COMP 561 - Research Project

Sagar Nandeshwar - Student ID: 260920948

Topics: Alignment algorithms for probabilistic sequences

Abstract

In this paper, I have proposed a modified BLAST algorithm for probabilistic database sequence. I have evaluated the algorithm on portion of chromosome chr22 of the predicted BoreoEutherian ancestor sequence, by generating small sequences from the genome and randomly adding indels and sequences. Finally, I have evaluated the algorithm for its running time and accuracy.

1. Introduction

The BLAST algorithm, which stands for Basic Local Alignment Search Tool, is a vital tool in bioinformatics used to find local similarity between short queries (such as short DNA sequences) and larger databases (such as genome).

However, in curtained situations the exact genome of a species is not known, therefore we determine nucleotides at a given position with some probability. However, BLAST algorithm does not work in such probabilistic genome sequences.

In this project, I develop an analog of Blast that works to align a short (non-probabilistic) query sequence against a long probabilistic genome.

Finally, I tested the proposed algorithm for its running time and accuracy.

2. Background

2.1 Needleman and Wunsch Algorithm[2]:

The Needleman and Wunsch Algorithm is a dynamic programing algorithm that is used to find global optimal alignment between two sequences.

2.2 BLAST Algorithm[1]:

Let q be the short sequence and D be the database. The BLAST Algorithm is based on the intuition that if q has a good alignment with region X in D, then q and X are likely to contain at least one perfect match of length at least w.

Steps:

1: Find Short Perfect matches

For every word of size w in query q, find perfect match in database D. This region of perfect match is called seed.

2: Ungapped alignment extension phase

Expend the seed in both left and right direction but adding nucleotides without any gap in alignment and calculate, alignment score. We will stop as soon as the score starts declining

3: Gapped extension phase

For all the alignment in ungapped alignment extension phase, that passes certain threshold we, run Needleman and Wunsch Algorithm, to find Gapped alignment.

3. Blast Algorithm for Probabilistic sequences:

3.1 Datasets:

Database: Portion of chromosome chr22 of the predicted BoreoEutherian ancestor sequence.

Probability: Confidence value (between 0 and 1) for each of the positions in the above sequence

queries: I generated a short-query sequence by randomly selection starting point from the Database and added substitutions and indels.

3.2 Scoring Scheme

For Phase 1 and Phase 2:

Let w be the word and g be the genome. let I bet set of all index positon of word w. Then,

$$\text{scorePhase1(w)} = \sum_{i \in I} \text{matchScore(wi, gj)},$$

where, j is the corresponding position in genome g for index i in word w

$$\text{matchScore(wi,gj)} = \begin{cases} \text{(confidence value at gj) - (1 - confidence value at gj), if wi == gj} \\ \text{(1 - confidence value at gj) - [(confidence value at gj) + (2/3)*(1-confidence value at gj)], otherwise} \end{cases}$$

The score scheme is same for scorePhase2(w)

For Phase 3

I follow the score scheme of modified NW algorithm scores.

3.3 BLAST Algorithm:

Given: query q and genome G

Phase 1: Find Perfect Match

- Create list of all possible words with size w from query q
- For each of the word, find perfect match in genome G
- For all the perfect match, I then calculate the score scorePhase1(wn) where w is word with perfect match in genome G

Phase 2: Un-gapped phase extension

- For each of the perfect match word wn in Phase 1, I proceed with ungapped phase extension.
- For each for word wn, I add a nucleotide to the left and right of wn until

$$\text{curLeftScore * threshold} < \text{maxLeftScore}$$

similarly,

curRightScore * threshold < maxRightScore

- I then initialize curLeftScore and curRightScore with zero and maxRightScore = curRightScore and maxLeftScore = curLeftScore
- I follow the matchScore scheme, as I move to the left or right.
- The total score of phase 2 is calculated as

scorePhase2 = scorePhase1(wn) + maxLeftScore + maxRightScore

Phase 3: Gapped phase extension

- For the alignment of phase 2 whose score is greater than threshold2

scorePhase2 > threshold2

- I proceed with NW algorithm (Needleman and Wunsch Algorithm), for both left and right side.
- The final score for phase 3 is calculated as

scorePhase 3 = scorePhase2 + NW algorithm score for left + NW algorithm score for right

3.4 NW algorithm

Initialization:

I initialize the M (dynamic programming matrix as follows), as follows

- Assing M(0,0) = 0
- Assign row 0 of DP linear gap penalty, such that

row[i+1,0] = row[i,0] + c

- Similarly, Assign col 0 of DP_mat linear gap penalty, such that

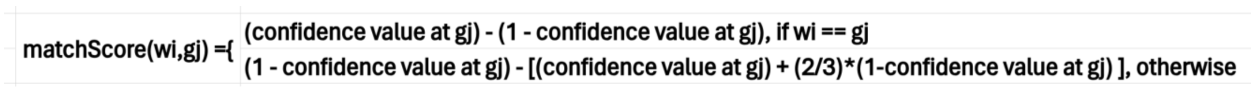
col[0,j+1] = row[0,j] + c

c = gap penalty; set it to -1

Filling matrix M:

M(i,j) = max{M(i-1,j-1) + matchScore(i, j), M(i-1,j) + c, M(i,j-1) + c}

Remember:



matchScore(wi,gj) ={	**(confidence value at gj) - (1 - confidence value at gj), if wi == gj**
	(1 - confidence value at gj) - [(confidence value at gj) + (2/3)*(1-confidence value at gj)], otherwise

Traceback:

Since size of genome >> size of word I select the max value of the last column and start traceback from there. I simply follow back the arrows and retrieve the sequence. Also, the max value of the last column is also the NM algorithm score.

4. Sample output:

1 Query:

CGGTGAAAGTGGTGTTGCCTGGAGGAGGGGCAAAGGGGCGGGCCAACTTGTGGGGG AGGTGCGGTGCCGGTCGACGCTGATTGACCGGGAGCTGAGGCC

Query Alignment:

CGGTGAAAGTG-GTGT-TGCCTGGAGGAGGGGCAAAGGGG-CG-GGCCAA-C-TTGTGGGGGAGGTGCGGTGCCGGTCGACGCTGATTGACCGAGCTGAGGCC

Genome Alignment:

CGGTGAAAGTGAGTGTCTGCCTGGAGGCGGGGCGAAGGGGCCGAGGCCAATCATTG T-GGGGA-GTGC-GTG-CGGTCGACGCTGATTGACAGAGCAAAGGCC

2 Query:

GCTCAGGGCGCTTCCTAGGGAAATACGCAAACCAGCAGCGGCGAGATCTTCTGGAG CCGCTTTTCTGGCAGAAGCTACTGTGCCCGCATCCAATTT

Query Alignment:

GCTCAGGG-C--GCTTCCTAGGGAAAT-ACGCAAACCAGCAGCGGC-GAG---ATC-T--TCTGGAGCCGCTTTTCTGGCAGAAGCTATGTGCCCGCATCCAATTT

Genome Alignment:

GAGCAGGGCCAGGC-TGCT-GAG-AGTCA-GC--TGCTGCAG-GGCTGAGCCAATCTTGCTCTGGAGCCGCCTGGCTGGCAACAGCTTTG-G---GAAGC-AAGAG

3 Query:

CCTGGCCCCTCAGCTGAGTAGCCCGGGGGCTGGAATCCCCTCCTGGTCCTGCGCCGC ACGCTCCCGGCCTGCGCGCCACCATCTGCACAGACCGG

Query Alignment:

CCTG--GCCCCTCAGCTGAGTAGCCCGG-GGGCT-GGAATCCCCTCCT--G-GTCCTGCGCCGCACGCTCCCGGCCTGCGCGCCACCATCGCACAGACCGG

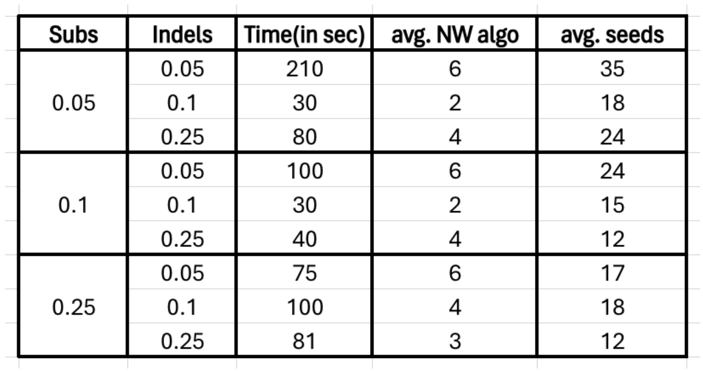
Genome Alignment:

CCCGCAGCCCCTCACCTGAGTAGCCCGGCGGGCTCGGGATCCCCTCCTCAGGGTCCC GCGCCGCACGCTTCC-TCCTGCGCGCCACCATCGC-C-GACCGG

5. Running Time

To test the running time of algorithms, I ran the program multiple times with different sets of indels probability, substitution probability, word length, and query size. I found that the major reason for the latency in the algorithm is because of the high number of seeds in ungapped extension phase and the no. of time it run NW algorithm in gapped extension phase.

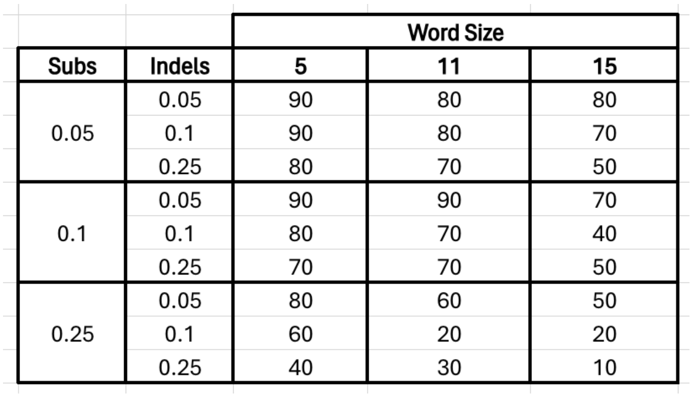
The running time word length 11, query sequence size ~ 50, threshold1 = 3 and threshold2 = 5



Subs	Indels	Time(in sec)	avg. NW algo	avg. seeds
0.05	0.05	210	6	35
	0.1	30	2	18
	0.25	80	4	24
0.1	0.05	100	6	24
	0.1	30	2	15
	0.25	40	4	12
0.25	0.05	75	6	17
	0.1	100	4	18
	0.25	81	3	12

6. Accuracy

I ran the following test 10 times and defined the accuracy for the experiment as the percentage of finding the database, in top 5 high scoring words.



		Word Size		
Subs	Indels	5	11	15
0.05	0.05	90	80	80
	0.1	90	80	70
	0.25	80	70	50
0.1	0.05	90	90	70
	0.1	80	70	40
	0.25	70	70	50
0.25	0.05	80	60	50
	0.1	60	20	20
	0.25	40	30	10

From the above graph we could see that the algorithm has a very high accuracy for short indels and mutations, and was able to identify alignments even with 25% mutation.

7. Choice of Parameter

For the choice of threshold, I found the following to be most feasible,

threshold1 = 3

threshold2 = 5

While threshold1, forces the algorithm to explore more in the gaped extension phase, threshold2, reduces the number of false positive words from the un-gapped tension phase and reduces the computation which is to be done with NW algorithm.

If we increase the threshold1, we will reduce the overall score of the alignments, and if we decrease the threshold1 we increase the no. of false positives words for phase 3. Similarly, if we increase the threshold2 we may miss many potential matches and may not find a match at all, whereas if we decrease the threshold2 we will have many words for NW algorithm, which would be computationally infeasible.

8. Conclusion

The proposed version of BLAST algorithm was able to find the region and align given sequences in probabilistic genome sequence. The algorithm could align short queries (length 50-100) with mutations of up to 25% in a feasible amount of time. With running time analysis, I also suspect the main reasons of latency could be the large number of seeds in ungapped extension phase and NW algorithm during gapped extension phase. I have also tested the algorithm with many different combinations of parameters (threshold1 and threshold2) and the reason that threshold1 = 3 and threshold2 = 5 are the most suitable for the short sequences. Lastly the algorithm has shown high accuracy of up-to 90% for minor indels and substitution.

9. Future Work

For the future work, I want to implement indexing, which could be used during the seed phase in reducing the searching time for perfect match. For the NW algorithm, we know that size of genome (n) is far larger than the size of queries (m), hence have a matrix (n*m), where there far more rows than column. For the future work I would want to reduce the size of rows with some estimation in order to reduce the no. computation.

Reference:

[1] Stephen Altschul; Warren Gish; Webb Miller; Eugene Myers; David J. Lipman (1990). "Basic local alignment search tool"

[2] Needleman, Saul B. & Wunsch, Christian D. (1970). "A general method applicable to the search for similarities in the amino acid sequence of two proteins". Journal of Molecular Biology.