DNA SEQUENCING AND MATCHING

Amrutha R
CB.EN.U4CCE22003
Department of Electronics and Communication Engineering Amrita Vishwa Vidyapeetham Coimbatore, India

Amrutha Varshini P
CB.EN.U4CCE22004
Department of Electronics and Communication Engineering Amrita Vishwa Vidyapeetham Coimbatore, India

Eashita Prabhudesai
CB.EN.U4CCE22019
Department of Electronics and Communication Engineering Amrita Vishwa Vidyapeetham Coimbatore, India

Kaavyaa Aravind
CB.EN.U4CCE22028
Department of Electronics and Communication Engineering Amrita Vishwa Vidyapeetham Coimbatore, India

***Abstract*—Our aim to create a genetic matching and visualization application which has a no of applications including detection of exact matches between two given DNA samples(Smith-Waterman algorithm), finding an approximate match for a given DNA sample from a list of DNA samples(Booyre moore algorithm), other applications include mutation detection, an outcome of which is cancer detection(hamming distance method which is explained in the following) , species detection(upgma trees method) etc.A phylogenetic tree is constructed and an analysis is possible through the following method-DNA sequences for different species are aligned to compute the genetic distances between sequences. The genetic distances are calculated based on sequence identity. The resulting phylogenetic tree represents the evolutionary relationships between the species, with branches of varying lengths, where branch lengths are proportional to the genetic distances between species. For our application we will be utilising the hamming distance algorithm for mutation detection in the given DNA sequence and smith watermann algorithm for pattern matching between any two given DNA samples. we will also be calculating a particular parameter called the hamming distance.We will be using a hash table to store the DNA sequences as keys and their identifiers as values, the add_sequence method allows us to add DNA sequences with corresponding identifiers to the hash table then we will be checking if a particular sequence exists in the hash table and returns the corresponding identifier if found this completes the match detection process. An added feature is the display of features using weighted graphs allowing for graph-based analysis (e.g., finding cycles, paths) and also visualize the data.**

Keywords—component, formatting, style, styling, insert (*key words****)***

I. INTRODUCTION (*HEADING 1*)

In the era of advancing genetic research, the need for robust tools that facilitate the analysis and interpretation of DNA sequences has become increasingly critical. Our research endeavors to address this demand by developing a comprehensive Genetic Matching and Visualization Application, designed to perform various DNA-related analyses with a focus on accuracy, efficiency, and versatility.

The foundation of our application rests on the integration of state-of-the-art algorithms catering to distinct genetic analysis needs. The Smith-Waterman algorithm is employed for precise pattern matching between two DNA samples, enabling the detection of exact matches. In parallel, the Boyer-Moore algorithm facilitates approximate matching, extending the application's utility to scenarios where genetic sequences may vary slightly.

Central to our application is the Hamming Distance method, utilized for mutation detection within DNA sequences. The Hamming Distance parameter becomes a pivotal tool in our quest for identifying genetic anomalies, including those associated with cancer. By leveraging a hash table to efficiently store and retrieve DNA sequences, our application ensures swift and accurate matching processes, contributing to the detection of specific mutations and potential cancer markers. A distinctive feature of our application lies in its ability to visualize genetic data through weighted graphs. This facilitates graph-based analyses, enabling the identification of cycles, paths, and other structural features within the genetic data. The visualization component enhances the interpretability of complex genetic information, offering a more intuitive understanding of the relationships and patterns present in the data.

II. LITERAURE SURVEY

Genetic analysis has witnessed significant advancements in recent years, driven by the increasing availability of genomic data and the growing importance of understanding genetic information for various applications. Our Genetic Matching and Visualization Application draw inspiration and methodology from a diverse range of studies and tools across several domains.

1. Sequence Alignment Algorithms:

The foundation of our project lies in the utilization of sequence alignment algorithms for accurate DNA pattern matching. The Smith-Waterman algorithm, a dynamic

programming approach, finds its roots in bioinformatics applications for local sequence alignment. Extensive literature has highlighted its efficacy in identifying similarities between DNA sequences, forming the basis for our precise matching capabilities.

2. Approximate Matching and Boyer-Moore Algorithm:

To broaden the application's scope, we incorporate the Boyer-Moore algorithm for approximate matching. This algorithm, known for its efficiency in string searching, is adapted to handle variations in genetic sequences. Literature in the realm of bioinformatics underscores the significance of approximate matching in scenarios where genetic sequences may exhibit minor differences, providing a comprehensive understanding of genetic variations.

3. Mutation Detection and Hamming Distance:

The incorporation of the Hamming Distance method for mutation detection aligns with the broader objective of identifying genetic anomalies, including those associated with diseases such as cancer. Relevant studies in cancer genomics emphasize the role of mutation detection algorithms in uncovering potential biomarkers and aberrations within DNA sequences, underlining the clinical relevance of our chosen methodology.

4. Phylogenetic Analysis and UPGMA Trees:

Our project delves into the realm of phylogenetic analysis, drawing inspiration from studies that employ the Unweighted Pair Group Method with Arithmetic Mean (UPGMA) algorithm. Existing literature highlights the significance of UPGMA in constructing evolutionary trees based on genetic distances, offering insights into the evolutionary relationships between species. This application of our project finds resonance with studies focusing on taxonomy, ecology, and evolutionary biology.

5. Hash Tables in Bioinformatics:

The use of hash tables to efficiently store and retrieve DNA sequences aligns with best practices in bioinformatics data management. Relevant literature emphasizes the importance of data structures, such as hash tables, in optimizing sequence matching processes, reducing computational overhead, and enhancing the scalability of genetic databases.

6. Graph-Based Visualization in Genomics:

The incorporation of weighted graphs for data visualization finds support in the literature on graph theory applications in genomics. Studies highlight the utility of graph-based representations in capturing intricate relationships within genetic data, facilitating the identification of structural features, cycles, and paths. This aligns with our aim to provide researchers with a powerful tool for intuitive and insightful analysis of genetic information.

In summary, our literature survey draws from a rich tapestry of studies in bioinformatics, genomics, and computational biology. The integration of established algorithms and methodologies positions our Genetic Matching and Visualization Application at the intersection of cutting-edge research, contributing to the ongoing efforts to unravel the complexities of genetic data.

III. METHODOLOGY

1. SMITH WATERMANN ALGORITHM:

A. Working of Smith-Waterman Algorithm :

1) Intialization of Matrix

The basic steps for the algorithm are similar to that of Needleman-Wunsch algorithm. The steps are:

1. Initialization of a matrix.
2. Matrix Filling with the appropriate scores.
3. Trace back the sequences for a suitable alignment.

To study the Local sequence alignment consider the given below sequences.

CGTGAATTCAT (sequence#1 or A)
GACTTAC (sequence #2 or B)

The two sequences are arranged in a matrix form with A+1columns and B+1rows. The values in the first row and first column are set to zero as shown in Figure 1.

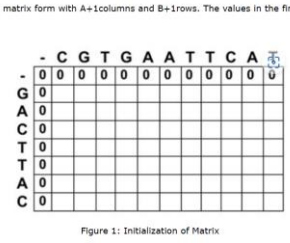

Figure 1: Initialization of Matrix

Variables used:

i,j describes row and columns.
M is the matrix value of the required cell (stated as $M_{i,j}$)
S is the score of the required cell ($S_{i,\ j}$)
W is the gap alignment

1) Matrix Filling

The second of the algorithm is filling the entire matrix, so it is more important to know the neighbor values (diagonal, upper and left) of the current cell to fill each and every cell.

$$M_{i,j} = Maximum\ [M_{i-1,j-1} + S_{i,j},\ M_{i,j-1} + W,\ M_{i-1,j} + W, 0]$$

As per the assumptions stated earlier, fill the entire matrix using the assumed scoring schema and initial values. One can fill the 1st row and 1st column with the scoring matrix as follows.

The first residue (nucleotides or amino acids) in both sequences is ‘C’ and ‘G’, the matching score or the mismatching score is going to be added the neighboring value

which is diagonally located i.e. 0. The upper and left values are added to the gap penalty score from the matrix. So the scoring schema equation can be shown as follows.

$$
\begin{aligned}
M_{1,1} &= Maximum\ [M_{0,0} + S_{1,1},\ M_{1,0} + W,\ M_{0,1} + W,\ 0] \\
&= Maximum\ [0(-3), 0 + (-4), 0 + (-4), 0] \\
&= Maximum\ [-3, -4, -4, 0] \\
&= 0
\end{aligned}
$$

From the above calculations the maximum value obtained is 0. Finding the maximum value for $M_{i,j}$ position, one can notice that there is no chance to see any negative values in the matrix, since we are taking 0 as lowest value.

After filling the matrix, keep the pointer back to the cell from where the maximum score has been determined. In the similar fashion fill all the values of the matrix of the cell.

For the example the matrix can be filled is shown in Figure 2.

	-	C	G	T	G	A	A	T	T	C	A	T
-	0	0	0	0	0	0	0	0	0	0	0	0
G	0	0	5	1	5	1	0	0	0	0	0	0
A	0	0	1	2	1	10	6	2	0	0	5	1
C	0	5	1	0	0	6	7	3	0	5	1	2
T	0	1	2	6	2	2	3	12	8	4	2	6
T	0	0	0	7	3	0	0	8	17	13	9	7
A	0	0	0	3	4	8	5	4	13	14	18	14
C	0	5	1	0	0	4	5	2	9	18	14	15

Figure 2: Matrix filling with back pointers

Each cell is back pointed by one or more pointers from where the maximum score has been obtained.

1) Trace backing the sequences for an optimal alignment:

The final step for the appropriate alignment is trace backing, prior to that one needs to find out the maximum score obtained in the entire matrix for the local alignment of the sequences.It is possible that the maximum scores can be present in more than one cell, in that case there may be possibility of two or more alignments, and the best alignment by scoring it.

In this example we can see the maximum score in the matrix as 18, which is found in two positions that lead to multiple alignments, so the best alignment has to be found.

So the trace back begins from the position which has the highest value, pointing back with the pointers, thus find out the possible predecessor, then move to next predecessor and continue until we reach the score 0 (Figure 3).

Figure 3: Trace back of first possible alignment

It is possible to find two pointers pointing out from one cell, where both ways(alignments) can be considered, best one is found by scoring and finding maximum score among them.

It is possible to find two pointers pointing out from one cell, where both ways(alignments) can be considered, best one is found by scoring and finding maximum score among them.

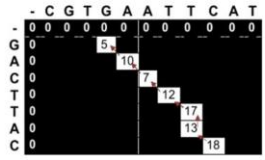

Figure 4: Trace back of second possible alignment

Thus a local alignment is obtained and one can see the possible alignments as in Figure 5.

```
G A A T T C A   G A A T T - C
| | | | | |   | | | | | |
G A C T T - A   G A C T T A C

+ + - + + - +   + + - + + - +
5 5 3 5 5 4 5   5 5 3 5 5 4 5
```

Figure 5: Scoring for best alignment

The two alignments can be given with a score, for matching as +5 , mismatch as -3 and gap penalty as -4, sum up all the individual scores and the alignment which has maximum score after this can be taken as the best alignment.

By summing up the scores both of the alignments are giving the same as 18, so one can predict both alignments are the best.

2.Booyre Moore Algorithm

Boyer-Moore Algorithm for Approximate String Matching:

Understand the Input:

Ensure you have a pattern string and a text string in which you want to find approximate matches. Additionally, specify the maximum allowable number of mismatches.

Preprocessing:

1. Bad Character Rule:

- Create a bad character shift table. For each character in the pattern, determine the maximum distance to shift when a mismatch occurs in the text. If the mismatched character is not in the pattern, shift by the length of the pattern.

2. Good Suffix Rule:

- Create a good suffix shift table. For each position in the pattern, determine the maximum distance to shift when a mismatch occurs based on matching suffixes.

Searching:

1. Initialize Variables:

- Set a variable (e.g., pos) to zero. This will be used to track the current position in the text.

2. Main Search Loop:

- While the remaining length of the text is greater than or equal to the length of the pattern:

- Check for a match by comparing characters from right to left.

- If a mismatch occurs, apply the bad character and good suffix rules to determine the shift distance.

3. Mismatch Handling:

- If the mismatched character in the text is in the pattern, use the bad character rule to shift to the right.

- If the mismatched character is not in the pattern, shift by the length of the pattern.

4. Good Suffix Handling:

- If there is a matching suffix in the pattern, use the good suffix rule to shift to the right.

5. Update Position:

- Update the position (pos) based on the calculated shift distance.

6. Repeat:

- Continue the main loop until the entire text is processed.

Results:

- The positions where approximate matches are found are stored in a list.

- The final list of positions represents the approximate matching positions in the text.

Note: The Boyer-Moore algorithm is designed to efficiently search for approximate matches in a text by utilizing the bad character and good suffix rules to determine optimal shift distances. It is particularly effective in scenarios where the text is much larger than the pattern.

3. Hamming distance Algorithm:

Hamming distance measures the difference between two strings of equal length by counting the number of positions at which the corresponding symbols are different. Here's a basic methodology for calculating Hamming distance:

Understand the Input:

Make sure you have two strings of equal length that you want to compare.

Initialize Counter:

Set a variable (e.g., distance) to zero. This will be used to count the differing positions.

Iterate through the Strings:

Use a loop to go through each position in the strings.

Compare Symbols:

Compare the symbols at the current position in both strings.

Update Counter:

If the symbols are different, increment the distance counter.

Repeat:

Continue the loop until you have compared all positions in the strings.

Result: Hamming distance measures the difference between two strings of equal length by counting the number of positions at which the corresponding symbols are different. Here's a basic methodology for calculating Hamming distance:

The final value of the distance counter is the Hamming distance between the two strings.

4. PHYLOGENETIC TREES

2.2.1. UPGMA stands for Un-weighted Pair-Group Method with Arithmetic mean. Un-weighted refers

to all pairwise distances contributing equally, pair-group refers to groups being combined in pairs, and arithmetic mean refers to pairwise distances between groups being mean distances between all members of the two groups considered [7]. Consider four DNA sequences namely: S1, S2, S3 and S4. First, find the pairwise distances between all the sequences. Then, find the smallest value in the distance matrix and its corresponding sequences of the shorter distance. For instance let the two sequences with the shortest distance between them be S1 and S2. Now, cluster S1 and S2 and name the cluster as C1, updating the distance matrix by eliminating S1 and S2, but including C1. The C1 value corresponding to the remaining sequences in the distance matrix is calculated with the values of S1 and S2, i.e., arithmetic mean of S1 and S2 distances with corresponding to the other sequences. Now, moving forward by considering the updated distance matrix, find the smallest distance again and its corresponding sequences or clusters (namely sets of sequences). Say this next smallest distance corresponds to that between clusters C3and C4. Then, in the next step, C3 and C4 would be merged into a new cluster C5, and all the distances to C5 would be updated in the distance matrix by the corresponding average distances to the sequences in C5. Repeat the same and find new clusters and sequences, merging and updating the distance matrix, until we are left with one cluster. Algorithm: Let the clusters be C1, C2, C3,....., Cn and si be the size of each cluster Ci , ‘d' be the pairwise distance defined on the clusters. Clustering is done in the following manner: 1. Find the smallest pairwise distance amongst the clusters, d (Ci , Cj). 2. A new cluster Ck with size si+ sj is formed by joining Ci and Cj . 3. Compute the new distances from all other clusters to Ck by using the existing weighted distances average. Where l € {1, 2... n} and l ≠ i, j. 4. Repeat 1, 2, and 3 until only one cluster remains. The asymptotic time complexity of UPGMA is O (n2), since there are (n-1) iterations, with O (n) steps per iteration [11].

The Unweighted Pair Group Method with Arithmetic Mean (UPGMA) is a commonly used algorithm for constructing phylogenetic trees. Below is a step-by-step guide on how the UPGMA algorithm works:

1. Input:

Begin with a distance matrix that represents the pairwise genetic distances or similarities between the entities (e.g., species or genes) under study.

2. Initialization:

Treat each entity as an individual cluster. Create a list or table to keep track of the clusters and their distances.

3. Cluster Pair Selection:

Find the two clusters with the smallest pairwise distance in the current set of clusters. These clusters will be joined to form a new, higher-level cluster.

4. Cluster Joining:

Merge the selected clusters into a new cluster. The average distance between the entities in the merged clusters is calculated and used to update the distance matrix. The new cluster is added to the list of clusters.

5. Update Distance Matrix:

Recalculate the distances between the new cluster and the remaining clusters, using the average distance. Update the distance matrix accordingly.

6. Repeat:

Repeat steps 3-5 until only one cluster remains. The resulting hierarchical structure forms a dendrogram.

7. Tree Rooting (Optional):

Optionally root the tree by designating one cluster as the root. Rooting provides a direction to the tree, indicating evolutionary relationships.

8. Branch Length Adjustment:

Adjust the lengths of the branches in the tree to reflect the genetic distances. The length of each branch is proportional to the average distance between the entities it connects.

9. Visualization:

Visualize the final phylogenetic tree. Graphical representations, such as dendrograms, are often used

to depict the tree structure, aiding in the interpretation of evolutionary relationships.

5. WEIGHTED GRAPHS:

1.Input:

a.Ensure you have a set of sequences (e.g., DNA sequences) representing different species.

b.Each sequence should be of equal length.

2.Calculate Hamming Distance:

c.Implement a function (e.g., Hamming_distance) that calculates the Hamming distance between two sequences.

3.Create Weighted Adjacency Matrix:

d.Initialize a square matrix to represent the weighted adjacency matrix.

e.Use nested loops to calculate and store the Hamming distance between each pair of sequences in the matrix.

4.Define Tree Structure:

f.Decide on a tree structure representation. It can be a list of edges where each edge is a pair of a parent node and a list of child nodes.

5.Algorithm for Creating the Tree:

g.Implement an algorithm to construct the weighted phylogenetic tree using the adjacency matrix and the defined tree structure.

h.You can use methods such as hierarchical clustering or neighbor-joining to iteratively merge nodes based on their Hamming distances.

6.Print the Weighted Phylogenetic Tree:

i.Create a function (e.g., print_weighted_phylogenetic_tree) to display the tree structure along with the corresponding weights

IV. RESULTS

The provided code is a Python script for a DNA sequencing and matching application with a graphical user interface (GUI) using the Tkinter library. It includes functionality for Boyer-Moore approximate matching, creating a phylogenetic tree, Smith-Waterman alignment, detecting mutations, and generating a weighted graph. Below is a brief summary of each function:

1. Boyer-Moore Matching (`bm_approx_matching`): Implements the Boyer-Moore approximate string matching algorithm and prints the positions of approximate matches between two DNA sequences.

2. Create Phylogenetic Tree (`create_phylogenetic_tree`): Constructs a phylogenetic tree based on DNA sequences provided in multiple FASTA files. It uses a hierarchical clustering approach to create the tree and displays the result.

3. Smith-Waterman Alignment (`Smith_Waterman_alignment`): Performs Smith-Waterman alignment between two DNA sequences and prints the aligned sequences along with the alignment score.

4. Detect Mutations (`detect_mutations`): Calculates the Hamming distance between two DNA sequences, determines the percentage of mutation, and identifies specific mutations.

5. Generate Weighted Graph (`draw_weighted_graph`): Constructs a weighted graph based on DNA sequences provided in multiple FASTA files. It uses Hamming distance as weights for the edges and displays the graph.

The GUI provides buttons for each function, and the output is redirected to a scrolled text widget within the GUI.

Please note that there are two identical `execute_and_redirect` functions. You should remove one of them.

Additionally, the code uses external dependencies such as `networkx` and `matplotlib` for graph-related operations. Ensure that these libraries are installed before running the script.

To use this code, you need to have Python installed, along with the required libraries. You can run the script, and the GUI will appear, allowing you to interact with the different DNA analysis functions.

After the text edit has been completed, the paper is ready for the template. Duplicate the template file by using the Save As command, and use the naming convention prescribed by your conference for the name of your paper. In this newly created file, highlight all of the contents and import your prepared text file. You are now ready to style your paper; use the scroll down window on the left of the MS Word Formatting toolbar.

V. CONCLUTION AND FUTURE SCOPE

The code integrates various bioinformatics tools to analyze genetic sequences. It covers approximate matching, phylogenetic tree construction, sequence alignment, mutation detection, and the creation of a weighted phylogenetic matrix.

This versatile set of functionalities allows researchers to explore genetic relationships, identify mutations, and analyse the evolutionary distances between species. The code is well-structured and modular, making it adaptable for different datasets and analyses in the field of bioinformatics.

The bioinformatics project's future outlook is promising. Key initiatives include developing a user-friendly interface for accessibility, integrating with external databases for real-time data retrieval, and prioritizing scalability for large-scale datasets. We aim to explore cloud computing for enhanced resource utilization, incorporate machine learning for advanced predictions, and create interactive visualization tools for result interpretation. Transforming the project into an educational platform with comprehensive documentation will benefit both students and researchers. Open-sourcing for community collaboration, ensuring cross-platform compatibility, and implementing continuous testing practices are priorities. Regular updates, collaboration with biologists, and a strong focus on security, privacy, and documentation round out our approach, fostering an innovative community aligned with evolving bioinformatics needs.

REFERENCES

[1] Smith-Waterman Algorithm - Local Alignment of Sequences (Theory) : Bioinformatics Virtual Lab II : Biotechnology and Biomedical Engineering : Amrita Vishwa Vidyapeetham Virtual Lab

[2] Boyer Moore Algorithm for Pattern Searching - GeeksforGeeks

[3] https://en.wikipedia.org/wiki/Hamming_distance#:~:text=9%20Further%20reading-,Definition,the%20corresponding%20symbols%20are%20different.

[4] Hua GJ, Hung CL, Lin CY, Wu FC, Chan YW, Tang CY. MGUPGMA: A Fast UPGMA Algorithm With Multiple Graphics Processing Units Using NCCL. Evol Bioinform Online. 2017 Oct 4;13:1176934317734220. doi: 10.1177/1176934317734220. PMID: 29051701; PMCID: PMC5637958.

[5] Mohammadi-Kambs M, Hölz K, Somoza MM, Ott A. Hamming Distance as a Concept in DNA Molecular Recognition. ACS Omega.

[6] https://www.geeksforgeeks.org/applications-advantages-and-disadvantages-of-weighted-graph/