BLAST Algorithm Analog for Alignment Against Probabilistic Sequences

Le Nhat Hung (260793376), Jovi Sidhu (260660052)
COMP 561
FALL 2019

Introduction and Background:

The normal BLAST algorithm is a Basic Local Alignment Search Tool that aims to find the optimal alignment of a query sequence against a target database sequence. This is done by first breaking the query sequence into substrings (words) of defined length. The word length will affect the number of hits (word alignments) found and the time it takes to find hits. Once a word (substring from the query sequence) is found, it is extended in both directions in what is called the ungapped phase, where the word alignment (seed) is extended in either direction, computing the score of the alignments and keeping track of the current maximum. The extension stops when the score deviates past a threshold from the maximum score found during the extension. After this, a gapped extension phase occurs, whereby using the Needleman-Wunsch algorithm, the optimal local alignment of the query against the database sequence can be found by extending with gaps in either direction from where the ungapped extension phase left off.

Where normal BLAST aligns a query sequence against a database sequence where each position in the database is one nucleotide, the goal of this project is to create an analog of BLAST that aligns the query against a sequence where each position has a partial confidence for each possible nucleotide (A, C, T, G). This probabilistic sequence requires various adjustments and considerations have to be made to BLAST in order for it to achieve the same function. The most basic operation of a nucleotide comparison between a position in the query and the database sequences is performed by looking the nucleotide probability at the position of the database for the corresponding nucleotide in the query. If the probability meets or exceeds a threshold probability value, then the comparison is seen as a "match", if the threshold is not met then the comparison is seen as a mismatch.

Methods:

The work on this BLAST analog began with understanding how BLAST works, and where the adjustments for the probabilistic sequences need to be made.

The regular BLAST parameters are as follows:

Word length: This defines the word size for the seed used in the first phase of the alignment.

Delta threshold: This defines the threshold for the difference between the score at the current point of ungapped extension from the maximal score so far during the ungapped extension.

HSP threshold: This defines the threshold score for alignments that have finished ungapped extension to move onto gapped extension.

Our implementation uses all the above parameters, in addition to a *hit threshold* parameter

Database indexing:

The first consideration, as mentioned above, required adjusting the comparison model to compare the query nucleotide to the nucleotide probability given by the probabilistic sequence for that given position. When determining our scoring model for the per-nucleotide comparisons, the main idea was to reward probabilities the more they exceeded the threshold, and penalize any scores in general that failed to meet the threshold but to make it more dynamic than having the threshold be a limit for making a binary pass/fail for match/mismatch. For the word alignment, each probability of each position for the length of the word alignment is multiplied together, so each word alignment will have a probability that is the product of each of it's alignment positions probabilities. This product has to meet the product of the word length multiplied by the log of the hit threshold:

$$seed\ threshold\ =\ word\ length\ *\ log(hit\ threshold)$$

Ungapped extension:

The ungapped extension operates similar to normal BLAST, except in order to keep track of a “current maximum” alignment score, we compute a score where positive scores are given for each comparison where the threshold is met by taking the value of the probability of the corresponding nucleotide if it exceeds or meets the threshold. A negative score is given where the threshold is not met by subtracting the probability for that corresponding nucleotide in the probability table at that position from the *hit threshold* value.

$$hit\ threshold = 0.8$$

$$q = ...AAGCT...$$

$$d = \begin{array}{c} A \\ T \\ G \\ C \end{array} \begin{bmatrix} & 0.9 & 0.6 & 0 & 0 & 0 & \\ \ldots & 0 & 0.2 & 0.2 & 0 & 0.1 & \ldots \\ & 0 & 0.1 & 0.8 & 0 & 0.9 & \\ & 0.1 & 0.1 & 0 & 1 & 0 & \end{bmatrix}$$

$$score = ... + 0.9 - (0.8 - 0.6) + 0.8 + 1 - (0.8 - 0.1) + ...$$

Figure 1 : Per-position comparison score in ungapped extension

This way, we are able to mimic the way normal BLAST operates by having a similar method for rewarding scores which can be adapted by the extension phases. During ungapped extension, we still stop the extension when the current best score deviates from the current score of the extension enough that it exceeds a defined “delta” value. The values for the hit threshold and the delta (current best score - current score at the extended end) is user defined, in the context of our project we experimented with different values for each which is explored in the Results and Discussion. We aimed to set more conservative constraints for these values as to encourage higher quality alignments.

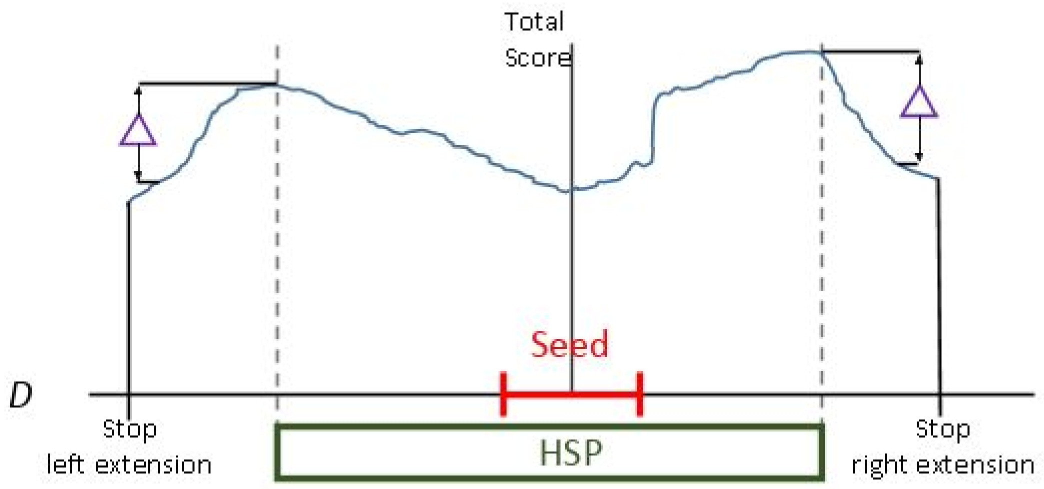

Figure 2 : Ungapped extension performed on both sides of the seed. The score is computed according to our previously described probabilistic per-position comparison scheme

Ungapped extension will then finally calculate the E-value from the HSP score. In regular BLAST, E-values is generated from the following formula:

$$E = Kmne^{-\lambda S}$$

Where *m* and *n* represent the query and database sequence lengths, and *K* and *lambda* are natural scaling factors, and *S* is the HSP score. *K* is for the search space size and *lambda* is for the scoring system. This would still provide a representation of the alignment that factors in the lengths of the query and the database Omitting further details about *K* and *lambda*, our implementation uses *K* = 0.711 and *lambda* = 1.37.

If an HSP's e-value is lower or equal than the e-value threshold of 2, this HSP will be qualified and allowed to proceed into **Gapped Extension**.

Gapped Extension:
Like regular BLAST, Needleman-Wunsch is used during gapped extension. Unlike regular BLAST, the match and mismatch scores are determined by the new probabilistic per-position comparison, as previously used in ungapped extension. The dynamic programming table expands layer by layer, keeping a max score so far, and stopping once the difference between the max score so far and current layer's max score exceeds the *delta threshold*.

The overall alignment score reported for an optimal local alignment of the query is thus determined by adding the score of seed to that of the HSP to that of the gapped extension.

Accuracy:
The algorithm outputs a percentage accuracy score for each alignment. This is calculated from:

$$\frac{\text{score of whole alignment}}{\text{score of database segment of alignment with gaps removed}}$$

Both scores are calculated from the probabilistic per-position comparison

Results:

When exploring how our algorithm performed, we performed various trials in order to examine how the parameters we defined shaped the structure and number of alignments found. The output files of our algorithm for the parameters discussed in this section are all included in the “output/” directory of project folder submitted with this report, the format of these output files includes the parameters in the name of the file as shown below, where Xs indicate the given values for those parameters.

alignments.q.X.fa.w.X.hit_thres:X.delta:X.hsp_thres:X.json

To speed up testing, we generated and used an index table for one word size, W=7. Due to limitations in time generating the index table for larger word lengths, we did all of our passes of our algorithm using word length 11. Given more time and more computation power, we would expand our analysis to a larger word size, such as W>11 to explore how it compares to our current findings.

We experimented with varying the hit-threshold and delta threshold parameters in order to observe any trends in how it affects the alignments produced. Lowering the hit-threshold from 0.9, to 0.8 or 0.6 resulted in more alignments and longer alignments on average, the lower the threshold value was. A few more alignments were seen at 0.8, and a magnitude more at 0.6 . Changing the threshold value from our self-defined basal value of 2.50 down to 1.00 caused fewer final alignments to be found that were as well much shorter in length. Changing the delta threshold from 2.50 to 10.00 caused the number of final alignments to decrease as well due to the alignments being longer and thus the ungapped extension would tolerate decreasing scores for much longer, by the time it would stop, these very poor scores would not meet the HSP threshold resulting in many HSPs not moving forward to the gapped extension phase.

Altering the HSP threshold has a great effect on the number of final alignments as it determines how many alignments can proceed to the gapped extension phase. In general a given HSP threshold creates a minimum possible alignment length due to how long an alignment would need to be to have a sufficient score to meet the HSP threshold. Since hit-threshold produces longer alignments, a low hit threshold and low HSP threshold would produce too many poor quality final alignments. A high hit-threshold that restricts sequence length out of the ungapped extension phase would not be paired well with a high HSP threshold.

This creates a basis for choosing a middle ground delta threshold that doesn't restrict growth during expansion phases (low delta-threshold = 1.00) and that also prevents scores from getting too low (high delta-threshold = 10.00). As well, from this our hit-threshold needs to not be too low as not tolerate low quality comparisons that have low confidence; a high hit-threshold produces good final alignments without diminishing the number of valid alignments found or their lengths.

Discussion:

In the process of creating the algorithm, we initially played around with different methods of scoring per-nucleotide comparisons, ungapped extension and ungapped extension. Initially we planned to keep the scoring of each stage as simple as possible to keep the algorithm straightforward.

As each major component, i.e. per-position comparison, ungapped and gapped extension phases, was completed, we backtracked and discussed ways to improve on what we had in order to provide a more robust score.
The first change was to, instead of giving a constant positive and negative value to the per-position comparison e.g. 1 and -1, we deducted the difference between the "hit" threshold and the position confidence if it was lower than the threshold, and added entire position confidence if it was greater or equal to the threshold. This was the first step into rewarding more for having a confidence exceeding the threshold, increasingly negative penalties to those that did not. This ties into our main idea when designing this algorithm, which is to make more meaningful decisions around how the positions are scored by using a dynamic model that takes into account the deviation of the probability from the threshold rather than just using the probability threshold to mimic the binary match/non-match of normal BLAST.

There were many additional areas that we could explore given more time in regards to addressing the accuracy of our algorithm.

Additionally, it was of interest to adapt the scoring model to incorporate an affine gap penalty instead of the linear gap penalty that we integrated into our algorithm. Normal BLAST makes regular use of an affine gap penalty, which would mostly have an affect on our overall scores as better alignments might be missed affecting our potential accuracy with just a linear penalty. Our gapped extension initially used standard Needleman-Wunsch with constant match and mismatch scores, which we then adapted to our probabilistic per-position comparison.

For per-position comparison, exploring an exponentially increasing threshold fail penalty - as opposed to our current linear method - would be interesting. Potential outcomes of this implementation could include premature stops in the ungapped extension phase, resulting in fewer potential HSPs and thereby fewer and shorter final alignments.

Time complexity:

For the database indexing phase, we build a table with $alphabet\ size^{word\ size}$ possible seeds, and checking $database\ size - word\ size + 1$ windows in the database sequence. For each window, we check $word\ size$ positions and $alphabet\ size$ probabilities for each position. This yields:

$$O(S^w(n-w+1)*Sw) = O(S^{w+1}n)$$

S: alphabet size

n: database size

w: word size

For all extension phases, we check $query\ length - w + 1$ windows of the query, for each of which we will check $n - w + 1$ windows of the database. For each database windows, we need to check w positions and S probabilities for each position. Finally, we are extending at most $query\ length$ positions for each window in the database sequence. This yields:

$$O((m-w+1)(n-w+1)m*Sw) = O(Sm^2n)$$

m: query length

Summing up, we get:

$$O(m^2n + S^wn)$$

However, as the database only needs to be indexed once for different *w* and *hit threshold*, we can disregard the second term in normal use.

Accuracy:

We evaluated the accuracy by generating a short sequence using the database's probabilities. This short sequence is then used as the query to run our algorithm. As expected, almost all seed hits extend up to the whole length of the query in the ungapped extension phase alone. These would become final alignments with 100% accuracies and E values reaching infinitesimally close to 0 (orders of magnitude of 10e-181).

We also generated completely random queries which, we tested against the E value threshold, almost never qualify and proceed to gapped extension, as their E values were never below 1.

These observations are congruent with our expectations of the algorithm's performance.

Conclusions:

The overall performance of algorithm met our expectations for accuracy and functionality. More tests could have been performed with a larger, structured approach to understand the effects of different variations of the parameters mentioned except with varying word size, but the analysis performed so far gives a good basic understanding as to the nuance of how the adjustments made to BLAST, to accommodate alignment against probabilistic sequences, affect its performance and output.

Diagrams:

As our algorithm was primarily focused on generating scores and alignments, there were not necessarily diagrams but instead printouts of the final alignments, in the form of json files, given certain initial conditions. These files as mentioned and explained earlier (please refer to the start of the results section) are formatted to be as self-explanatory as possible, and are included in the project folder under the "output" sub folder.

Any and all code used to generate the results explored and mentioned in the report is in included in the project folder that was submitted alongside this PDF.

Bibliography:

[1] *Wheeler, David, and Medha Bhagwat. "Chapter 9: BLAST QuickStart." Comparative Genomics: Volumes 1 and 2., U.S. National Library of Medicine, 1 Jan. 1970,* ***Used in understanding and confirming the basic method in which blast operates.*** *https://www.ncbi.nlm.nih.gov/books/NBK1734/.*

[2] *"The Statistics of Sequence Similarity Scores." National Center for Biotechnology Information, U.S. National Library of Medicine,* ***Used for understanding how the expect value is calculated and its significance.*** *https://www.ncbi.nlm.nih.gov/BLAST/tutorial/Altschul-1.html.*

[3] *Professor Mathieu Blanchette. Ungapped Extension Phase Graph [Figure 2]*