An Efficient Implementation of Cosine Distance on Minimal Absent Word Sets using Suffix Automata

Mohammad Tamimul Ehsan[1][0009−0009−5786−5367], Sk. Sabit Bin Mosaddek[1][0009−0009−5500−9743], and M Saifur Rahman[1][0000−0002−9887−4456]

Bangladesh University of Engineering and Technology

Abstract. The Cosine Distance on Minimal Absent Word Sets, CD-MAWS in short, was recently introduced for alignment-free phylogeny estimation. Here we introduce a refined CD-MAWS method, significantly reducing computational complexity from $\mathcal{O}(\max(km^3n, km^2n\log n))$ to $\mathcal{O}(\max(m^2n, kmn))$ while maintaining tree quality. Here, m is the number of species, n is the size of the whole genome of a species, and k is the maximum length of a minimal absent word (MAW). This advancement is achieved through a revised cosine distance calculation method, binary encoding of MAWs, and the adoption of suffix automata for MAW generation, addressing the main computational bottleneck and setting a better runtime for alignment-free phylogenetic analysis.

Keywords: Phylogeny · CD-MAWS · Suffix Automata

1 Introduction

The study of phylogeny is crucial for understanding evolutionary relationships among species. The Cosine Distance on Minimal Absent Word Sets (CD-MAWS) [1] offers an alignment-free approach for phylogeny reconstruction from whole genome analysis. However, CD-MAWS is hampered by its computational inefficiency. Our improvements aim to streamline this process, offering a more practical solution for genomic analysis. Building on the foundational work of the CD-MAWS method, our research extends the application of Minimal Absent Words (MAWs) in phylogenetic analysis by addressing computational bottlenecks and enhancing methodological efficiency.

We introduce suffix automata for MAW generation in lexicographical order and leverage this ordered generation for determining the intersection set of MAWs between species pairs in an efficient way. Furthermore, we propose to encode each nucleotide in two bits so as to process any MAW of up to a certain length in a very efficient way in various operations. We thus reduce computational complexity while maintaining accuracy. Our work opens up new avenues for large-scale genomic studies, offering a faster and more scalable tool for unraveling the complexities of evolutionary histories.

2 Background and Related Works

2.1 Background

Phylogenetic analysis is essential for understanding evolutionary relationships among species. Traditional methods, such as multiple sequence alignment (MSA), have been widely used but come with significant computational challenges and limitations, particularly when dealing with large datasets or highly divergent sequences. To address these issues, alignment-free methods have gained attention in recent years [2,3,4,5]. These methods bypass the need for sequence alignment, thus reducing computational complexity.

Minimal Absent Words (MAWs) are a novel concept in alignment-free phylogenetic analysis. MAWs are sequences that are conspicuously absent from a given genome but present in related genomes. These absent sequences provide unique phylogenetic signals that help in constructing evolutionary relationships. The use of MAWs, combined with mathematical measures such as cosine distance, offers a promising approach for phylogeny estimation, balancing computational efficiency and accuracy.

2.2 Related Works

Several alignment-free methods have been proposed and benchmarked in recent studies. Zielezinski et al. (2019) [10] conducted a comprehensive benchmarking of alignment-free sequence comparison methods, highlighting the strengths and weaknesses of various approaches. Their study provides a foundation for evaluating new methods such as the one proposed in this paper.

One notable alignment-free method is the use of k-mers [11], which involves counting the frequency of fixed-length substrings within sequences. This method has been widely adopted due to its simplicity and effectiveness. However, it may not fully capture the phylogenetic signals present in the sequences, especially for more complex datasets. Another significant contribution to the field is the work of Liu et al. (2009) [9], who developed a rapid and accurate method for large-scale coestimation of sequence alignments and phylogenetic trees. Although not alignment-free, their approach sets a high standard for accuracy and computational efficiency.

In addition to k-mers and large-scale coestimation methods, techniques like spectral methods [12] and information-theoretic measures [13] have been explored for phylogenetic analysis. These methods offer alternative ways to capture evolutionary signals in sequence data without relying on traditional alignment procedures.

Minimal absent words (MAWs) are increasingly important in alignment-free sequence analysis [14,15,16,17,18]. They efficiently distill essential genomic sequence information, enabling comparisons without traditional alignment methods. MAWs support distance-based techniques such as Length Weighted Index [17] and Jaccard Distance [15], which aid in reconstructing phylogenies. Additionally, they facilitate the creation of composition vectors used in cosine similarity

computation [19]. The CD-MAWS approach [1] specifically demonstrates their effectiveness through experiments on biological and simulated datasets, showcasing their potential in reconstructing species phylogeny. However, as the number of MAWs increases, memory constraints can prevent the method from generating vectors needed for cosine distance calculations, and the running time can also become prohibitively expensive. In this study, we integrated suffix automata and encoding techniques to address these limitations, achieving significantly improved runtime and memory usage.

3 Preliminary Concepts

3.1 Minimal Absent Words

Minimal Absent Words (MAWs) are words that do not appear in a string, but all their proper substrings do. For example, *acg*, *gac*, and *gct* are examples of MAWs for the string $x = actgcga$. However, *acta* is an absent word but not a MAW of x. Notably, proper substrings are derived by deleting one or more characters from the beginning or end of the word. Proper prefixes and suffixes are similarly defined. The number of MAWs of a string of size n is $O(n)$ [20].

3.2 Cosine Distance for MAW Vectors

Cosine distance measures the cosine of the angle between two vectors, representing the presence or absence of MAWs in genomic sequences. It ranges from 0 (identical orientation, implying high similarity) to 1 (orthogonal orientation, implying low similarity). This metric is advantageous for its normalization, accounting for differences in sequence lengths and compositions. The distance can be shown as:

$$\textit{CD-MAWS}(x, y) = 1 - \frac{V^x.V^y}{|V^x||V^y|}$$

Here V^x (V^y) is a vector representation of the sequence x (y), such that the value assigned to vector V^x (V^y) along dimension w identifies whether w is a MAW of x (y) or not. These vectors are also known as composition vectors.

3.3 Length of MAWs

Aurell et al. [6] derived lower and upper limit on the length within which the bulk of MAWs of a string of length N lie, assuming a random model:

$$l_{min} = \frac{\ln N - \ln \ln N}{\ln 4}$$

$$l_{max} = \frac{2 \ln N + \ln 9}{\ln 4}$$

The human genome contains roughly 3.2 billion base pairs (3.2×10^9 bp), providing the blueprint for human biology [21]. *Tmesipteris truncata* (aka *Tmesipteris oblanceolata*) is a fern ally endemic to eastern Australia holds the record for the largest sequenced genome at approximately 160 billion base pairs (160×10^9 bp) [22]. So, by the formula given above the bulk of the MAW for human will be within range [13,34] and for the fern it will be within range [16,39]. This information is crucial for our analysis as it shows the length of majority of the MAW is very much managable and small in size.

3.4 Suffix Automata

Suffix Automata A suffix automaton is an efficient data structure used to represent the substring index of a given string. It enables the storage, processing, and retrieval of compressed information about all the string's substrings. The suffix automaton for a string S is the smallest directed acyclic graph with a designated initial vertex and a set of "final" vertices, such that every path from the initial vertex to any final vertex corresponds to a suffix of the string. [23] [24].

Here is an example of a tree of suffix links in the suffix automaton build for the string "abcbc". The nodes are labeled with the longest substring from the corresponding equivalence class.[1]

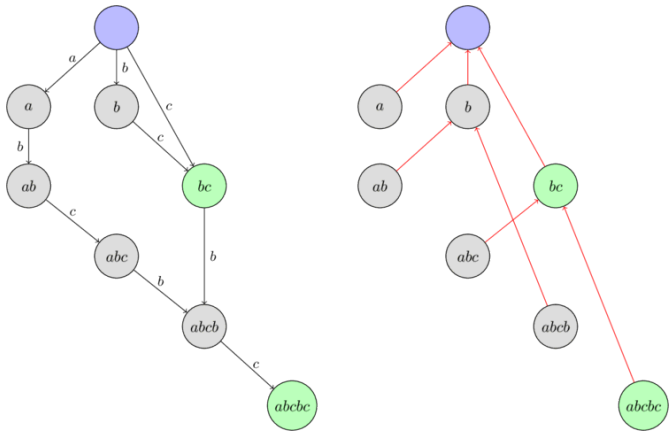

Fig. 1. A tree of suffix links in the suffix automaton build for the string "ABCBC". The nodes are labeled with the longest substring from the corresponding equivalence class

[1] `https://cp-algorithms.com/string/suffix-automaton.html`

End Position Equivalency A substring can occur in multiple positions of a string. For each substring, we can find the set of positions where the corresponding substring has ended. This set does not uniquely define a substring. In fact, multiple substrings can have the same set of end positions. If two substrings have the same set of end positions, they are considered in an equivalent class. For example, in string "ABCBC", the substring "ABCB" and "BCB" belong to the same equivalent class as both of them have the same set of end positions. Each node in Suffix automaton represents an equivalent class. It can easily be derived from the definition of suffix automaton. As the suffix automaton accepts all the suffixes, from each node there will be some path to the terminal node representing a set of suffixes. Thus, if two strings reach a node, they will have the same path to the end which means wherever those two strings have ended, they end at the exact same positions. Otherwise, there would have been different sets of suffixes starting from their endpoint.

Suffix Link For two different substrings, if their set of end positions does not exactly match, then either one of them has to be a subset of the other or there will be no common end position among them. This is because, if two substring has at least one common end position, it clearly signifies that one of the substring is a suffix of the other. Thus, the shorter substring will have a larger set of end positions. To keep track of the largest suffix that does not belong in the equivalent class, we introduce a suffix link between them. Formally, there will be a suffix link from an equivalent class to another if and only if the former class has a proper subset of end positions of the later one and the later class has the smallest set of end positions among all possible other sets. For example, in string "ABCB", the class which represents "ABC" and class which represent "ABCBC" has a suffix link toward the class with "BC".

4 Proposed Methodology

We break down our work into two primary segments. First part being the distance calculation ie CD-MAWs. This segment assumes that the MAW set is already calculated and works on that. The second part concerns the calculation of the MAW set of a given string using suffix automata. The description of the proposed methodology and comparison are done seperately for the two segments.

4.1 Generating MAW using Suffix Automata

Before we proceed, we would like to remind readers that linear-time algorithms for generating Minimal Acyclic Words (MAWs) already exist, as described by Barton et al [20]. In this work, we propose a methodology for generating MAWs in lexicographically sorted order. Although our method is slower than the linear-time algorithm when generating MAWs alone, it achieves significantly better runtime when the output is required to be sorted.

We will use the concept of suffix links and suffix tree to compute MAWs of a string. For instance, we are trying to find if a string $t = xabcdy$ is a MAW of another string s. We at first move to the node that contains the string $xabcd$. If we do not find this substring of t in s, then t is not a MAW of s. If the substring is present, then we attempt to proceed using edge y. If we find a node following edge y then t itself is present in s and thus is not a MAW. If we do not find it, then we move back using the suffix link. The property of suffix link dictates that we are in a node which is the longest possible suffix of previous node which is present in the string. If the string size in the node following the suffix link is less than $|t| - 2$, then t is not a MAW because it is smaller than the longest proper suffix of t. Otherwise, we again try to proceed using y from that node. If we can, it means there is a substring $abcdy$ present in s. Thus, by the definition, t is a MAW of s.

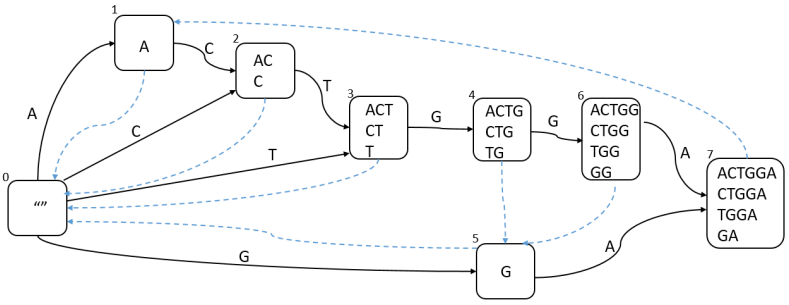

Fig. 2. Suffix automaton of 'ACTGGA'

For example, consider the word $x = ACTGGA$. We check if two strings $s = GAC$ and $t = CTGA$ are MAWs or not. For GAC, we traverse the trie up to GA to reach node 7. The longest prefix of s is thus a substring of the original word. But from there we cannot move forward with edge C. Therefore, s is not included in x. We then move backward using the suffix link to reach node 1. The length of the word in that node is one, thus having the potential of constructing the longest proper suffix of s. Node 1 has an outgoing edge C, so AC is included in x. Thus, both the longest proper prefix and longest proper suffix of s are included in x but not s itself, making s a MAW of x.

Similarly, for t, we move to node 4 to find CTG by traversing through node 0, 2, 3 using edges C, T, G respectively. Next, we try to move forward using edge A but it is absent. So, $CTGA$ is not present in x but CTG is, which is the longest proper prefix of t. Now we move back using suffix link to node 5. However, this node only holds a length 1 string. Although it has a forward edge

of A to construct a suffix of t, it is not going to be the longest proper suffix of t. Therefore, t is not a MAW of x.

4.2 Encoding MAWs

To compare two MAWs, which are character arrays or strings, we need to iterate over their length, resulting in a complexity of $O(k)$. However, as there are 4 characters in the DNA set: A, C, G, and T, we can encode them in binary as A (00), C (01), G (10), and T (11). Thus, each MAW can be encoded in a binary string. In many programming languages, including C++, a MAW of length 32 can thus fit in a 'long long' integer. On a 64-bit computer, the comparison can be done in constant time, and on a 32-bit computer, it takes only twice the time, which is also constant. For MAWs with larger size, the encoding can be done with two variable or a 128 bit integer.

4.3 Two Pointer Method

In the original CD-MAWS method, the proposed approach was to create a composition vector using MAWs for each species, and then calculate the pairwise cosine distances. However, by examining the structure of the cosine similarity, we observe that the numerator of the ratio implies the number of common MAWs, while the denominator is the product of the number of MAWs in each sequence. Without composing the actual vectors, we can directly find the denominator value as the size of the MAW set for each species. For the numerator, we can use a two-pointer method to find the common MAWs between two species. For this, the MAWs need to be sorted. After sorting, we iterate over them using the two-pointer technique to find the number of common MAWs.

5 Algorithm

The algorithm for constructing a suffix automaton is well-known and, therefore, is not detailed in this paper. The pseudo-code for generating MAWs is provided in Algorithm 1. Additionally, the authors assume that the two-pointer method is straightforward and have thus opted to omit it as well.

6 Runtime Analysis

At first we define the variables required for further analysis

m = the total number of taxa in the dataset
n = the size of DNA (the number of base pair) of a taxa
k = the maximum size of a MAW (the number of base pair)

The overall process is divided into 3 stages, i) building the suffix automata, ii) generating MAWs, iii) calculating cosine distance

Algorithm 1 Generating MAW from Suffix Automaton

```
1: procedure GenerateMAW(node, word, len)
2:     if len ≥ maxK then
3:         return
4:     end if
5:     for c ∈ {A, C, G, T} do
6:         u ← node.ForwardLink(c)
7:         if u is null then
8:             if len + 1 < minK then
9:                 continue
10:            end if
11:            v ← node.BackwardLink
12:            if v is not null and v.len + 1 ≥ len then
13:                Write word + c to output
14:            end if
15:        else
16:            GenerateMAW(u, word + c, len + 1)
17:        end if
18:    end for
19: end procedure
```

The time complexity of building suffix automaton is $\mathcal{O}(n)$. In order to generate m suffix automaton the time complexity will be $\mathcal{O}(nm)$

Then to generate the MAWs, we run a dfs with max depth k. The find a MAW of length l we find if its longest prefix of length $l-1$ is present in the string as substring and check if it can be extended any further. The total number of substring of length l in a string of length n is $n-l+1$. So, the upper bound on the number of substring of size less than k is

$$\sum_{l=1}^{k} n-l+1$$

$$= n + (n-1) + (n-2) + ... + (n-(k-1))$$

$$= n*k - k*(k-1)/2$$

So, the total number of search space to generate MAW is bounded by $\mathcal{O}(kn)$. As we have to generate the MAWs for m taxa, so the total time complexity to generate all the sets is $\mathcal{O}(knm)$.

In the final stage, we find the cosine distance between the taxa. To find the cosine distance between two taxa, we apply the method mentioned in 4.3. As we run a linear search on the set of MAW of the two taxa, the runtime is equal to the size of the MAW set of a taxa. As we have mentioned earlier, the number of MAW of a string is $\mathcal{O}(n)$ and we need to compare the equality of two MAW in $\mathcal{O}(k)$ ie their length. we need to So, the time complexity of finding the cosine distance between two taxa is $\mathcal{O}(kn)$. As we have a total m taxa in the dataset,

so there will $m * (m-1)/2$ pair of taxa. So, the time complexity of this stage is $\mathcal{O}(knm^2)$.

So, in total the time complexity of the whole process is

$$\mathcal{O}(nm) + \mathcal{O}(knm) + \mathcal{O}(knm^2)$$
$$= \mathcal{O}(knm + knm^2)$$

We can also compute the distances using the encoded MAWs, which simplifies the final stage of comparison to a constant-time operation. However, encoding the MAWs results in them becoming unsorted, negating any advantage of using the suffix automaton-based approach. Consequently, we revert to the MAW generation method described by Barton et al [20].

Using this approach, generating MAWs requires $\mathcal{O}(nm)$ time for m taxa, each with n base pairs. After generation, the MAWs are encoded and individually sorted, which adds an additional $\mathcal{O}(knm)$ for encoding and $\mathcal{O}(nm \log n)$ for sorting m taxa, each with n base pairs. Finally, the distance calculation between the taxa is performed in $\mathcal{O}(nm^2)$ time.

So, in total the time complexity of the whole process is

$$\mathcal{O}(nm) + \mathcal{O}(knm) + \mathcal{O}(nm \log n) + \mathcal{O}(nm^2)$$
$$= \mathcal{O}(knm + mn \log n + nm^2)$$

7 Comparative Analysis of MAW Generation

In this section we compare the execution time and memory usage of our proposed method of MAW generation based on Suffix Automata with the MAW generation algorithm proposed by Barton et al. [20] All the comparisons were conducted in a system with Microsoft Windows 10 Pro x64-based PC with WSL Kernel Version: 5.15.133.1-microsoft-standard-WSL2. The Linux Distribution in WSL was Ubuntu 22.04.2 LTS. The processor of the machine is AMD Ryzen 5, 3600 Mhz, 6 Core(s), 12 Logical Processor(s), and RAM was 8 GB. The programming language used is C++ with gnu gcc compiler.

7.1 Execution Time

Table 1 and Figure 3 compare the execution times of the MAW generation algorithms proposed by us and Barton et al. with an extra sorting step for varying input sizes. Our proposed algorithm shows asymptotic improvements, particularly for large datasets.

The original CD-MAWS algorithm did not have any constraint on the order of the MAWs required in the MAW set. The algorithm of Barton et al. [20] also does not produce the MAW set in any particular order. However, given that our proposed algorithm processes a sorted list, the MAW set is to be sorted to facilitate the distance calculation step. As the suffix automata already outputs the MAW set in sorted order, it is asymptotically faster for this particular use case. For an input size of 10^7, our method shows a clear asymptotic advantage.

Table 1. Execution time comparison (in seconds) between the MAW generation algorithms of Barton et al.[20] and our proposed one with Suffix Automata

Input Size (n)	Barton et al.	Sorting	Total	Proposed
10^2	0.0190	0.0068	0.0258	0.3695
10^3	0.0200	0.0090	0.0290	0.3485
10^4	0.0295	0.0218	0.0513	0.3553
10^5	0.1298	0.1133	0.2430	0.4573
10^6	1.2658	1.0448	2.3105	1.9138
10^7	14.5448	11.1833	25.7280	18.0760

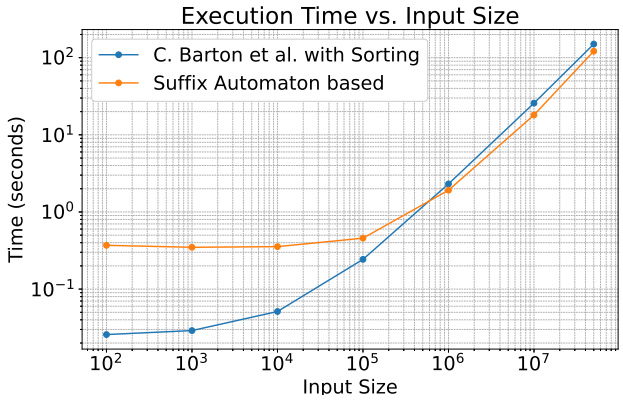

Fig. 3. Execution time comparison between the MAW generation algorithms of Barton et al. [20] and our proposed one with Suffix Automata.

7.2 Memory Usage

Table 2 and Figure 4 summarize the peak memory consumption, showing significant reductions in memory usage with the modified algorithm.

Table 2. Peak memory usage (in MB) comparison between the MAW generation algorithms of Barton et al. [20] and our proposed one with Suffix Automata.

Input Size (n)	Barton et al.	Proposed
10^1	3.77	3.33
10^2	3.82	3.36
10^3	3.84	3.43
10^4	4.46	3.76
10^5	16.13	12.36
10^6	253.26	78.89
10^7	2479.24	620.97

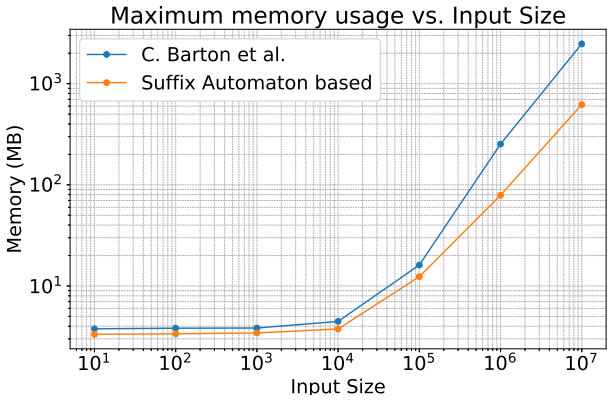

Fig. 4. Peak memory usage (in MB) comparison between the MAW generation algorithms of Barton et al. [20] and our proposed one with Suffix Automata.

8 Comparative Analysis of Cosine Distance Calculation

Previous CD-MAWs implementation introduced a method of composition vector where the length of the vector will be the number of MAW across all DNA. This vector can easily get very large in size. Thus it took more time to create those vectors and also to work with them. As shown in this section, the proposed two-pointer method and encoding techniques significantly reduce the runtime

and memory footprint compared to the original CD-MAWS method. All the experiments in this section were conducted in the same computing system as mentioned in Section 7.

Since the original paper focused solely on finding the cosine distance without considering the MAW generation step, we also chose to skip the MAW generation step in order to make a fair comparison with the original approach

8.1 Data Generation

We generated a simulated dataset with varying numbers of species using the Seq-gen Monte Carlo simulation tool [8]. The data comprised 1000 DNA sequences, each having 10000 nucleotides. We then prepared various sub-samples and ran our experiments 5 times on each of them and averaged the execution time and peak memory usage.

8.2 Execution Time

A comparative runtime analysis of the original CD-MAWS method and our proposed method using suffix automata can be found in Table 3. We show two variants of our method: CD-MAWS using suffix automata (CD-MAWS-SA) and another one which also applies bit encoding for nucleotides (CD-MAWS-SA-Encoded). The corresponding graph of this data is presented in Figure 5.

Table 3. Execution Time comparison (in seconds) for different versions of CD-MAWS for varying number of Taxa.

Taxa Count	CD-MAWS	CD-MAWS-SA	CD-MAWS-SA-Encoded
100	2.34	0.37	0.31
200	6.83	1.52	0.91
300	13.55	3.33	1.79
400	23.18	5.89	3.06
500	33.97	9.15	4.63
600	47.72	13.29	6.46
700	63.41	17.87	8.60
800	82.62	23.52	11.10
900	104.12	29.71	14.03
1000	129.76	36.53	17.01

8.3 Peak Memory Usage

The composition vector in CD-MAWS generated heavy memory usage in the earlier implementations. By only working with the MAWs (and avoiding the

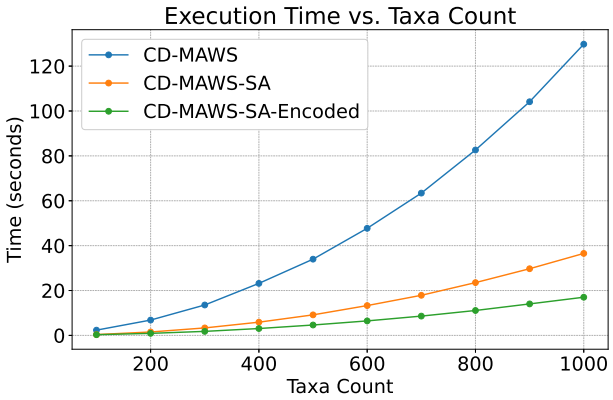

Fig. 5. Execution Time comparison (in seconds) for different versions of CD-MAWS for varying number of Taxa.

composition vectors), we have significantly decreased the memory footprint. Additionally, encoding the MAWs further reduces memory usage. The memory usage comparison is provided in Table 4, with the corresponding graph presented in Figure 6.

Table 4. Peak memory consumption (in MB) for different versions of CD-MAWS for varying number of Taxa

Taxa Count	CD-MAWS	CD-MAWS-SA	CD-MAWS-SA-Encoded
100	90.07	14.81	8.79
200	199.08	25.99	15.00
300	371.25	37.54	20.96
400	449.67	48.80	26.85
500	526.91	59.90	32.89
600	914.80	71.50	38.66
700	1008.31	82.65	45.53
800	1102.43	98.06	52.14
900	1198.46	111.13	57.99
1000	1286.33	116.76	62.69

9 Results on Biological Dataset

We utilized datasets obtained from the AF project [25] to evaluate the performance of our improved method. Specifically, we tested our method on five datasets and compared its performance against the method described in CD-MAWS [1]. Detailed information about the datasets is provided in Table 5, while

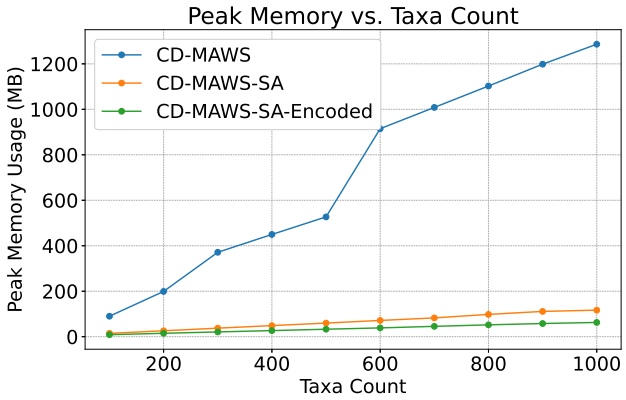

Fig. 6. Peak memory consumption (in MB) for different versions of CD-MAWS for varying number of Taxa

the time taken to execute both methods on these datasets is presented in Table 6.

Table 5. Summary of the AFProject [25] datasets used in this study.

Dataset	**Number of Species**	**Total Sequence Size (MB)**
Fish mtDNA	25	0.42
E.coli/Shigella	29	144.3
E.coli/Shigella (HGT)	27	134.70
Yersinia (HGT)	8	37.50
Simulated (HGT)	33	74.40

10 Discussion

The enhanced CD-MAWS method presented in this study significantly improves computational efficiency as well as memory consumption. By employing a two-pointer method on the sorted MAW set, encoding MAWs in binary, and utilizing suffix automata for MAW generation, we have reduced the computational complexity from $\mathcal{O}(\max(km^3n, km^2n\log n))$ to $\mathcal{O}(\max(m^2n, kmn))$. These improvements address the primary computational bottlenecks of the original CD-MAWS method, making it more suitable for large-scale genomic studies.

Compared to previous alignment-free methods, such as k-mers and spectral methods, our approach provides a more nuanced understanding of evolutionary relationships through the use of MAWs. This method not only retains the advantages of being alignment-free but also improves the scalability of phylogenetic

Table 6. Execution Time comparison (in seconds) between CD-MAWS [1] and our improved one for AFProject [25] dataset

Dataset	**CD-MAWS**	**Proposed**
Fish mtDNA	1.86	0.24
E.coli/Shigella	5.60	1.00
E.coli/Shigella (HGT)	4.66	0.91
Yersinia (HGT)	1.11	0.126
Simulated (HGT)	15.54	2.51

Table 7. Execution time comparison (in seconds) between the MAW generation algorithms of Barton et al. [20] followed by sorting in lexicographical order and our proposed one with Suffix Automata for AFProject [25] dataset

Dataset	**Barton et al.**	**Proposed**
Fish mtDNA	0.67	0.293
E.coli/Shigella	179.54	103.921
E.coli/Shigella (HGT)	168.84	99.99
Yersinia (HGT)	48.25	27.77
Simulated (HGT)	96.16	57.50

estimations. However, our study has limitations. The assumption of a maximum MAW length of 32, while practical, may not be optimal for all datasets. For longer MAWs, we propose using hashing. To minimize the probability of hash collisions, we propose employing double hashing. Because of use of hashing, equality checking can still be performed in constant time. We plan to experiment with this approach in future. Adaptive approaches to determining the appropriate MAW length based on the specific characteristics of the data shold also be investigated. Additionally, while the binary encoding of MAWs enhances comparison speed, it may require further optimization for different hardware architectures.

Future research could focus on extending the CD-MAWS method to other types of genomic data, such as RNA or protein sequences, and integrating it with other phylogenetic tools to provide a more comprehensive analysis framework. Exploring the use of machine learning techniques to predict evolutionary relationships based on MAWs could also be a promising direction.

11 Conclusion

In this paper, we presented an enhanced version of the CD-MAWS method for phylogenetic estimation, achieving a significant reduction in computational complexity while maintaining high accuracy. Our improvements, including refined cosine distance calculations, binary encoding of MAWs, and the use of suffix automata for MAW generation, have addressed the major computational challenges of the original method.

These advancements make the CD-MAWS method a more practical and scalable tool for large-scale genomic analyses, capable of efficiently handling modern datasets. The impact of our work lies in its potential to facilitate more accurate and faster phylogenetic estimations, contributing to a deeper understanding of evolutionary relationships. We have made our implementation of CD-MAWS open-source and freely available at `https://github.com/TamimEhsan/cd-maws-sa`. Looking forward, we anticipate that our method will be integrated into broader phylogenetic analysis pipelines and adapted for various types of genomic data. The continued development and refinement of alignment-free methods like CD-MAWS will be crucial for advancing the field of phylogenetic analysis and unlocking new insights into the evolutionary history of life.

Disclosure of Interests. The authors state that there is no competing interest for this study.

References

1. Anjum, N, Nabil, R. L., Rafi, R. I., Bayzid, M. S. and Rahman, M. S.: CD-MAWS: An Alignment-Free Phylogeny Estimation Method Using Cosine Distance on Minimal Absent Word Sets. Journal **IEEE/ACM Trans. Comput. Biol. Bioinformatics** 20, 1 (Jan.-Feb. 2023), 196–205. `https://doi.org/https://doi.org/10.1109/TCBB.2021.3136792`
2. Yi, H. and Jin, L.: Co-phylog: an assembly-free phylogenomic approach for closely related organisms," Journal **Nucleic Acids Research**, vol. 41, p. e75, Apr 2013.
3. Klotzl, F. and Haubold, B.: Phylonium: fast estimation of evolutionary distances from large samples of similar genomes. Journal **Bioinformatics**, vol. 36, no. 7, pp. 2040–2046, 2020.
4. Ondov, B. D., Starrett, G. J., Sappington, A., Kostic, A., Koren, S., Buck, C. B. and Phillippy, A. M.: Mash screen: high-throughput sequence containment estimation for genome discovery. Journal **Genome Biology**, vol. 20, no. 1, p. 232, 2019.
5. Sarmashghi, S., Bohmann, K., Gilbert, M. T. P., Bafna, V. and Mirarab, S.: Skmer: assembly-free and alignment-free sample identification using genome skims. Journal **Genome Biology**, vol. 20, no. 1, pp. 1–20, 2019.
6. Aurell, E., Innocenti, N. and Zhou, H.-J.:"The bulk and the tail of minimal absent words in genome sequences". Journal **Physical Biology**, vol. 13, no. 2, p. 026004, 2016.
7. Collins, K. and Warnow, T.: "Pasta for proteins". Journal **Bioinformatics**, vol. 34, pp. 3939–3941, 11 2018.
8. Rambaut, A. and Grass, N. C.: Seq-gen: an application for the Monte Carlo simulation of DNA sequence evolution along phylogenetic trees. Journal **Bioinformatics**, vol. 13, no. 3, pp. 235–238, 1997.
9. Liu, K., Raghavan, S., Nelesen, S., Linder, C. R. and Warnow, T.: Rapid and accurate largescale coestimation of sequence alignments and phylogenetic trees. Journal **Science**, vol. 324, no. 5934, pp. 1561–1564, 2009.
10. Zielezinski, A., Girgis, H.Z., Bernard, G. et al.: Benchmarking of alignment-free sequence comparison methods. Journal **Genome Biology** 20, 144 (2019). `https://doi.org/https://doi.org/10.1186/s13059-019-1755-7`

11. Ondov, B. D., Treangen, T. J., Melsted, P., Mallonee, A. B., Bergman, N. H., Koren, S. and Phillippy, A. M.: "Mash: fast genome and metagenome distance estimation using minhash," Journal **Genome Biology**, vol. 17, no. 1, p. 132, 2016
12. Abeysundera, M., Field, C., Gu, H.: Phylogenetic Analysis Based on Spectral Methods, Journal **Molecular Biology and Evolution**, Volume 29, Issue 2, February 2012, Pages 579–597, https://doi.org/https://doi.org/10.1093/molbev/msr205
13. Liu, Z., Meng, J., Sun, X.: A novel feature-based method for whole genome phylogenetic analysis without alignment: Application to HEV genotyping and subtyping, Journal **Biochemical and Biophysical Research Communications**, Volume 368, Issue 2, 2008, Pages 223-230, ISSN 0006-291X, https://doi.org/https://doi.org/10.1016/j.bbrc.2008.01.070
14. Akon, M., Akon, M., Kabir, M., Rahman, M. S., and Rahman, M. S.: "ADACT: a tool for analysing (dis)similarity among nucleotide and protein sequences using minimal and relative absent words," Journal **Bioinformatics**, 2020. In Press.
15. Rahman, M. S., Alatabbi, A., Athar, T., Crochemore, M.and Rahman, M. S.: "Absent words and the (dis)similarity analysis of DNA sequences: an experimental study," Journal **BMC Research Notes**, vol. 9, p. 186, Mar 2016.
16. Silva, R. M., Pratas, D., Castro, L., Pinho, A. J., and Ferreira, P. J.: "Three minimal sequences found in ebola virus genomes and absent from human DNA," Journal **Bioinformatics**, vol. 31, no. 15, pp. 2421–2425, 2015.
17. Chairungsee, S. and Crochemore, M.: "Using minimal absent words to build phylogeny", Journal **Theoretical Computer Science**, vol. 450, pp. 109–116, Sep 2012.
18. Garcia, S. P. and Pinho, A. J.: "Minimal absent words in four human genome assemblies". Journal **PLoS One**, vol. 6, p. e29344, Dec 2011.
19. Belazzougui, D. and Cunial, F.: "A framework for space efficient string kernels". Journal **Algorithmica**, vol. 79, no. 3, pp. 857–883, 2017.
20. Barton, C., Heliou, A., Mouchard, L. et al: "Linear-time computation of minimal absent words using suffix array". Journal **BMC Bioinformatics** 15, 388 (2014). https://doi.org/https://doi.org/10.1186/s12859-014-0388-9
21. International Human Genome Sequencing Consortium. Initial sequencing and analysis of the human genome. Journal **Nature** 409, 860–921 (2001). https://doi.org/https://doi.org/10.1038/35057062
22. Fernández, Pol et al.: "A 160 Gbp fork fern genome shatters size record for eukaryotes". Journal **iScience**, Volume 27, Issue 6, 109889
23. Blumer, A., Blumer, J., Ehrenfeucht, A., Haussler, D., McConnell, R.: "Building the minimal DFA for the set of all subwords of a word on-line in linear time". (1984) In: Paredaens, Journal **Automata, Languages and Programming**. ICALP 1984. Lecture Notes in Computer Science, vol 172. Springer, Berlin, Heidelberg. https://doi.org/https://doi.org/10.1007/3-540-13345-3_9
24. Maxime Crochemore, Christophe Hancart. "Automata for matching patterns". Rozenberg G., Salomaa A. "Handbook of Formal Languages", 2, Linear Modeling: Background and Application, Springer-Verlag, pp.399-462, 1997. ffhal-00620792f
25. Zielezinski, A., Girgis, H.Z., Bernard, G. et al. "Benchmarking of alignment-free sequence comparison methods". Journal **Genome Biology** 20, 144 (2019). https://doi.org/https://doi.org/10.1186/s13059-019-1755-7