TÉCNICO LISBOA

CSA-MEM: Enhancing Circular DNA Multiple Alignment through Text Indexing Algorithms

André Filipe Cisneiros Ferreira Salgado

Thesis to obtain the Master of Science Degree in

Computer Science and Engineering

Supervisor: Prof. Ana Teresa Correia de Freitas

Examination Committee

Chairperson: Prof. Diogo Manuel Ribeiro Ferreira
Supervisor: Prof. Ana Teresa Correia de Freitas
Member of the Committee: Prof. Luís Manuel Silveira Russo

October 2023

Declaration
I declare that this document is an original work of my own authorship and that it fulfills all the requirements of the Code of Conduct and Good Practices of the Universidade de Lisboa.

Acknowledgments

I express my gratitude to Instituto Superior Técnico and Universidade de Lisboa for providing me with the educational foundation to finish my Master's degree. In particular, I am deeply appreciative of my supervisor, Professor Ana Teresa Freitas, for her invaluable teachings and the time she invested in guiding me. Additionally, I extend my thanks to Professor Francisco Fernandes for his mentorship, dedication, and support.

I am thankful for the opportunity to collaborate with inspiring colleagues at INESC and the support of Fundação para a Ciência e a Tecnologia, projects PRELUNA (Grant PTDC/CCIINF/4703/2021) and UIDB/50021/2020, especially for the opportunity to write an article and present my work at the ISBRA conference 2023, held in Wroclaw.

I also want to acknowledge and appreciate all the friends who have crossed my path in Portugal and abroad, while during my Erasmus experience at Universidad Politécnica de Madrid. Working alongside you has been a privilege and I have learnt so much from each of you.

Finally, to my mother, father, and Beatriz I want to express my deepest gratitude for your love and unwavering support.

Abstract

In the realm of Bioinformatics, the comparison of DNA sequences is essential for tasks such as phylogenetic identification, comparative genomics, and genome reconstruction. Methods for estimating sequence similarity have been successfully applied in this field. The application of these methods to circular genomic structures, common in nature, poses additional computational hurdles. In the advancing field of metagenomics, innovative circular DNA alignment algorithms are vital for accurately understanding circular genome complexities. Aligning circular DNA, more intricate than linear sequences, demands heightened algorithms due to circularity, escalating computation requirements, and runtime. This research proposes CSA-MEM, an efficient text indexing algorithm to identify the most informative region to rotate and cut circular genomes, thus improving alignment accuracy. The algorithm uses a circular variation of the FM-Index and identifies the longest chain of non-repeated maximal subsequences common to a set of circular genomes, enabling the most adequate rotation and linearisation for multiple alignment. The effectiveness of the approach was validated in five sets of mitochondrial, viral, and bacterial DNA. The results show that CSA-MEM significantly improves the efficiency of multiple sequence alignment, consistently achieving top scores compared to other state-of-the-art methods. This tool enables more realistic phylogenetic comparisons between species, facilitates large metagenomic data processing, and opens up new possibilities in comparative genomics.

Keywords

Circular DNA; Multiple Alignment; Text Indexing; FM-Index.

Resumo

Na área de Bioinformática, a comparação de sequências de ADN é essencial para tarefas como identificação filogenética, genómica comparativa e reconstrução de genomas. Os métodos para estimar a semelhança entre sequências têm sido aplicados com sucesso neste campo. A aplicação destes métodos a estruturas genómicas circulares, comuns na natureza, apresenta desafios computacionais adicionais. No domínio da metagenómica, algoritmos inovadores de alinhamento de ADN circular são vitais para compreender com precisão as complexidades de genomas circulares. O alinhamento de ADN circular, mais complexo do que sequências lineares, requer a utilização de algoritmos mais sofisticados devido ao aumento dos requisitos computacionais e tempo de execução associado à circularidade destes elementos. Este estudo propõe o CSA-MEM, um algoritmo eficiente de indexação de texto para identificar a região mais informativa do genoma para os rodar e cortar, melhorando assim a precisão do alinhamento. O algoritmo utiliza uma variação circular do FM-Index e identifica a cadeia mais longa de subsequências máximas não repetidas comuns a um conjunto de genomas circulares, permitindo a rotação mais adequada e a linearização para o alinhamento múltiplo de sequências circulares. A eficácia da abordagem foi validada em cinco conjuntos de ADN mitocondrial, viral e bacteriano. Os resultados mostram que o CSA-MEM melhora significativamente a eficiência do alinhamento de múltiplas sequências, alcançando consistentemente pontuações elevadas em comparação com os métodos já existentes. Esta ferramenta permite realizar comparações filogenéticas mais precisas entre espécies, facilita o processamento de grandes volumes de dados metagenómicos, e introduz novas possibilidades na comparação de genomas.

Palavras Chave

ADN Circular; Alinhamento de múltiplas sequências de ADN; Indexação de Texto; FM-Index;

Contents

List of Figures

List of Tables

List of Algorithms

Acronyms

DNA	Deoxyribonucleic acid
BWT	Burrows-Wheeler Transform
HIS	Heaviest Increasing Subsequence
LCP	Longest Common Prefix
LIS	Longest Increasing Subsequence
LMS	Leftmost S-type
MEM	Maximal Exact Matches
MUM	Maximal Unique Matches
MSA	Multiple Sequence Alignment
RLFM	Run-Length FM-Index
RNA	Ribonucleic Acid
SA	Suffix Array
SA-IS	Suffix Array Induced Sorting

1

Introduction

Contents

Recent advances in metagenomics have propelled the field to new frontiers, allowing researchers to explore microbial communities with unprecedented depth and breadth [2]. However, as datasets become larger and more intricate, the challenge of addressing circular Deoxyribonucleic acid (DNA) alignment, efficient data processing, accurate taxonomic classification, and integration of multiomics data is of paramount importance [3]. Overcoming these challenges will not only advance our understanding of microbial ecosystems, but will also enable innovative applications in biotechnology, environmental science, and personalised medicine [4].

With the expansion in scope and scale of the metagenomics field, comes the need to tackle the challenges posed by the analysis of large and diverse metagenomic datasets. One of the main challenges is the efficient alignment of circular DNA within these complex datasets. Circular genomes are common in many microorganisms, and aligning them is crucial for understanding their structure and function.

1.1 Motivation

Genomic analysis stands as an indispensable instrument in the study of biological and genetic information inherent to living organisms, and the sector has gone through a profound paradigm shift due to groundbreaking advances in high-throughput sequencing technologies [5] [6]. Since DNA serves as the fundamental repository of genetic information that encodes vital components such as proteins, regulatory elements, and other functional molecules essential for the existence of life, a deep understanding of its intricate structure and functionality is crucial to understand the genetic basis of traits, diseases, and evolutionary relationships between species [7] [6] [8].

The study of genetic information provides valuable information on evolutionary relationships. By comparing similarities and differences in DNA among different species, researchers can construct phylogenetic trees that depict the evolutionary history of organisms. These trees help trace common ancestors, identify genetic adaptations, and study the divergence of species over time [9] [10].

Furthermore, the analysis of DNA sequences is essential for comparative genomic studies within populations and among individuals. Genetic diversity is the driving force behind adaptation and evolution, as it allows species to respond to environmental changes and challenges [11] [12] [13]. Researchers can use DNA sequencing and alignment techniques to identify genetic variations, such as single nucleotide polymorphisms (SNPs) or insertions/deletions, which are crucial to understanding the genetic structure of living beings [14] [15].

In the context of functional genomics, DNA sequencing plays a central role in the identification of genes and regulatory elements [16]. Properly aligned sequences allow researchers to annotate genes, promoters, enhancers, and other functional regions in genomes [17]. This information is invaluable for studying gene expression, regulation, and the mechanisms underlying various cellular processes [18]. Identifying conserved sequences between species through multiple sequence alignment also helps pinpoint highly conserved functional elements that are critical for maintaining essential biological functions [16] [19] [20] .

In some specific cellular organelles and extrachromosomal elements, DNA can be found to have a closed loop or circle structure, as opposed to the linear composition of DNA present in the chromosomes of most organisms. Circular DNA molecules, known as plasmids, play an important role in conferring adaptive advantages, such as antibiotic resistance and virulence, to bacteria [21].

Their circular configuration allows efficient replication, maintenance and transfer of genetic information within these organelles and extrachromosomal elements, contributing to their specialised roles in cellular processes and adaptation [22] [23] [24] [25] [26]. Circular mitochondrial DNA plays an essential role in the survival and energy production of eukaryotic cells [27] and has long been used for phylogenetic analyses [28]. Smaller structures known as extrachromosomal circular DNA are considered a hallmark of genomic flexibility in eukaryotes [29]. Furthermore, the circular nature of DNA in

some viruses has a major impact on their replication and infection strategies [30]. Understanding these complex mechanisms is crucial for the development of antiviral therapies and vaccines [31].

In order to sequence circular DNA, researchers start by separating the DNA molecules from bacterial cells using techniques such as centrifugation or gel electrophoresis. Once the DNA has been isolated, it can be sequenced using methods similar to those of linear DNA [23] [26] [32]. The result of this process in metagenomics is always a dataset with millions of short reads, making the assembly and alignment steps very challenging from a computational point of view.

Existing DNA alignment algorithms, often designed for linear genomes, such as ClustalW [33], may struggle to handle the unique characteristics of circular DNA, leading to misinterpretations and inaccuracies in the analysis [34]. The special importance of circular DNA presents a unique challenge to compare its sequences. Because circular sequences can start from any point, it adds complexity. This distinctive feature makes traditional linear-centric multiple alignment algorithms inadequate because they lack the adaptability to effectively cope with the inherent circular structure. The outcome is a possible loss of critical genetic information during the alignment process [1].

Identifying similarities between two or more circular sequences can be very challenging, mainly because of the need for more significant sequence conservation between different circular molecules. Unlike linear DNA, circular DNA has no clearly defined beginning or end, making it difficult to define its orientation and edge points during the alignment process with the reference genome. Multiple alignment candidates may result from two circular sequences, making it difficult to determine which alignment is the most relevant [35]. Circular DNA sequences can also vary significantly in size and composition, with some containing highly repetitive or highly conserved regions, adding the difficulty of providing reliable alignments which can lead to a standardisation problem in terms of position coordinates [35].

Furthermore, as metagenomic datasets increase in size and complexity, issues related to data management and analysis, processing speed, and computational resources become increasingly pressing. Efficient algorithms are needed to address challenges related to assembly quality, binning, and functional annotation, which are vital to extract meaningful biological information from the sheer volume of metagenomic data [36].

Beyond identifying the optimal rotation for each sequence in a multiple circular DNA alignment, it is also necessary to address the challenge of handling large volumes of data. In this context, new algorithms that provide better solutions in terms of space and time efficiency are necessary.

1.1.1 Problem Statement

Circular sequences, such as those found in mitochondrial DNA and viral genomes, are ubiquitous in the biological world. Their unique circular nature mirrors the continuous and cyclical processes of life. However, as mentioned above, when it comes to computational analysis, circularity becomes a formidable obstacle. Unlike linear sequences with well-defined start and end points, circular sequences wrap around themselves, which presents a challenge when aligning them with conventional linear reference sequences [1].

To address this conundrum, researchers have resorted to cutting circular sequences at arbitrary points to transform them into linear representations, enabling compatibility with existing sequence alignment tools [1]. Yet, this seemingly simple task is fraught with intricacies. For example, in the context of a viral genome, when selecting an arbitrary cutting point in proximity to a gene's coding region, inadvertent division of the gene into two fragments may occur, potentially resulting in its unrecognizability during subsequent alignment processes. Conversely, if the chosen cutting point is too distant from the gene, there is a risk of losing critical contextual and regulatory elements, thereby compromising the biological significance of subsequent analytical procedures.

The issue becomes even more pronounced in comparative genomics. Consider the scenario in which circular sequences of different virus strains are examined. In one strain, the gene responsible for drug resistance might conveniently fall within a linearised segment, facilitating straightforward alignment and subsequent analysis. However, in another strain, a slight variation in the location of the cutting point could lead to the fragmentation of the same gene into multiple fragments, making it extremely challenging to identify the drug resistance gene through conventional alignment methods, as expounded in prior research [37] [38].

To further illustrate this point, consider a circular DNA molecule, such as a bacterial plasmid. Plasmids often serve as carriers for pivotal genes, including antibiotic resistance genes, which can have significant implications for public health. Cutting a plasmid at an arbitrary location can lead to the loss of these vital genetic markers or disrupt the reading frames, making the analysis misleading or inconclusive [21].

Figure 1.1 vividly demonstrates these challenges. It shows one circular sequence represented in three distinct rotations due to different cutting points. In rotation 0, a biologically significant sequence element is intact and easily recognisable. However, in rotations 1 and 2, that same sequence element is fragmented or lost due to the arbitrary placement of the cut.

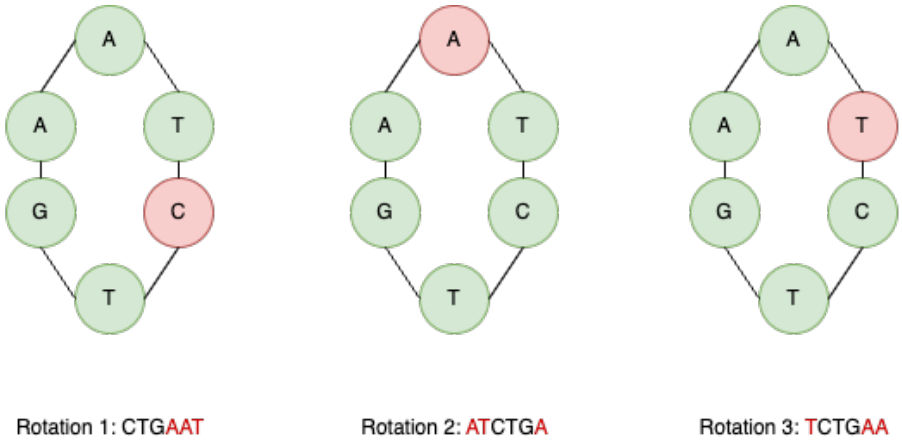


Figure 1.1: Circular DNA comparison on the same sequence but with three different cuts.

In essence, the problem at hand is not merely about converting circular sequences into linear forms, but rather about doing so intelligently and systematically to preserve biological significance. It is about ensuring that the chosen cut points maximise the common alignment across various rotational orientations. The implications of addressing this problem extend beyond bioinformatics and computational biology; they touch upon the accuracy and reliability of genetic and genomic analyses, with potential impacts on fields ranging from medicine to agriculture.

1.2 Objectives

To effectively address the challenges associated with circular multiple sequence alignment in large datasets, the following objectives were delineated:

- (1) Study and adapt advanced data structures such as the FM-Index [39] to **achieve a computationally efficient exact solution for circular sequence matching**;
- (2) Propose and implement an algorithm for the **effective identification of the longest chain of non-repeated maximal exact matches (MEMs) common to a set of circular DNA sequences**, in an approach similar to the CSA tool [1] to facilitate a seamless rotation and linearisation process, specifically designed for multiple circular alignment applications.
- (3) Implement and integrate all the necessary algorithms and data structures to **create an open source tool known as CSA-MEM**.
- (4) Ensure that the resulting tool is **the most efficient in terms of memory requirements and execution time**, with a special focus on **the treatment of large volumes of data**.

1.3 Organization of the Document

The work presented in this thesis is organised into five chapters as follows.

- **Chapter 2 - Relevant Background** introduces essential background concepts and the current state of the art relevant to this thesis.
- **Chapter 3 - Methodology** describes the methodological framework used to identify maximal exact circular matches and determine optimal cutting positions within circular sequences.
- **Chapter 4 - Results and Discussion** presents the results obtained from the analysis of real data sets and provides an in-depth discussion of these findings.
- **Chapter 5 - Conclusions** concludes the document, offering final remarks and suggestions for future work.

1.4 Publications

This research was presented at the **International Symposium on Bioinformatics Research and Applications (ISBRA)** held in Poland and subsequently published in the corresponding proceedings of the **Springer Lecture Notes in Computer Science book series** (LNBI, volume 14248). Furthermore, it was featured in a poster session during the **2nd Microbiome PT Summit** in Lisbon.

2

Relevant Background

Contents

This section stands as a pillar that establishes the core computational principles that serve as the foundation for this thesis. A deep understanding of these principles, rooted in the realms of computer science, mathematics, and related fields, is essential to navigate this research.

The necessary terminology and concepts that form the vocabulary of this work are expounded upon, providing a robust foundation for understanding the core theoretical principles of circular DNA sequence alignment, driving the research problem and its proposed solutions. In accordance with these imperative notions, the topic of sequence alignment is then introduced along with the current state-of-the-art, shedding light on the most recent developments and trends within the field.

2.1 String Matching

String matching algorithms have been proven to be indispensable for locating occurrences of prevalent patterns within reference texts while simultaneously counting their frequency of occurrence. These algorithms are highly versatile and have gained widespread adoption in various domains, including information retrieval, data compression, security, pattern recognition, and more [40]. Notably, the field of bioinformatics has greatly benefited from these algorithms, particularly in the context of DNA sequence alignment, where, as shown before, the primary objective is to uncover intricate patterns encoded within strings.

Deoxyribonucleic acid, often abbreviated as DNA, constitutes the fundamental genetic material that encodes the intricate instructions governing the development and functioning of living organisms. The molecular structure of DNA consists of a unique assembly of four distinct nucleotide bases, namely adenine (A), cytosine (C), guanine (G) and thymine (T). Functional segments within DNA, recognised as genes, partake in a complex process that involves transcription and translation, which ultimately leads to the synthesis of precise proteins. These genetic elements inherently harbour the indispensable information requisite for the determination of the amino acid sequence within proteins, an intricate molecular choreography essential for various biological processes. As such, the quest for a computational representation of DNA assumes paramount significance in contemporary scientific research. This representation typically manifests itself as a text string denoted T with a specific length of n, offering a computational avenue for exploring and deciphering the intricate genetic code.

$$T = T[1], T[2], ..., T[n]$$

In bioinformatics, it is assumed that the last character of a DNA sequence is always a special terminator character $\$$ which is lexicographically smaller than all other characters, resulting in the final alphabet:

$$\Sigma = \{\$, A, C, G, T\}$$

2.1.1 Exact String Matching

Definition 1. *"Given a string P called the pattern and a longer string T called the text, the exact matching problem is to find all occurrences, if any, of pattern P in text T [41]."*

Taking into account the set of characters $[l, r]$, with r always larger than l, and a reference string T, the substring of T equivalent to $[l, r]$ is denoted as $T[l, r]$. In the realm of exact string matching, especially when dealing with DNA sequences, two primary functions come into play to solve these types of problems: locating and counting the occurrences of a specific pattern. If we treat $[l, r]$ as

a pattern denoted by P, the result of the function $locate(P)$ comprises the positions within the text T where instances of the pattern begin. Meanwhile, the function $count(P)$ yields the number of times the range $[l, r]$ appears within T.

As an illustration, consider $P = AAT$ and $T = AATCTGAAT$. The result of $locate(P)$ corresponds to positions 1 and 7 where the pattern P begins within T. Simultaneously, the result of $count(P)$ will be 2, indicating that the pattern P appears twice in T.

From a biological point of view, researchers may be interested in finding a particular gene or regulatory element within a genome, and, commonly, exact string matching algorithms are used to locate the exact sequence of nucleotides that correspond to these genetic elements. This process is fundamental to understanding the functions, regulatory mechanisms, and genetic variations that can cause diseases. An exemplification of this concept can be observed when potential start codons are sought in a protein-coding mRNA sequence. In such instances, the focus lies on identifying all instances where the initiation of an exact match between a pattern and where a mRNA sequence begins [42].

Apart from gene identification, the significance of exact string matching is evident in tasks like genome assembly, where the primary objective is the reconstitution of an organism's entire genomic sequence. Given the large volume of fragmented DNA data produced by sequencing technologies, the use of exact string matching algorithms is integral to the precise alignment of overlapping sequences, enabling scientists to assemble the genome [43] [44].

2.1.2 Approximate String Matching

Definition 2. *Search for substrings of the text that are no more than k characters away from the pattern, with a given distance d, where k is not an integer larger than the size of the pattern. [45].*

Approximate string matching, analogous to exact string matching, operates on a given reference string, also denoted T, and a specific pattern, referred to as P. However, the key distinction lies in the degree of similarity or dissimilarity allowed between the pattern and the substrings of the reference string. Instead of looking for exact matches, approximate string matching aims to identify instances where P closely resembles substrings of T within a specified tolerance. This tolerance can encompass variations introduced by insertions, deletions, substitutions, or other character-level discrepancies. The goal is to pinpoint and quantify the occurrences of patterns that are not necessarily identical but are within the defined similarity threshold, which makes this a crucial problem in applications such as DNA sequence alignment, where accommodating variations in the data is essential for accurate results.

In the context of DNA sequence alignment, "approximate" refers to the acceptance of errors or variations within valid matches. These errors can encompass nucleotide substitutions, insertions, deletions, or other genetic mutations. Sequence alignment, which involves the process of lining up characters in DNA sequences, takes on a broader definition here, allowing for not only perfect matches, but also

mismatches. Furthermore, it accommodates the concept of gaps, where characters from one DNA sequence are matched with spaces introduced in the opposing sequence. This flexibility in aligning sequences is critical for capturing the evolutionary history and genetic relationships among species, individuals, or even viruses [41].

The transition from problems primarily focused on substrings to those concerning subsequences marks a fundamental shift in the scope of analysis. A substring is a sequence of characters that must be contiguous within a larger string. In contrast, a subsequence is a sequence that can be embedded within a string without the requirement of contiguity. For example, the pattern $P = AAT$ is a subsequence, but not an exact substring within $T = ACAGT$.

$$AC\,A\,GT, d = 1$$

This shift from exact substrings to subsequences is a logical consequence of moving from exact matching to inexact matching, where accommodating variations in the data becomes paramount. It allows the identification of similarities that extend beyond contiguous stretches of genetic information [41].

It is essential to have efficient algorithms that can accurately perform these tasks, as DNA sequences can be very long and analysing them can be time-consuming.

2.2 Text Indexing

Text indexing is an exceptionally efficient approach to string matching that involves the construction of data structures designed to improve the querying and retrieval of particular words or phrases, at the cost of some additional storage space [46]. Within the realm of DNA alignment, it serves as a rapid way to pinpoint specific information within an extensive assortment of sequences. Current state-of-the-art text indexing methods incorporate data structures such as space-efficient adaptations of the Suffix Array [47], the Burrows-Wheeler Transform (BWT) [48], and the FM-Index [39]. These will be further elucidated in a subsequent discussion to enhance the understanding of the proposed solutions for performing string matching tasks.

Suffix Array

The Suffix Array (SA) [47] is a data structure that stores references to all suffixes of a string T, allowing efficient querying and manipulation of these suffixes. It is a powerful tool for compressing and pattern-searching between DNA sequences.

Given the alphabet Σ and a sequence $T = TGCCTTTG\$$ of size $n = 9$, the suffix array $SA[1, n]$ can be built by first storing the starting positions of each suffix in T, as presented in Figure 2.1.

Figure 2.1: Visual representation of the Suffix Array of T = TGCCTTTG$.

SA									
1	T	G	C	C	T	T	T	G	$
2	G	C	C	T	T	T	G	$	T
3	C	C	T	T	T	G	$	T	G
4	C	T	T	T	G	$	T	G	C
5	T	T	T	G	$	T	G	C	C
6	T	T	G	$	T	G	C	C	T
7	T	G	$	T	G	C	C	T	T
8	G	$	T	G	C	C	T	T	T
9	$	T	G	C	C	T	T	T	G

However, to take full advantage of the suffix array, it must be sorted in lexicographic order, as shown in Figure 2.2.

Figure 2.2: Visual representation of the lexicographically ordered Suffix Array of T.

SA									
9	$	T	G	C	C	T	T	T	G
3	C	C	T	T	T	G	$	T	G
4	C	T	T	T	G	$	T	G	C
8	G	$	T	G	C	C	T	T	T
2	G	C	C	T	T	T	G	$	T
7	T	G	$	T	G	C	C	T	T
1	T	G	C	C	T	T	T	G	$
6	T	T	G	$	T	G	C	C	T
5	T	T	T	G	$	T	G	C	C

The suffix array can be constructed in $O(n)$ time, requiring memory equivalent to the sequence size, $O(n)$ space [49]. This method can be applied for pattern matching employing common search techniques, such as the binary search algorithm, to identify the indices that contain the start of each pattern within the array. However, the versatility of a suffix array goes beyond this elementary search technique, as it can be leveraged for building supplementary data structures that significantly improve the efficiency of pattern matching tasks, with the Longest Common Prefix (LCP) array among suffixes being a notable example.

LCP

The array of Longest Common Prefixes (LCP) in a sequence T is a data structure of size n that stores the length of the Longest Common Prefixes between pairs of consecutive suffixes.

The $LCP[1, n]$, where $LCP(P, P')$ is the length of the longest common prefix between strings P and P', satisfies:

$$LCP[i] = \begin{cases} LCP(T[SA[i-1], n], T[SA[i], n]), & i \neq 1 \\ 0, & i = 1 \end{cases}$$

For example, considering again the sequence $T = TGCCTTTG\$$ and its suffix array, the LCP value between the two consecutive suffixes TTG and $TTTG$ can be computed by comparing their characters until a mismatch is found or the end of one of the strings is reached. In this case, as can be seen from the suffixes represented in green at the positions $SA[8]$ and $SA[9]$ of Fig. 2.2, the LCP value between them is 2 for the prefix TT, which results in LCP$[9] = 2$.

The resulting LCP array for the complete example would be:

$$LCP[1, 9] = [0, 0, 1, 0, 1, 0, 2, 1, 2]$$

Burrows-Wheeler Transform (BWT)

The BWT [48] is an algorithm that rearranges a string of characters into a new one more suited to text processing and data compression methods, while also being reversible, meaning that it can be used efficiently to return to the original string.

One of the key benefits of the BWT is that it groups similar characters together, resulting in a more compressible string. Large sequences, which can consume a lot of storage space, can benefit from more compressible strings. In bioinformatics, the BWT is frequently used to efficiently manipulate and store DNA sequences, significantly reducing the amount of storage they require. The resulting sequence also has many beneficial statistical properties that can help identify patterns and trends within a text.

The BWT can be obtained by constructing the suffix array of the original string, containing all these sorted rotations of the string, and finally taking its last column, which is called the L column. The first column, called the F column, is also essential and is used in pattern matching algorithms. The BWT satisfies:

$$L[i] = \begin{cases} T[SA[i] - 1], & SA[i] \neq 1 \\ \$, & SA[i] = 1 \end{cases}$$

To illustrate, consider the example of Fig. 2.3 based on the last sequence mentioned on the other data structure examples:

Figure 2.3: Visual representation of the sorted text rotations to better understand the concept of BWT.

F								L
$	T	G	C	C	T	T	T	G
C	C	T	T	T	G	$	T	G
C	T	T	T	G	$	T	G	C
G	$	T	G	C	C	T	T	T
G	C	C	T	T	T	G	$	T
T	G	$	T	G	C	C	T	T
T	G	C	C	T	T	T	G	$
T	T	G	$	T	G	C	C	T
T	T	T	G	$	T	G	C	C

In this example, the BWT is:

$$BWT = GGCTTT\$TC$$

FM-Index

The FM-Index [39] is a compressed full-text substring index built on top of the BWT that can be used to efficiently search and manipulate large volumes of text. Relying on only the $O(n)$ space to store the BWT characters and some other efficient data structures, the FM-Index is a good choice for applications where space is a priority, as in DNA sequence matching. The FM-Index has several statistical properties that help identify the patterns and trends within the DNA sequences in constant time. In addition, it enhances the reversibility of the BWT, resulting in a more efficient conversion of the transformed string back into the original.

The FM-Index creation process begins with the application of the Burrows-Wheeler transform (BWT) algorithm to the text. Alongside the implicit storage of the suffix array and the BWT, an additional summation array is introduced. This new structure captures the $rank$ of each character in the alphabet, which can be translated to the number of occurrences of the character c in the BWT up to position i, and is represented by the function $rank_c(L, i)$. The reverse operation, denoted as $select_c(L, j)$, yields the position i within the BWT corresponding to the j-th instance of character c.

To $locate$ and $count$ a specific pattern P, the FM-Index algorithm employs a backward search strategy by iteratively applying the *LF-mapping* procedure [39]. Given an index position, this consists of calculating the new index position corresponding to the suffix to the left (in text order) of the current one, using the information in L and F and an auxiliary array $C[c]$ that stores the starting position in the index of each character c, and then setting:

$$LF(i) = C[L[i]] + rank_{L[i]}(L, i)$$

Figure 2.4: Visual representation of the auxiliary data structures required to perform the LF-Mapping steps during pattern search.

F								L		A	C	G	T
$	T	G	C	C	T	T	T	G		0	0	1	0
C	C	T	T	T	G	$	T	G		0	0	2	0
C	T	T	T	G	$	T	G	C		0	1	2	0
G	$	T	G	C	C	T	T	T		0	1	2	1
G	C	C	T	T	T	G	$	T		0	1	2	2
T	G	$	T	G	C	C	T	T		0	1	2	3
T	G	C	C	T	T	T	G	$		0	1	2	3
T	T	G	$	T	G	C	C	T		0	1	2	4
T	T	T	G	$	T	G	C	C		0	2	2	4

In a concise and lucid overview, the search process begins by locating the last character of the pattern P in the initial column of alphabetically sorted suffixes. Once the interval containing this character is identified, the algorithm proceeds by looking for the next character to the left within the pattern P and finding its corresponding occurrences in the same interval on the BWT. Subsequently, the LF-mapping is used, yielding the new positions of this pair of letters in the F column. This process is then reiterated until either the entire pattern P is found or a mismatch arises.

In the context of searching for the pattern $P = CTT$ within the FM-Index, the process involves recognising the sequence of 'T's in the initial column of lexicographically sorted suffixes. To ascertain the following character in the pattern (another 'T'), the relevant interval in the BWT is explored. Subsequently, if any relevant characters ('T') were identified, they become subject of the LF-mapping to determine their position within the initial column again, with this process being repeated. Upon finding the last character of the pattern, in this case, the character 'C,' in the initial column, the suffix array (SA) is used to locate all occurrences of the pattern in the reference text.

R-Index

The R-Index [50], also referred to as Run-Length FM-Index (RLFM), is an extension of the FM-Index, preserving its commitment to facilitate efficient pattern searching in extensive texts. However, its distinguishing feature lies in further optimising storage requirements. Notably, the R-Index surpasses the performance of the FM-Index by allowing pattern counting in $O(m)$ time, where m denotes the pattern length, and by utilising fewer resources.

The R-Index is based on the concept of run-length encoding of the Burrows-Wheeler transformed text. Run-length encoding is a data compression technique that represents runs of successive identical items with a single count of the number of times that element appears. In the case of the R-Index, it is used to decrease the amount of space necessary to keep the transformed text while maintaining the

capacity to perform effective searches. In addition to run-length encoding, the R-Index also combines the reverse search (Backwards Search), enabling the counting of queries on alphabets to be completed much faster.

Overall, the R-Index is a string matching index that combines the FM-Index efficiency with the space-saving capabilities of the run-length encoding. The equivalent run-length encoding of the Burrows-Wheeler transformed text for $BWT = GGCTTT\$TC$ is:

$$BWT_{RLE} = G2CT3\$TC$$

BR-Index

In order to find every instance of a pattern within a text, it is frequently helpful to combine forward and backward searches. For instance, we might perform a forward search to find the first instance of a pattern and a backward search to find all subsequent instances. Combining both searches can help the process run more smoothly, especially when looking for trends among circular DNA sequences.

The FM-Index allows for the computation of the backward search. However, this extension is unidirectional, meaning that it cannot perform forward searches, which, as we mentioned earlier, can be very useful for comparing strings. To address this limitation, Y. Arakawa et al. proposed the BR-Index [51], an improved version of the FM-Index that enables both forward and backward search using structures based on the Bi-directional BWT [52] and the R-Index. The BR-Index employs the BWT of the inverted reference string T, allowing for the interruption of a backward search at any time to initiate a forward search if needed. In addition to this significant improvement, considering that it could neglect the space complexity of the index, the BR-Index also uses the R-Index.

The Bi-Directional BWT is visually demonstrated in the figures 2.5 and 2.6, taking into account the sequence $T = TGCCTTTG\$$.

Figure 2.5: Visual representation of the sorted suffixes to better understand the concept of the forward BWT.

F								L
$	T	G	C	C	T	T	T	G
C	C	T	T	T	G	$	T	G
C	T	T	T	G	$	T	G	C
G	$	T	G	C	C	T	T	T
G	C	C	T	T	T	G	$	T
T	G	$	T	G	C	C	T	T
T	G	C	C	T	T	T	G	$
T	T	G	$	T	G	C	C	T
T	T	T	G	$	T	G	C	C

Figure 2.6: Visual representation of the sorted reversed suffixes to better understand the concept of the reversed BWT.

F'								L'
$	G	T	T	T	C	C	G	T
C	C	G	T	$	G	T	T	T
C	G	T	$	G	T	T	T	C
G	T	$	G	T	T	T	C	C
G	T	T	T	C	C	G	T	$
T	$	G	T	T	T	C	C	G
T	C	C	G	T	$	G	T	T
T	T	C	C	G	T	$	G	T
T	T	T	C	C	G	T	$	G

2.3 Finding Maximal Matches

In metagenomics, finding Maximal Matches (Maximal Exact Matches (MEM) and Maximal Unique Matches (MUM)) between sequences is essential for assembling and analysing genomes. MEMs, or maximal exact matches, are sequences that match precisely between a read and the genome, but cannot be extended in any direction [53]. Research has shown that MEMs are the most useful starting point for aligning short and long reads [54]. MUMs, or maximum unique matches, on the other hand, are long segments of genomic sequences that are identical in both genomes and occur only once in each of them [54]. Using MEMs and MUMs as prospective anchors for the alignment process can avoid computationally expensive alignment extensions and provide an efficient way to construct multiple-sequence alignments.

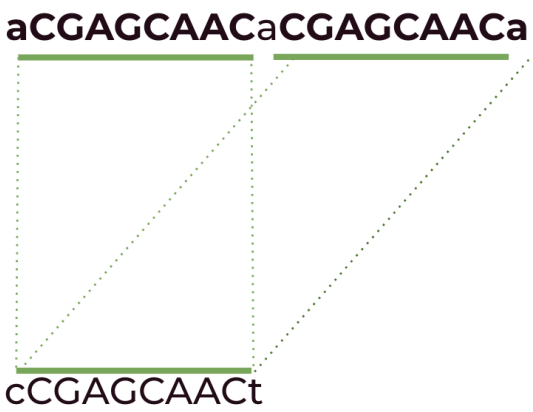
Furthermore, MUMs help generate pangenome graphs in progressive techniques such as progressive Mauve and progressive Cactus [54]. They can also measure the amount of a specific strain present in a sample and compare genomic similarity efficiently. By adding information on population variation, the ability to locate MEMs and MUMs cheaply and on a broad scale can ease the use of assembled high quality genomes and relieve reference genome bias [54].

One of the most widely used tools for this purpose is the MUMmer system [55], specifically its genome sequence aligner nucmer. The initial MUMmer release is a computational tool that uses a suffix tree's data structure to detect the different subsequences inside a given sequence, which are then used to quickly find the maximal matches between two DNA sequences. The approach combines three fundamental ideas: the usage of a suffix tree, the Longest Increasing Subsequence (LIS) and Smith-Waterman alignment [56], all combined to produce a complete system for large-scale genome alignment.

The latest version, MUMmer4 [57], offers significant improvements over previous versions, including a more efficient 48-bit suffix array data structure and the ability to process input query sequences in parallel. MUMmer4 can now accommodate any biologically reasonable length input sequence, making

it suitable for matching large genomes and high-throughput sequencing data. It can also align short and long reads with varied error rates to a reference genome and complete genomes, making it one of the most versatile and efficient genome alignment programmes available in metagenomics.

Figure 2.7: An example of a MEM.



2.3.1 slaMEM

SlaMEM [58] is a competitive option for identifying maximal matches in the context of multiple sequence alignment. Through validation using real-world data, it was proven to be a valuable tool for comparative genomics efforts based on maximal exact matches (MEMs). It uses the concept of *LCP-Intervals*, where an interval of size l is named an $l-interval$ and denoted by $l-[i,j]$, where $1<=i<=j<=n$ and respects the following properties:

- $LCP[i]<l$
- $LCP[k]>=l$ for all k with $(i+1)<=k<=j$
- $LCP[k]=l$ for at least one k with $(i+1)<=k<=j$
- $LCP[j+1]<l$ if $j\neq n$

The first significant LCP interval that encloses a particular LCP interval is considered its parent interval. Parent intervals are helpful to perform right-contraction when performing the backward search procedure over the FM-Index. When finding MEMs for a pattern $P[1,...,m]$, if a mismatch occurs before the first position is reached, say at position $1<k<m$, the base algorithm would have to restart the matching from the second last position to search for $P[1,...,m-1]$, but taking the current interval (corresponding to $P[k,...,m]$) it is possible to take its parent interval (corresponding at least to $P[k,...,m-1]$) and continue the search from there.

The slaMEM algorithm uses the NSV and PSV arrays developed by Fischer et al. [59] to determine the parent interval of a given LCP interval. These arrays indicate the position of the first lower value in the LCP array below or above the current position. Both arrays are defined as follows [58]:

- $PSV[i] = max\{k : (0 <= k < i$ and $LCP[k] < LCP[i])$ or $k = 0\}$
- $NSV[i] = min\{k : (i < k <= n$ and $LCP[k] < LCP[i])$ or $k = n\}$

With this information, the parent interval of an LCP interval can be efficiently determined [58].

- Parent($l - [i, j]) = (LCP[k]) - [PSV[k], (NSV[k] - 1)]$, with

  $k = i,$ if $LCP[i] >= LCP[j + 1]$

  $k = j + 1,$ if $LCP[j + 1] > LCP[i]$

Unfortunately, the current implementation of this tool cannot identify maximal matches in circular sequences, only in linear sequences. To circumvent this constraint and continue the research, the methodology section will introduce modifications designed to enable the accommodation of sequence circularisation.

2.4 Sequence Alignment

As introduced earlier, sequence alignment is a fundamental concept in bioinformatics and molecular biology, serving as a critical tool to compare and analyse DNA sequences. In this process, the sequences are aligned to identify similarities and differences, with the ultimate goal of elucidating the underlying biological and structural significance.

The two primary types of sequence alignment are global alignment and local alignment. Global alignment seeks to align the sequences in their entirety, often for the comparison of sequences with a high degree of similarity [41]. One of the most commonly used algorithms for global alignment is the Needleman-Wunsch algorithm, which employs a dynamic programming method to find the optimal alignment by maximising a similarity score [60].

Definition 3. *"A (global) alignment of two strings S1 and S2 is obtained by first inserting chosen spaces (or dashes), either into or at the ends of S1 and S2, and then placing the two resulting strings one above the other so that every character or space in either string is opposite a unique character or a unique space in the other string [41]."*

As an illustration of global alignment, examine the alignment of strings $TGACAT$ and $TACAC$:

```
TGACAT
T-ACAC
```

In this alignment, both characters T and C result in mismatches, while G in the first sequence corresponds to a gap in the second sequence.

In contrast, local alignment aims to identify short and closely related segments in longer sequences. Smith-Waterman's algorithm is an easy-to-use local alignment method that enables the discovery of subregions with the highest degree of similarity [61].

The accuracy of sequence alignment is greatly influenced by the choice of alignment algorithm and scoring system. When aligning sequences, a scoring matrix, such as BLOSUM (for proteins) [62], is used to assign values to matches and mismatches between nucleotides or amino acids. Gap penalties are also defined to account for the introduction of gaps (insertions or deletions) in the alignment, as these can significantly affect the overall alignment quality. Moreover, the choice of alignment algorithm depends on the specific task and the characteristics of the sequences. Heuristic algorithms such as BLAST [63] are suitable to quickly identify similar sequences in large databases, while more rigorous but slower dynamic programming algorithms such as Needleman-Wunsch [60] and Smith-Waterman [64] are used for precise alignments.

2.5 Circular Sequence Alignment

As previously stated, when aligning circular DNA sequences, researchers encounter a new noteworthy challenge. Conventional alignment methods do not consider the circular nature of these sequences, leading to suboptimal scoring results. This limitation is significant as circular DNA sequence alignments are essential in various biological applications. Because of that, different specialised tools designed exclusively to improve the alignment of circular DNA sequences have been created using different methods and data structures, such as those mentioned before.

2.5.1 Cyclope

The Cyclope algorithm [65], developed in 2006, is an old and slow computer method to improve the alignment and analysis of cyclic sequences, focussing on viroids and other tiny circular RNAs. It employs a cyclic alignment technique, which involves generating a series of pairings that retains the order of the sequence in both strings. Aligns the sequences and finds conserved Ribonucleic Acid (RNA) sec-

ondary structures using a mix of dynamic programming and graph-theoretic approaches. It also solves the problem of circularity, secondary structure development, and inconsistent treatment of end-gaps when aligning circular sequences as linear. The method was tested on a wide range of small circular nucleic acids, including viral RNA genomes, viroids, and several ssDNA viruses from the Parvoviridae, Circoviridae, and Geminiviridae families.

2.5.2 CSA

CSA [1] presents a powerful approach to improve the alignment of multiple circular DNA sequences, particularly mitochondrial DNA (mtDNA) sequences. With linear time complexity, the method employs a cyclic variation of the suffix tree to efficiently identify an optimal rotation among all potential rotations of any circular genomic sequence.

The first step in implementing the algorithm is to create a Cyclic Suffix Tree, the suffix tree-like structure that reflects all rotations of a given sequence rather than all of its suffixes, as in the case with standard suffix trees. The building process is based on Ukkonen's [66] suffix tree construction approach, but with specific changes in how suffix links and loose leaves are implemented. Without additional effort, the method generates all rotations of the sequence in linear time.

Suffix links at the leaf nodes are seen as linking sequential rotations rather than successive suffixes. The leaves are modified to have the same path length from root to end, resulting in all leaves being at the same depth, equal to the length of the original sequence.

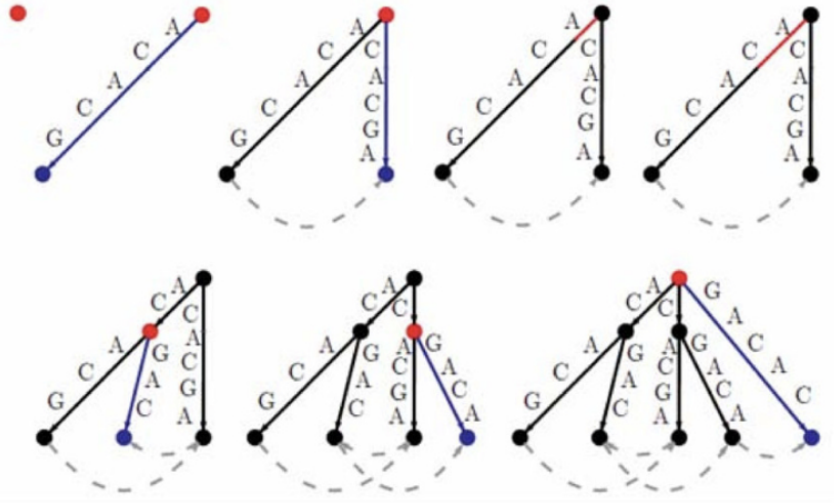


Figure 2.8: Example of the step-by-step construction of the cyclical suffix tree for the sequence ACACG, taken from [1].

Furthermore, while accessing characters from edge labels, if a pointer indicates a location k that extends beyond the end of the original sequence $(i.e., k > n)$, the method subtracts the sequence size

$(i.e., k = k - n)$ from that position. In some circumstances, the algorithm must conduct an additional run over the original string as a last step to construct the last missing rotation.

The main disadvantages of this approach include its maximum of 32 sequences limitation and the high memory consumption of its cyclic suffix tree data structure, which is based on a rudimentary suffix tree data structure implementation, while other much more memory efficient indexing data structures are now currently available.

2.5.3 MARS

MARS [67], which stands for Multiple circular sequence Alignment using Refined Sequences, is a tool implemented using the C++ programming language to improve circular alignments using refined sequences. It follows a three-stage heuristic approach, and in the same way as CSA, it takes a set of sequences as input, and outputs a newly set of rotated sequences. These rotated sequences are then used as input to an Multiple Sequence Alignment (MSA) algorithm to obtain an improved alignment.

The algorithm has three main stages. The first stage has the objective of computing the cyclic edit distance between two sequences. Through a heuristic method, it computes a matrix where each cell holds two values, the edit distance between sequences, and the sequence rotation that minimises the edit distance. The primary objective is to find the optimal rotation, denoted as r, for a sequence x that minimises the $\beta - blockwiseq - gram$ distance when compared to sequence y.

Then, to refine the rotation and obtain a more accurate value for r, the algorithm uses an input parameter to create refined sequences of a specific length from the rotated sequences x and y. Enlarging the values of the input parameter results in an improvement of the refinement of the rotation, but slows down the computational process because it results in longer sequences.

The refined sequences can be divided into three components:

- The first component represents a prefix of x, extracted from a portion of the β blocks.
- The second component is a sequence of the same length as the prefix, consisting of a placeholder character ($\$$).
- The third component is a suffix of x, also taken from a portion of the β blocks. The same process is applied to the sequence y, resulting in a refined sequence in the same format, denoted $y0$, $y1$, and $y2$.

The second stage involves constructing a guide tree using the edit distance data from the matrix. The guide tree is constructed using a neighbour-joining which produces a binary tree that represents the relationships between the sequences based on their edit distances.

The guide tree is then used in the progressive alignment process to determine the order of sequence alignment. Progressive alignment is a method to align multiple sequences by iteratively aligning pairs

of sequences and combining them into a profile. The profile is then aligned with another sequence or profile until all sequences are aligned. At this stage, three possible cases of alignment may occur: sequence with sequence, sequence with a profile, and profile with another profile. Each alignment is scored using a scoring scheme that considers the similarity of letters in the alignment and the presence of gaps (insertions or deletions of letters). The alignment with the highest score is selected as the best alignment, and the rotations of the sequences or profiles involved are refined to improve alignment accuracy.

In general, the MARS method provides a comprehensive approach to improving the alignment of multiple circular sequences by considering rotations, edit distances, and progressive alignment. In terms of disadvantages besides being an heuristic method which means it does not produce necessarily an optimal solution, it also relies on some dynamic programming elements and on a progressive alignment which slows down the process, having in this way a big space for improvement.

2.5.4 Other Approaches

Beyond the methods mentioned before, some alternative methods also employ unique data structures and algorithmic approaches tailored to the specific challenges of circular sequence. However, it is imperative to note that these alternative techniques have been rendered obsolete and have demonstrated inferior performance when compared with the previously discussed tools. Therefore, only a concise review of these alternative circular alignment enhancement techniques will be shown, with a focus on their key features, the datasets on which they have been tested, and their ability to boost alignment scores when the resulting sequences are integrated with multiple sequence alignment tools, just to have a global idea of how they can help the development of the research in question.

The Circal method, as introduced by Fritzsch et al. [68], employs cyclic list alignments of gene orders and implements a progressive alignment strategy. This approach is specifically applied to investigate the phylogenetic relationships among protostomes using exclusively mitochondrial genome configurations. The dataset encompasses a comprehensive collection of metazoan mitochondrial genomes. Phylogenetic inferences are established through the alignment of mitochondrial gene orders, employing the Circal method, followed by the reconstruction of maximum parsimony trees. Furthermore, this approach has shown its efficacy in accommodating larger chloroplast genomes, exemplified by a dataset that includes 60 complete mitochondrial genomes of vertebrates, hemichordates, and cephalochordates. The inherent inefficiency of this package can be attributed to the algorithm it utilises, characterised by a sophisticated gap-cost function. Because of that, it is constrained to handling sequences of relatively limited length, typically comprising fewer than a thousand characters, primarily due to the algorithm's demanding time complexity, which would negatively impact the purpose of this research.

In addition to the previously established method, a more recent approach has been developed to ad-

dress the specific challenges of circular sequence alignment. BEAR, as described by its creators [69], is a dedicated computational tool designed to enhance Multiple Circular Sequence Alignment (MCSA) through the use of maximum-likelihood-based phylogenies. The authors introduce two distinct methodologies within this framework. The initial approach involves an extension of an approximate circular string matching algorithm, which facilitates the execution of approximate circular dictionary matching. This similar computational process is similar to the MARS methodology [67] and also yields a matrix with the edit distances between two sequences, but for this case, a hierarchical agglomerative clustering method is applied to the entire collection of values to determine the appropriate rotations for each sequence cluster.

The second method introduced by BEAR is particularly well-suited for sequences exhibiting higher levels of divergence. This method employs an algorithm tailored for fixed-length approximate string matching, systematically assessing each pair of sequences to identify the most analogous factors of a predetermined fixed length. These identified factors then serve as pivotal elements in determining the appropriate rotations for all input sequences, employing the same hierarchical agglomerative clustering approach outlined previously. BEAR was evaluated in the same datasets as CSA and MARS, and in the current investigation, its utilisation was precluded due to encountered compilation errors, along with a preference for the more recent and effective tool MARS, which incidentally was introduced by one of the coauthors of BEAR.

3

Methodology

Contents

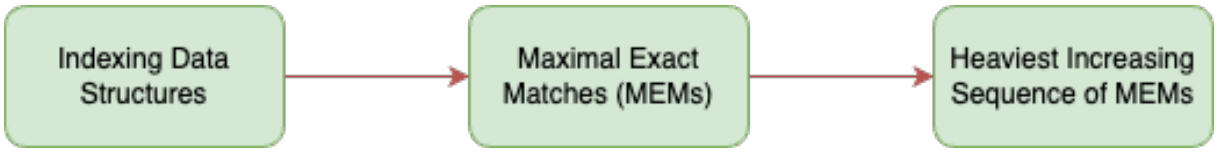
This section delineates the methodology behind CSA-MEM. Its overarching objective is to enhance the accuracy of multiple sequence alignment, particularly when applied to circular DNA.

The approach elucidated herein is organised into a meticulous three-step process with the primary aim of identifying and extracting the longest common chain of nucleotides from a comprehensive sequence dataset. It should be noted that the CSA algorithm [1] has previously established the state-of-the-art accuracy of finding the starting positions of this chain to improve circular MSA. However, CSA-MEM advances the field by removing CSA's limitations regarding both sequence number and size by using more efficient indexing data structures and searching for maximal exact matches.

The first step of CSA-MEM involves the building of specialised indexing data structures. This is achieved by designing a FM-Index variant modified to excel in detecting circular DNA patterns. After establishing this foundational step, a thorough search over the index is conducted to identify all Maximal Exact Matches (MEMs) that are common across all sequences. These MEMs serve as universally

applicable puzzle pieces for each sequence. Subsequently, these puzzle pieces are systematically organised to obtain the longest common chain of nucleotides that will define a new cutting region which optimises the alignment of the sequences.

Figure 3.1: CSA-MEM approach.



3.1 Indexing

In the pursuit of identifying the optimal rotation for the alignment of circular DNA sequences, CSA-MEM strategically searches for MEMs across multiple sequences using the procedural essence of the slaMEM algorithm [58]. However, to seamlessly accommodate circularity, substantial changes were made to the underlying data structures within the approach. Both SA and BWT were subject to meticulous modifications, allowing for an uninterrupted search of the sequences even after encountering the predefined terminal points of the reference sequence. This pivotal enhancement empowers the method to sustain the search for substrings beyond sequence textual boundaries, deeply enriching CSA-MEM circular DNA analysis capabilities.

3.1.1 Suffix Array

The circular nature of the DNA sequences was considered by removing the terminator character present in the standard SA. The modified SA captures the order of all now rotated substrings, enabling the construction of a circular BWT, an essential step for efficient MEM detection on circular sequences.

The algorithm chosen to construct and sort the suffix array is the Suffix Array Induced Sorting (SA-IS) algorithm [49], enriched with a customised modification and a preprocessing step.

3.1.1.A SA-IS Algorithm

The SA-IS algorithm [49], as mentioned above, is a fundamental method to efficiently construct the suffix array of a given string. This algorithm was developed to overcome the limitations of traditional SA construction algorithms, particularly in terms of time complexity. Using a technique called *induced sorting*, SA-IS achieves a worst-case time complexity of $O(n)$, a significant improvement over the nonlinear complexity of the previous approaches.

The core concept of SA-IS is the *induced sorting* algorithm, which is a strategy to sort the suffixes of a string. It involves first sorting certain smaller substrings of the input string and then deducing the sorted order of the remaining suffixes from these smaller sorted elements. This is achieved efficiently by leveraging the properties of the substrings that are already sorted.

The algorithm can be broken down into the following key steps:

L-type/S-type Suffixes The algorithm scans the sequence linearly, classifying each suffix as L-type or S-type. An L-type suffix is lexicographically larger than the next suffix, whereas an S-type suffix is smaller. This classification is crucial for understanding the subsequent steps of the algorithm, especially during the induced sorting phase.

It is possible to efficiently classify each suffix as either S-type or L-type in time $O(n)$, by satisfying the following conditions:

- A suffix starting at position i is an S-type if:
  1. $S[i] < S[i+1]$, or
  2. $S[i] = S[i+1]$ and $suf(S, i+1)$ is S-type.
  3. $S[i] = \$$
- A suffix starting at position i is an L-type if:
  1. $S[i] > S[i+1]$, or;
  2. $S[i] = S[i+1]$ and $suf(S, i+1)$ is L-type.

Taking as an example $T = GTGCCTTT\$$, the suffix $CTTT\$$ is called an S-type suffix, since it lexicographically precedes $TTT\$$, the suffix in the position immediately after it. On the other hand, the suffix $T\$$ is called an L-type suffix, since it lexicographically succeeds $\$$, the suffix in the position immediately after it.

Figure 3.2: Visual representation of the L-type and S-type suffixes of the string GTGCCTTT$

String:	G	T	G	C	C	T	T	T	$
Suffix Type:	S	L	L	S	S	L	L	L	S

Inducing L-type and S-type Order Furthermore, the induced sorting step employs the fundamental concept of *Leftmost S-type (LMS) suffixes*. This notion is essential to the method's efficiency, where LMS suffixes, once systematically sorted, allow the determination of the relative order of all remaining suffixes.

An LMS suffix exhibits distinctive attributes. It is notably characterised by its position in the sequence. It is always preceded by an L-type suffix and is inherently categorised as an S-type suffix itself. Also, to classify suffixes as LMS, note that the initial suffix in a given string cannot be designated as an LMS suffix, as it is restricted to adopting either an S or an L classification. Additionally, the identity of the terminal suffix, denoted by the sentinel symbol "$", is assumed to be an LMS-type suffix in all strings.

Figure 3.3: Visual representation of the LMS suffixes of the string GTGCCTTT$

String:	G	T	G	C	C	T	T	T	$
Suffix Type:	S	L	L	S	S	L	L	L	S
LMS Suffixes:				*					*

As mentioned above, the core theorem of the SA-IS algorithm states that the sorted arrangement of LMS suffixes facilitates the process of sorting all other suffixes in linear time, which can be achieved by the following three linear scans:

1. A first reverse scan over the sorted LMS suffixes, placing them at the ends of their buckets;

   Here, the term bucket is used to delimit the interval in the SA corresponding to all the suffixes starting with the same initial character, resulting in one bucket for each alphabet character. With LMS suffixes properly arranged in their respective buckets, the placement of any newly discovered suffix in the list complies with two fundamental principles. When a new S suffix is revealed, it is positioned before all S-type suffixes in the list that share the same initial character. On the contrary, when a new L suffix is found, it is placed after all L-type suffixes with the same initial character.

   In cases where two suffixes, identified by the indices i and j, begin with the same character, and i is an L-type suffix while j is an S-type suffix, the suffix starting at position i takes precedence in lexicographic order over the one starting at position j. This theorem solidifies the idea that L-type suffixes precede S-type suffixes in the sorted sequence because the second character of L-type suffixes is smaller than the first, in contrast to S-type suffixes, where the next character is larger.

Giving rise to the following final two linear scans:

2. A forward pass is executed across the suffix array, directing L-type suffixes to the forefront of their designated buckets.

Algorithm 3.1: Induced Sorting of L Suffixes

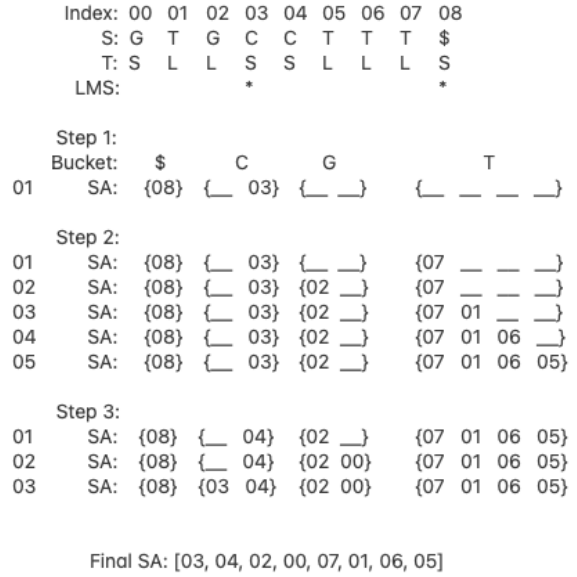
```
begin
    for each index i in SA do
        if SA[i] > 0 and SA[i] − 1 is L-type then
            put SA[i] − 1 at the next free slot at the front of the bucket S[SA[i]−1]
```

3. A subsequent reverse pass is carried out over the suffix array, placing S-type suffixes at the conclusions of their designated buckets, all the while ensuring the appropriate resetting of end positions for each bucket.

Algorithm 3.2: Induced Sorting of S Suffixes

```
begin
    for each index i in SA, starting from the end do
        if SA[i] > 0 and SA[i] − 1 is S-type then
            put SA[i] − 1 at the next free slot at the end of the bucket S[SA[i]−1]
```

Figure 3.4: Example of the induced sorting algorithm in practice.



3.1.1.B Circular SA-IS Algorithm

In order to generate a circular suffix array, certain adjustments were imperative, in addition to the absence of the terminator character $'\$'$. These modifications increase the number of steps and cast additional complexity to the algorithm. However, it is worth noting that their impact is generally irrelevant in practical scenarios.

To employ the SA-IS algorithm while simultaneously constructing a circular suffix array, two prerequisites must be satisfied. Initially, this new algorithm can only be applied to sequences ending in an L-type suffix, meaning that the terminal character in the sequence must be lexicographically larger than the value of the sequence's initial element. However, in the context of circular sequences, this condition poses no impediment since they can easily be rotated to achieve this prerequisite. Using the algorithm 3.3 with the text $TGCCTTTG$, this preprocessing step results in the string $GTGCCTTT$.

Algorithm 3.3: Preprocessing step to rotate the sequence and make it end on a L-Type suffix

```
begin
    for each character c in the alphabet Σ, starting from the lexicographical bigger character do
        for each index i in S, starting from the end do
            if S[i] == c and S[i] > S[i − 1] then
                rotate sequence to end in S[i];
```

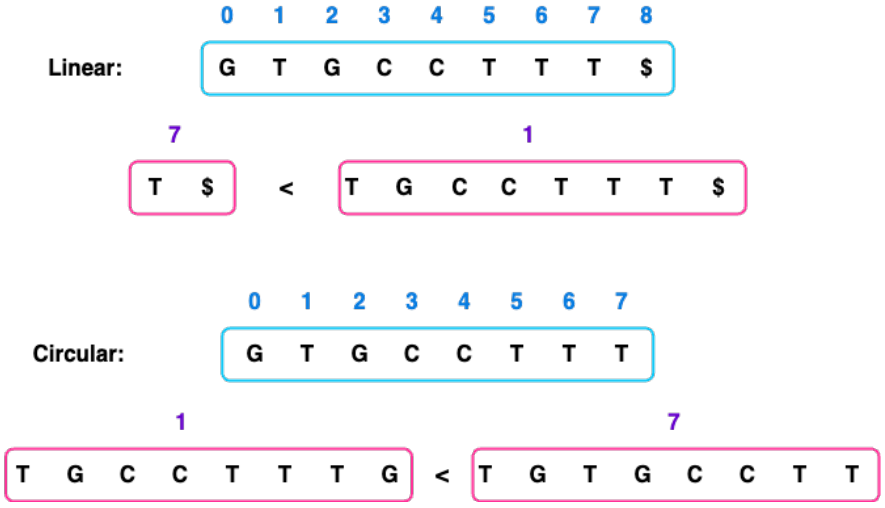
Subsequent to this sequence preprocessing, it is imperative to acknowledge an adjustment in the algorithm. In the original configuration, from a lexicographical perspective, the algorithm considers the final suffix to be the smallest. However, in the context of circular cases, this assumption is not always true, as visually elucidated in Figure 3.5, which depicts such sequences.

Henceforth, to establish the ordering of L-type suffixes, an additional step is introduced to verify and ensure the accountability of the circular nature of this type of sequences, as exemplified in Algorithm 3.4.

Algorithm 3.4: Extra sorting step of L-type suffixes

```
begin
    for each index i in the last bucket of SA, starting from the end do
        for each element j of the bucket after i do
            if suffix(S[SA[j]]) < suffix(S[SA[j + 1]]) then
                break;
            swap SA[j] with SA[j+1]
```

Figure 3.5: Example showing that the last suffix in circular sequences is not necessarily the smallest.



This extra consideration does not disrupt the operation of the induction step of the S-type suffixes, thus maintaining the assurance of accurate ordering, as they emanate from the correct arrangement of the L-Type suffixes. Figure 3.6 provides a visual representation of the algorithm in action following these modifications.

Figure 3.6: Example of the circular induced sorting algorithm in practice.

```
      Index: 00 01 02 03 04 05 06 07 08
          S: G  T  G  C  C  T  T  T  $
          T: S  L  L  S  S  L  L  L  S
       LMS:          *              *

      Step 1:
     Bucket:  $     C     G           T
01     SA:  {08} {__ 03} {__ __}   {__ __ __ __}

      Step 2:
01     SA:  {08} {__ 03} {__ __}   {07 __ __ __}
02     SA:  {08} {__ 03} {02 __}   {07 __ __ __}
03     SA:  {08} {__ 03} {02 __}   {07 01 __ __}
04     SA:  {08} {__ 03} {02 __}   {07 01 06 __}
05     SA:  {08} {__ 03} {02 __}   {07 01 06 05}

      Step 3:
01     SA:  {08} {__ 03} {02 __}   {01 07 06 05}

      Step 4:
01     SA:  {08} {__ 04} {02 __}   {01 07 06 05}
02     SA:  {08} {__ 04} {02 00}   {01 07 06 05}
03     SA:  {08} {03 04} {02 00}   {01 07 06 05}

Final SA: [03, 04, 02, 00, 01, 07, 06, 05]
```

3.1.2 BWT

The circular BWT also omits the terminator character of the transform, avoiding complications arising from circularity, but maintaining the core algorithms of FM-Index unaffected and retaining their efficiency. An example is shown in Figures 3.7 and 3.8, which depicts both the regular and circular version without the terminator symbol for the text $TGCCTTTG\$$. The BWT strings for both versions are $TGCT\$TGTC$ and $GCTTGTTC$, respectively.

Figure 3.7: Visual representation of the regular suffix array to better understand the concept of BWT.

F								L
$	G	T	G	C	C	T	T	T
C	C	T	T	T	$	G	T	G
C	T	T	T	$	G	T	G	C
G	C	C	T	T	T	$	G	T
G	T	G	C	C	T	T	T	$
T	$	G	T	G	C	C	T	T
T	G	C	C	T	T	T	$	G
T	T	$	G	T	G	C	C	T
T	T	T	$	G	T	G	C	C

Figure 3.8: Visual representation of the circular suffix array to better understand the concept of circular BWT.

F							L
C	C	T	T	T	G	T	G
C	T	T	T	G	T	G	C
G	C	C	T	T	T	G	T
G	T	G	C	C	T	T	T
T	G	C	C	T	T	T	G
T	G	T	G	C	C	T	T
T	T	G	T	G	C	C	T
T	T	T	G	T	G	C	C

3.1.3 FM-Index

In this section, an illustrative example is presented to underline the heightened efficiency of the FM-Index in the context of circular pattern detection, a hitherto unaddressed challenge. The example uses a sequence labelled $S = GTGCCTTT\$$ and a pattern designated as $P = TGT$. This serves as a compelling demonstration of the adaptability and effectiveness of the FM-Index when applied to novel circular data structures, encapsulated in a set of figures referenced as 3.9, 3.10, and 3.11.

In its conventional configuration, the FM-Index uses a backward search strategy to locate a specific pattern, and for the circular pattern search that is no exception. For the pattern P, the algorithm starts by seeking the occurrence of the character T (the last) within the occurrence table, between the green interval that spans from the first position to the last position, as T represents the first character in the pattern search. In this specific case, it becomes evident that the letter T manifests itself on four occasions within the terminal row of the occurrence table. Consequently, an inspection of the column F, highlighted in yellow, reveals four distinct occurrences of the letter T, effectively narrowing the search interval for the subsequent character.

Figure 3.9: Visual representation of the LF-Mapping step for the first character of the backwards pattern search, corresponding to the character T of pattern TGT in the example.

F							L	A	C	G	T
C	C	T	T	T	G	T	G	0	0	1	0
C	T	T	T	G	T	G	C	0	1	1	0
G	C	C	T	T	T	G	T	0	1	1	1
G	T	G	C	C	T	T	T	0	1	1	2
T	G	C	C	T	T	T	G	0	1	2	2
T	G	T	G	C	C	T	T	0	1	2	3
T	T	G	T	G	C	C	T	0	1	2	4
T	T	T	G	T	G	C	C	0	2	2	4

To search for the next character of the pattern (G), the examination is restricted to the occurrence table within the interval defined by the first and last positions of T in column F. As depicted in green in Figure 3.10, this interval starts in row 5 and ends in row 8. The letter G appears only once within this interval, being the second occurrence of that character in the BWT which, through the LF-Mapping procedure, also corresponds to the second G of the F column. When inspecting its position in column F, highlighted in yellow, the presence of the pattern GT can be identified.

Figure 3.10: Visual representation of the LF-Mapping step for the second character of the backwards pattern search, the pattern GT of TGT.

F							L	A	C	G	T
C	C	T	T	T	G	T	G	0	0	1	0
C	T	T	T	G	T	G	C	0	1	1	0
G	C	C	T	T	T	G	T	0	1	1	1
G	T	G	C	C	T	T	T	0	1	1	2
T	G	C	C	T	T	T	G	0	1	2	2
T	G	T	G	C	C	T	T	0	1	2	3
T	T	G	T	G	C	C	T	0	1	2	4
T	T	T	G	T	G	C	C	0	2	2	4

Now, the circular distinctive feature enters into play, since the search when using the non-circular version of the FM-Index, would have terminated because it would not be possible to find the following character to the left, leading to the conclusion that only the pattern GT could be identified in the sequence. However, due to the incorporation of a circular BWT, this impediment is overcome, allowing the discovery of the pattern TGT. Consequently, within the occurrence table and within the green interval where the pattern GT is located in Figure 3.11, the search continues to find the occurrence of the subsequent letter (T). A single occurrence is found, the second T in column F. As can be seen in the yellow highlighted section, it becomes evident that the pattern TGT is present in the circular suffix $TGTGCCTT$, beginning before the last position of the sequence and returning to the sequence's inception, thereby making possible the identification of pattern P.

Figure 3.11: Visual representation of the LF-Mapping step for the third character of the backwards pattern search, the pattern TGT of TGT.

F							L	A	C	G	T
C	C	T	T	T	G	T	G	0	0	1	0
C	T	T	T	G	T	G	C	0	1	1	0
G	C	C	T	T	T	G	T	0	1	1	1
G	T	G	C	C	T	T	T	0	1	1	2
T	G	C	C	T	T	T	G	0	1	2	2
T	G	T	G	C	C	T	T	0	1	2	3
T	T	G	T	G	C	C	T	0	1	2	4
T	T	T	G	T	G	C	C	0	2	2	4

3.2 Finding Maximal Exact Matches

SlaMEM [58] is the groundwork tool responsible for the MEM search step that allows CSA-MEM to find the longest chain of nucleotides common to all sequences in the dataset. Its main algorithm is based on a method outlined in the work developed by Ohlebusch et al. [70] and relies on the backward matching concept of the FM-Index. The algorithm presented has no changes except when searching for the parent intervals, which sometimes needs to extend the interval beyond the linear limits of the sequence with the help of the circular BWT.

Algorithm 3.5: MEM searching

```
begin
    (l, [i, j]) = (0, [0, n − 1]);
    for each index k in n, starting from the end do
        [nexti, nextj] = BackwardSearchStep(P[k], [i, j]);
        while [nexti, nextj] == [] do
            l − [i, j] = Parent_c(l − [i, j]);
            [nexti, nextj] = BackwardSearchStep(P[k], [i, j]);
        (l, [i, j]) = (l + 1, [nexti, nextj]);
        (l′, [i′, j′]) = (l, [i, j]);
        (previ′, prevj′) = (j′, j′ + 1);
        while l′ ≥ minLength do
            while previ′ ≥ i′ do
                if (k == 0) or BWT_c[previ′ ≠ P[k − 1]] then
                    output (l′, SA_c[previ′], k);
                previ′ = previ′ − 1;
            while prevj′ ≤ j′ do
                if (k == 0) or BWT_c[prevj′ ≠ P[k − 1]] then
                    output (l′, SA_c[prevj′], k);
                prevj′ = prevj′ + 1;
            l′ − [i′, j′] = Parent(l′ − [i′, j′]);
```

The core operational principle of slaMEM, Algorithm 3.5, involves the backward processing of each query sequence, as required by the FM-Index. In the search for patterns, the algorithm identifies each LF-Mapping interval $[i, j]$ while simultaneously tracking back their parent intervals $[i', j']$ until the current size of that match falls below a specified minimum threshold. During this process of tracing parent intervals, the algorithm also examines the potential for pattern extension in both the left and right directions.

The algorithm ensures the correct output of the maximal extension to the right through the steps on the first while (lines 4-6), invoking a leftward contraction when the pattern is not found, thus ensuring left-maximality, by inspecting in the parent intervals for matches that do not possess the currently searched character on the second set of while loops (10-19).

The runtime complexity of slaMEM for the circular MEM search process remains the same, which is $O(m + R_L + M_L * t_{SAc})$, where m represents the length of the query, R_L and M_L correspond to the number of right maximal matches and MEMs, respectively, each with a minimum size of L, and t_{SAc} represents the time needed to obtain a value from the SA, in this case the circular SA, which is constant using the FM-Index.

3.2.1 Shared MEMs among all sequences

The output from slaMEM does not guarantee that all the gathered MEMs are common to all sequences because it compares one query with the reference at a time. However, for CSA-MEM, only the MEMs that are common to the entire set are considered relevant. To address this, a data structure has been integrated into the tool, a list of the same length as the reference string. Each element in this list corresponds to a character in the reference, and to solve the issue, the query number of the sequence currently under examination is marked as a reference at the equivalent position where a MEM is found in the reference string. The condition for marking is that this position must have been previously marked by the query that was searched before.

As a result, whenever a new query is analysed, the algorithm will store MEMs that are common to the previous query. By conducting two consecutive searches on the sequences, the results of the second pass will only produce MEMs that are common to all queries, as can be seen in Figure 3.12.

Figure 3.12: Visual representation of the shared MEM data structure in action for the reference string $S = ACGTT$ and the queries $q1 = CGTT$ and $q2 = GT$.

							A	C	G	T	T	
1:	q1	C	G	T	T	$\rightarrow$	0	1	1	1	1	MEM : CGTT
2:	q2			G	T	$\rightarrow$	0	1	2	2	1	MEM : GT
3:	q1	C	G	T	T	$\rightarrow$	0	1	3	3	1	MEM : GT
4:	q2			G	T	$\rightarrow$	0	1	4	4	1	MEM : GT

In this example, there are two queries, making it necessary to perform four iterations to identify the common MEMs across all sequences. The initial iteration compares the query $q1$ with the reference S, leading to the identification of the MEM $CGTT$. In response, the new list is marked with the number 1 at all positions corresponding to the MEM's location in the reference.

Subsequently, in the second iteration, the query $q2$ is compared to the reference S, resulting in the discovery of the MEM GT. Given that the equivalent positions in the new list are already marked with the previous query's number (1), this MEM can also be marked as common because it appears in both sequences, now designated with the number 2. However, as we have distinct MEMs in query 1 ($CGTT$) and query 2 (GT), it is evident that the MEM from query 1 is not common with query 2. Consequently, an additional pass across all queries is required to trim the MEMs to ensure their full commonality.

In the third iteration, the query 1 is re-evaluated, and the MEM $CGTT$ is encountered once more. However, this time, it is not entirely retained. Instead, in the reference positions of the new list, only the pattern GT is retained, as it is associated with the number 2 from the previous query, signifying its commonality with $q2$

As a result, the MEMs from the last two iterations are outputed, presenting all the common MEMs, as highlighted in yellow in the figure.

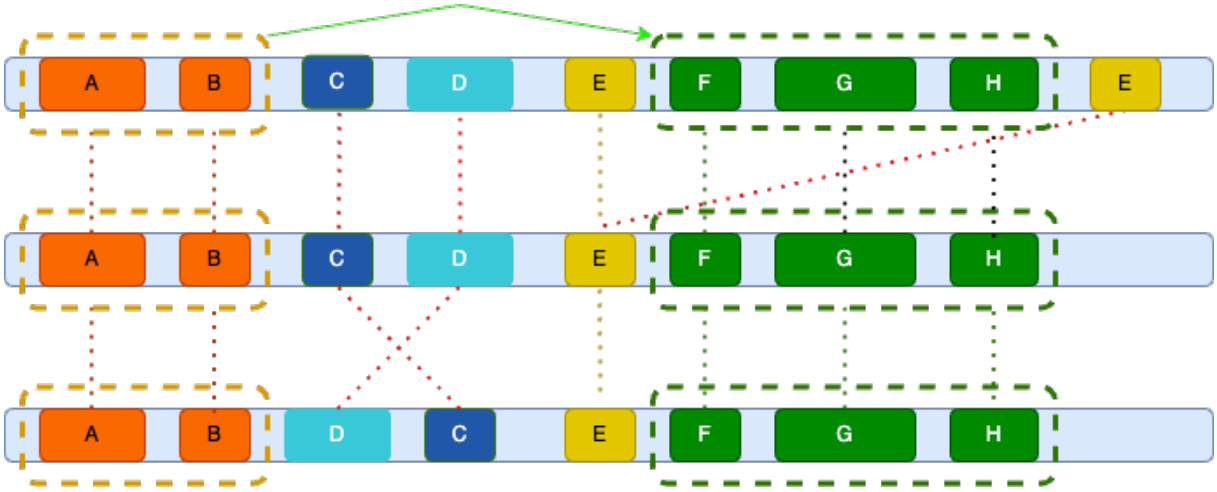
3.3 Most Significant Common Subsequence Chain of MEMs

As proven in the CSA paper by Fernandes et al. [1], one key technique to determine an optimal rotation to align a set of sequences involves identifying the most significant chain of shared nucleotides among it. This is crucial, as it denotes the most informative region universally present across all of them.

To achieve this, it is necessary to identify the longest subsequence of characters common to all target sequences earmarked for rotation. Executing this task allows the identification of a common segmentation point where a cut, if made, would enhance multiple sequence alignment scores, contrasting against a chosen arbitrary cut point.

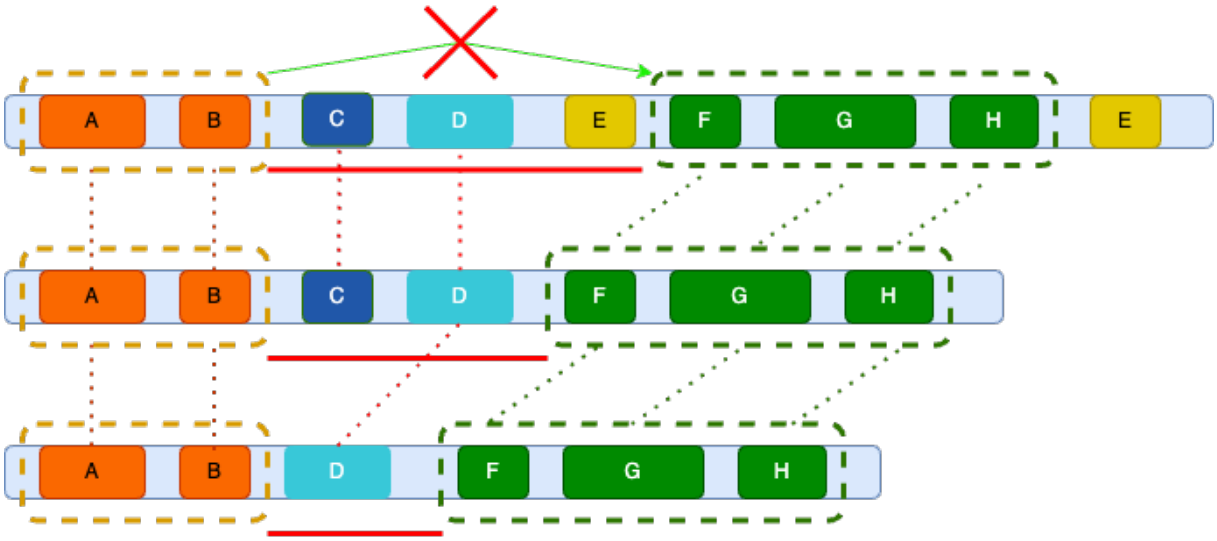
Figure 3.13 provides a visual representation that helps clarify the process of determining the optimal cut-off point for the sequences. Each block in the figure represents a common sequence of characters. The core of the algorithm is to identify the longest common subsequence among these blocks. In this instance, the beginning of the longest subsequence ($ABFGH$) is marked by the block A. Consequently, the image includes sequences starting at this point to facilitate a clearer observation of potential enhancements in the alignment.

Figure 3.13: Example of 4 block chains (A+B, C, D and F+G+H) derived from a set of longest common subsequences, from 3 sequences, resulting on ABFGH as the most significant chain.



This decision is also based on the fact that between blocks B and F, the relative distances remain consistent across all sequences. However, as shown in Figure 3.14, when this consistency is broken and the distances vary, the most favourable cutting point is moved to the position F. In this scenario, F, G, and H emerge as the most valuable common subsequences due to their equidistant blocks in all sequences.

Figure 3.14: Example of 4 block chains (A+B, C, D and F+G+H) derived from a set of longest common subsequences, from 3 sequences, resulting on FGH as the most significant chain.



The essential requirement to identify the longest common subsequence of nucleotides is improved by searching for MEMs since they constitute shorter sequences shared between two distinct strings, and because of this, they can be translated as the sequence blocks mentioned before. Each illustrated block in the figures corresponds to a visual representation of a MEM and that elucidates the concept's significance within the scope of this project.

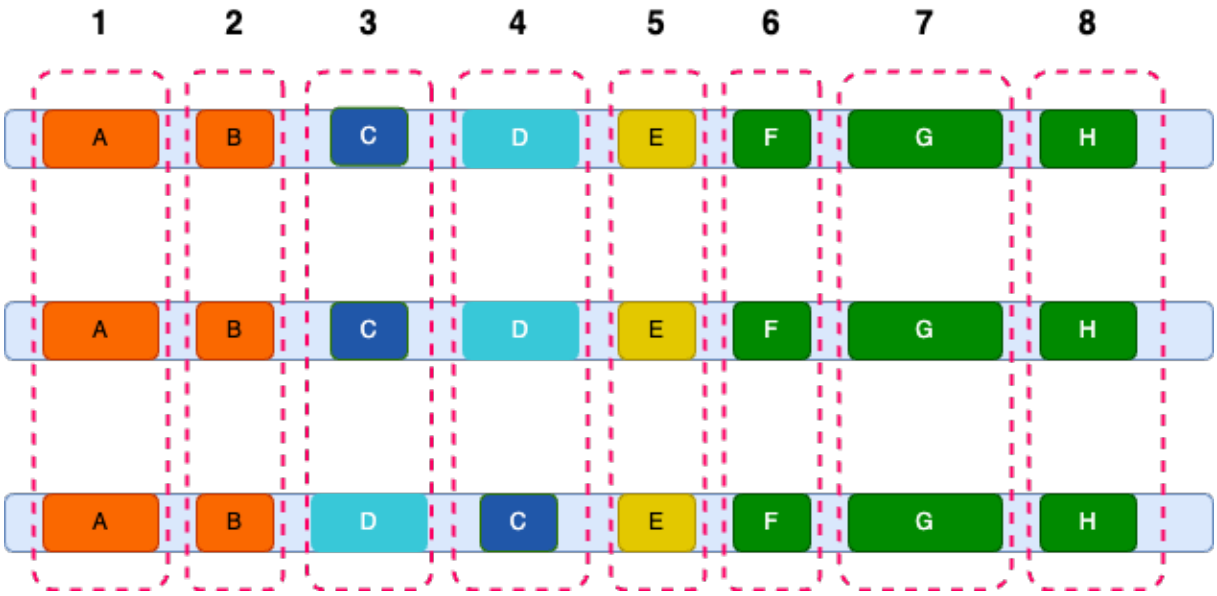
As demonstrated previously, MEMs usually contain data concerning their dimensions and initial positions within both the reference and query strings. This information facilitates the discrimination of unique MEMs, ascertaining their sequence order, and assessing the relative distance between them, all of which are prerequisites for achieving the proposed solution.

Hence, due to the availability of size data for each MEM, the problem is simplified. Attention has now shifted towards identifying the heaviest common increasing subsequence among MEMs, resulting in a faster problem resolution due to the reduction in dimensionality, transitioning from searching through complete nucleotide sequences to the number of common MEMs.

Still, considering the persistent lack of an efficient solution for identifying the heaviest common increasing subsequence within a large set of multiple elements, it is evident that attaining a perfect optimisation within a reasonable timeframe remains unfeasible. However, the possibility of an optimal rotation persists through another simplification of the problem. The crux of the matter lies in streamlining the process to focus exclusively on the identification of the heaviest increasing subsequence. This can be achieved if an equal number of unique MEMs is achieved within each sequence by eliminating coincident MEMs, thereby avoiding the presence of redundant blocks. This strategic pruning refines the focus onto more significant segments, resulting in a substantial enhancement of both the rotation's efficacy and the

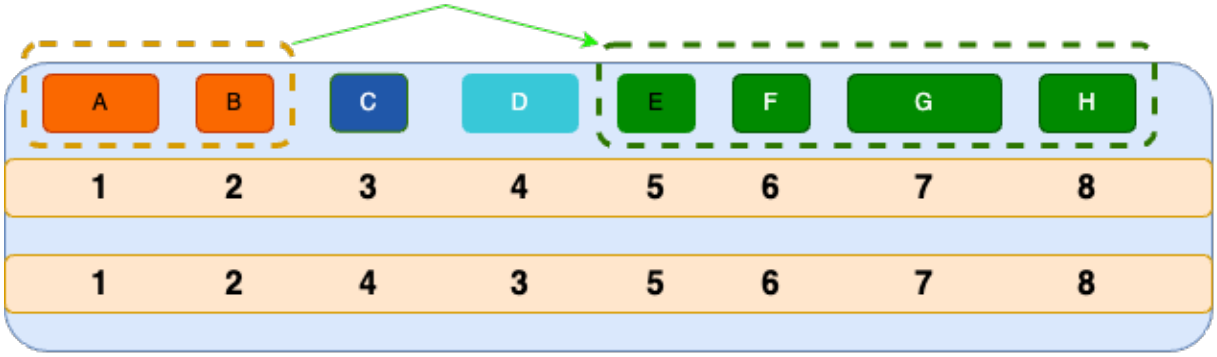
search process's efficiency. With this approach, it becomes possible to consolidate the MEM sequences into a single cohesive unit, paving the way for the search for the heaviest increasing MEM sequence, as illustrated in Figure 3.15.

Figure 3.15: The 3 sequences after the removal of the less significative non-unique MEMs.



The presence of a single unified sequence of MEMs simplifies the computational process, enabling the indirect derivation of the heaviest common increasing subsequence by applying the heaviest increasing subsequence algorithm. This is achieved by ensuring that each block initiates in ascending order, not only within the reference string, but also across all queries, as shown in the chosen subsequence of Figure 3.16. This strategy guarantees the identification of a substantial common subsequence to improve multiple sequence alignment in a computationally efficient manner.

Figure 3.16: All the sequences combined into one while storing the original query positions.



3.3.1 Heaviest Increasing Subsequence Algorithm

Given this new paradigm, the algorithm chosen to address the problem resulted from the study and understanding of the solution presented by Jacobson and Vo in their work titled "Heaviest increasing/-common subsequence problems" [71]. To identify the heaviest increasing subsequence, the article suggests that it is essential to have a sorted list. In this case, the list must be sorted by the position of each MEM in the reference string, as already mentioned and illustrated in Figure 3.16. Furthermore, it is imperative to consider an auxiliary list, denoted as L, consisting of objects of the same type that represent the end of a subsequence, while also taking into account the following operations:

- $insert(L, o)$: insert the object o into the list L
- $delete(L, o)$: delete the object o form the list L
- $next(L, o)$: find the least element strictly larger than o in L
- $prev(L, o)$: find the largest element strictly smaller than o in L
- $max(L)$: find the maximal element in L
- $min(L)$: find the minimal element in L

The authors emphasise that these operations can be executed efficiently in $O(log\ n)$ time using balanced tree data structures, where n is equivalent to the count of objects involved. It is crucial to grasp that the functions $next$, $prev$, max, and min will produce the equivalent of NULL in C when they are unable to find any element.

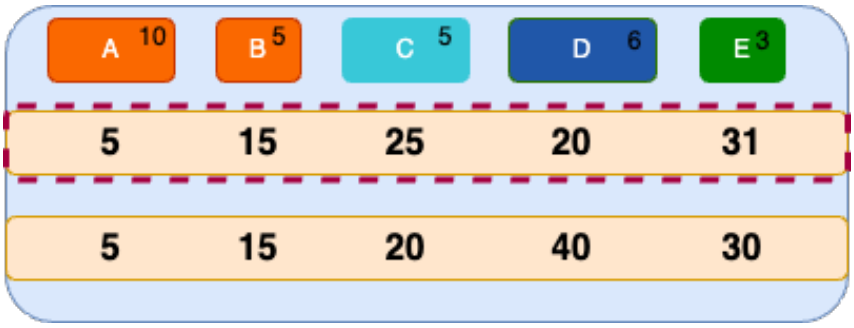
Of particular note, the functions $next$ and $prev$ do not require the argument to be previously contained within L. However, it is imperative that the argument be of the same type. For convenience, $next(L, NULL)$ corresponds to $min(L)$, and $prev(L, NULL)$ is equivalent to $max(L)$.

The concept of the Heaviest Increasing Subsequence (HIS) stems from the development of the algorithm chosen to compute the longest increasing subsequence LIS. However, for the sake of this thesis's simplicity, only the HIS algorithm will be discussed in detail. To gain a better understanding of the algorithm, it is recommended reading the article to understand how the longest increasing subsequence is calculated.

Algorithm 3.6: HIS($\sigma_1\sigma_2...\sigma_n$, weight function Ω)

begin
- $L = NULL$;
- **for each** index i **in** n **do**
  - $(s, v) = prev(L, (\sigma_i, 0))$;
  - $(t, w) = next(L, (s, v))$;
  - **while** $(t, w)! = NULL$ **do**
    - **if** $v + \Omega(i, \sigma_i) < w$ **then**
      - $break$;
    - $delete(L, (t, w))$;
    - $(t, w) = next(L, (t, w))$;
  - **if** $(t, w) == NULL || \sigma_i < t$ **then**
    - $insert(L, (\sigma_i, v + \Omega(i, \sigma_i)))$;
    - $node[\sigma_i] = newnode(\sigma_i, node[s])$;

Figure 3.17: New chain of MEMs.



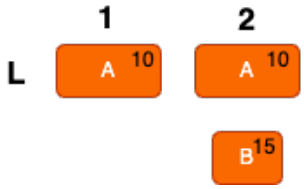
The algorithm, as presented in Algorithm 3.6, is illustrated by the use of a new instance of a MEM sequence. Figure 3.17 shows the new chain of MEMs, including their respective sizes within each block. This subsection is dedicated to providing insight into the operation of the heaviest increasing subsequence algorithm within the realm of MEMs. Consequently, it is imperative to emphasise that the computation exclusively considers the query positions of the initial query sequence, as surrounded by red dashed lines in Figure 3.17.

1. The algorithm starts by adding the first element of the MEM sequence to the list denoted as L.

Figure 3.18: Representation of list L after the first iteration.

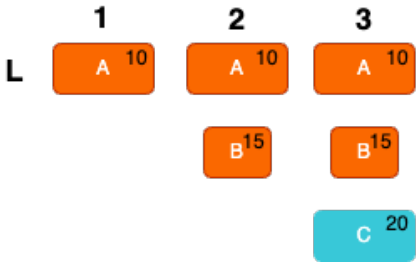
2. Then, since the following element appears next in the query, it should be appended to the list L after the first element, with a total weight of 15 in this case.

Figure 3.19: Representation of list L after the second iteration.



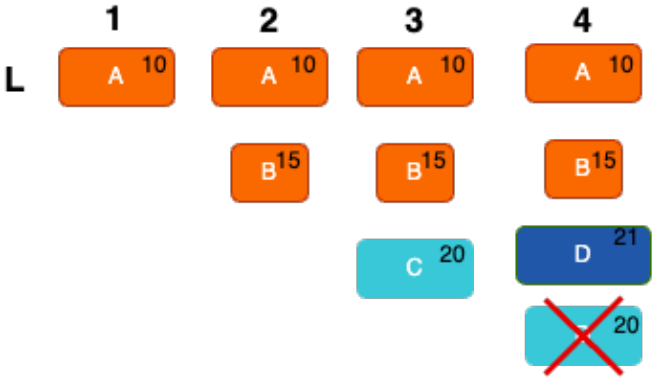
3. The subsequent Mem C also follows B in the query, so it is appended to L at the last position, forming a new subsequence (ABC) with a total sum of 20.

Figure 3.20: Representation of list L after the third iteration.



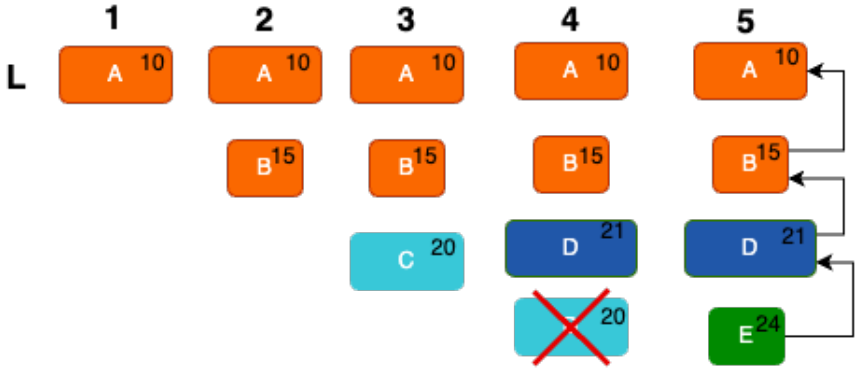
4. Now, the key feature of the algorithm comes into action. The next MEM in the reference string is the block D, which comes before C in the query. Therefore, D is added to the list L after block B, not after block C, forming the subsequence ABD, not $ABCD$. This new subsequence has a total weight of 21, which is larger than the previous subsequence, meaning that the subsequence that ends in C can be deleted, since every MEM that follows will also follow D, joining that subsequence, since ABD has a sum of 21 and ABC only has a sum of 20.

Figure 3.21: Representation of list L after the fourth iteration.



5. Subsequently, the search continues taking into account these conditions, but now with a smaller list since purposeless subsequences are now not considered, improving the time complexity of the search.

Figure 3.22: Representation of list L after the final iteration.



6. The final result of the search for this example is $ABDE$. This result is obtained by searching for the links of the last block of the list L, since it is the end of the heaviest subsequence.

3.3.2 CSA-MEM Heaviest Common Increasing Subsequence Algorithm

After conducting some research, it became clear that the algorithm discussed in the previous subsection cannot be used in its entirety to identify the heaviest common increasing subsequence of MEMs. This limitation arises from the fact that the algorithm disregards certain subsequences of smaller sizes. The issue at hand pertains to the fact that some MEMs within certain sequences may be contiguous, while in others, they are not, so the algorithm should not discard any subsequence. For illustration, by referring to Figure 3.17, it becomes evident that in the first query, element C would be omitted, while in the other query it would not. Furthermore, in the event of finding the common heaviest increasing subsequence, its exclusion leads to an erroneous outcome, since C belongs to the heaviest common increasing subsequence $ABCE$. This happens because the block E follows D in the first query, but not in the second.

In order to reach Algorithm 3.7 and address this issue, it became necessary to remove the feature that involves the removal of subsequences, thereby slightly compromising the time complexity of the algorithm. Additionally, the following operations had to be modified:

- $next(L, o)$: Identify the smallest element in the list L that is strictly greater than o, considering its weight.
- $prev(L, o)$: Locate the largest element in the list L that is strictly smaller than o in terms of query position, ensuring that the identified object has all query positions smaller than the input and similar

distances between them.

- $insert(L, o)$: Incorporate the object o into the list L according to the criterion determined by $next(L, o)$.

Algorithm 3.7: HCIS($\sigma_1\sigma_2...\sigma_n$, weight function Ω)

begin

$L = NULL$;

for each index i **in** n **do**

$(s, v) = prev(L, (\sigma_i, 0))$;

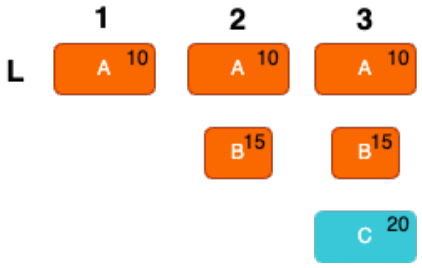
$(t, w) = next(L, (s, v))$;

$insert(L, (\sigma_i, v + \Omega(i, \sigma_i)))$;

$node[\sigma_i] = newnode(\sigma_i, node[s])$;

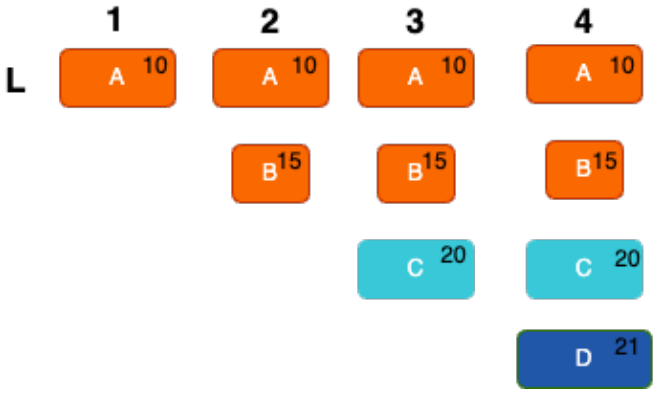
1. In the same example as in the HIS subsection, the algorithm proceeds to populate and establish connections within the L list in a manner consistent with the previous procedure, up to the examination of the fourth MEM, as visually represented in Fig. 3.23.

Figure 3.23: Representation of list L after the third iteration.



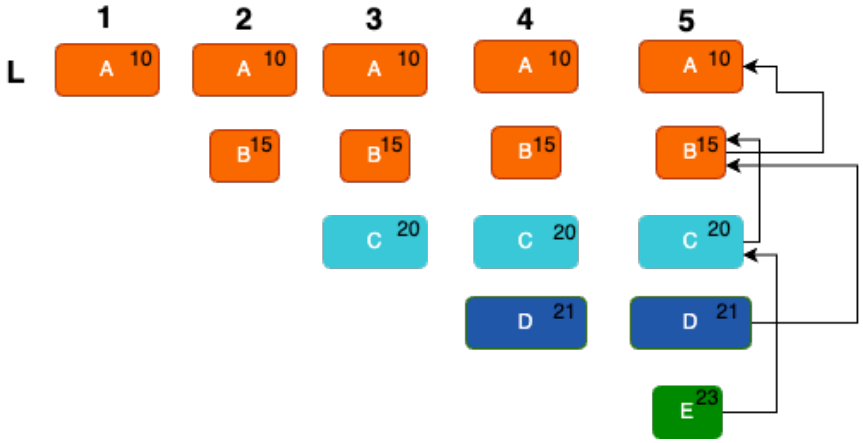
2. However, in the current scenario, the block D is also appended to the end of the List, given that its cumulative sum of weights exceeds the sum of block C. In particular, it remains exclusively related to block B, thus constituting the subsequence ABD. This approach ensures that the blocks maintain an ordered arrangement based on their significance, without eliminating any potentially valuable components for future reference. It is pertinent to emphasise that linkages are only established when they meet the validity criteria, requiring consecutive query positions with consistent interposition distances. In instances where the dataset exhibits a uniform inter-block distance of 10 for all queries, except for a singular query with a distance of 1000, a connection will not be established. This is due to the potential introduction of numerous gaps during sequence alignment.

Figure 3.24: Representation of list L after the fourth iteration, adding the new heaviest block after the second heaviest block and not removing C.



3. The same happens to block E, which does not connect with D but connects with C. Using the regular algorithm, C would have been removed from L before, becoming unrecognisable at this stage, resulting in loss of information. The subsequence ABD would have been heavier than ABE. But when applying the necessary changes to the algorithm to consider the block C, $ABCE$ actually becomes the heaviest common increasing subsequence.

Figure 3.25: Representation of list L after the fifth iteration, resulting in the heaviest common increasing subsequence $ABCE$.



4. For the heaviest common increasing subsequence, the final result of the search for this example is $ABCE$ with a total weight of 23.

4

Results and Discussion

Contents

This section delves into an exploration and analysis of CSA-MEM's performance results. It is crucial to recognise that some tools have previously established very good state-of-the-art approaches for enhancing the alignment of circular multiple sequences. However, CSA-MEM goes beyond this by overcoming the limitations associated with dealing with a large number of sequences and sizable data.

The content in this section includes not only the results of the comparisons using CSA-MEM but also a comprehensive explanation of the reasoning that led to choosing specific details of the methodology. The comprehensive explanation is followed by a thorough assessment of CSA-MEM's capabilities, involving a detailed evaluation of its performance across various scenarios. And, at the end of the section, the comparative analysis is conducted in which CSA-MEM is measured against other advanced circular preprocessing alignment tools. Ultimately, this comprehensive evaluation sheds light on the effectiveness of CSA-MEM and its potential impact within the broader scientific community.

4.1 Evaluation Methodology

The selected evaluation methodology is designed to comprehensively assess the performance of the new algorithm compared to existing solutions and pinpoint areas that need improvement. Various metrics were used to gain insight into the algorithms capacity of handling different datasets, with a special focus on memory usage and time efficiency. To guarantee a fair and standardised comparison, a detailed description of the hardware used for the test of the solution is also provided in Section 4.1.2.

4.1.1 Metrics

To conduct a thorough performance evaluation of the solutions, the following metrics were evaluated:

- **Running time**: Measuring the time required to analyse a dataset is a critical factor in evaluating the program's effectiveness and its ability to preprocess extensive sequences.
- **Space consumption**: Quantifying the required storage space for accommodating the program's results significantly influences the assessment of its capacity to rotate more extensive DNA sequences.
- **Memory consumption**: Assessing the program's operational RAM memory requirements constitutes a critical parameter that impacts the understanding of both the solution's efficiency and resource demands.
- **Annotation quality**: Evaluating the alignment accuracy with respect to proper references and construction of accurate phylogenetic relationships provides a more in-depth overview of the program's accuracy.

4.1.2 Hardware

The system was carefully chosen having in mind the compliance with the mentioned criteria to make sure that the solution can process large datasets efficiently and deliver reliable and standardised results of its performance:

- **Processor**: The system is powered by a powerful Intel (R) Xeon(R) Silver 4214R CPU @ 2.40GHz of 40 cores.
- **Capacity**: The machine has 256GB of RAM, which is enough memory to allow the analysis of the datasets in hands.
- **Enviroment**: The experiment runs on the Ubuntu Linux 20.04.4 LTS x86_64 distribution, ensuring a stable and consistent platform for all experiments.

4.2 FM-Index variants explored

Before the final decision of using slaMEM and due to the advantage of using the FM-Index algorithm to find common patterns between sequences, it was necessary to test and decide if any other variation would perform better when analysing large metagenomic datasets. Because of this, during this thesis, a collaboration was established with the authors of the BR-Index (professor Sadakane from the University of Tokyo) [51], who have provided access to the open source version of the BR-Index algorithm and an unpublished version of it, the full-BR-Index. To understand the potential of these tools, a comparative analysis between them was performed using seven real DNA sequences, as shown in Table 4.1.

Table 4.1: Datasets used to perform the index comparison.

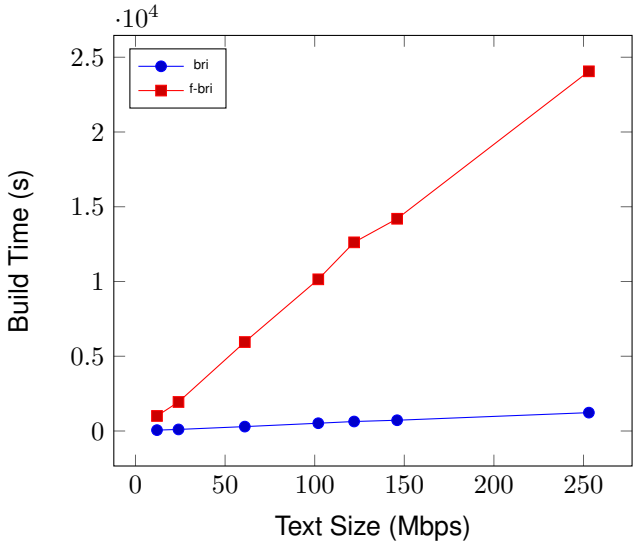
Genome	**Text size** (Mbps)
Saccharomyces cerevisiae	12
Plasmodium falciparum	24
Danio rerio (Chrom.1)	61
Caenorhabditis elegans	102
Arabidopsis thaliana	122
Drosophila melanogaster	146
Homo sapiens (Chrom.1)	253

Figure 4.1 demonstrates the relationship between building time and text size for BR-Index and full-BR-Index. It can be seen that as the text size increases, the building time for both algorithms also increases as expected. However, when comparing the two algorithms, it is revealed that the full-BR-Index has a construction time consistently higher than the BR-Index. This suggests that, while both algorithms exhibit a similar trend in building time with respect to text size, full-BR-Index may have the worst overall performance.

Additionally, following Figure 4.2, it is evident that the full-BR-Index always needs more disk space than the BR-Index. It is necessary because the second technique employs more data structures to make the search faster. The necessity for more space is not a problem when processing shorter sequences, because the increase in index space is not noticeably worse, but when dealing with larger datasets, BR-Index would be a more realistic option in terms of resources needed to store these indexed sequences.

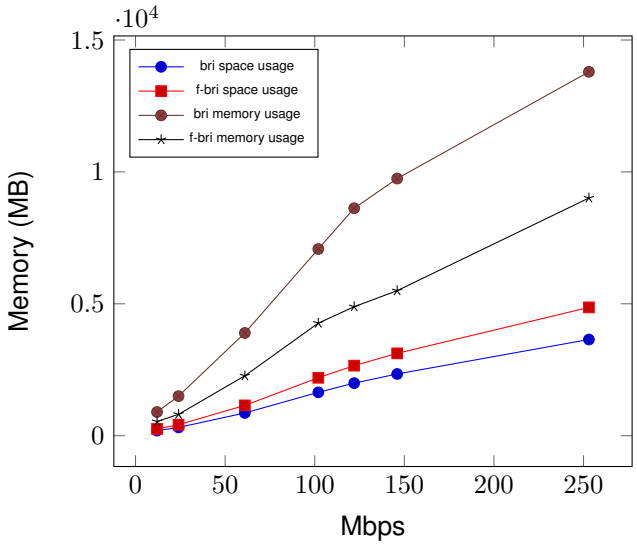
However, on the other hand, the full-BR-Index uses less volatile memory to build the index, making it more memory efficient, which can be a really strong advantage. For example, the first approach requires a maximum of 13794 MB of memory when the index space is 3647 MB, whereas the second algorithm uses 4864 MB of disk space and 9011 MB of memory. With increasing index space, the memory requirements of the two methods diverge more and more.

Figure 4.1: Comparison of time performance of both index algorithms.



Nevertheless, this improvement in building memory is still insufficient for processing large metagenomic datasets, which can reach sizes on the order of many gigabytes. This problem suggests that additional optimisation in terms of index building algorithms or a different approach may be necessary to process these sequences effectively.

Figure 4.2: Comparison of space and memory of the both index algorithms.



Due to this reason and as it was mentioned before, the algorithm chosen to perform the pattern matching task ended up being slaMEM, which had already been proven to be suitable for it [58] and also because no extraordinary modifications were necessary to it, except for the use of the circular suffix array and BWT.

4.3 Final Results

4.3.1 Datasets

To ascertain the effectiveness of the approach, a collection of real-world metagenomic data was assembled to test and compare the performance of the final solution with the current state of the art. The resulting information provides a more comprehensive evaluation of the solution's capabilities in solving the problem, especially with regard to larger sequences.

The tests were carried out on five distinct datasets. The first three sets contained mitochondrial DNA (mtDNA) for 16 primates, 12 mammals, and a more diverse set of 19 sequences. This third set is composed of a mixture between the 16 primates and the addition of 3 more distant evolutionary species (Drosophila melanogaster, Gallus gallus, and Crocodylus niloticus). The fourth set contains viral DNA in the form of circular genomes of 9 viruses recently discovered in a small metagenomics study [72]. The fifth set was used to evaluate the tool's efficiency on extensive datasets, in this case consisting of 35 sequences of Escherichia coli (E. coli). The first two datasets were previously used to benchmark CSA, BEAR and MARS in their respective works. The third dataset was also used in both CSA and BEAR. Some statistics for each of these datasets are available in Table 4.2.

Table 4.2: General properties of each one of the datasets used in the benchmarks.

Dataset	Number of sequences	Average size (bp)	Total size (KB)
Mammals	12	16,777	204
Primates	16	16,581	269
Diverse	19	16,759	323
Viruses	9	3,130	29
E. coli	35	5,703,382	190,370

4.3.2 Alignment Quality

The quality of the rotations was assessed using a linear multiple sequence alignment process, feeding the rotated sequences into ClustalW to obtain the alignment score. An example of the multiple sequence alignment produced by ClustalW on one of the datasets, before and after rotation with CSA-MEM, is presented in Fig. 4.3 and in Fig. 4.4 respectively. The gaps are represented in blue, and alignment conservation increases from green to red. The sequence start and end positions meet at the top centre of their circular representations.

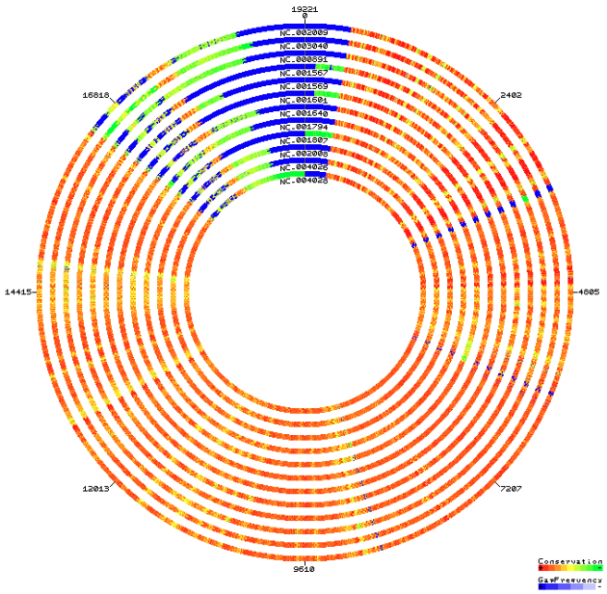


Figure 4.3: Circular representation of the multiple sequence alignment outcome for the mammalian dataset before rotating the sequences with CSA-MEM.

The gaps and unmatched portions at both loose ends of the unrotated sequences are clearly visible in the first image, after which they fade away and fit together to produce a much more meaningful alignment after the sequences are rotated, in the second image proving how beneficial it is to use CSA-MEM to preprocess the sequences before multiple circular sequence alignment.

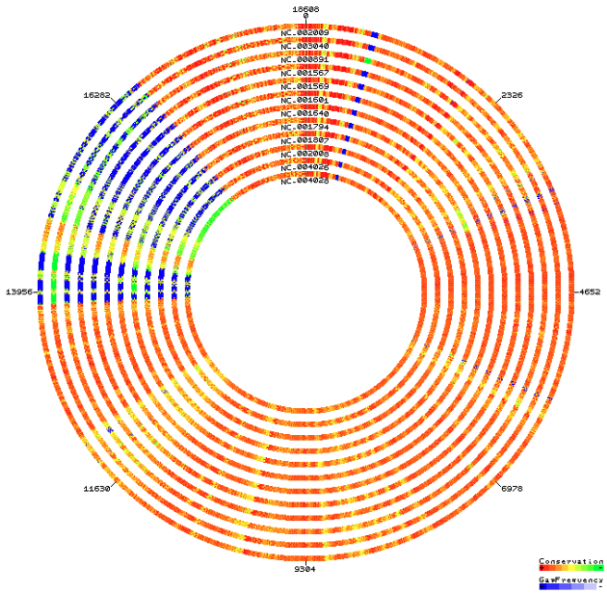
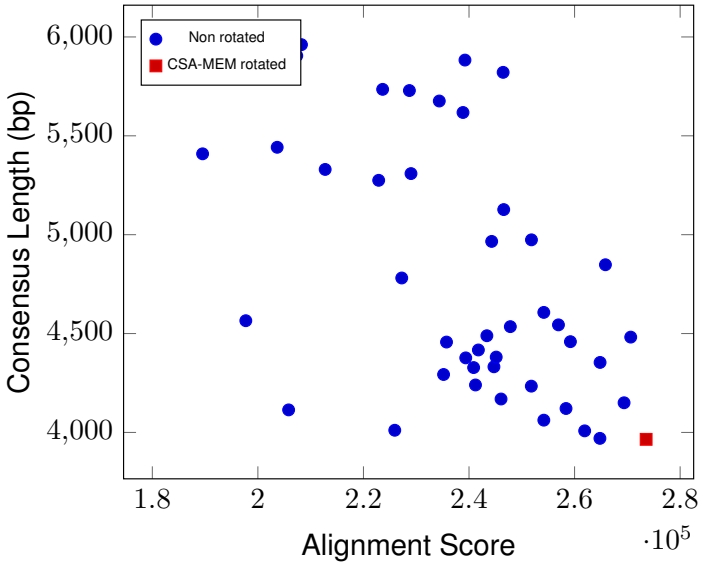


Figure 4.4: Circular representation of the multiple sequence alignment outcome for the mammalian dataset after rotating the sequences with CSA-MEM.

In order to get a sense of the improvement in statistical significance relative to a random situation, tests were also carried out with 50 sets of control sequences with random cuts (using the set of viruses described before), as was done with CSA in its original paper, but with different data. For these sets, the alignment scores and consensus length obtained from the random rotations were compared to the improved CSA-MEM rotation.

The enhanced efficiency of CSA-MEM once again became evident. When used with these sets, all consistently produce the same initial cutting rotation, which significantly outperforms the scores of the random test sets. In Figure 4.5, the graph illustrates the relationship between the consensus sequence length in base pairs (bp) and the alignment score for the 50 test sets using ClustalW for multiple sequence alignment, with the scores remaining constant when CSA-MEM was used as a preprocessing step and always better than random rotations.

Figure 4.5: Comparison between 50 random rotations against CSA-MEM rotation.

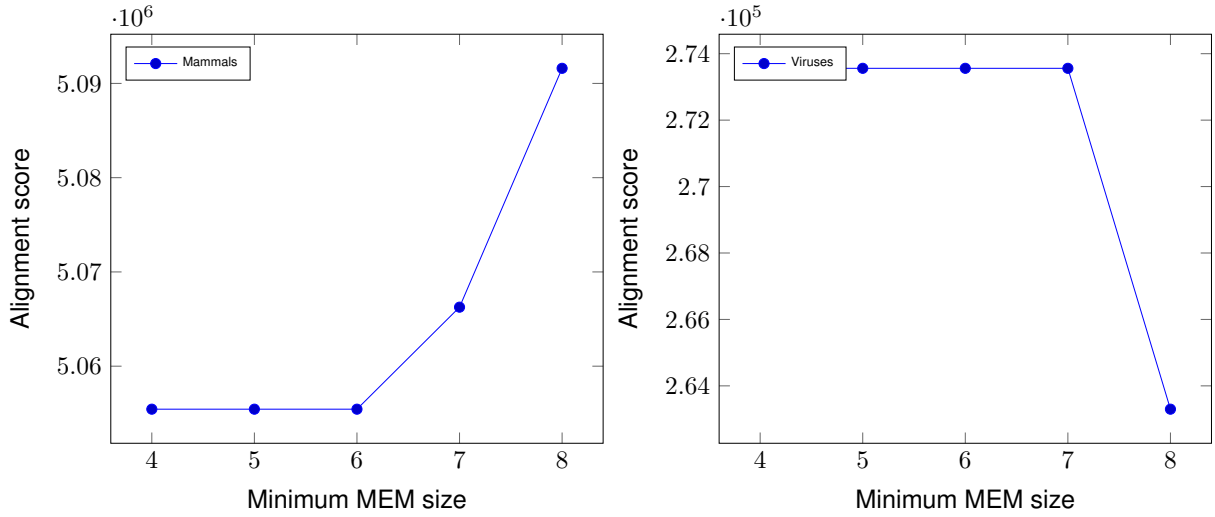


4.3.3 Efficiency vs. Accuracy Trade-off

CSA-MEM is designed with the concept of versatility in mind. This is achieved by incorporating an input parameter intended to determine the minimum size of each detected MEM. Such an inclusion imparts greater flexibility to the tool, especially in cases involving extensive and highly similar datasets, where there is no need to search for excessively small blocks. This, in turn, positively impacts the tool's computational complexity in terms of time efficiency.

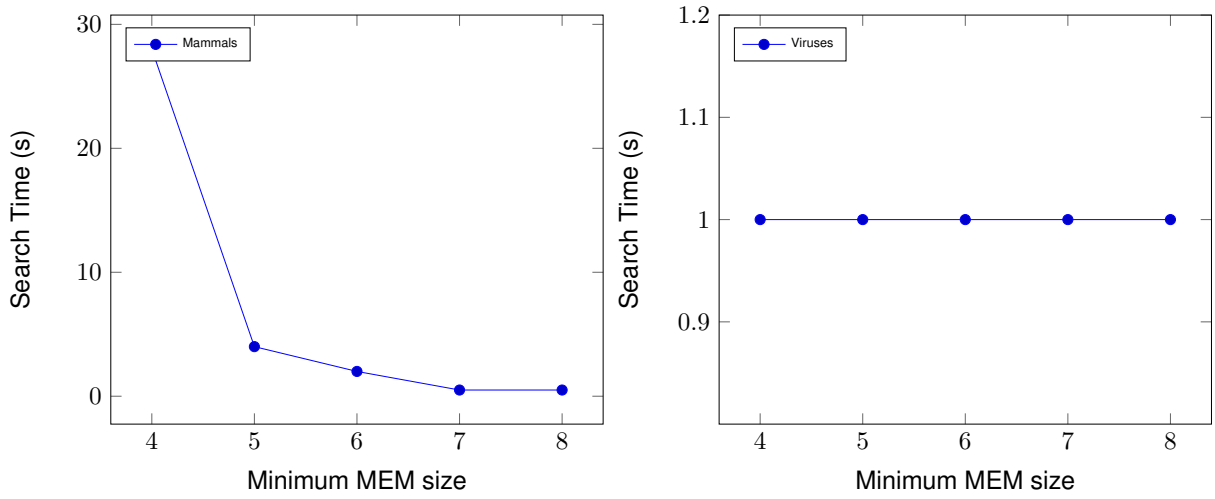
This versatility is clearly evident in the following Figures 4.6 and 4.7. For the Mammals dataset, which is relatively extensive and exhibits a high degree of similarity, the time required to compute the sequence cutting positions decreases significantly as the minimum size required for MEM detection increases. Furthermore, this adjustment also exerts a positive influence on the resultant alignment quality, most likely attributable to the reduction of less informative regions gathered when increasing the minimum size for MEMs. This, in turn, influences the decision-making process related to the dimensionality reduction problem inherent in the computation of the heaviest common increasing subsequence of MEMs, which can generalily explain the improvement of alignment quality.

Figure 4.6: Comparison between the resulting scores from different minimum MEM sizes for the mammals and viruses datasets respectively.



On the other hand, when datasets exhibit substantial diversity, it is advisable to reduce the minimum requirement for the MEM size. This is particularly relevant when the task at hand involves identifying common blocks among many highly disparate sequences. In such cases, smaller blocks may be prevalent, and an excessively high minimum size could result in the neglect of many critical blocks, as can be seen in Figure 4.6 for the viral dataset. In the most extreme circumstances, it could even result in the complete absence of detected blocks, thereby precluding the determination of an appropriate realignment strategy to enhance multiple-sequence alignment. Nonetheless, this effect is only discernible when studying large sequences. As evidenced in Figure 4.7, the required time to analyse the viral data set consistently remained below 1 second for all designated minimum MEM sizes.

Figure 4.7: Comparison between the resulting search time in seconds from different minimum MEM sizes for the mammals and viruses datasets respectively.



4.3.4 Benchmarks

CSA-MEM was evaluated against CSA, Cyclope, and MARS on the five mentioned datasets using the metrics described to assess their respective performance. Table 4.3 presents the results for the first four datasets.

Table 4.3: Comparative benchmarks between the four analysed circular sequence alignment software tools on four distinct datasets considering different metrics.

		Alignment score	Consensus length (bp)	Running time (s)	Memory used (MB)
Primates	CSA-MEM	**10474101**	17446	≤ 1	**30**
	CSA	10471600	**17444**	≤ 1	91
	Cyclope	10472551	17447	2820	4670
	MARS	10468302	17458	215	1183
Mammals	CSA-MEM	**5067971**	18608	≤ 1	**30**
	CSA	5049488	**18202**	≤ 1	75
	Cyclope	5020605	18268	1620	4797
	MARS	5033469	18460	168	1340
Diverse	CSA-MEM	13116753	19784	7	**30**
	CSA	12942609	**19751**	≤ 1	103
	Cyclope	**13132115**	19889	4260	5419
	MARS	13031632	20172	342	1562
Viruses	CSA-MEM	273560	3965	≤ 1	**30**
	CSA	248401	3980	≤ 1	33
	Cyclope	**276949**	**3963**	34	340
	MARS	257741	4051	6	96

On the smaller datasets, CSA-MEM achieves the highest alignment scoring results in the first and second datasets, while being the second best in the remaining ones. Although Cyclope shows a little

better scoring results, the computational costs are prohibitive for it to be used with current large data volumes. For instance, in the Diverse dataset the requirements of Cyclope in terms of both time and space are more than 1000x those of CSA-MEM. In terms of alignment consensus size, CSA consistently produces the most compact alignments in most tests. This implies that the chaining algorithm employed in CSA preserves a greater number of common blocks compared to the one utilised in CSA-MEM, although without a substantial disparity.

CSA and CSA-MEM are the fastest algorithms due to their reliance on text indexing data structures and their strategy based on common substrings. In some cases, CSA achieves better running times than CSA-MEM, possibly due to the reason already mentioned about the minimum size required of the MEMs for more diverse datasets. Both MARS and Cyclope are less efficient due to the use of time-consuming dynamic programming and progressive pairwise alignment algorithms. CSA-MEM memory consumption is always around 30MB, indicating that this should be the overhead for maintaining its data structures for the relatively small datasets considered.

In the context of larger datasets, such as the E. coli dataset, CSA-MEM demonstrates a great advantage in terms of both processing time and memory usage. CSA-MEM was capable of generating the optimal rotation in under 2 minutes, consuming less than 70 MB of memory. In sharp contrast, the alternative methods did not produce any results even after extended hours of processing. As a consequence of these outcomes, we have omitted their presentation in the table.

5

Conclusion

Contents

This concluding section offers a comprehensive overview and analysis of the themes and findings presented in the previous sections. An in-depth exploration of the implications associated with the comprehension of genetic relationships and evolutionary history encoded within circular DNA, as well as the challenges and complexities that are posed by circular sequences.

Furthermore, in this section, an examination of the limitations of CSA-MEM will be conducted, along with areas where further research and development is deemed necessary, such as the optimisation of the algorithm utilised for the identification of the longest chain of characters. Future research directions will also be explained, which include large-scale algorithm testing, metagenomic analysis, and potential theoretical considerations pertinent to the newly created indexes.

This concluding section, in its ultimate objective, strives to synthesise the key takeaways and contributions of the research, thereby emphasising the potential of the research to stimulate advancements within its respective scientific domains.

5.1 Conclusions

Understanding the genetic relationships and evolutionary history encoded in circular DNA molecules has a profound impact on human health, environmental studies, and the comprehension of the complexity and diversity of life. The intricate nature of circular sequences, such as those found in mitochondrial DNA, viral genomes, and plasmids, poses a significant challenge in the field of computational biology. The circular topology of these sequences facilitates efficient processes of genetic replication, maintenance, and transmission but simultaneously complexifies their alignment with conventional linear alignment tools commonly applied in genomic analysis. This issue becomes highly pronounced in comparative genomics, where the choice of arbitrary cutting points can lead to fragmentation of vital genetic elements, potentially compromising the accuracy and reliability of genetic and genomic analyses.

The central problem is not merely converting circular sequences into linear representations but doing so intelligently and systematically to preserve their biological significance. The choice of cutting points is of paramount importance, aiming to maximise the alignment across various rotational orientations. Addressing this problem has implications that extend far beyond the realm of bioinformatics and computational biology, affecting fields as diverse as medicine and agriculture.

To address this challenge, CSA-MEM was proposed to demonstrate that the use of efficient indexing data structures and string matching algorithms for circular sequences yields superior benchmark scores with minimal computational demands. It was empirically demonstrated that it is feasible to use indexing data structures such as the FM-Index to find circular patterns, thereby effectively identifying the longest chain of non-repeated maximal matches common to a set of circular DNA sequences in order to improve their sequence alignment. This novel approach not only enhances the rotation strategy, but also optimises space and time complexities, enabling a more thorough analysis of circular DNA sequences.

Furthermore, it is worth noting that this research has also been presented at the International Symposium on Bioinformatics Research and Applications (ISBRA) and published in the Springer Lecture Notes in Computer Science book series (LNBI) with the title ”CSA-MEM: Enhancing Circular DNA Multiple Alignment Through Text Indexing Algorithms” [73]. It was also showcased in a poster session during the second Microbiome PT Summit in Lisbon, highlighting its relevance and potential impact in the broader scientific community. Researchers can now analyse larger and complex datasets more comprehensively and in a timely manner, allowing the extraction of new meaningful patterns and relationships which can empower them to uncover novel discoveries, gain a deeper understanding of biological systems, and ultimately advance their respective fields of research.

5.2 System Limitations and Future Work

CSA-MEM is constrained in its ability to locate the longest chain of characters because of its capacity to solely analyse unique blocks. This constraint arises from the lack of an ideal solution for calculating the heaviest increasing subsequence across multiple sequences, necessitating further exploration and resolution of this ongoing computational challenge.

Furthermore, the effectiveness of the tool can be hindered by the minimum size input of MEMs set by the users. It is advisable for users to possess prior knowledge of the dataset to ensure the tool's most efficient use.

Upcoming research efforts will mainly focus on extensive algorithm testing through the establishment of a comprehensive database containing various circular genome datasets. These datasets will help identify constraints in the algorithms used in comparative genomics.

In addition, this algorithm will be applied to the analysis of metagenomic datasets that lack alignment within reference databases. It will aid in exploring antibiotic resistance elements and supporting dataset assembly for the training of machine learning algorithms targeting the human microbiome, specifically in the context of colon cancer, during ongoing projects.

Lastly, future work will include the publication of a more theoretical paper discussing the creation and implementation of the circular FM-Index, given its potential implications beyond the immediate problem. Although the need for circular patterns may not be common outside the field of bioinformatics, situations do arise where circular patterns must be identified, such as in data logging systems, which frequently employ circular data structures for information storage.

Bibliography

[1] F. Fernandes, L. Pereira, and A. T. Freitas, "Csa: an efficient algorithm to improve circular dna multiple alignment," *BMC bioinformatics*, vol. 10, no. 1, pp. 1–13, 2009.

[2] I. Laudadio, V. Fulc, L. Stronati, and C. Carissimi, "Next-generation metagenomics: Methodological challenges and opportunities," *OMICS*, vol. 23, no. 7, pp. 327–333, 2019.

[3] R. Grossi, C. S. Iliopoulos, R. Mercas, and et al., "Circular sequence comparison: algorithms and applications," *Algorithms Molecular Biology*, vol. 11, no. 12, 2016.

[4] C. A. Dulanto and J. P. Dekker, "From the Pipeline to the Bedside: Advances and Challenges in Clinical Metagenomics," *The Journal of Infectious Diseases*, vol. 221, no. Supplement 3, pp. S331–S340, 2019.

[5] T. Lappalainen, A. J. Scott, M. Brandt, and I. M. Hall, "Genomic analysis in the age of human genome sequencing," *Cell*, vol. 177, no. 1, pp. 70–84, 2019.

[6] P. S. Lee and K. H. Lee, "Genomic analysis," *Current Opinion in Biotechnology*, vol. 11, no. 2, pp. 171–175, 2000.

[7] A. Travers and G. Muskhelishvili, "DNA structure and function," *The FEBS journal*, vol. 282, no. 12, pp. 2279–2295, 2015.

[8] J. E. Krebs, E. S. Goldstein, and S. T. Kilpatrick, *Lewin's genes XII*. Jones & Bartlett Learning, 2017.

[9] D. F. Robinson and L. R. Foulds, "Comparison of phylogenetic trees," *Mathematical biosciences*, vol. 53, no. 1-2, pp. 131–147, 1981.

[10] B. G. Hall, *Phylogenetic trees made easy*. WH Freeman, 2004.

[11] A. L. Abdel-Mawgood, "DNA based techniques for studying genetic diversity," *Genetic diversity in microorganisms*, pp. 95–122, 2012.

[12] M. Nei and W.-H. Li, “Mathematical model for studying genetic variation in terms of restriction endonucleases.” *Proceedings of the National Academy of Sciences*, vol. 76, no. 10, pp. 5269–5273, 1979.

[13] L. B. Jorde and S. P. Wooding, “Genetic variation, classification and ’race’,” *Nature genetics*, vol. 36, no. Suppl 11, pp. S28–S33, 2004.

[14] J. L. Hartman IV, B. Garvik, and L. Hartwell, “Principles for the buffering of genetic variation,” *Science*, vol. 291, no. 5506, pp. 1001–1004, 2001.

[15] S. T. Sherry, M.-H. Ward, M. Kholodov, J. Baker, L. Phan, E. M. Smigielski, and K. Sirotkin, “dbsnp: the ncbi database of genetic variation,” *Nucleic acids research*, vol. 29, no. 1, pp. 308–311, 2001.

[16] M. Brudno, M. Chapman, B. Göttgens, S. Batzoglou, and B. Morgenstern, “Fast and sensitive multiple alignment of large genomic sequences,” *BMC bioinformatics*, vol. 4, pp. 1–11, 2003.

[17] J. Ernst, P. Kheradpour, T. S. Mikkelsen, N. Shoresh, L. D. Ward, C. B. Epstein, X. Zhang, L. Wang, R. Issner, M. Coyne *et al.*, “Mapping and analysis of chromatin state dynamics in nine human cell types,” *Nature*, vol. 473, no. 7345, pp. 43–49, 2011.

[18] X. Ge, H. Zhang, L. Xie, W. V. Li, S. B. Kwon, and J. J. Li, “Epialign: an alignment-based bioinformatic tool for comparing chromatin state sequences,” *Nucleic acids research*, vol. 47, no. 13, pp. e77–e77, 2019.

[19] L. Przybyla and L. A. Gilbert, “A new era in functional genomics screens,” *Nature Reviews Genetics*, vol. 23, no. 2, pp. 89–103, 2022.

[20] S. Fields, Y. Kohara, and D. J. Lockhart, “Functional genomics,” *Proceedings of the National Academy of Sciences*, vol. 96, no. 16, pp. 8825–8826, 1999.

[21] A. Carattoli, “Plasmids and the spread of resistance,” *International Journal of Medical Microbiology*, vol. 303, no. 6, pp. 298–304, 2013.

[22] G. M. Wahl, “The importance of circular DNA in mammalian gene amplification,” *Cancer Res*, vol. 49, no. 6, pp. 1333–1340, 1989.

[23] D. R. Helinski and D. Clewell, “Circular DNA,” *Annual review of biochemistry*, vol. 40, no. 1, pp. 899–942, 1971.

[24] Y. Li, K.-W. Hsiao, C. A. Brockman, D. Y. Yates, R. M. Robertson-Anderson, J. A. Kornfield, M. J. San Francisco, C. M. Schroeder, and G. B. McKenna, “When ends meet: Circular DNA stretches differently in elongational flows,” *Macromolecules*, vol. 48, no. 16, pp. 5997–6001, 2015.

[25] J. B. Noer, O. K. Hørsdal, X. Xiang, Y. Luo, and B. Regenberg, "Extrachromosomal circular acDNA in cancer: history, current knowledge, and methods," *Trends in Genetics*, vol. 38, no. 7, pp. 766–781, 2022.

[26] W. Bauer and J. Vinograd, "Circular DNA," *Basic principles of nucleic acid chemistry (T'so, ed.)*, vol. 2, pp. 265–303, 2012.

[27] J. L. O. Pohjoismäki and S. Goffart, "Of circles, forks and humanity: Topological organisation and replication of mammalian mitochondrial dna," *BioEssays*, vol. 33, no. 4, pp. 290–299, 2011.

[28] L. Pereira, F. Freitas, V. Fernandes, J. B. Pereira, M. D. Costa, S. Costa, V. Máximo, V. Macaulay, R. Rocha, and D. C. Samuels, "The diversity present in 5140 human mitochondrial genomes," *The American Journal of Human Genetics*, vol. 84, no. 5, pp. 628–640, 2009.

[29] L. Yang, R. Jia, T. Ge, S. Ge, A. Zhuang, P. Chai, and X. Fan, "Extrachromosomal circular DNA: biogenesis, structure, functions and diseases," *Signal transduction and targeted therapy*, vol. 7, no. 1, p. 342, 2022.

[30] M. J. Tisza, D. V. Pastrana, N. L. Welch, B. Stewart, A. Peretti, G. J. Starrett, Y.-Y. S. Pang, S. R. Krishnamurthy, P. A. Pesavento, D. H. McDermott *et al.*, "Discovery of several thousand highly diverse circular DNA viruses," *Elife*, vol. 9, 2020.

[31] L. Zhao, K. Rosario, M. Breitbart, and S. Duffy, "Chapter three - eukaryotic circular rep-encoding single-stranded dna (cress dna) viruses: Ubiquitous viruses with small genomes and a diverse host range," ser. Advances in Virus Research, M. Kielian, T. C. Mettenleiter, and M. J. Roossinck, Eds. Academic Press, 2019, vol. 103, pp. 71–133.

[32] R. Radloff, W. Bauer, and J. Vinograd, "A dye-buoyant-density method for the detection and isolation of closed circular duplex DNA: the closed circular DNA in hela cells." *Proceedings of the National Academy of Sciences*, vol. 57, no. 5, pp. 1514–1521, 1967.

[33] J. D. Thompson, T. J. Gibson, and D. G. Higgins, "Multiple sequence alignment using clustalw and clustalx," *Current protocols in bioinformatics*, no. 1, pp. 2–3, 2003.

[34] Y. Zhang, Q. Zhang, J. Zhou, and Q. Zou, "A survey on the algorithm and development of multiple sequence alignment," *Briefings in Bioinformatics*, vol. 23, no. 3, 2022.

[35] L. A. Ayad and S. P. Pissis, "Mars: Improving multiple circular sequence alignment using refined sequences," *BMC Genomics*, vol. 18, 1 2017.

[36] S. Pan, X.-M. Zhao, and L. P. Coelho, "SemiBin2: self-supervised contrastive learning leads to better MAGs for short- and long-read sequencing," *Bioinformatics*, vol. 39, no. Supplement 1, pp. i21–i29, 2023.

[37] S. N. Cohen, A. C. Chang, and L. Hsu, “Nonchromosomal antibiotic resistance in bacteria: genetic transformation of escherichia coli by r-factor DNA,” *Proceedings of the National Academy of Sciences*, vol. 69, no. 8, pp. 2110–2114, 1972.

[38] P. M. Bennett, “Plasmid encoded antibiotic resistance: acquisition and transfer of antibiotic resistance genes in bacteria,” *British journal of pharmacology*, vol. 153, no. S1, pp. S347–S357, 2008.

[39] P. Ferragina and G. Manzini, “Opportunistic data structures with applications,” in *Proceedings 41st annual symposium on foundations of computer science*. IEEE, 2000, pp. 390–398.

[40] D. Adjeroh, T. Bell, and A. Mukherjee, “Exact and approximate pattern matching,” *The Burrows-Wheeler Transform: Data Compression, Suffix Arrays, and Pattern Matching*, pp. 187–263, 2008.

[41] D. Gusfield, “Algorithms on stings, trees, and sequences: Computer science and computational biology,” *Acm Sigact News*, vol. 28, no. 4, pp. 41–60, 1997.

[42] B. Haubold and T. Wiehe, “Biological sequences and the exact string matching problem,” *Introduction to Computational Biology: An Evolutionary Approach*, pp. 43–63, 2006.

[43] S. Kim and A. M. Segre, “Amass: A structured pattern matching approach to shotgun sequence assembly,” *Journal of Computational Biology*, vol. 6, no. 2, pp. 163–186, 1999.

[44] B. Branchini, S. Breschi, A. Zeni, and M. D. Santambrogio, “Fast genome analysis leveraging exact string matching,” in *2022 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, 2022, pp. 136–139.

[45] Z. Galil and R. Giancarlo, “Data structures and algorithms for approximate string matching,” *Journal of Complexity*, vol. 4, no. 1, pp. 33–72, 1988.

[46] P. Ferragina, G. Manzini, V. Mäkinen, and G. Navarro, “Compressed representations of sequences and full-text indexes,” *ACM Transactions on Algorithms (TALG)*, vol. 3, no. 2, pp. 20–es, 2007.

[47] U. Manber and G. Myers, “Suffix arrays: a new method for on-line string searches,” *siam Journal on Computing*, vol. 22, no. 5, pp. 935–948, 1993.

[48] M. Burrows, “A block-sorting lossless data compression algorithm,” *SRS Research Report*, vol. 124, 1994.

[49] G. Nong, S. Zhang, and W. H. Chan, “Linear suffix array construction by almost pure induced-sorting,” in *2009 data compression conference*. IEEE, 2009, pp. 193–202.

[50] V. Mäkinen and G. Navarro, “Succinct suffix arrays based on run-length encoding,” in *Annual Symposium on Combinatorial Pattern Matching*. Springer, 2005, pp. 45–56.

[51] Y. Arakawa, G. Navarro, and K. Sadakane, “Bi-directional r-indexes,” vol. 223. Schloss Dagstuhl-Leibniz-Zentrum fur Informatik GmbH, Dagstuhl Publishing, 6 2022.

[52] T. W. Lam, R. Li, A. Tam, S. Wong, E. Wu, and S.-M. Yiu, “High throughput short read alignment via bi-directional bwt,” in *2009 IEEE International Conference on Bioinformatics and Biomedicine*. IEEE, 2009, pp. 31–36.

[53] “Finding maximal exact matches using the r-index,” *Journal of Computational Biology*, vol. 29, pp. 188–194, 2 2022.

[54] S. Giuliani, G. Romana, and M. Rossi, “Computing maximal unique matches with the r-index,” vol. 233. Schloss Dagstuhl- Leibniz-Zentrum fur Informatik GmbH, Dagstuhl Publishing, 7 2022.

[55] A. L. Delcher, S. Kasif, R. D. Fleischmann, J. Peterson, O. White, and S. L. Salzberg, “Alignment of whole genomes,” *Nucleic acids research*, vol. 27, no. 11, pp. 2369–2376, 1999.

[56] T. Smith and M. Waterman, “Identification of common molecular subsequences,” *Journal of Molecular Biology*, vol. 147, no. 1, pp. 195–197, 1981. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0022283681900875

[57] G. Marçais, A. L. Delcher, A. M. Phillippy, R. Coston, S. L. Salzberg, and A. Zimin, “Mummer4: A fast and versatile genome alignment system,” *PLoS computational biology*, vol. 14, no. 1, p. e1005944, 2018.

[58] F. Fernandes and A. T. Freitas, “slamem: efficient retrieval of maximal exact matches using a sampled lcp array,” *Bioinformatics*, vol. 30, no. 4, pp. 464–471, 2014.

[59] J. Fischer, V. Mäkinen, and G. Navarro, “Faster entropy-bounded compressed suffix trees,” *Theoretical Computer Science*, vol. 410, no. 51, pp. 5354–5364, 2009.

[60] S. B. Needleman and C. D. Wunsch, “A general method applicable to the search for similarities in the amino acid sequence of two proteins,” *Journal of molecular biology*, vol. 48, no. 3, pp. 443–453, 1970.

[61] T. F. Smith, M. S. Waterman *et al.*, “Identification of common molecular subsequences,” *Journal of molecular biology*, vol. 147, no. 1, pp. 195–197, 1981.

[62] D. W. Mount, “Using blosum in sequence alignments,” *Cold Spring Harbor Protocols*, vol. 2008, no. 6, pp. pdb–top39, 2008.

[63] ——, “Using the basic local alignment search tool (blast),” *Cold Spring Harbor Protocols*, vol. 2007, no. 7, pp. pdb–top17, 2007.

[64] M. J. Griffiths, M. J. Shafi, S. J. Popper, C. A. Hemingway, M. M. Kortok, A. Wathen, K. A. Rockett, R. Mott, M. Levin, C. R. Newton *et al.*, "Genomewide analysis of the host response to malaria in kenyan children," *Journal of Infectious Diseases*, vol. 191, no. 10, pp. 1599–1611, 2005.

[65] A. Mosig, I. L. Hofacker, and P. F. Stadler, "Comparative analysis of cyclic sequences: Viroids and other small circular rnas," in *German Conference on Bioinformatics*, D. Huson, O. Kohlbacher, A. Lupas, K. Nieselt, and A. Zell, Eds. Bonn: Gesellschaft für Informatik e.V., 2006, pp. 93–102.

[66] E. Ukkonen, "On-line construction of suffix trees," *Algorithmica*, vol. 14, no. 3, pp. 249–260, 1995.

[67] L. A. Ayad and S. P. Pissis, "Mars: improving multiple circular sequence alignment using refined sequences," *BMC genomics*, vol. 18, no. 1, pp. 1–10, 2017.

[68] G. Fritzsch, M. Schlegel, and P. F. Stadler, "Alignments of mitochondrial genome arrangements: Applications to metazoan phylogeny," *Journal of Theoretical Biology*, vol. 240, no. 4, pp. 511–520, 2006.

[69] C. Barton, C. S. Iliopoulos, R. Kundu, S. P. Pissis, A. Retha, and F. Vayani, "Accurate and efficient methods to improve multiple circular sequence alignment," in *Experimental Algorithms: 14th International Symposium, SEA 2015, Paris, France, June 29–July 1, 2015, Proceedings 14*. Springer, 2015, pp. 247–258.

[70] E. Ohlebusch, S. Gog, and A. Kügel, "Computing matching statistics and maximal exact matches on compressed full-text indexes," in *String Processing and Information Retrieval: 17th International Symposium, SPIRE 2010, Los Cabos, Mexico, October 11-13, 2010. Proceedings 17*. Springer, 2010, pp. 347–358.

[71] G. Jacobson and K.-P. Vo, "Heaviest increasing/common subsequence problems," in *Annual Symposium on Combinatorial Pattern Matching*. Springer, 1992, pp. 52–66.

[72] E. Fehér, E. Mihalov-Kovács, E. Kaszab, Y. S. Malik, S. Marton, and K. Bányai, "Genomic diversity of cress DNA viruses in the eukaryotic virome of swine feces," *Microorganisms*, vol. 9, no. 7, p. 1426, 2021.

[73] A. Salgado, F. Fernandes, and A. T. Freitas, "Csa-mem: Enhancing circular dna multiple alignment through text indexing algorithms," in *Bioinformatics Research and Applications: 19th International Symposium, ISBRA 2023, Wrocław, Poland, October 9–12, 2023, Proceedings*. Berlin, Heidelberg: Springer-Verlag, 2023, p. 509–517. [Online]. Available: https://doi.org/10.1007/978-981-99-7074-2_41