COMP561 Final Project: A Probabilistic BLAST Algorithm

Grace Hu
260776936

Xingyu Chen
260786048

1 Introduction

Basic Local Alignment Search Tool (BLAST)[1] is a sequence similarity search program that can be used to quickly search a sequence database for matches to a query sequence. It is one of most broadly used tool in the field of bioinformatics. However, the traditional approach only considers the fixed genome: the optimal alignment is computed from the genome that has exact nucleotide at each position. In practice, it's hard to know exactly the genome of a species. The identity of each nucleotide may have some uncertainty at certain positions in the sequence data induced by experimental error. Traditional BLAST does not treat the reference sequence differently in the presence of probabilistic information, when we do not know with total certainty the exact base at certain position of the sequence.

An local alignment search tool that utilizes probabilistic information in the reference sequence data not only allow us to find better alignments, but also contributes to the analysis of ambiguous genome data in the real world. To answer this need, we present a probabilistic adaptation of BLAST by modifying the methods for indexing the database and aligning query and reference sequence. We test our algorithm on a DNA sequence data: a portion of CHR22, a predicted Boreoeutherian ancestor sequence, of 604,466 nucleotides is used as our reference sequence. Each nucleotide in the given sequence has an associated confidence value on the probability of that nucleotide being in that position in the real genome. We assume that each position in the sequence has equal chance of being one of the other three nucleotides.

The code for our implementation is on Github at gracexwho/probabilistic-blast. We consulted JiaShun-Xiao/BLAST-bioinfor-tool's deterministic Smith-Waterman algorithm to develop our own modified Smith-Waterman algorithm.

1	2	3	4	5	6	7	8	9	10
C	A	A	C	T	A	A	C	C	A
0.99	1.0	1.0	0.99	0.98	1.0	1.0	0.99	0.99	1.0

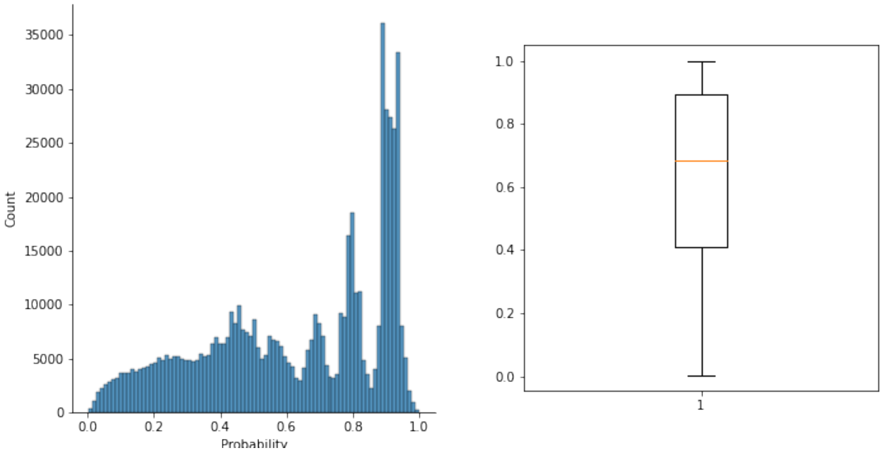
Table 1: First 10 nucleotides of the sequence with probabilities

2 Methods

BLAST runs in three steps: (1) it preprocesses the reference sequence by constructing a database of words of length w (w-mers) mapped to their position on the genome sequence; (2) upon input of a query sequence, it divides the query into words and returns a list of starting positions, whereupon it determines the high scoring pairs (HSPs) via ungapped extension; and finally (3) for each HSP, it extends the alignment using local sequence alignment in a gapped extension phase and selects the alignment(s) with the highest score [1]. To include the probability information in this process, we modified the way of comparing nucleotides in the local sequence alignment algorithm and developed new methods to index word database and find HSPs.

2.1 Build an Index of W-Mers

While examining the dataset, we noticed that the average probability of a nucleotide was 68.2%, and that 25% nucleotides had a probability of over 89.4% (Figure 1). This means that while most nucleotides have a relatively high probability, there are some positions where the nucleotide there is uncertain. For example, if the probability of the nucleotide being ’C’ in position 1 is only 40%, this means that by our assumptions, there’s a 20% chance the nucleotide at that position may actually be ’A’, ’T’, or ’G’. We made sure to take this into account while building the index/database.



(a) Frequency Dist. of Probabilities

(b) Box Plot of Percentage Quartiles

Figure 1: Analysis of CHR22 Boreoeutherian Genome Data

As part of the first step in any BLAST algorithm, we created a mapping of every possible w-length nucleotide sequence (w-mer) to its start position in the genome sequence and saved this index in a Pandas dataframe in a .csv file. For a w-mer of length 7, there 4^7 possible permutations. We would then go through the reference genome position by position with a sliding window of size w, and note down the starting position of each w-mer encountered in a database file titled ’BoreoEutherian_**w**.csv”, where w=7, 9, 11.

To take into account the probabilistic nature of the genome sequence, we assume that if the probability of a nucleotide exceeds 0.5 (50% probability), this nucleotide is considered to be deterministic. Otherwise, this position is treated as a wildcard character, and we will store all 4 variations of the w-mer in the database.

Let’s say that for w=4, the w-mer ’AAAA’ has probabilities 0.9, 0.3, 0.8 and 0.9 in each position. The second ’A’ with probability 0.3 cannot be considered deterministic, so we expand this w-mer to a set of w-mers 'A**A**AA', 'A**C**AA', 'A**G**AA', 'A**T**AA'. The starting position of the current sliding window is then stored in the database for all four w-mers.

2.2 Find High Scoring Pairs

2.2.1 Filtering W-mer Hits

To identify high scoring regions, we first break down the query to overlapping words by the given word length w. This means we would end up with query_length/w # of w-mers. Then we store the starting positions returned by all possible w-mers on the query. For each w-mer in the query, there will be many positions returned because just by random chance, you would expect $604466/4^w$ hits. But only one of these hits is the right location of that w-mer in the query. The right location should have many "neighbours" in other w-mers. For example, if location i of one w-mer is the right location, then in the position lists of other w-mers, we should find i+1,i+2,...until the end of the query assuming there are no mutations between the query and the genome. So to improve the running time of our algorithm, we go through all possible positions, and filter out the positions with 0 neighbours. The filtered out positions are highly unlikely to be the right location of the query, and in theory, we only need to find 1 out of the query_length/w right positions in order to end up with the right alignment.

2.2.2 Ungapped Linear Extension

Then we executed ungapped linear extension on the remaining, highly probable locations. We modified the scoring methods of matching and mismatching nucleotides: Let $\mathrm{P_i}^{\mathrm{max}}$ be the max probability of the ith nucleotide on reference sequence G. Let $\mathrm{G_i}^{\mathrm{max}}$ be the ith nucleotide with the largest probability on G, and let $\mathrm{Q_j}$ be the nucleotide on the position j of the query. Let $\mathrm{score_i}$ be the score of the ith nucleotide on G. If $\mathrm{G_i}^{\mathrm{max}}=\mathrm{Q_j}$,

$$score_i = P_{\mathrm{i}}^{\mathrm{max}}$$

Otherwise,

$$score_i = \frac{1 - P_{\mathrm{i}}^{\mathrm{max}}}{3}$$

We extend the alignment from both sides of the w-mer and add up the scores of each position, then divide by the extension length to obtain the average score. If the average score exceeds our threshold, we send this sequence to be an HSP, and pass it on to the gapped extension phase.

2.3 Gapped Local Sequence Alignment

The gapped extension algorithm we used is a modified version of the Smith-Waterman local alignment algorithm [2] that computes the alignment score for each candidate HSP. We used a weighted scoring scheme for matches and mismatches in the alignment, taking into account the probabilistic nature of the genome data.

The overall score of the candidate genome sequence at each position is then calculated by adding the score of the HSP with score of the local alignment. We compare the overall scores of all positions passed to stage 3, and the alignment with the highest overall score is returned as the optimal alignment. In the end, the program reports the alignment of query and genome data as well as the start index of optimal alignment position.

```
def score_gapped(self,seq1,seq2,pos1,pos2):
    match = 1
    mismatch = -1
    if seq1[pos1] == seq2[pos2]:
        return self.probs[pos1]*match + (1-self.probs[pos1])*mismatch
    else:

        return self.probs[pos1]*mismatch + (1-self.probs[pos1])/3*match
```

Figure 2: Weighted Scoring Scheme for Smith-Waterman Local Alignment

3 Results

3.1 Evaluation

There were various parameters we could fine-tune in our BLAST algorithm: the length of a word (w), what threshold does the score of an ungapped alignment need to surpass in order to be considered an HSP, and the linear gap cost. We conducted an assessment similar to GridSearchCV where we tested different values of parameters and chose the ones with the best accuracy (Figure 3). We settled on the parameters $w = 9$, $threshold = 0.7$, $gap = -0.75$ after many rounds of testing.

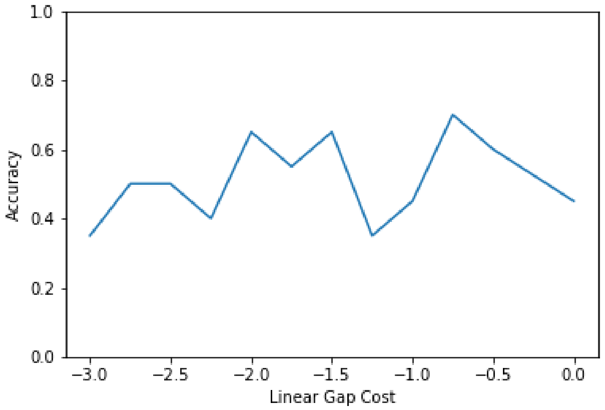


Figure 3: Effect of Varying Gap Cost on BLAST Accuracy for w=9, threshold=0.7

However, the average mammalian mutation rate is 2.210$^{(}$9) per base pair per year [3]. The common ancestor of Boreoeutherian mammals lived 100 to 80 million years ago [4], which means that if we're querying modern mammalian sequences against the Boreoeutherian genome, there would be a 20% mutation rate. Meanwhile, due to the limitations of our computational machines and the lack of memory on Deepnote, we're only testing our algorithm on short query sequences of length 50-200 nucleotides. A 20% mutation rate would mean that 10 of the 50 nucleotides in the query sequence have been altered. The w-mer lengths we tried were between 7-11 (wee did not

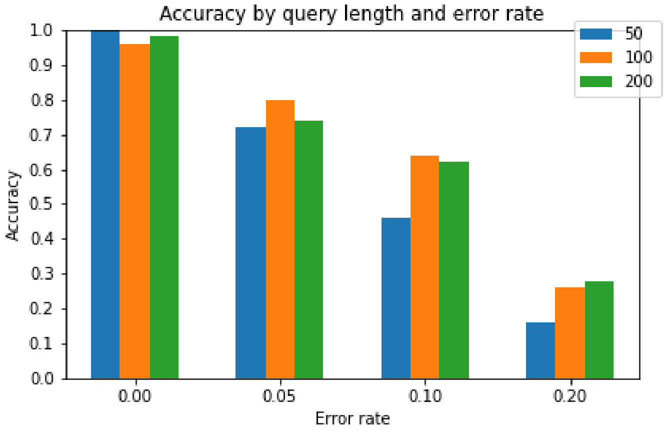


Figure 4: BLAST Algorithm Accuracy for Different Sequence Lengths

have enough memory to store a larger .csv file), and with a mutation rate of 20%, this means that there could have been a completely unfamiliar w-mer inserted into the query sequence.

To evaluate our algorithm, we generated short queries by picking an arbitrary position and length, and generated nucleotides based on the provided probability at each position on the genome sequence. Moreover, we added random substitutions, deletions and insertions based on the chosen mutation rates. Since the generated query probably contains multiple indels, the actual starting position does not actually equals the input position. Let the real start position be i, the number of insertions be m, and the number of deletions be n. Therefore, if the predicted position falls in the [i-m+n,i+m-n], we consider the predicted position to be accurate.

Our algorithm performed well when the mutation rate was set to 0 %; it achieved accuracies over 95% for queries of different lengths 4. As we increased the mutation rate, the accuracy of our algorithm decreased. At a mutation rate of 10% the accuracy rate was only around 60-70%. At a muation rate of 20%, the accuracy rate was only around 10-30% (Figure 4).

3.2 W-Mer Filtering Efficiency

To test the efficiency of our method of filtering w-mers before passing them to HSP localization, we run our filtering function on random queries with different length and rates of mutations(Figure.5). If the actual position in the genome where the query was taken from was present in the list of filtered w-mers (after removing w-mers with 0 neighbours), then we counted that as a hit. We plotted the accuracy of this filtering step in (Figure 5). For query with length greater tan 50 nucleotides, even with the highest error rate, the probability that the filtered positions contain the real positions is 100%.

3.3 Time and Space Complexity

Let l be the length of the Boreoeutherian genome sequence (604466 nucleotides), and q be the length of the query sequence. Let w be the chosen word length.

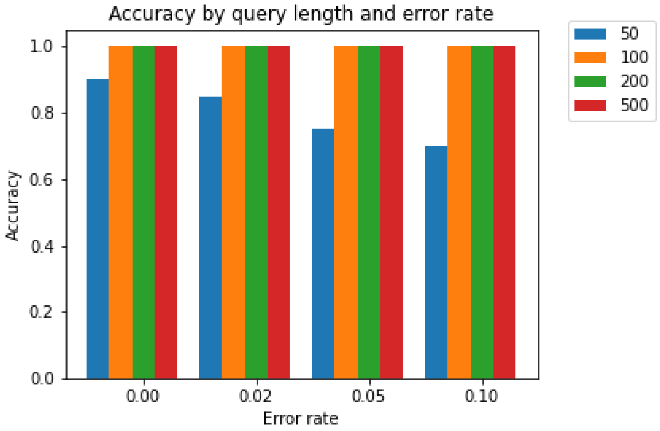


Figure 5: Accuracy of W-Mer Filtering Step

The time complexity for generating an index of w-mers is $O(l \cdot w)$, since we scan through the entire genome sequence once with a sliding window of length w. However, for each word in the genome, we assess the probability at each position to see if it's lower than 0.5. If it is, then we consider it a wildcard character and follow the pocess mentioned in Section 2.1. The database contains 4^w rows, so the space complexity of the index is $O(4^w)$. We also need to store the genome sequence and its associated probabilities in memory, so the space complexity of that storage is $O(l)$.

The time complexity of finding HSP is split into two stages: Filtering for W-mer Hits, and Ungapped Linear Extension.

For filtering the w-mer hits, we generate q words of length w from the query, and return the list of indices from the database. Accessing a Pandas dataframe via df.loc is $O(4^w)$. There are an average of $\frac{l}{4^w}$ possible indices for each word, and we go through all of these indices for each word and return the ones with 1+ neighbours. So the time complexity of this stage is $O(q \cdot 4^w \cdot \frac{l}{4^w}) = O(l \cdot q)$.

Next, we go through ungapped linear extension for each of the $\frac{l}{4^w}$ positions found. The HSP algorithm is $O(q)$, since the furthest it needs to extend a w-mer is the length of the entire query. So the time complexity of this stage is $O(\frac{l \cdot q}{4^w})$

For the final stage, gapped extension, we call the Smith-Waterman algorithm for all HSPs above the threshold, so a maximum of $O(\frac{l}{4^w})$ calls are made to S-W. S-W has a $O(l \cdot q)$ time complexity [2], and also a $O(l \cdot q)$ space complexity. The tiem complexity of this stage is $O(\frac{l \cdot l \cdot q}{4^w})$.

Together, the overall time complexity of our probabilistic BLAST algorithm is:

$$O(l \cdot w) + O(l \cdot q) + O(\frac{l \cdot q}{4^w}) + O(\frac{l \cdot l \cdot q}{4^w}) = O(l^2 \cdot q)$$

3.4 Limitations

Although being guaranteed with certain level of optimality, this program does contain limitations regarding its implementation. We tried using other deterministic algorithms on the Boreoeutherian sequence, and obtained similar alignments as our own S-W algorithm. Combined with the fact that we introduced mutations into our query sequences to represent real life situations, this means that there were many indels that may have caused the algorithm to fail to return the optimal alignment at high mutation rates.

Because Smith-Waterman is a local alignment algorithm and aligning all of the query sequence with a short stretch of the genome sequence is more similar to a global alignment, it may not be able to perform that well. In addition, we used a linear gap penalty cost instead of an affine gap penalty cost due to computational resources limitations (an affine gap penalty scheme would require 3 arrays of length $O(l \cdot w)$ compared to 1 for a linear gap penalty scheme). When printing out our alignments, we saw some cases where the S-W algorithm inserted unnecessary gaps near the end of the sequence (Figure 6). When we tried using other deterministic implementations of S-W, we got the same results. This seems to be an inherent limitation of the linear gap penalty scheme S-W algorithm.

```
Scanning Query... CGACACTCCCAGCTAGTGAGGAGCTCAAAGTCAAGCACTTAGAAGAGATA
Query:  C---G-----------ACACTCCCAGCTAGTGAGGAGCTCAAAGTCAAGCACTTAG-A-AG--A-G-------ATA
Genome: CTAAGACAAATTTACAACACTCCCAGCACGTGAGGAGCTCAAAGTCAAGCACTTAGAAGAGATATGTTAAATTATA
```

Figure 6: Incorrect S-W alignment generated by our algorithm at the correct position

4 Conclusion

We were able to devise a BLAST algorithm for probabilistic genome data by using a mixture of weighted scores and the insertion of wildcard characters during the index creation phase. Our algorithm performed extremely well when the mutation rate was 0 and query sequences were derived exactly from the genome sequence. However, as mutation rates increased, the accuracy of the algorithm decreased significantly. There are still many areas that can be improved, and there are definitely other approaches to handling the probabilistic data that can be explored in future work.

References

[1] Stephen F. Altschul, Warren Gish, Webb Miller, Eugene W. Myers, and David J. Lipman. Basic local alignment search tool, Feb 2007.

[2] William R Pearson. Searching protein sequence libraries: comparison of the sensitivity and selectivity of the smith-waterman and fasta algorithms. *Genomics*, 11(3):635–650, 1991.

[3] Sudhir Kumar and Sankar Subramanian. Mutation rates in mammalian genomes. *Proceedings of the National Academy of Sciences*, 99(2):803–808, 2002.

[4] William J Murphy, Eduardo Eizirik, Stephen J O'Brien, Ole Madsen, Mark Scally, Christophe J Douady, Emma Teeling, Oliver A Ryder, Michael J Stanhope, Wilfried W de Jong, et al. Resolution of the early placental mammal radiation using bayesian phylogenetics. *Science*, 294(5550):2348–2351, 2001.