Aho-Corasick algorithm in motif pattern matching

Mariam Waleed†
202100089
s-mariam.waleed@zewailcity.edu.eg

Alaa Mohamed
202101529
s-alaa.elmansy@zewailcity.edu.eg

ABSTRACT

In several fields, including network security, biology, and text processing. One popular and very effective method for concurrently identifying many patterns in a given text simultaneously is the Aho-Corasick algorithm. This algorithm achieves linear time complexity for pattern matching by combining the ideas of trie data structures and finite automata. It builds Aho-Corasick trie from given motifs that would be matched against a given sequence.,by adding each motif pattern in a separate branch in the trie and then it traverses and searches throughout the trie by building failure links it creates failure links and output links. Each node in the trie represents a prefix of one or more patterns. The failure links help in the improvement of pattern matching as when a mismatch arises during pattern matching and the traversal by normal edges of the trie, these failure links help in shifting to the longest appropriate suffix of the current state. The algorithm`s speed is improved by avoiding unnecessary comparisons and moving through the trie quickly by following the failure connections. The output links enable the algorithm to locate and effectively store pattern occurrences or positions in the text. The Aho-Corasick algorithm traverses trie following the transitions depending on the characters in the input test, during the pattern matching phase. Finally, the output of this algorithm is the patterns that were found in the sequence, its occurrences and positions. So, Aho-carsick is one of the most efficient algorithms in pattern matching as it processes multiple patterns with linear complexity.

KEYWORDS

Aho-Corasick algorithm, Motif pattern matching, Failure links, Output links, Trie data structure.

Introduction

Motifs

Motifs are short (6-8 bases long), and encompass patterns seen in RNA sequences, such as splicing signals, and DNA patterns found in enhancer regions or promoter motifs. Genetic activity is controlled in response to environmental differences. Transcription factors, or regulatory proteins, are attracted to the right target gene via motif-mediated processes. Other RNAs that employ a mix of DNA sequence and structure, nucleosomes that identify motifs based on their GC content, and microRNAs that bind to patterns provided by complementarity are also capable of recognizing motifs. Once bound, they have the ability to either stimulate or inhibit the related genes`s expression. examination of the areas upstream of genes known to have related activities identifies a large number of regulatory motifs. Regulatory motif analysis and discovery have advanced significantly with our present computational capabilities and continue to be at the forefront of genomic research[1].

Aho-Corasick algorithm

Alfred V. Aho and Margaret J. Corasick created the Aho-Corasick algorithm, a string-searching method, in the field of computer science in 1975[2]. This type of algorithm looks for strings inside an input text that belong to a finite set of patterns. It is one of the most efficient algorithms in pattern matching as it can take multiple patterns and search for them in a given sequence in a linear time complexity O(n+m+z) where n is the total length of all patterns, m is the length of the text in which the patterns will be searched, z is the number of occurrences of each pattern. It starts by building a trie from the patterns to be searched for with extra linkages connecting the different internal nodes. These additional links facilitate quick transitions to other branches of the trie that share a common suffix and then traverse this trie and build the failure links from each node. As a result, the automation can move between string matches without the need for backtracking. Then, it builds the failure function to construct failure links between nodes and the last step is building a search and output function that will return the found patterns with their occurrences and positions[3].

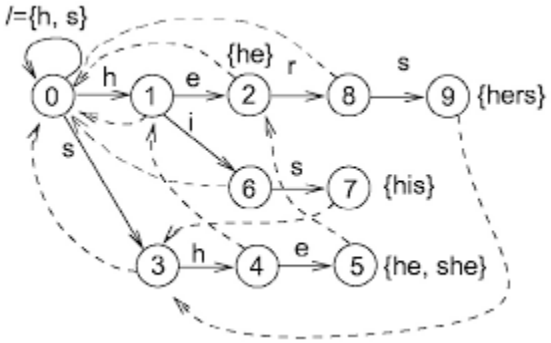


Figure 1: Aho-Corasick algorithm illustration dashed arrows represent failure links[3].

Problem definition

In several fields, including network security, biology, and text processing, motif finding is a prevalent problem. The objective is to efficiently detect all instances of a collection of target patterns(motifs) within a huge text or dataset. This problem may be effectively solved using the Aho-Corasick algorithm, which makes use of the trie data structure. The input of this algorithm is A collection of desired patterns or motifs expressed as strings {S1, S2, S3,....}, A string (sequence) of DNA or RNA. The output will be the whole set of times each motif appears in the sequence and the start position of it. The goal is to create a productive technique to locate every instance of the target theme in a given sequence by utilizing the trie data structure and the Aho-Corasick algorithm, Reduce the amount of time and space needed to locate motifs, Effectively handle a large number of motifs and large-scale datasets. The limitations of this algorithm are: The text`s length, n is flexible and has the potential to be substantially longer than the lengths of each motif separately, The number of motifs,m, to vary significantly, The motifs can have different lengths and include any invalid characters other than normal.

Related Work

One of the main areas of interest for bioinformatics researchers has always been solving motif-finding problems. In recent years a multitude of techniques, algorithms, and instruments have emerged. A few popular techniques have been taken into account.The only previous knowledge needed by the greedy algorithm CONSENSUS[4][5] is the size of the intended motif. In general, it functions by extracting every potential subsequence discovered in the sequences that has the appropriate length. The positional weight matrix (PWM) for each set is then determined by repeatedly combining these subsequences and retaining the better ones at each stage.The Gibbs sampling[6] method makes educated estimations about the location of motifs in each input sequence by first estimating their locations. It is not a greedy algorithm because it selects motif positions in a semi-random manner, although the early predictions have an impact: A family of algorithms known as Expectation Maximization (EM)[7] calculates the values of a collection of unknowns based on a set of parameters (referred to as the "Expectation step") using the estimated values, which known as ("Maximization step") are then refined over a number of repetitions. SP-STAR the combinational method[8] was put out by Sze and Pevzner in 2000. It first selects an appropriate scoring formula to determine how good a theme is. Next, it selects the finest instance of each K-mer that appears in the sample for each sequence, compiling these occurrences to create an initial motif. After that, each original motif is improved using a local improvement heuristic. Numerous novel techniques and algorithms have been developed for this problem.

other methods used to solve this problem are[9]: **Enumerative Algorithms**: one of the algorithms that are used alongside p**robabilistic algorithm** which is built based on statistical models to enhance accuracy. this method searches for each combination to find the desired motif. **Heuristic Algorithms**: Typically relies on intuition to search for the motif which makes it fast but sometimes can not work probably with local optima. **Meta-heuristic Algorithms**: for example, particle swarm optimization (PSO) and artificial bee colony (ABC) algorithms, rely on inspiration from nature leading to efficient motif finding from large solution spaces.**Tree-based Methods**: Weeder and FMotif algorithms built with tree structures to enhance motif discovery based on k-mers and motifs evaluation in DNA sequences.**Graph Theoretic-based Methods**: the sequence is represented on the graph cliques and the edges correspond to the sequence's similarity such as WINNOWER and Pruner algorithms.**Clustering-based Methods**: put a similar group on the cluster to find the motif such as CisFinder clustering. **Entropy-based Algorithms[9]**: such as DREME and EPP use entropy-based measures to analyze the information of each position on the sequence. **Genetic Algorithms (GA)[9]**: GA is an algorithm inspired by evolution processes. It involves selection, crossover, and mutation operations to find better motif candidate. **Consensus Algorithms[9]**: seek to identify motifs based on consensus sequences derived from input sequences.

These different algorithms and data structures help researchers to discover highly valuable biological insights and regulatory elements[9].

Intuition

The Aho-Corasick algorithm is used in the suggested motif-searching method, which has a number of benefits over other cutting-edge algorithms. It is good for motif discovery or finding when it is necessary to identify several motifs inside a given sequence or multiple sequences since it effectively looks for multiple patterns at once. This approach seeks to offer a scalable and quick fix for motif identification by utilizing the algorithm`s capacity to build a trie data structure and effectively match patterns. It has a lot of benefits such as: creating a single automation to represent all patterns instead of going over each pattern one at a time and requiring redundant checks, This gets much quicker, particularly when there are a lot of patterns. Average linear Time complexity: the complexity of this algorithm is linear and proportional to the length of the text and patterns. Effective handling of overlaps: when patterns appear twice in a text, the Aho Corasick method can handle these situations effectively. When a mismatch arises, the failure function enables the automation to "jump" to pertinent states, preventing backtracking and pointless computation. Other algorithms like Kunth-Morris-Pratt(KMP), may be more appropriate for specific pattern-matching tasks, however, they are limited to matching single patterns or have issues with overlapping patterns.

Proposed method:

in this project the tries data structure was built to store the motif sequences along with the implementation of the Aho-Coarsick automaton algorithm to search for the occurrence of certain motifs here is the breakdown of the code:

trie implementation:

the trie first started with nod struct that initializes the tries nodes which consist of (character, isleaf, failure pointer, a vector for child nodes, length to) in addition to the main functions (indexing: initialize the index of each nucleotide to use it when inserting a sequence, add: to add the new sequence, seqExsist: to check if the sequence exists or not, prefixExsist: to check if the prefix is already in) .

add the function takes the sequences to iterate over each nucleotide and insert it when it does not exist then indicates the last character as a leaf and stores its length.

indexing: it uses the switch case to initialize the index of each nucleotide and return -1 if the sequence has another character rather than A,T,C,G.

seqExsit: the function takes a constant sequence then starts from the main root that points to the root of the trie, the function then iterates through the sequence and calls the indexing function to see the index of each character, and if it is valid or not, if the index = -1 or the curr->child[index] ==nullptr this indicated the child node of this index is null which means it does not exist so the function return false, otherwise it iterates through the whole sequence till it find the leaf and if the sequence found it return true.

prefixExsit: check if the prefix exists by taking the constant pointer to a character array similar to the previous function it starts from the main root and then iterates while the (*str) is not null (\0) then works typically as the previous function but with no need to reach or check the leaf any prefix is found will return true

for the implementation of the algorithm itself, FAILFUNC was made to construct the failure links and search to return the occurrence of each sequence;

For the failure function, it constructed the failure links that were mentioned above to fail on the same charger that has the longest suffix. it helps to make the better search by navigating each level of the trie using a queue data structure, for each child if it carries the desired nucleotide the failure link points to the parent and for each node when in the second level it fails on the main node.

search function: take the query sequence and iterate through each character on the trie if it is equal to the charger it moves to this node if not it moves to the other child following the failure link. When traversing if it found the leaf this means end of the sequence.

algorithm complexity analysis

for the trie construction time complexity for insertion is O(L) while L is the pattern length. for the algorithm the time complexity is O(N + M + Z) while N is the length of the pattern sequence, M is the total length of all sequences in the trie, Z is the total number of occurrences of all patterns found in the pattern sequence.

Pseudo code:

```
TrieConstruction(Patterns)
    Trie ← a graph consisting of a single node root
    for each string Pattern in Patterns
        currentNode ← root
        for i ← 1 to |Pattern|
            currentSymbol ← i-th symbol of Pattern
            if there is an outgoing edge from currentNode with label currentSymbol
                currentNode ← ending node of this edge
            else
                add a new node newNode to Trie
                add a new edge from currentNode to newNode with label currentSymbo
                currentNode ← newNode
    return Trie
```

```
PrefixTrieMatching(Text, Trie)
    symbol ← first letter of Text
    v ← root of Trie
    while forever
        if v is a leaf in Trie
            return the pattern spelled by the path from the root to v
        else if there is an edge (v, w) in Trie labeled by symbol
            symbol ← next letter of Text
            v ← w
        else
            output "no matches found"
            return
```

```
TrieMatching(Text, Trie)
    while Text is nonempty
        PrefixTrieMatching(Text, Trie)
        remove first symbol from Text
```

In addition to the failure function which makes the failure link of each node.and trie construction **t**he trie consists of the main node which is empty then you use the add function to insert a sequence it iterates over the sequence and checks the prefix exists or not if not it create a new node and store the character on this node if the character already on it moves to the next nucleotide.

Data

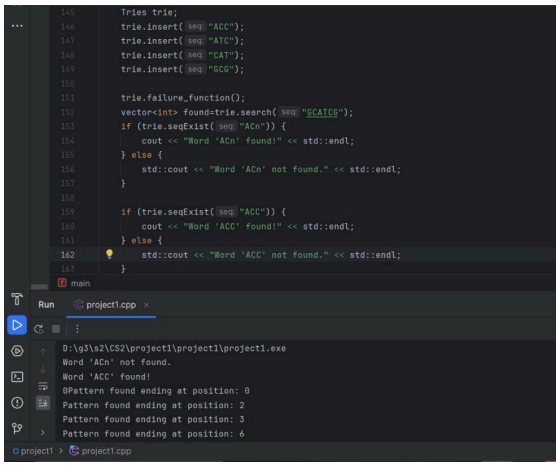
Biological sequences:

known short sequences were inserted into the tries and one other motif was used to search in it for these motif patterns to check which is present and which is not in addition to knowing their positions and occurrences. No data files were used.

Questions

This algorithm try to answer some questions including: Which motifs or repeating patterns can be found in a specific sequence or group of sequences? In which occurrences they are present? In which starting position does the motif present?

Results

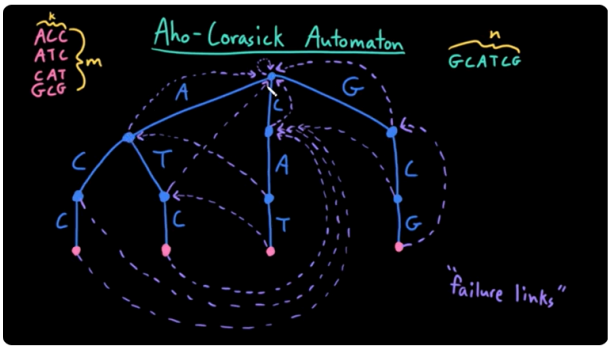


As appears in the attached photo first we inserted the sequences with the insert function (its name was just changed to add but the function is the same). then the function seqExsist which indicates if a sequence exists on the tree or not it appears that both functions work correctly because when I searched for ACn which was not inserted the output was not found and when I used ACC the output says this function was found. for the search function and failure function it works and it was tested using "GCATCG" and the output was the vector indicating the occurrence index of each part of this sequence that was inserted before.

One of the main challenges that we faced was how to initialize the node and its index till we reached the solution of using the switch case. before that I tried to write this part based on the ASC|| code however it does not work here because we deal only with 4 characters, not the whole alphabet so instead the A was set to be 0, T:1, C:2, G: 3, and if there is any other character it will -1. Another problem was how to indicate the longest prefix then the length variable was added to the node to indicate the length of the leaf.

Validation

As mentioned above the code was validated using already known examples to check whether each function worked or not and the final result was true it indicated which sequence exists or not and the index or occurrence of each sequence



Advantages and Disadvantages

The Aho-Corasick algorithm has a lot of advantages:

-Its linear time complexity O(n+m+z) as the required time to process and give output is very small compared to other algorithms. There is just one analysis of each character in the text regardless of the input symbol. States are reached through the determination steps[10].

and disadvantages:

It stores the transition roles from the deterministic finite state by using additional storage[11]

Conclusion and Discussion

To sum up, the Aho-Corasick algorithm-based motif searching approach that has been suggested has a number of advantages, including flexibility, scalability, searching for multiple patterns a the same time, and linear time complexity. This algorithm uses the ability of quick pattern-matching to build a trie data structure, which facilitates the detection of several motifs at once and sheds light on the recurrent

patterns in the dataset. The algorithm is sensitive to input parameter changes and assumes fixed-length motifs, among other drawbacks.

Suggestions and Future Work

apply this algorithm in a whole genome for example and try it on different types of data and construct the trie using different motifs. modify the algorithm to handle and accept various lengths of motifs using dynamic programming approaches.

REFERENCES

[1] Introduction to regulatory motifs and gene regulation," *Biology LibreTexts*, Oct. 05, 2020. https://bio.libretexts.org/Bookshelves/Computational_Biology/Book%3A_Computational_Biology_-_Genomes_Networks_and_Evolution_(Kellis_et_al.)/17%3A_Regulatory_Motifs_Gibbs_Sampling_and_EM/17.02%3A_Introduction_to_regulatory_motifs_and_gene_regulation

[2] A. V. Aho and M. J. Corasick, "Efficient string matching," *Communications of the ACM*, vol. 18, no. 6, pp. 333–340, Jun. 1975, doi: https://doi.org/10.1145/360825.360855.

[3] Author links open overlay panelJakub Botwicz *, & AbstractOne of the main objectives for pattern matching methods study is their serious role in the real-world applications. (2017, May 11). *B07: Aho-Corasick algorithm implementation in hardware for network intrusion detection*. IFAC Proceedings Volumes. https://www.sciencedirect.com/science/article/pii/S1474667017305979

[4] G. D. Stormo, "DNA binding sites: representation and discovery," *Bioinformatics*, vol. 16, no. 1, pp. 16–23, Jan. 2000, doi: https://doi.org/10.1093/bioinformatics/16.1.16.

[5] G.Z. Hertz and G.D. Stormo. Identification of consensus patterns in unaligned dna and protein sequences: a largedeviation statistical basis for penalizing gaps. The Third International Conference on Bioinformatics and Genome Research, 201—216, 1995

[6] Lawrence C E, Altschul S F, Boguski M S, Liu J S, Neuwald A F and Wootton J C. Detecting subtle sequence signals: a Gibbs sampling strategy for multiple alignments. Science, 262, 208-214, 1993.

[7] Cardon, L.R. and Stormo, G.D. (1992) An ExpectationMaximization (EM) Algorithm for Identifying ProteinBinding Sites with Variable Lengths from Unaligned DNA Fragments. J. Mol. Biol. 223:159-170.

[8] Pevzner P A and Sze S H. Combinatorial approaches to finding subtle signals in DNA sequences. Proceedings of the Eighth International Conference on Intelligent Systems for Molecular Biology (ISMB 2000), 269-278, 2000.

[9]F. A. Hashim, M. S. Mabrouk, and W. Al-Atabany, "Review of Different Sequence Motif Finding Algorithms," *Avicenna Journal of Medical Biotechnology*, vol. 11, no. 2, pp. 130–148, 2019, Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6490410/

[10]S. Hasib, M. Motwani, and A. Saxena, "Importance of Aho-Corasick String Matching Algorithm in Real World Applications." Accessed: May 26, 2024. [Online]. Available: https://ijcsit.com/docs/Volume%204/vol4Issue3/ijcsit2013040318.pdf.

[11]H. Kim, K.-I. Choi, and S.-I. Choi, "A Memory-Efficient Deterministic Finite Automaton-Based Bit-Split String Matching Scheme Using Pattern Uniqueness in Deep Packet Inspection," *PLOS ONE*, vol. 10, no. 5, p. e0126517, May 2015, doi: https://doi.org/10.1371/journal.pone.0126517.