi et al.'s review in order to compare this technique with existing ones, both optimal and approxmiated. This entailed the following steps.

1. For each pair (a, b) of the N sequences, use the algorithm previously described to compute the approximately best rotation of sequence b.
2. Using EMBOSS Needle for pairwise sequence alignment with default parameters, compute a similarity score for (a_i, b), placing the result in cell [a, b] of an N x N similarity score matrix ((F et al., 2019).
3. Scale the similarity score matrix by converting the similarity score measures to a distance relative to the maximum similarity score in the matrix.
4. Utilize neighbor-joining clustering on the scaled similarity scores matrix to create a phylogenetic tree.
5. Compare the tree to the known, actual phylogenetic tree.

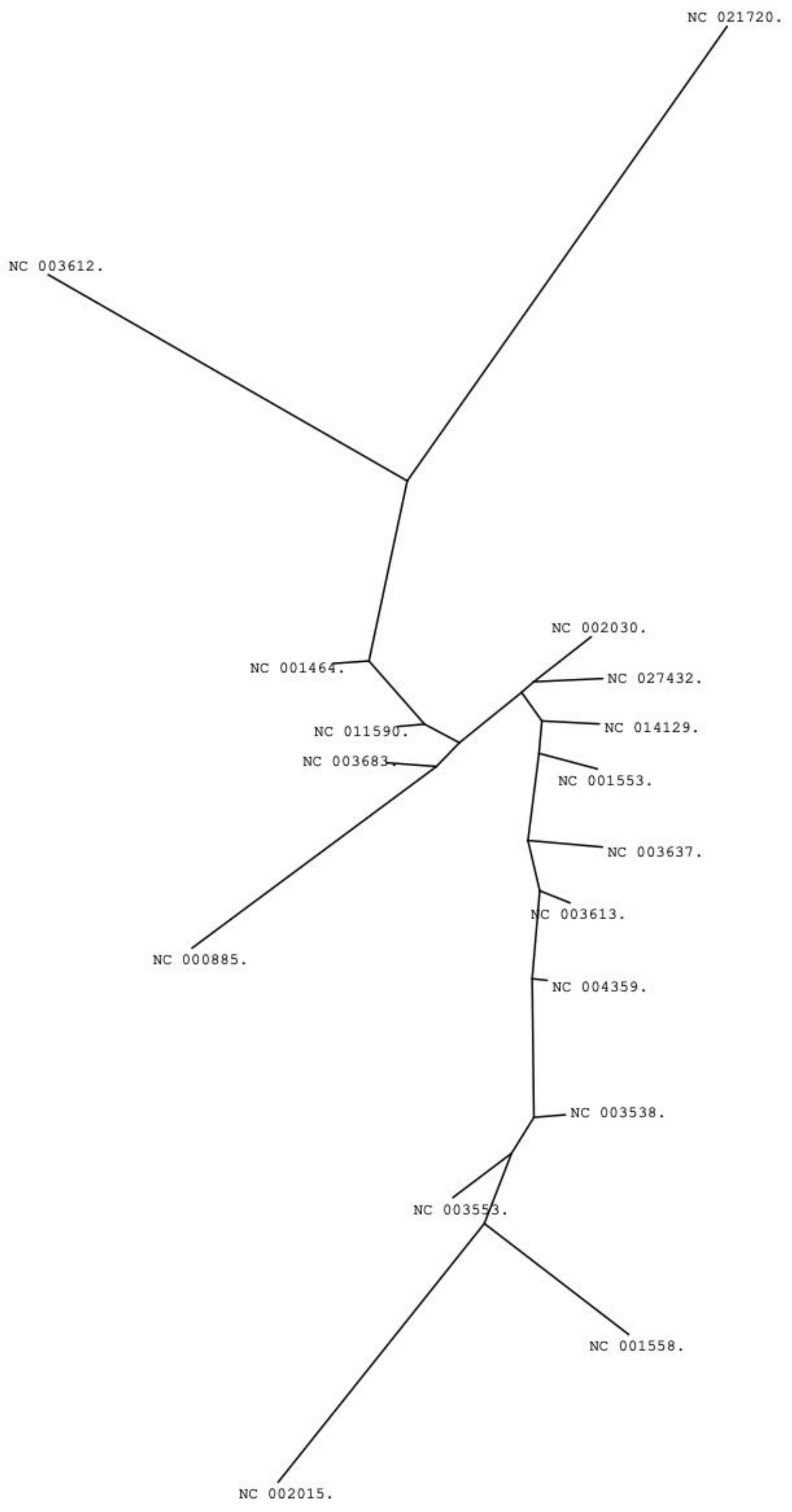
The matrix that follows contains the results of performing caLSH with each sequence as the alignment. I utilized neighbor-joining to cluster the lineages via the Phylip package (Dereeper et al., 2008)

Viroid Phylogenetic Distance Matrix

	NC_000885.1	NC_003683.1	NC_011590.1	NC_001464.1	NC_003612.1	NC_021720.1	NC_001558.1	NC_002015.1	NC_003553.1	NC_003538.1	NC_004359.1	NC_003613.1	NC_003637.1	NC_002030.1	NC_027432.1	NC_014129.1	NC_001553.1
NC_000885.1	0.00	0.41	0.41	0.41	0.41	0.41	0.43	0.44	0.41	0.44	0.41	0.39	0.41	0.45	0.40	0.36	0.42
NC_003683.1	0.41	0.00	0.42	0.41	0.44	0.40	0.43	0.41	0.41	0.41	0.43	0.40	0.44	0.41	0.39	0.37	0.42
NC_011590.1	0.41	0.42	0.00	0.42	0.41	0.41	0.42	0.43	0.41	0.41	0.42	0.39	0.41	0.41	0.39	0.40	0.43
NC_001464.1	0.41	0.41	0.42	0.00	0.42	0.40	0.43	0.43	0.39	0.40	0.41	0.39	0.43	0.42	0.40	0.38	0.44
NC_003612.1	0.41	0.44	0.41	0.42	0.00	0.40	0.43	0.42	0.40	0.40	0.44	0.40	0.44	0.42	0.40	0.38	0.42
NC_021720.1	0.41	0.40	0.41	0.40	0.40	0.00	0.40	0.42	0.40	0.41	0.40	0.38	0.40	0.41	0.41	0.37	0.41
NC_001558.1	0.43	0.44	0.42	0.43	0.43	0.40	0.00	0.44	0.42	0.44	0.42	0.39	0.43	0.42	0.39	0.36	0.44
NC_002015.1	0.44	0.00	0.43	0.43	0.41	0.42	0.44	0.00	0.41	0.44	0.41	0.41	0.42	0.44	0.41	0.37	0.44
NC_003553.1	0.41	0.41	0.39	0.39	0.40	0.40	0.42	0.41	0.00	0.41	0.41	0.38	0.41	0.41	0.39	0.37	0.43
NC_003538.1	0.44	0.43	0.41	0.40	0.40	0.41	0.44	0.44	0.41	0.00	0.41	0.39	0.43	0.44	0.40	0.36	0.44
NC_004359.1	0.41	0.43	0.42	0.41	0.44	0.40	0.42	0.41	0.41	0.41	0.00	0.38	0.43	0.42	0.40	0.38	0.42
NC_003613.1	0.39	0.40	0.39	0.39	0.40	0.38	0.39	0.41	0.38	0.39	0.38	0.00	0.41	0.40	0.44	0.37	0.41
NC_003637.1	0.41	0.44	0.41	0.43	0.44	0.40	0.43	0.42	0.41	0.43	0.43	0.41	0.00	0.43	0.40	0.36	0.43
NC_002030.1	0.45	0.41	0.41	0.42	0.42	0.41	0.42	0.44	0.41	0.44	0.42	0.40	0.43	0.00	0.42	0.37	0.42
NC_027432.1	0.40	0.39	0.39	0.40	0.40	0.41	0.39	0.41	0.39	0.40	0.40	0.44	0.40	0.42	0.00	0.39	0.40
NC_014129.1	0.36	0.37	0.40	0.38	0.38	0.37	0.36	0.37	0.37	0.36	0.38	0.37	0.36	0.37	0.39	0.00	0.37
NC_001553.1	0.42	0.42	0.43	0.44	0.42	0.41	0.44	0.44	0.43	0.44	0.42	0.41	0.43	0.42	0.40	0.37	0.00

Leveraging the distance matrix above, I utilized the Phylip software package to construct a tree based on nearest-neighbor clustering (Dereeper et al., 2008). This tree can be viewed in Figure 4. This tree is in agreement with the tree produced by Grossi et al.'s caCSC algorithm in that paper, indicating that caLSH is a valid method for producing alignments that can be used in phylogenetic tree construciton.

Figure 2: Phylogenetic tree constructed from caLSH distance matrix



Conclusion

In conclusion, the utilization of locality-sensitive hashing for the compression of biological sequences in order to find the best possible sequence linearization for pairwise alignment is promising. On samples of both longer, mitochondrial sequences and shorter, viroid sequences, the caLSH algorithm performed faster and with quite reasonable accuracy, improving upon the baseline alignment of database linearizations of genomes to a significant degree. Much further work exists in the exploring whether the algorithm can be used generally for sequence alignment improvement, including examination of the refining step, which produced good results for some sequences and actually reduced similarity between the sequences for some pairings. Additionally, I leveraged existing Python libraries to handle the parsing of FASTA-formatted files, one of many formats used for biological sequences. These may have reduced the overall efficiency of my program, which could ideally take advantage of dynamic programming to increase speed if rewritten with that goal in mind. Implementation in a language such as OCaml was a stretch goal discussed that I believe would render the algorithm even more efficient.

References

A. Mosig, IL Hofacker, PF Stadler. (2006). Comparative Aligment of Cyclic Sequences: Viroids and other Small Circular RNAs. *German Conference on Bioinformatics*, *83*, 93–102.

Benson, D. A., Karsch-Mizrachi, I., Lipman, D. J., Ostell, J., Rapp, B. A., & Wheeler, D. L. (2000). GenBank. *Nucleic Acids Research*, *28*(1), 15–18. https://doi.org/10.1093/nar/28.1.15

Dereeper, A., Guignon, V., Blanc, G., Audic, S., Buffet, S., Chevenet, F., Dufayard, J.-F., Guindon, S., Lefort, V., Lescot, M., Claverie, J.-M., & Gascuel, O. (2008). Phylogeny.fr: Robust phylogenetic analysis for the non-specialist. *Nucleic Acids Research*, *36*(Web Server issue), W465-469. https://doi.org/10.1093/nar/gkn180

F, M., Ym, P., J, L., N, B., T, G., N, M., P, B., Arn, T., Sc, P., Rd, F., & R, L. (2019). The EMBL-EBI search and sequence analysis tools APIs in 2019. *Nucleic Acids Research*, *47*(W1), W636–W641. https://doi.org/10.1093/nar/gkz268

Fernandes, F., Pereira, L., & Freitas, A. T. (2009). CSA: An efficient algorithm to improve circular DNA multiple alignment. *BMC Bioinformatics*, *10*, 230. https://doi.org/10.1186/1471-2105-10-230

Grossi, R., Iliopoulos, C. S., Mercas, R., Pisanti, N., Pissis, S. P., Retha, A., & Vayani, F. (2016). Circular sequence comparison: Algorithms and applications. *Algorithms for Molecular Biology : AMB*, *11*. https://doi.org/10.1186/s13015-016-0076-6

Needleman, S. B., & Wunsch, C. D. (1970). A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, *48*(3), 443–453. https://doi.org/10.1016/0022-2836(70)90057-4

Pruitt, K. D., Tatusova, T., & Maglott, D. R. (2007). NCBI reference sequences (RefSeq): A curated non-redundant sequence database of genomes, transcripts and proteins. *Nucleic Acids Research*, *35*(Database issue), D61-65. https://doi.org/10.1093/nar/gkl842

Rec06.pdf. (n.d.). Retrieved April 3, 2020, from http://courses.csail.mit.edu/6.006/spring11/rec/rec06.pdf