WHOLE GENOME ALIGNMENT VIA ALTERNATING LYNDON FACTORIZATION TREE TRAVERSAL

A THESIS SUBMITTED TO

THE GRADUATE SCHOOL OF ENGINEERING AND SCIENCE

OF BILKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR

THE DEGREE OF

MASTER OF SCIENCE

IN

COMPUTER ENGINEERING

By

Mahmud Sami Aydın

July 2023

Whole Genome Alignment via Alternating Lyndon Factorization Tree Traversal
By Mahmud Sami Aydın
July 2023

ABSTRACT

WHOLE GENOME ALIGNMENT VIA ALTERNATING LYNDON FACTORIZATION TREE TRAVERSAL

Mahmud Sami Aydın
M.S. in Computer Engineering
Advisor: Can Alkan
July 2023

The Whole Genome Alignment Problem (WGA) is an important challenge in the field of genomics, especially in the context of pangenome construction. Here we propose a novel indexing structure called the **Alternating Lyndon Factorization Tree (ALFTree)**, which incorporates both spatial and lexicographical information within its nodes. The ALFTree is a powerful tool for WGA, as it can efficiently store and retrieve information about large DNA sequences.

We present an algorithm, namely **Idoneous**, specifically designed to construct the ALFTree from a given DNA sequence. The algorithm works by generating intervals of specific sizes, identifying matches within these intervals, and performing a sanity check through alignment procedures. The algorithm is efficient and scalable, making it a valuable tool for WGA.

Some of the key features of the ALFTree are 1) compact and efficient data structure for storing large DNA sequences; 2) efficient retrieval of information about specific regions of a DNA sequence; 3) ability to handle both spatial and lexicographical information; and 4) scalability to large DNA sequences.

Our experimental results on different genomes highlight the effects of parameter selections on coverage and identity. Idoneous demonstrates competitive performance in terms of coverage and provides flexibility in adjusting sensitivity and specificity for different alignment scenarios.

The ALFTree has the potential to significantly improve the performance of WGA algorithms. We believe that the ALFTree is a valuable contribution to the field of genomics, and we hope that it will be used by researchers to accelerate the pace of discovery.

Keywords: Whole Genome Alignment, Indexing, Lyndon Factorization.

ÖZET

ALMAŞIK LYNDON FAKTORİZASYON AĞACINDA GEZİNEREK TÜM GENOM HİZALAMA

Mahmud Sami Aydın
Bilgisayar Mühendisliği, Yüksek Lisans
Tez Danışmanı: Can Alkan
Temmuz 2023

Tüm Genom Hizalama Problemi (WGA), özellikle pangenom oluşturma bağlamında genomik alanında önemli bir zorluktur. Burada, düğümlerinde hem mekansal hem de alfabetik bilgileri bir araya getiren yeni bir indeksleme yapısı olan **Almaşık Lyndon Faktörizasyon Ağacı (ALFAğacı)** öneriyoruz. ALFAğacı, büyük DNA dizileri hakkında bilgi depolamak ve geri almak için etkili bir araçtır.

Belirli bir DNA dizisinden ALFAğacını oluşturmak için **Idoneous** adını verdiğimiz bir algoritma sunuyoruz. Algoritma, belirli boyutlardaki aralıkları oluşturarak, bu aralıklar içinde eşleşmeleri belirleyerek ve hizalama işlemleri aracılığıyla bir doğrulama kontrolü gerçekleştirerek çalışır. Algoritma verimli ve ölçeklenebilir olduğundan, WGA için değerli bir araçtır.

ALFAğacının önemli özellikleri şunlardır: 1) büyük DNA dizilerini depolamak için kompakt ve verimli bir veri yapısı; 2) belirli bir DNA dizisinin belli bölgeleri hakkında bilgiyi etkili bir şekilde geri alabilme; 3) hem mekansal hem de alfabetik bilgilere uyum sağlama yeteneği; ve 4) büyük DNA dizilerine ölçeklenebilme.

Farklı genomlardaki deneysel sonuçlarımız, parametre seçimlerinin kapsama ve benzerlik üzerindeki etkilerini vurgulamaktadır. Idoneous, kapsama açısından rekabetçi bir performans sergilemekte ve farklı hizalama senaryoları için hassasiyet ve özgüllük ayarlamasında esneklik sağlamaktadır.

ALFAğacı, WGA algoritmalarının performansını önemli ölçüde artırma potansiyeline sahiptir. ALFAğacının genomik alanına değerli bir katkı olduğuna inanıyor ve araştırmacıların keşif hızını hızlandırmak için kullanmasını umuyoruz.

Anahtar sözcükler: Tüm Genom Hizalama, İndeksleme, Lyndon Faktörizasyon .

Acknowledgement

I would like to express my deepest gratitude and appreciation to all who have contributed to the completion of this master's thesis.

First and foremost, I am immensely thankful to my supervisor, Can Alkan, for their guidance, expertise, and unwavering support throughout the entire research process. Can Alkan's valuable insights, constructive feedback, and constant encouragement have been instrumental in shaping this thesis and my academic growth. I am truly grateful for their patience, dedication, and commitment to my success.

I would also like to acknowledge the valuable contributions of my thesis committee members, Cevdet Aykanat and Aybar Can Acar. Their expertise, feedback, and scholarly guidance have significantly enhanced the quality of this thesis.

Furthermore, I want to express my sincere gratitude to Abdullah for their significant contribution in suggesting parameter names for this thesis. Abdullah's creative thinking and thoughtful suggestions played a vital role in selecting descriptive and appropriate names that accurately represent the parameters in the research.

Additionally, I am grateful for the elaborate support provided by Tarık during the poster presentation of this research. Tarık's ability to explain complex concepts in a clear and concise manner has been invaluable in effectively communicating our findings to the audience. Their assistance in creating an informative and visually appealing poster greatly enhanced the impact of our presentation.

I would also like to extend my heartfelt appreciation to my colleagues Ömer, Onur, Ecem, Klea, Rafi, Gözde, Ezgi, Ricardo, and Zülal. Their support, collaboration, and camaraderie have been invaluable throughout this research journey. Their constructive discussions, insightful perspectives, and unwavering support have played a crucial role in shaping my ideas and enhancing the overall quality of this thesis.

I am grateful for the financial support provided by the Scientific and Technological Research Council of Turkey (TÜBİTAK) under the 2210-A program. Their support has enabled me to conduct this research and contributed significantly to this thesis's successful completion.

Additionally, I would like to express my gratitude to my friends Faruk et al.(354,355,601), Mehmet Emin Albayrak, and all the members of the Bilkent Board Game Enthusiasts and the Storytelling and Mythology Club. Their support, friendship, and shared interests have provided a supportive and inspiring community outside of academia. Their diverse perspectives, engaging discussions, and enjoyable company have enriched my life and provided a much-needed balance during this research journey.

Lastly, I am deeply grateful for the love, support, and understanding of my family. I would like to thank my parents, Nuran and Bahattin, for their belief in my abilities and continuous encouragement. Throughout this journey, my siblings, Semanur, Abdulkadir, and Fatmanur, have been a constant source of inspiration and support. Their presence has been a pillar of strength and motivation. Additionally, I am grateful for the joy and laughter brought into my life by my nieces and nephews. I want to express my heartfelt appreciation to my loved one, Şeyma. Their steady support, understanding, and love have been instrumental in my journey. They have provided me with continuous support and encouragement, and their presence in my life has been an endless source of inspiration and motivation.

Thank you all for your invaluable contributions.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Genomics

All living organisms have a common genetic material that contains genetic instructions that regulate growth, development, metabolism, and reproduction. For most complex genomes this material is DNA; however, simpler organisms such as viruses instead have RNA molecules. This genetic material is crucial to understanding biological function, intra- and interspecies variation, genetic abnormalities that may give rise to diseases, and evolutionary relationships between species. To better understand and identify the similarities and differences among genomes, we need to compare genomic sequences of both individuals of the same species and also between different species.

Various different methods have been proposed to calculate genetic variation since the first efforts of DNA & RNA sequencing studies. The first metric used for variation is qualifying heterozygosity, that is, the differences between maternally and paternally inherited DNA in diploid cells (e.g., in humans), which can be generalized allelic variation in polyploid genomes (e.g., in some plants). In this metric, variation is quantified as the average rate of heterozygosity observed in each individual of a given population or species. The next metric used to indicate genetic variation is statistical variance, which is proportional to the squared distance between an individual and a consensus genome of a population. Later, several other metrics including Bayesian statistics, entropy, and F statistics are used to indicate genetic variation.

Individuals of the same species show very high identity in their genetic material. For example, the 1000 Genomes Project showed that genomes of human individuals even from distant populations are >99% identical [1]. Similarly, a study on dog genomes demonstrated that domestic dog genomes are $> 99.8\%$ identical [2] even though dogs show a high rate of phenotypic variation.

The variable sections of the genome, which correspond to the observed differences, cause the diversity within the species. These differences can be single nucleotide variation (SNV) or larger-scale structural variation (SV). SNVs are substitutions of a single nucleotide in a specific location of an allele. Those variations in a single nucleotide can be classified according to its annotation, such as coding vs. non-coding. If an SNV is identified in a non-coding region, it likely does not cause phenotypic modification. On the other hand, if an SNV is within the coding region of a gene, it may have three different possible outcomes, where one of them is called *synonymous* and the two others are called *non-synonymous*. Synonymous substitution occurs if the altered nucleotide does not change the codon that determines the amino acid in the protein product. The two non-synonymous variations, however, result in changes in the coded amino acid. The first subtype of a non-synonymous mutation is *missense*, which results in a changed but full-size protein that may cause a disease or a malfunction in cell metabolism. The second subtype, called *nonsense*, results in an immature stop in the coded protein, which therefore truncates it, which again, in turn, may disrupt cell metabolism and cause diseases. Searching for these variations is the standard practice to identify genotype-disease relations and to help diagnose patients with genetic diseases.

Structural variants are large-scale rearrangements that affect DNA segments of length from 50 bp to several megabases. These include *insertions, deletions, duplications, inversions, transpositions* (i.e., *mobile element insertions*) and *translocations*. Insertions add some new content to the DNA compared to the reference sequence. Deletions are the mirror cases of insertions as they remove DNA compared to the reference. Duplications are copy events where DNA segments are repeated elsewhere in the genome. Insertions, deletions, and duplications are also collectively called *copy number variation*. Inversions correspond to genomic segments that are inverted in orientation at the same loci. Transpositions, also called mobile element insertions, are copy events of previously characterized common repeat elements such as Alu and L1Hs sequences. Translocations are the migrations of the segments of a sequence from one chromosome to another. A study shows that the rate of structural variations is 8% in one generation per haplotype [3], whereas another study claims that structural variants can affect up to 20% of a human genome [4].

The divergence between genomes of different species is informative of the evolutionary relationship between those species. For example, there is a 1.23% single nucleotide variation along the human and chimpanzee genomes, where this difference within the species is not greater than 1.06%. Approximately 29% of the orthologous genes, which are derived from a common ancestor and have high similarity in sequence and functionality, between human and chimpanzee genomes are exactly the same. Most of the remaining orthologous genes do not have more than two different amino acids. [5]

To be able to perform the comparisons and analyses explained above, one needs to "read" the genome of a given individual. The earliest attempts to identify nucleotides that make up the genetic material, or the *first-generation sequencing* technology, Sanger sequencing, was both time-consuming and prohibitively expensive. Still, it was the main driving force to generate the human reference genome, as well as the reference genomes for several other species including chimpanzee, marmoset, baboon, mouse, and rat. Starting in 2006, several new-generation DNA sequencing platforms started to replace Sanger sequencing, which offered massive parallelization to offer high-throughput DNA sequencing

(HTS). However, they also generated shorter and more error-prone readouts, increased redundancy, and sequencing became cheap, widely available, and affordable, which in turn spearheaded large-scale genome variation characterization studies such as the 1000 Genomes Project.

There are two main strategies for genome analysis given a sequenced read set. First, similar to constructing a *superstring*, we can “stitch” short DNA reads together and use the data redundancy to eliminate as many errors as possible. This strategy is called *de novo sequencing*, and it is the only method to apply if there does not exist a prebuilt reference genome of the organism under evaluation. In the case where a reference genome exists, the second strategy called *resequencing* is the method of choice. Here, we simply compare each short DNA sequence read with the reference, through mapping and alignment, and identify differences while reducing errors through statistical analysis of redundant alignment information. Both *de novo* sequencing and resequencing include finding matching patterns between strings. This problem is called Sequence Alignment Problem, which is explained in Section 1.2.

1.2 Sequence Alignment Problem

Here we define two versions of the Sequence Alignment Problem: Pairwise Sequence Alignment Problem and Multiple Sequence Alignment Problem.

1.2.1 Pairwise Sequence Alignment Problem

The Pairwise Sequence Alignment Problem can be defined as follows:

Given two sequences, find the alignment with the highest alignment score. The score is calculated according to awarding the number of matching nucleotides and penalizing gaps which can be seen as deletion on one sequence and insertion on the other, as well as mismatch between nucleotides.

Also, this version of the problem has several types: global, global-extension, semi-global, overlap, extension, and local. For different scenarios, different types of the problem are solved. The two most common pairwise alignment types are global and local alignment. Both have a solution with dynamic programming.

Global alignment finds a full alignment between the two sequences so that each sequence is aligned from the first position to the last position. The solution to the problem is proposed by Needleman-Wunsch [6]. The algorithm is based on dynamic programming with table size $N \cdot M$ where N and M are the sequence lengths. The recurrence function of the dynamic programming table is as follows.

$$s_{i,j} = max \begin{cases} s_{i-1,j-1} + 1 & \text{if } v_i = w_j \\ s_{i-1,j-1} - \mu & \text{if } v_i \neq w_j \\ s_{i-1,j} - \sigma \\ s_{i,j-1} - \sigma \end{cases}$$

where v_i and w_j are i^{th} and j^{th} positions on sequence v and w respectively. It traces back the alignment from the right bottom corner to the left top corner. This algorithm takes $O(N \cdot M)$ time since calculating each entry in the table takes constant time and the run time of the traceback procedure is bounded by $O(N + M)$.

Local alignment finds an alignment between the two sequences without considering the tails of each sequence. Therefore, there is no penalty when parts of the sequences are aligned if it is not between two aligned regions. The solution to the problem proposed by Smith-Waterman [7]. The algorithm also does dynamic programming with the same table size, and it is a simple extension of the Needleman-Wunsch algorithm. The recurrence function of the dynamic programming table is as follows:

$$s_{i,j} = max \begin{cases} s_{i-1,j-1} + 1 & \text{if } v_i = w_j \\ s_{i-1,j-1} - \mu & \text{if } v_i \neq w_j \\ s_{i-1,j} - \sigma \\ s_{i,j-1} - \sigma \\ 0 \end{cases}$$

where v_i and w_j are the i^{th} and j^{th} positions of sequences v and w respectively. The traceback interval starts from the maximum value on the table to 0. This algorithm takes $O(N \cdot M)$ time since calculating each entry on the table takes constant time.

1.2.2 Multiple Sequence Alignment Problem

The Multiple Sequence Alignment (MSA) Problem can be defined as follows:

Align k sequences such that the total cost of pairwise alignments *inferred* from the MSA is minimum. The cost function is similar to the cost function of the pairwise sequence alignment.

The MSA problem is shown to be MAX SNP-hard [8] therefore optimal MSA construction is not feasible. The naive attempt to solve MSA uses a k dimensional dynamic programming table with the length of each dimension being the length of one sequence. In this table, each entry is calculated by comparing 2^k previous entries. Assuming that the length of each sequence is N, the run time becomes $O(2^k \cdot N^k)$. Therefore heuristics are applied to achieve a polynomial run time while sacrificing optimality. Profile-based progressive alignment can construct MSA in $O(k \cdot N^2)$ time.

1.3 Burst of Assembly data

With the advent of HTS and advancements in *de novo* assembly algorithms, numerous assembled genomes of various organisms, including humans, have become available. Several consortia and projects are dedicated to enhancing the confidence of these genomes while expanding the diversity of species represented within genomic libraries. For instance, initiatives such as the Human Genome Project, the 1000 Genomes Project, and The Genome Reference Consortium not only aim to establish more reliable human genome references but also strive to incorporate a broader range of human populations into these references. Furthermore, endeavors such as the Great Ape Genome Project [9], The Chimpanzee Sequencing and Analysis Consortium [5], and the Orangutan Genome Project [10] focus on exploring the genomes of species closely related to humans. The Vertebrate Genomes Project [11], The 10000 Plant Genomes Initiative [12], and the Earth Biogenome Project [13] are involved in the exploration of genomes across various species. Consequently, the analysis and comparison of these numerous genomes have led to the emergence of the Whole Genome Alignment Problem as a pressing concern.

1.4 Whole Genome Alignment Problem

The Whole Genome Alignment Problem pertains to the alignment of strings at the scale of entire chromosomes or whole genomes. Standard alignment problem solutions encounter several challenges that render them inadequate for addressing this problem, particularly in terms of scalability and the resolution of structural variants.

Scalability becomes a significant concern when dealing with large strings, as existing alignment algorithms lack linear or sub-linear time complexity. For instance, storing an uncompressed haploid human genome with each nucleotide allocated 2 bits would require approximately 750 MB of memory. If we were to construct a dynamic table employing 32-bit integers for the entire genome, a minimum of 36 exabytes of memory would need to be allocated. Even when assuming knowledge of each chromosome, the smallest of which is chromosome 22, a memory allocation of 10 petabytes would be required. Consequently, standard approaches are not scalable for whole genome alignment. Although there may be potential techniques to reduce the size of the dynamic table, the resulting memory access demands would be time-consuming, leading to poor algorithm scalability. While the banded version of pairwise alignment necessitates a linear space dynamic table, it only functions when the sequences possess precisely the same starting point and exhibit minimal variation from the main diagonal of the dynamic table. Unfortunately, this is not the case for sequences within the domain of whole genome alignment.

Another crucial issue in whole genome alignment involves the presence of structural variants within the sequences. Global alignment fails to reveal more than one structural variant, despite thousands of such variants existing within the human genome. Local alignments can identify structural variants, but the quadratic time complexity required to check each entry for such variants renders this approach unsuitable. Consequently, more sophisticated methods are employed to address this problem, such as the seed-extend-chain framework. This framework utilizes exact matching regions (seeds) to establish alignments between sequences. However, the uniform-like distribution of seeds within the method results in quadratic time complexity for alignment. Thus, seeding techniques must ensure a non-uniform distribution of seeds to mitigate this issue.

For multiple sequence alignment in the context of whole genomes, graph data structures and guiding trees with specialized indexes are utilized. However, the primary challenge in the first method lies in effectively handling large graphs and generating alignments from loosely connected graphs, even when a consensus is achieved. The extraction of pairwise alignments from such consensus pangenome graphs proves difficult. The second method, known as Cactus [14] and its more recent iteration, Minigraph-Cactus [15], relies on the assumption of having guiding trees. While it is feasible to construct such trees for inter-species cases due to the availability of analyses and guiding trees for species, creating a guiding tree between individuals within a species is as challenging as pairwise alignment, particularly when dealing with reference-free genomes.

1.5 Motivation

Existing solutions for the pairwise whole genome alignment problem lack flexibility in terms of alignment size. On one hand, there are exceedingly small matches, such as seeds or maximum unique matches, which exclusively consist of exact matches. While these matches can be rapidly identified, they do not accommodate any mutations. Additionally, due to their short lengths, they are insufficiently sensitive for capturing long approximate matches. On the other hand, there exist very large alignments that require significant time due to the processes of gap-filling and alignment. **The objective of this research is to strike a balance between these two extremes, aiming to harness the sensitivity of approximate matches while maintaining computational efficiency.**

1.6 Thesis Statement and Contribution

The thesis statement of this study asserts that *a scalable and localized reference-free whole genome alignment can be achieved with an approximate string matching and subsampling index structure while ensuring a desired level of sensitivity and specificity.* To accomplish this, we have developed a comprehensive framework comprising data preprocessing, indexing, and matching stages. Notably, we propose a novel indexing structure called the **Alternating Lyndon Factorization Tree (ALFTree)**, which incorporates both spatial and lexicographical information within its nodes. Furthermore, we present an algorithm specifically designed to construct the ALFTree from a given DNA sequence in FASTA format. Additionally, we introduce the **Idoneous** algorithm that generates intervals of specific sizes on two ALFTrees, identifies matches within these intervals, and performs a sanity check through alignment procedures.

Chapter 2

Background

In this chapter, we discuss related works and data structures that contribute to designing ALFTree, Monoid Factorization Methods, and run-length encoding.

2.1 Related Works

This section contains solutions for the pairwise whole genome alignment problem. As mentioned before, two different approaches exist to solve the problem. Two solutions are the fast solution with exact matches and the slow solution with approximate matches.

The state of the art for the former one is MUMmer. It finds the maximum unique exact matches via suffix array data structure. Even though it is less sensitive because of exactness, it finds matches very fast. Namely, it has high false negatives but has the advantage of time efficiency in order of magnitudes.

The state of the art for the latter one is NUCmer. It starts with maximum unique exact seeds and makes possible chaining and gap-fillings in order to extend and combine seeds to get longer alignments and high coverage. This solution is more sensitive since it contains approximate matches as well. However, because of the chaining and gap-filling part, it consumes lots of time even if it is the best method for that coverage rate.

2.2 Data Structures

This section covers min-max heap, spatial indexing, and syncmers in short.

2.2.1 Min-max Heap

The min-max heap is a binary tree that combines the properties of both a min-heap and a max-heap. This tree is characterized by its *min-max ordering*, where values at nodes located on even levels are smaller than or equal to the values stored at their descendants (if any), while values at nodes on odd levels are greater than or equal to their descendants. It is important to note that the root node is considered to be at level zero [16].

A notable characteristic of this data structure is the decreasing marginality of nodes as we traverse deeper into the tree. In this context, marginality refers to the probability of a node being distant from the center of the value space, which is defined by the interval between the minimum and maximum values present in the tree. This property can be demonstrated for a set of node values that are uniformly distributed.

Consider a scenario where we have n nodes, each with values uniformly distributed between 0 and 1. In the absence of location knowledge, we can infer that the marginality is proportional to the variance of the distribution. For a uniform distribution between 0 and 1, the variance is known to be 1/12.

As we traverse deeper into the tree, specifically reaching the $2n^{th}$ level node, it can be observed that k of its descendants possess values smaller than the node, while k of them have values greater than the node. This implies that the node itself functions as the median value among $2n+1$ samples drawn from a uniform distribution spanning the interval between 0 and 1. Consequently, the density function of the node can be expressed as:

$$f(x) = \frac{x^n(1-x)^n}{B(n+1, n+1)}$$

This density function corresponds to the beta distribution, where the numerator represents the combination of choosing n elements from the interval $(0, x)$ and n elements from the interval $(x, 1)$. The denominator incorporates a factor of the beta function, denoted as $B(n+1, n+1)$, to ensure that the sum of the density functions equals 1. In beta distribution, the parameters are $\alpha = n+1$ and $\beta = n+1$. It is worth noting that the variance of the beta distribution is defined as follows:

$$Var[X] = \frac{\alpha\beta}{(\alpha+\beta)^2(\alpha+\beta+1)}$$

in the given parameters we have,

$$Var[X] = \frac{(n+1)^2}{(2(n+1))^2(2n+3)}$$

$$Var[X] = \frac{1}{8n+12}$$

Hence, it can be demonstrated that as the depth increases, the variance progressively diminishes.

This property proves advantageous when profiling a subset of a numerical list. Instead of solely considering the smallest and largest numbers, opting for mid-range numbers arranged spatially provides more informative insights into the list. The specificity of mid-range numbers exceeds that of marginal numbers, as demonstrated by the entropy calculations for the corresponding beta distributions. For marginal numbers, the entropy is calculated with parameters $\alpha = 1$ and $\beta = 2n+1$, while for mid-range numbers, the entropy is calculated with $\alpha = n+1$ and $\beta = n+1$. Since the entropy of the latter is greater than that of the former, sampling from the mid-range numbers yields more substantial information.

In this study, we did not employ a binary min-max heap; however, our devised tree structure exhibits the aforementioned property. The algorithm does not explicitly focus on leaf nodes, but rather retrieves elements that are typically either leaves or deeper inner nodes. This is attributed to the high children-to-parent ratio within the constructed tree.

2.2.2 Spatial Indexing

A spatial index is a data structure designed to facilitate efficient spatial queries and operations. Many indexing systems lack support for multiple ordering, which is essential for spatial queries. Spatial indexes, on the other hand, are specifically designed to accommodate multiple ordering queries, as spatial information generally involves at least two dimensions. This characteristic enables the compact representation of spatially proximate data, ensuring that each entry has easily reachable neighborhoods.

Two main approaches are commonly employed in the construction of spatial indexes: the space-driven approach and the data-driven approach. In the space-driven approach, the spatial index is organized based on the space or coordinate system. The space is divided into blocks using regular grids or sets of cells, with data points allocated to their corresponding blocks. Although this approach is not data-driven, the efficient design of grid or cell divisions depends on the amount of data occupying each cell. Examples of this indexing structure include grid index and quadtree. On the other hand, the data-driven approach organizes the spatial index based on the data points themselves. The division of data blocks is determined by the characteristics of data distribution. In essence, the selection of tree nodes and their child-parent relationships in the index are influenced by the data's distribution. This approach is exemplified by R-tree and kd-tree. Both approaches offer advantages and disadvantages. The space-driven approach provides a consistent range at runtime, but its performance suffers when the data is non-uniform due to a lack of knowledge about the data. Conversely, the data-driven approach offers flexibility in accommodating different data distributions with good performance, but it incurs the cost of changing interval ranges and dynamic allocations, making it slower.

Spatial indexing finds its most common application in the field of computational geometry. It enables functionalities such as locating objects within specific areas, searching for the k nearest neighbors, and identifying intersections between objects. Although these problems are traditionally associated with computational geometry, other fields also adapt their problems to incorporate spatial or geometric aspects. For instance, a sequence mapping tool called Sigmap [17] transforms k-mers into k-dimensional points to identify spatially close k-mers, where closeness is determined by a small Hamming distance between the two k-mers.

In this study, we draw inspiration from spatial indexes to create a geometric space based on different lexicographical orders as its dimensions, using the data-driven approach.

2.2.3 Syncmers

Comparing entire long strings is both costly and impractical. To address this issue, the subsampling method is employed, wherein specific substrings are selected in a manner that promotes greater intersection among similar strings. Metrics such as Jaccard similarity prove useful in profiling strings by comparing sets. Subsampling methods can be classified into two categories: those using predefined substring sets, also known as universal hitting sets, and those that choose substrings by scanning the local portion of the string. These categories are referred to as context-free subsampling and context-dependent subsampling, respectively. Typically, subsamples are selected from k-mer samples, where k-mers represent substrings of length k within a string. For instance, the 3-mers of the string “CATTTGA” are “CAT”, “ATT”, “TTT”, “TTG”, and “TGA”.

Among the widely used subsampling methods is the *minimizer* approach. In this method, each k-mer is hashed to an integer value. For the context-free approach, k-mers with hash values smaller than a specified threshold are selected.

In the context-dependent approach, a window size is determined, and for each window in the string, the k-mer with the smallest hash value is chosen. The first approach ensures that if a subsample does not exist in a string, it truly does not occur in the string. This property helps avoid false negativity in string similarity analysis. However, this approach lacks a constraint on the distance between subsamples, which may result in overlooking specific substrings that do not contain any k-mer with a hash value below the threshold. The second approach ensures that the distance between two subsamples cannot exceed the size of the window. Nevertheless, it does not guarantee that the presence of a k-mer in a subsample of a string implies its presence in any other string containing that subsample. Another subsampling method, known as "syncmer", attempts to mitigate these issues.

Syncmer selects a k-mer based on the position of the s-mer (a subsequence of length s) within it, which possesses the minimum hash value. Syncmers have different versions, each satisfying specific conditions based on positional rules, but all versions depend on the parameters k and s. One version, called closed syncmer, has the smallest s-mer in either the starting or ending position. Syncmers can be calculated within a narrow range, making them context-free. Moreover, for sufficiently long strings, there will always be syncmers, thus overcoming the drawback of the context-free approach in the minimizer scheme. The term "long enough" is upper bounded by k, s, and the size of the alphabet. Given the small alphabet size in DNA sequences, this upper bound is small.

In this study, syncmer is not directly utilized; however, the employed frameworks imply that the selected blocks are analogous to syncmers, albeit without the specific constraint on the chosen k-mer size.

2.3 String Parsing Methods

A string parsing method is employed to partition a given string into mutually exclusive substrings that collectively cover the entire string.

In this section, we explore two distinct approaches to string parsing. The first approach, known as locally consistent parsing, focuses on generating substrings of balanced lengths while adhering to the constraint of avoiding periodic substrings within a specified length. On the other hand, the second approach, called Lyndon Factorization, leverages the relative positions of the substrings by adopting an idea derived from the algebraic concept.

2.3.1 Locally Consistent Parsing

There exist various constructions of Locally Consistent Parsing (LCP), and in our study, we utilize the version defined by the (τ, δ)-partitioning, as it offers better suitability for facilitating block investigations. The definition of (τ, δ)-partitioning is as follows:

Definition 2.3.1 *A set of positions $P \subseteq [n]$ is referred to as a (τ, δ)-partitioning set of string S, with $1 \leq \tau \leq \delta \leq n$, if and only if it satisfies the following properties:*

1. ***Local Consistency*** *- For any two indices $i, j \in [1 + \delta \ldots n - \delta]$ such that $S[i - \delta \ldots i + \delta] = S[j - \delta \ldots j + \delta]$, we have $i \in P \iff j \in P$.*
2. ***Compactness*** *- Let $p_i < p_{i+1}$ be two consecutive positions in $P \cup \{1, n+1\}$. Then, one of the following conditions holds:*
   (a) *(Regular block) $p_{i+1} - p_i \leq \tau$.*
   (b) *(Periodic block) $p_{i+1} - p_i > \tau$, and the substring $u = S[p_i \ldots p_{i+1} - 1]$ is an aperiodic string with a period length $\rho_u \leq \tau$.*

[18, 19]

This factorization provides a consistent seed, implying that if a block appears somewhere in the string, it will be present in every occurrence of the string.

Additionally, the blocks are uniformly distributed throughout the factorization, ensuring that the lengths of each block do not differ significantly in terms of magnitude.

2.3.2 Lyndon Factorization

The Lyndon Factorization is built upon the algebraic structure known as a *monoid*. A monoid is defined as follows:

Definition 2.3.2 *A monoid is a set M equipped with a binary operation that is associative and possesses a neutral element denoted by 1_M. [20]*

The set of all strings over an alphabet forms a monoid, where the binary operation is concatenation and the neutral element is the empty string. Hence, strings can be regarded as free monoids in relation to their algebraic counterparts.

Definition 2.3.3 *A factorization of the free monoid A^* is a family $(X_i)_{i \in I}$ of subsets of A^+ indexed by a totally ordered set I, such that every word $w \in A^+$ can be uniquely expressed as*

$$w = x_1 x_2 \cdots x_n$$

with $x_i \in X_{j_i}$, and

$$j_1 \geq j_2 \geq \cdots \geq j_n$$

[20]

This implies the existence of totally ordered sets of non-empty strings that can be uniquely combined in a monotonic order to generate any string. The total ordered sets are determined by Lyndon Words and the lexicographical order, defined as follows:

Definition 2.3.4 (Lyndon Word) *Lyndon Word intuitively can be defined as a word with minimum lexicographical order of all its rotations.*

More formally, a Lyndon word $l \in \Sigma^+$ is a word such that $l = uv$ with $u, v \in \Sigma^+$ implies that $l < vu$.

Theorem 2.3.1 *Every word $w \in \Sigma^+$ admits a unique factorization as a sequence of decreasing Lyndon words:*

$$w = l_1^{n_1} l_2^{n_2} \cdots l_k^{n_k} \tag{2.1}$$

where $n_i \geq 1$ and l_i is a Lyndon word for all i such that $1 \leq i \leq k$. [21]

Due to the decreasing order in the factorization, the likelihood of starting a new block decreases, resulting in larger blocks ordered by magnitude from left to right. Additionally, the blocks exhibit contextuality, where a block may not appear in every occurrence of the string but rather when it is consecutively repeated.

2.3.3 Comparison between Methods

Both LCP and Lyndon Factorization lack built-in recursive factorization for creating a min-max heap data structure. Although LCP has some versions created in a bottom-up fashion, both methods can be manipulated using hashing or order functions to achieve the desired outcome, which can be easily implemented.

The main distinction between LCP and Lyndon Factorization lies in their contextuality. LCP represents a context-free factorization, whereas Lyndon Factorization is context-sensitive. Context-sensitive systems establish matches between blocks that exhibit longer regions of similarity compared to context-free systems. However, context-free systems only need to be calculated once, whereas context-sensitive systems must be recalculated for each instance. It should be noted that both factorization techniques operate in linear time since scanning the sequence is necessary. Thus, the order in which the sequence is given does not affect the efficiency of the process. Consequently, LCP is more suitable when dealing with streaming data, whereas Lyndon Factorization excels in cases where the seeds are more sensitive and require a broader region representation. Given that whole genome alignment does not require streaming capabilities and prioritizes sensitivity, this work adopts Lyndon Factorization as the chosen string parsing method.

2.4 Run-Length Encoding

The definition of Run-Length Encoding(RLE) as follows:

Definition 2.4.1 (Run-Length Encoding) *Run-Length Encoding is a lossless data compression method that stores runs as a data value and counts, where a run refers to a sequence with only one data value.*

The run time of the Run-Length Encoding (RLE) algorithm is linear, as it processes each character in the encoding individually, and the time complexity is proportional to the length of the input. RLE's effectiveness in compression depends heavily on the data's characteristics. Consequently, the resulting compression ratio can vary significantly and can take on any positive value. This means that RLE can produce highly compressed data, but it can also result in data that is larger than the original.

It is worth noting that an alternative data structure called the FM-index can be utilized for data sets with low entropy, such as a book written in a natural language [22]. The FM-index is intelligently designed to efficiently handle long runs of repeated characters, leading to higher data compressibility. This data structure takes advantage of the inherent redundancy in natural language texts, allowing for more effective compression techniques to be applied.

RLE is particularly advantageous when working with data containing repetitive letters or when sequencing data contains repetitive regions known as homopolymers. In the context of Lyndon factorization, excessive repetitiveness results in a high number of blocks, leading to inefficiency. By eliminating repetitions, RLE proves to be a valuable tool for improving the efficiency of our algorithm.

Chapter 3

Methods

In this thesis, we present Idoneous, an algorithm designed for whole genome alignment with target mapping length and error rate considerations. The algorithm encompasses various stages, including preprocessing of the sequence data, construction of characteristic data structures, which take the form of specialized trees, and a mapping procedure that involves traversing these trees.

To facilitate efficient synchronization between seeds and chaining, we introduce a novel data structure called the "Alternating Lyndon Factorization Tree" (ALFTree). This data structure incorporates spatial lexicographical information, enabling seamless coordination between seeds. Furthermore, the ALFTree offers the capability of approximate string matching with specific substring length ranges and varying edit distance approximation rates. This feature proves advantageous in scenarios where adjustments to sensitivity and specificity are required. A comprehensive definition and explanation of the ALFTree can be found in subsection 3.1.3.

3.1 The Idoneous Algorithm

The Idoneous algorithm includes the following steps:

1. Alphabet transformation through Run-Length Encoding (RLE).
2. Recursive Lyndon factorization with distinct lexicographical orders to facilitate tree construction.
3. Generation of traversal intervals for both trees.
4. Identification of matches for each interval pair using the seed-chain-filter framework.
5. Sanity check for matches through alignment between the identified matches.

Figure 3.1 illustrates an overview of the entire process.

3.1.1 Idoneous Parameters

The Idoneous algorithm depends on the following parameters:

- block_size (b): minimum size of an inner node in the ALFTree.
- chaining_threshold (c): minimum number of seeds required to build a valid chain.
- division_length (d): approximate distance between the starting positions of two intervals in a sequence.
- division_window (a): number of division traversed in one matching step.
- max_indel_rate (e): maximum indel rate (used as $1/e$) between two link lengths, where a link length corresponds to the distance between two consecutive seeds.

- k-mer size (k): length of k-mers at the starting positions of the blocks for comparisons.

We explain these parameters further while presenting the details of Idoneous.

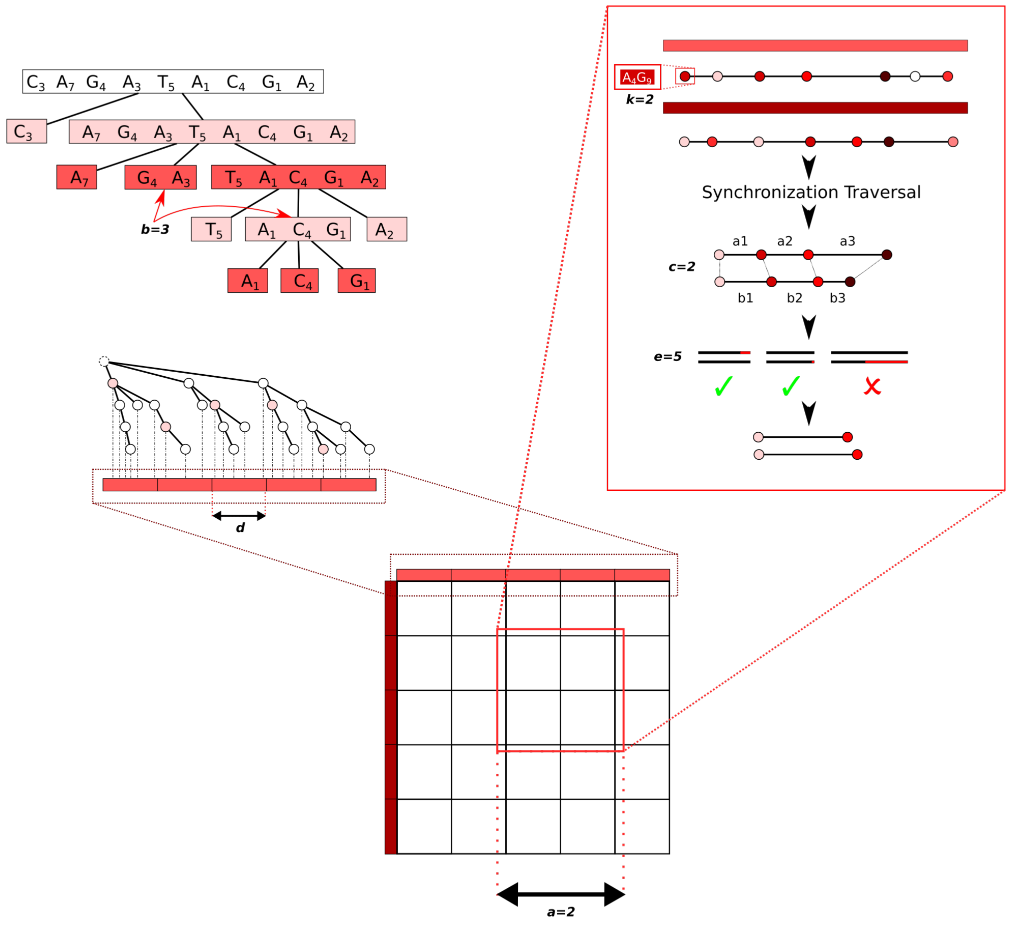

Figure 3.1: Overview of Idoneous: Construction of tree for the string "$C_3A_7G_4A_3T_5A_1C_4G_1A_2$", Generation of the traversal intervals for a tree, Identification of matches in a square for each interval pair of red and maroon rectangles, shown in detail which includes synchronization traversal, chaining and filtering to get matches

3.1.2 Alphabet Transformation

The primary objective of the alphabet transformation is to alleviate the presence of repetitive Lyndon words, such as homopolymers like "AAAA", which would otherwise result in factorizations like “A”, “A”, “A”, “A” and lead to a crowded and inefficient tree structure. Additionally, this transformation offers several advantages, including lossless compression, maintaining a nearly consistent comparator with the original alphabet, and facilitating faster comparisons compared to the original alphabet.

In the case of aligning strings with the four-letter alphabet A, C, G, and T, we perform an alphabet transformation using Run-Length Encoding (RLE) to convert it into a 253-letter alphabet. The maximum allowed run length in this transformation is 63. Each letter in this transformed alphabet is represented by the original letter concatenated with its corresponding run length. For example, “AAAA” becomes “A_4”. However, it is important to note that we cannot use a run length of 64 due to the requirement of an end character.

The lexicographical order of the transformed alphabet depends on the lexicographical ordering of the original four-letter alphabet and the run length. The ordering of the original alphabet is preserved, and longer run lengths are assigned smaller lexicographical orders. Thus, the canonical ordering is as follows:

$$A_{63} < A_{62} < \cdots < A_1 < C_{63} < \cdots < C_1 < G_{63} < \cdots < T_2 < T_1$$

While this canonical ordering is suitable, we require at least two different lexicographical orders to alternate the factorization. Remarkably, achieving this requirement is possible by permuting the original alphabet, resulting in a total of 24 possible permutations. This “reverse canonical order” is formulated as follows:

$$T_{63} < T_{62} < \cdots < T_1 < G_{63} < \cdots < G_1 < C_{63} < \cdots < A_2 < A_1$$

The formal definition of the lexicographical order transformation is as follows, from the general alphabet Σ to Σ_{rle} with a maximum run length of max:

For any total order in Σ, there exists another total order in Σ_{rle} that satisfies the following conditions:

- $\forall X \in \Sigma$ and $\forall i, j$ such that $0 < i < j \leq max$, we have $X_i >_{rle} X_j$
- $\forall X, Y \in \Sigma$ and $\forall i, j$ such that $X < Y$, we have $X_i <_{rle} Y_j$

3.1.3 Tree Construction via Alternating Lyndon Factorization

The Lyndon factorization offers the advantage of uniquely partitioning a string into blocks that are confined to specific lexicographical Euclidean subspaces. This partitioning provides valuable information regarding the potential presence of a given k-mer within these blocks. However, due to the uniqueness of the Lyndon factorization, it becomes impossible to parse a Lyndon word into other Lyndon words without altering the lexicographical order. Moreover, it is essential to ensure that the parsed blocks are as small as possible, imposing an efficiency constraint. To address these requirements, we propose the design of a tree structure known as the Alternating Lyndon Factorization Tree.

Definition 3.1.1 (Alternating Lyndon Factorization Tree) *The Alternating Lyndon Factorization Tree represents a string using two distinct lexicographical orders and adheres to the following conditions:*

- *The root node represents the entire string.*
- *All nodes, except the root node, correspond to Lyndon words according to their depth in the tree.*
- *A leaf node is reached when the length of the represented string falls below a specified threshold, denoted as b (the block size).*
- *The children nodes of a given node represent the Lyndon factorization of the string using the alternative lexicographical order.*

This tree structure enables the organization of Lyndon words within a hierarchical framework, allowing for efficient processing and exploration of the factorized blocks in a controlled manner. In Figure 3.2, construction of the ALFTree for a given string is demonstrated.

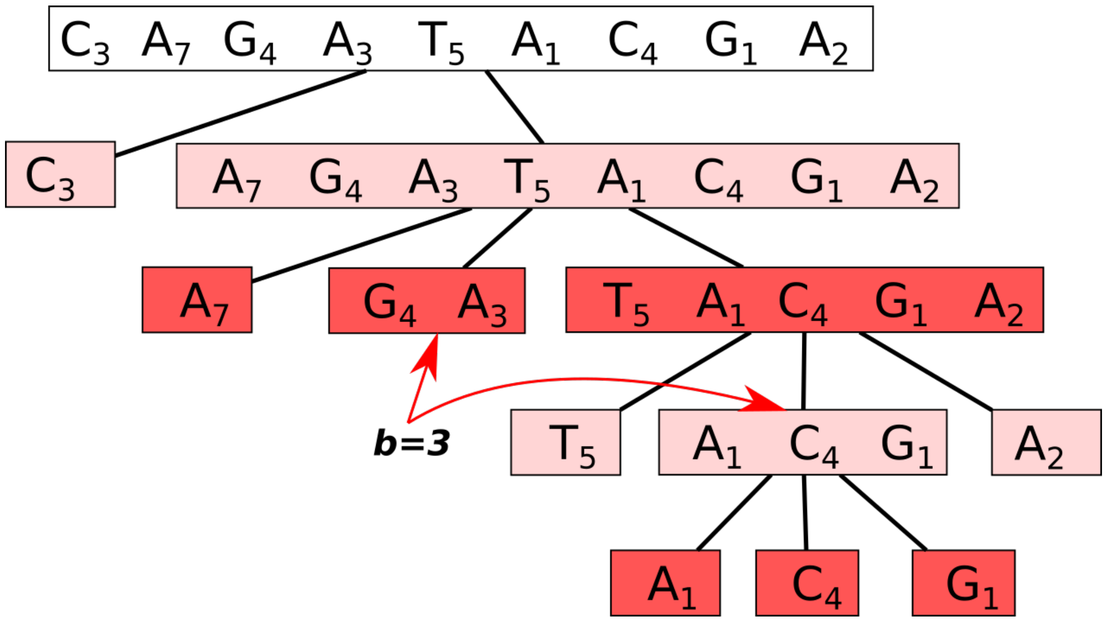

Figure 3.2: Example of Alternating Lyndon Factorization Tree for the string "$C_3A_7G_4A_3T_5A_1C_4G_1A_2$" with canonical and reversed canonical lexicographical order where b = 3. The white block is the root. Pink and Red blocks are Lyndon word blocks for different lexicographical orders.

The construction of the Alternating Lyndon Factorization Tree can be achieved with time complexity of $O(Nh)$ by employing the recursive implementation of Duval's algorithm [23], where N represents the length of the string and h denotes the depth of the tree. Furthermore, the space complexity of the tree is determined to be $O(N \cdot \log(b)/b)$, taking into account that within the range of each block of size b, there can only exist a maximum of two inner nodes. On average, an inner node is associated with $log(b)$ leaf nodes.

The pseudocodes for the tree construction algorithm and Duval's algorithm are given in Algorithm 1 and Algorithm 2 respectively.

```
**Data:** String S, Block Size b, lexicographical order ord
**Result:** Alternating Lyndon Factorization Tree of S
Create Node N represents S; if $Size(S) > b$ then
    Factors ← Duval's Algorithm(S, ord); foreach $factor \in Factors$ do
        add TreeConstruct($factor$, b, alternate(ord) ) as Child to N;
    end
end
return N;
```

Algorithm 1: Tree construction algorithm (TreeConstruct)

```
**Data:** String S, lexicographical order ord
**Result:** Lyndon Factorization of S
Set comparators according to ord;
$n \leftarrow size(S)$;
for $l \leftarrow 0$ to $n$ do
    $r \leftarrow l$;
    $p \leftarrow l + 1$;
    while $r < n$ do
        if $s[r] > s[p]$ then
            break;
        else if $s[r] = s[p]$ then
            increment r and p;
            continue;
        else
            $r \leftarrow l$;
            increment p;
            continue;
    end
    while $l \leq r$ do
        report $s[l \cdots l + p - r]$;
        $l \leftarrow l + p - r$;
    end
end
```

Algorithm 2: Duval's Algorithm

3.1.4 Generating Traversal Intervals

As the synchronization and matching procedures operate in a linear manner on the tree, it becomes challenging to identify all reversed matches between two strings. For instance, when considering the strings AB and ba, with corresponding similar substring pairs A-a and B-b, only one of the pairs, either A-a or B-b, can be detected as a match. This limitation arises due to the inability of the traversal process to move backward on the tree. In order to overcome this issue, we introduce the concept of generating smaller intervals. To achieve this, we select a parameter, referred to as the division length, denoted as d, which enables the generation of intervals for each string. These intervals are obtained by dividing the original intervals of approximately size d, with the selection of the first inner node encountered after a distance of d. An ALFTree and representation of the intervals is illustrated in Figure 3.3.

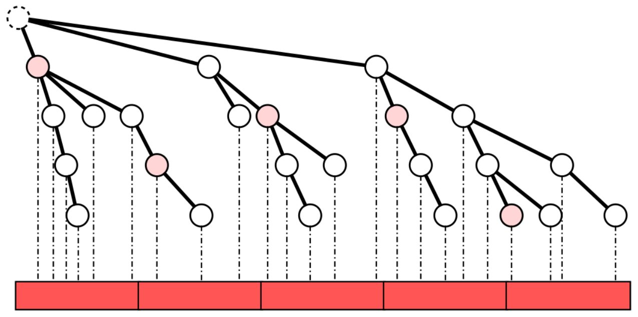

Figure 3.3: An Alternating Lyndon Factorization Tree. The dash-line node represents the root node, and only the inner nodes are depicted, each governed by their respective reference position. The starting node of each interval is distinguished by a distinct color marking.

3.1.5 Matching Framework

The matching framework operates on the generated interval pairs, taking them as input and producing corresponding approximate matches as output. This framework encompasses three key procedures: seeding, chaining, and filtering. In the seeding procedure, seeds are identified by synchronizing the intervals through tree traversal, employing k-mer and length comparisons node-by-node. Once the seeds are identified, chains are constructed by determining the distances between these seeds. The chaining procedure incorporates two parameters: c, representing the minimum number of seeds required for a valid chain, and e, which serves as a threshold to restrict the number of indels between two seeds. If the difference between the interval pair exceeds the average length by a value greater than e^{-1}, the interval pair is considered erroneous. Subsequently, the filtering procedure is employed to eliminate shorter chains and erroneous matches. The remaining chains represent regions that serve as matches within the entire process. This entire process is shown in Figure 3.4.

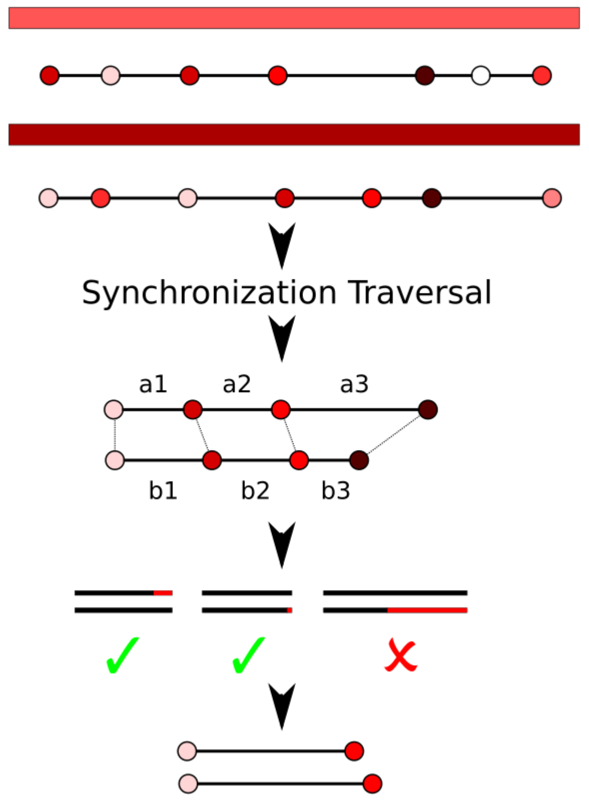

Figure 3.4: An illustration exemplifying the matching framework. The rectangles in the figure represent the generated intervals, while the colored circles denote the tree nodes, with different colors indicating different k-mers. The dashed lines indicate the identified seeds. The bold black lines represent the intervals, and the bold red lines indicate the difference between these intervals. The chosen parameters for this example are $c = 2$ and $e = 5$.

3.1.5.1 Synchronization Traversal

The synchronization traversal algorithm is given in Algorithm 3 with the advance function is responsible for identifying the next inner node with the closest position within the tree.

The synchronization traversal process is characterized by its lack of symmetry. However, owing to the consistent and negative correlation between node size

```
Data: Node n , Node m
Result: synchronized nodes
while $n.kmer \neq m.kmer$ do
    if $n.kmer > m.kmer$ XOR $n.size > m.size$ then
        Advance(n) ;
    else
        Advance(m) ;
end
return n and m ;
```

Algorithm 3: Synchronization Traversal

and lexicographical order, nodes with similar sizes that exhibit the same k-mer ordering will achieve synchronization. While not readily apparent, this can be observed by considering a simplified scenario involving the comparison of a single metric. It becomes evident that the k-mers are ordered through Lyndon factorization, assuming a lack of depth, resulting in two lists of decreasing numbers. By continuously advancing the larger element, the position where two lists share the same element can be determined. This method functions analogously to merge sort.

In the case of two-dimensional traversal, the process becomes more intricate. If Node n is positioned higher in the tree, it will initially become deeper. If Node m is at a deeper point, synchronization will either occur, or one of the conditions will become false. In the former case, synchronization is achieved. In the latter case, Node m goes deeper again, resulting in a zigzag pattern to facilitate synchronization. If Node m is located at a higher point, Node n continues to traverse deeper until it reaches a node that is higher than Node m. At this point, the previous condition is fulfilled, resolving the situation. Since a total order is lacking, there is no guarantee that synchronization will encompass all nodes. However, when a locally similar list of nodes is present, synchronization proves effective.

The nature of synchronization traversal gives rise to gaps preceding matches in the initial regions of the intervals. To mitigate these gaps, instead of halting at the first interval, we traverse a certain number of intervals. This parameter is denoted as a.

3.1.6 Alignment using the Needleman–Wunsch Algorithm

To perform a sanity check on the matches, we used the Edlib[24] library to calculate global alignment. If the alignment distance of a matched region exceeds a predefined threshold, we filter out the match without attempting to align the entire region.

Chapter 4

Experimental Methodology

This part includes data, criteria, coverage & distance results, and run time results.

4.1 Data

We first tested the accuracy of Idoneous and compared it with MUMmer and NUCmer using the two highest quality human genomes: the human reference genome (GRCh38) [25, 26] and the complete human genome assembly released by the Telomere-to-Telomere Consortium (T2T-CHM13) [27]. We then tested Idoneous by comparing several *diploid* draft human genome assemblies to GRCh38 (Table 4.1). Finally, we used Idoneous to generate WGA of human and chimpanzee genomes. We only tested Chromosome 22 of each genome, it is easy to extend this to the whole genome since each chromosome can be identified.

4.2 Criteria

We evaluated Idoneous, MUMmer, and NUCmer under three criteria: coverage, distance, and run time. Coverage represents the extent to which the genome

Table 4.1: Genome assemblies used to test Idoneous.

Name	**Species**	**No. of Scaffolds***	**Assembly Length***	**Ungapped Length**	**Diploid**	**Citation**
GRCh38	Human	36	50,818,468	39,159,782	No	[25, 26]
T2T-CHM13	Human	1	51,454,416	51,454,416	No	[27]
HG002-maternal	Human	1	542,94,140	48,944,140	No	[28]
HG002-paternal	Human	1	50,257,608	48,707,608	No	[28]
panTro6	Chimpanzee	4	33,698,415	33,420,349	No	[5]

* All tests are conducted on chromosome 22 only.

is mapped to the other genome. Distance measures the average ratio of edit distance between two matched seeds per nucleotide, providing an assessment of the alignment of the entire genome. Run time denotes the amount of CPU time consumed by the algorithm using a single thread. In the first criterion, a higher value is indicative of better performance, while for the latter two metrics, a lower value indicates superior results.

4.3 Coverage & Distance Results

This section presents the coverage results obtained by manipulating various parameters and offers comparisons across different data sets.

4.3.1 Effects of Parameter Selection

This section focuses on the analysis of coverage and distance results pertaining to the parameters: *division window, block size, chaining threshold, division length, max indel rate,* and *k-mer size.*

4.3.1.1 Division Window (a)

The *division window* parameter demonstrates a positive correlation with coverage, wherein an increase in its value leads to an extended potential matching range. As a result, the ratio of the synchronization gap decreases, as the synchronization gap itself remains unaffected by the incremental adjustments to the division window. Empirical experiments provide evidence of a sublinear connection between the augmentation of this parameter and the corresponding rise in coverage, as visually represented in Figure 4.1. Moreover, Figure 4.2 depicts a downward trend in distance.

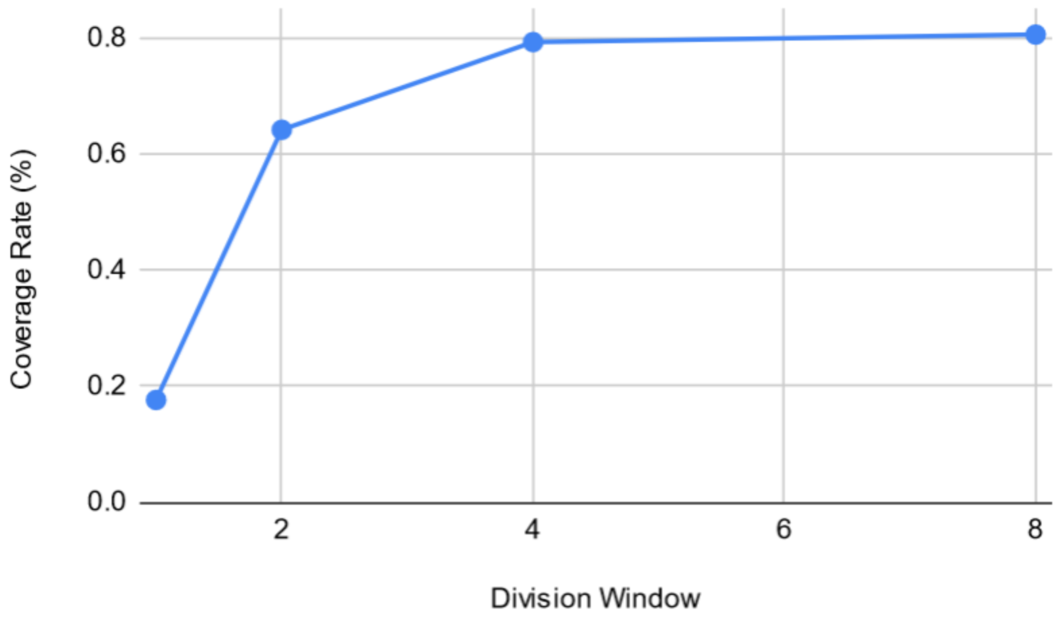

Figure 4.1: Relationship between the Division Window and coverage rate with parameters b = 2000, c=2, d=8000, e= 200, and k=2

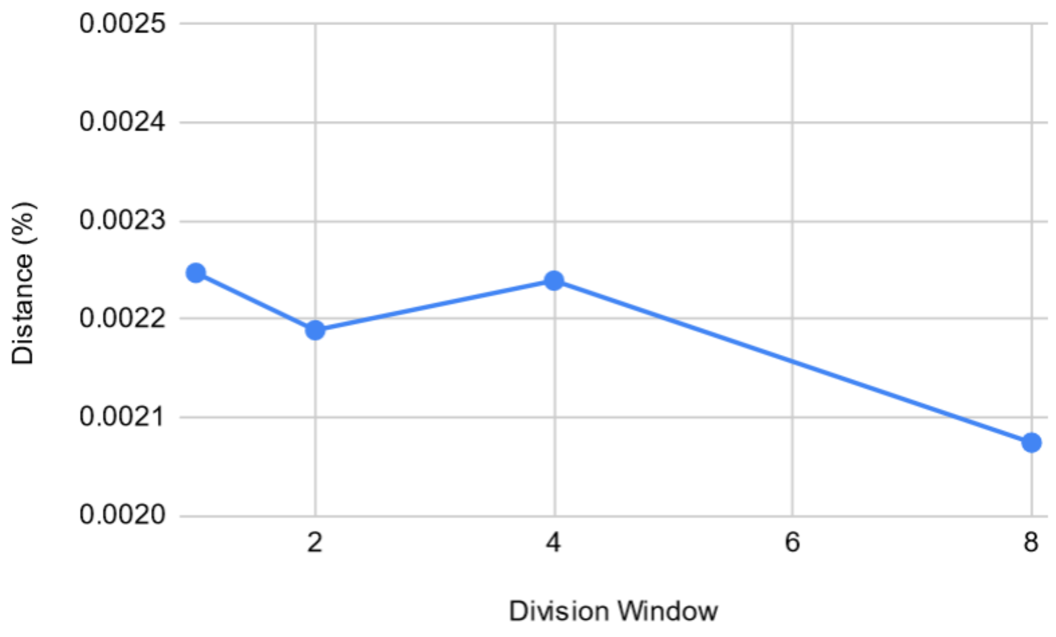

Figure 4.2: Relationship between the Division Window and distance with parameters b = 2000, c=2, d=8000, e= 200, and k=2

4.3.1.2 Block Size (b)

The *block size* can serve as an indicator of the Idoneous algorithm's level of granularity, where a reduction in block size prompts the algorithm to search for finer regions in order to identify matches. Consequently, smaller matches can be detected within smaller block sizes, while larger, coarser regions may fail to yield any matches. Therefore, a smaller value of the block size parameter (b) enables a broader coverage of regions. Furthermore, due to the exclusion of unnecessary unmatched neighborhoods within smaller regions, their corresponding distances are also minimized. This observation is supported by the findings presented in Figure 4.3, which illustrate a decline in the coverage rate as the block size increases, aligning with the underlying theory. Additionally, Figure 4.4 demonstrates a positive correlation between the block size and the distance, indicating a positive relationship.

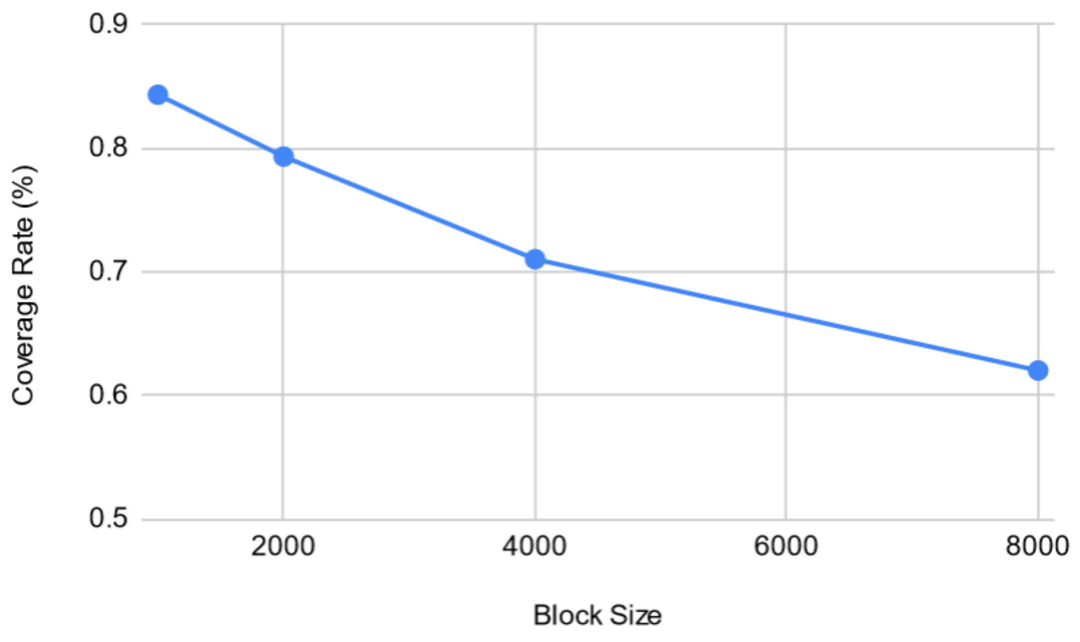

Figure 4.3: Relationship between the block size and coverage rate with parameters a = 4, c=2, d=8000, e= 200, and k=2

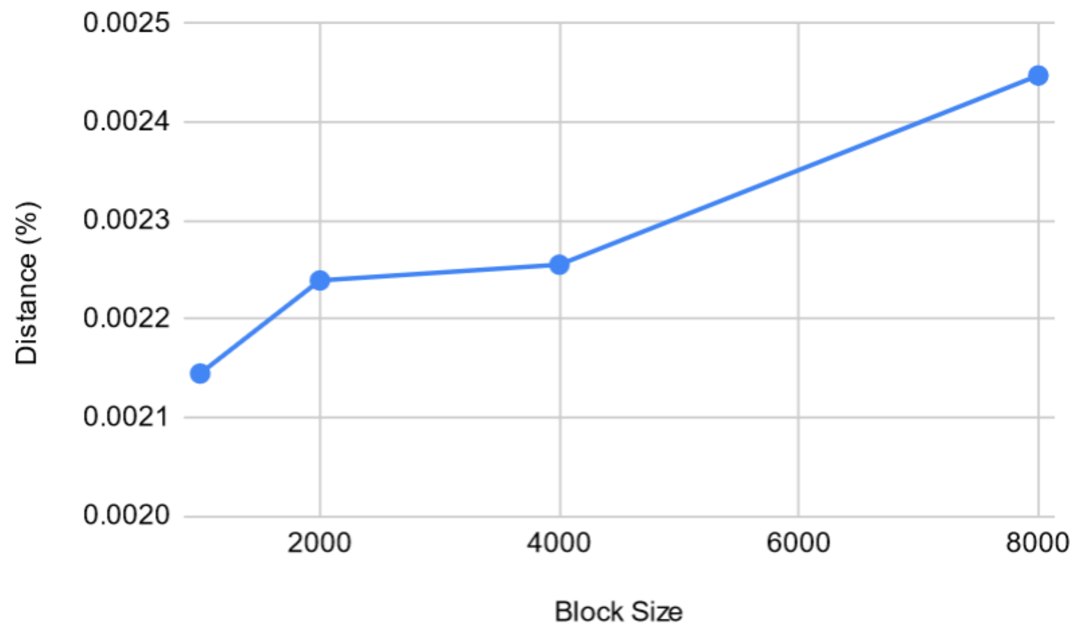

Figure 4.4: Relationship between the block size and distance with parameters a = 4, c=2, d=8000, e= 200, and k=2

4.3.1.3 Chaining Threshold (c)

The *chaining threshold* can be viewed as a geometric trail that offers a reduced false positive rate. As it necessitates more precise matches in the form of sequential seeds, the coverage rate diminishes with longer chaining. This inverse relationship between coverage and the chaining number is illustrated in Figure 4.5. Conversely, the reduction in false positive occurrences leads to a decrease in the distance between seeds. This phenomenon is demonstrated in Figure 4.6.

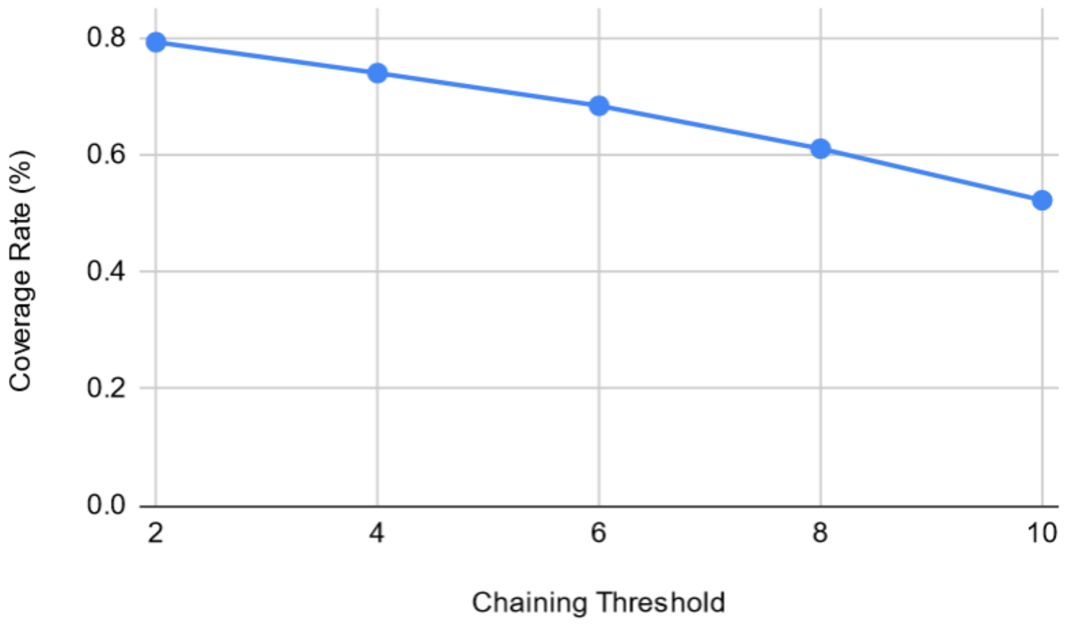

Figure 4.5: Relationship between the chaining number and coverage rate with parameters a = 4, b=2000, d=8000, e= 200, and k=2

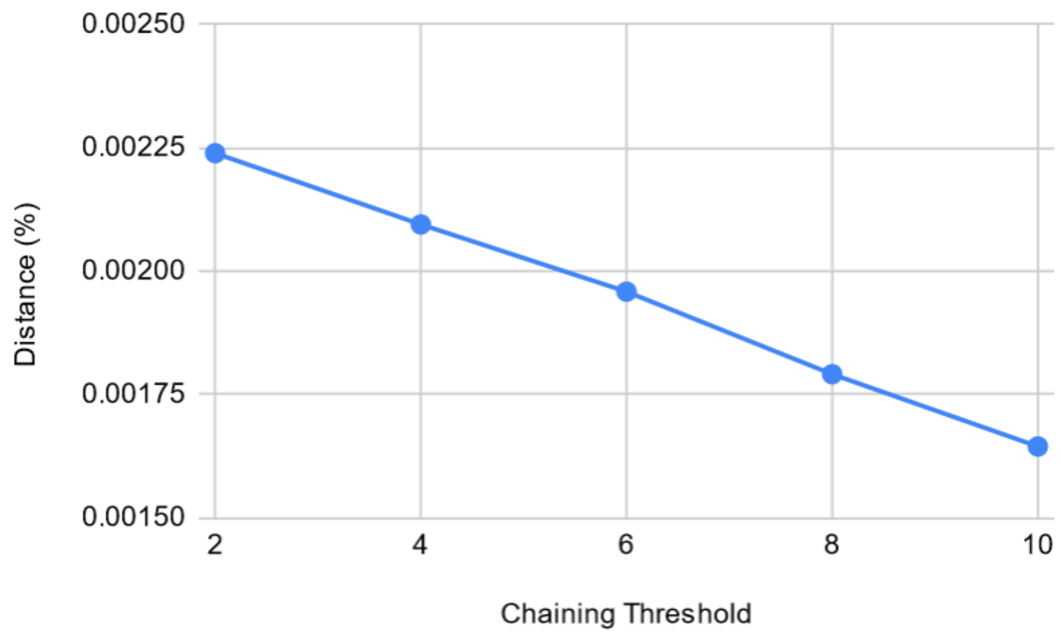

Figure 4.6: Relationship between the chaining number and distance with parameters a = 4, b=2000, d=8000, e= 200, and k=2

4.3.1.4 Division Length (d)

Similar to the block size, the *division length* also plays a crucial role in determining the Idoneous algorithm's granularity. A larger division size leads to a coarser analysis, resulting in a decrease in the coverage rate. This decline can be attributed to the same underlying reason as the block size, where the algorithm may fail to identify smaller matches within larger divisions. Consequently, there is an increasing trend in the distance between matches as the division size expands. These findings are consistent with the experimental results presented in Figure 4.7 and Figure 4.8.

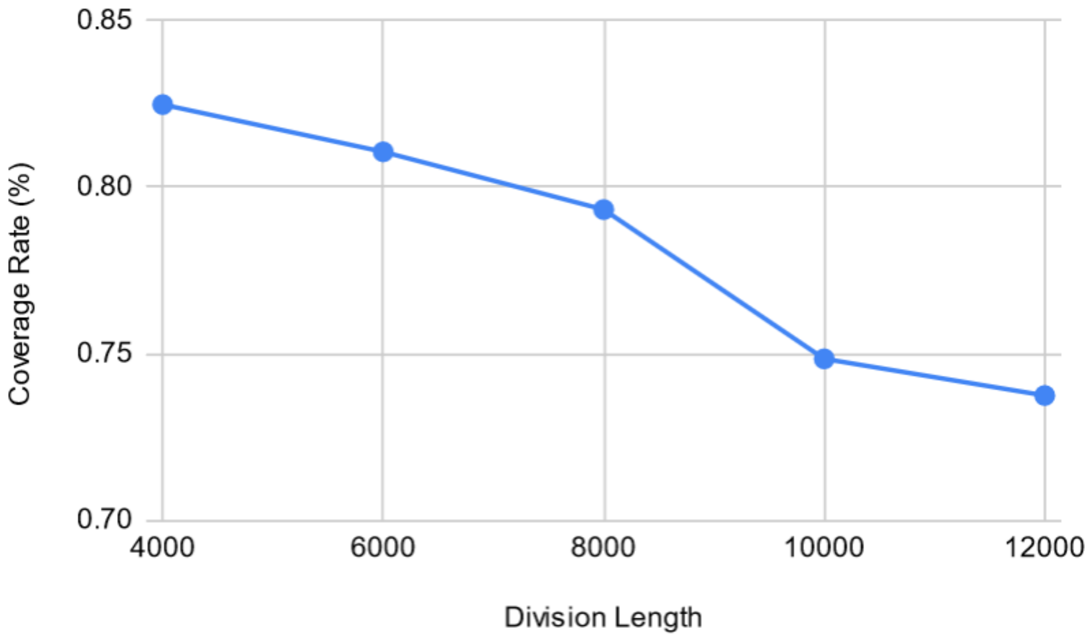

Figure 4.7: Relationship between the division size and coverage rate with parameters a = 4, b=2000, c=2, e= 200, and k=2

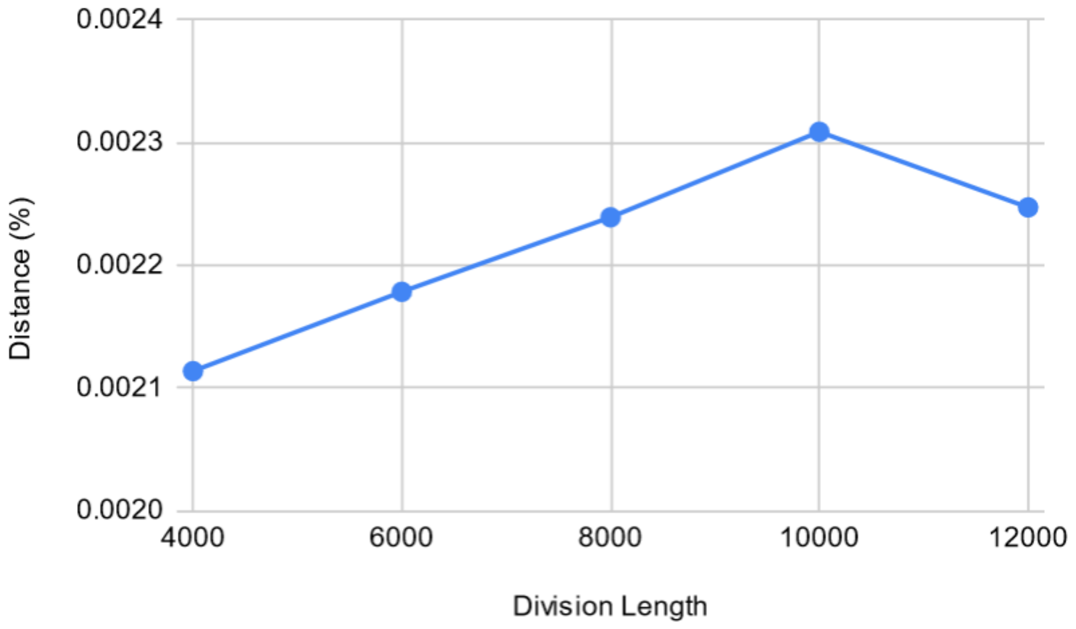

Figure 4.8: Relationship between the division size and distance with parameters a = 4, b=2000, c=2, e= 200, and k=2

4.3.1.5 Max Indel Rate (e)

The *max indel rate* serves as an additional filtering parameter to reduce false positives. When the max indel rate, denoted as e, is increased, the coverage decreases due to the requirement for accepted matches to exhibit greater similarity. Similarly, the distance between matches decreases for the same reason. As anticipated, the experimental results shown in Figure 4.9 and Figure 4.10 clearly demonstrate an inverse relationship between the coverage rate and the distance, with respect to the parameter e.

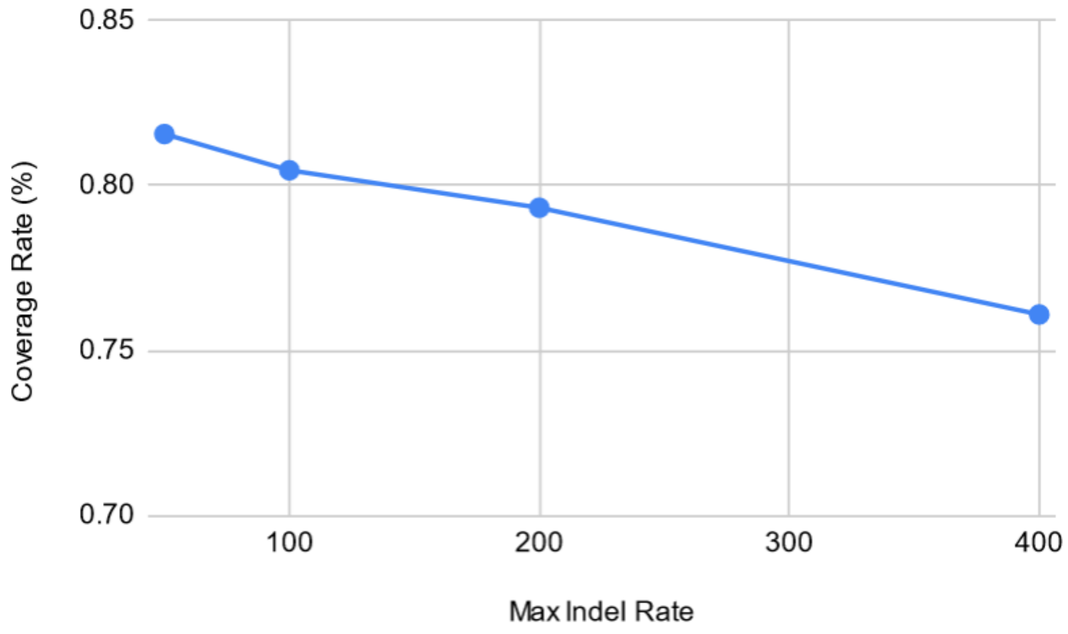

Figure 4.9: Relationship between the error rate and coverage rate with parameters a = 4, b=2000, c=2, d= 8000, and k=2

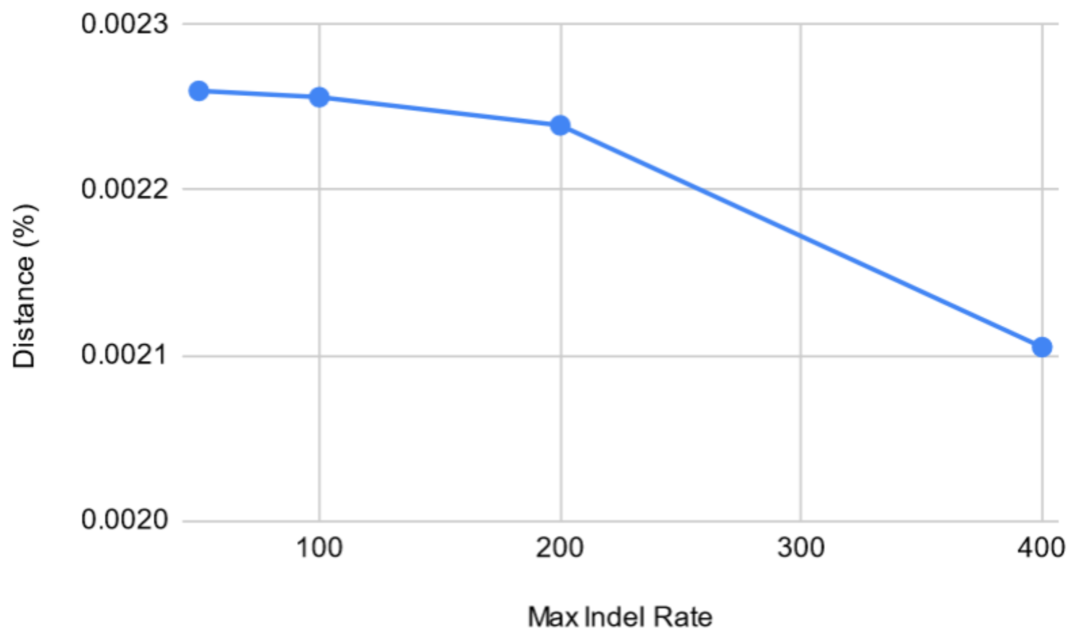

Figure 4.10: Relationship between the error rate and distance with parameters a = 4, b=2000, c=2, d= 8000, and k=2

4.3.1.6 K-mer Size (k)

Theoretically, an increase in the *k-mer size* enhances the specificity of the algorithm, leading to a lower coverage rate and shorter distance between matches. However, due to the incorporation of chaining and spatial distance checks, these metrics remain constant for reasonable k-mer sizes. In an alternative experimental setup, the behavior of the k-mer size would resemble that of parameters c and e. This constancy is visually depicted in Figure 4.11 and Figure 4.12, illustrating the consistent coverage rate and distance.

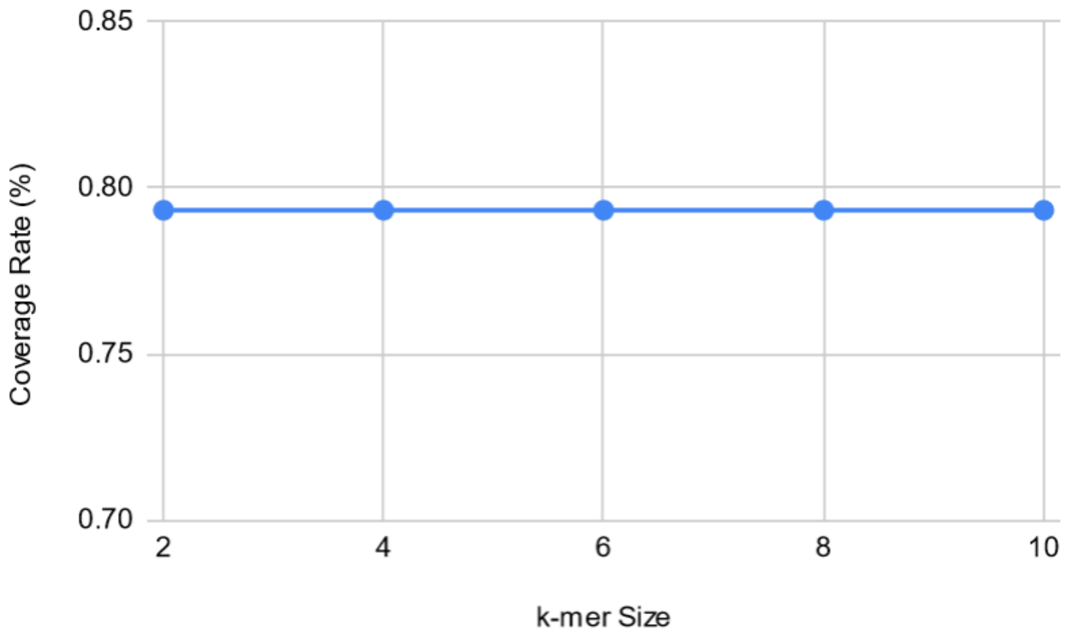

Figure 4.11: Relationship between the k-mer size and coverage rate with parameters a = 4, b=2000, c=2, d= 8000, and e=200

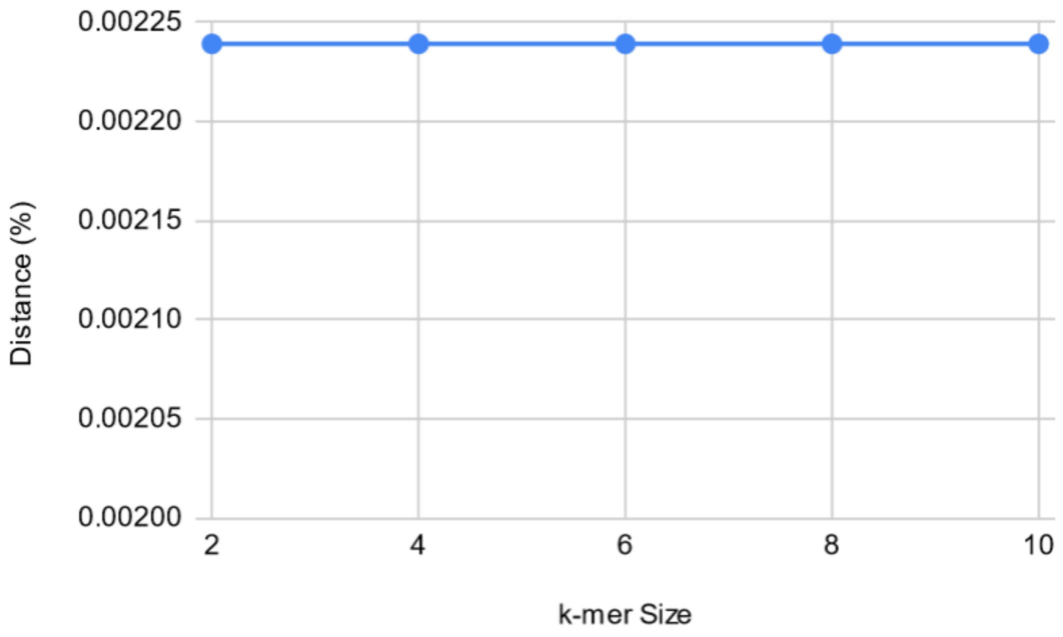

Figure 4.12: Relationship between the k-mer size and distance with parameters a = 4, b=2000, c=2, d= 8000, and e=200

4.3.2 Different data sets

In this section, GRCh38 assembly aligned with T2T-CHM13 assembly, two haploids of a human genome namely HG002-maternal and HG002-paternal genomes, and panTro6 genome assembly. Consistency between results and coverage of each algorithm examined. Coverage of each tool for each data set is shown in 4.2.

Table 4.2: Number of covered nucleotides in GRCh38 assembly against given data set

data set	Idoneous	MUMmer	NUCmer
T2T-CHM13	30,367,599	24,350,168	38,280,735
HG002-maternal	31,339,114	24,350,168	38,157,637
HG002-paternal	29,922,355	18,125,340	38,254,675
panTro6	757,787	1,085,841	32,097,210

For the human vs human alignment, our algorithm lies between MUMmer and NUCmer in terms of coverage. For the human vs chimpanzee alignment, our algorithm has not a good performance and it is a negative outlier.

4.3.2.1 GRCh38 vs T2T-CHM13

To generate the alignment between GRCh38 and T2T-CHM13 assembly, we used the following parameters for the Idoneous algorithm: $a = 4, b = 2000, c = 2, d = 8000, e = 200$ and $k = 2$. In this experiment we observed 78% percent coverage. Figure 4.13 shows that near position chr22:20,000,000 there is a translocation event.

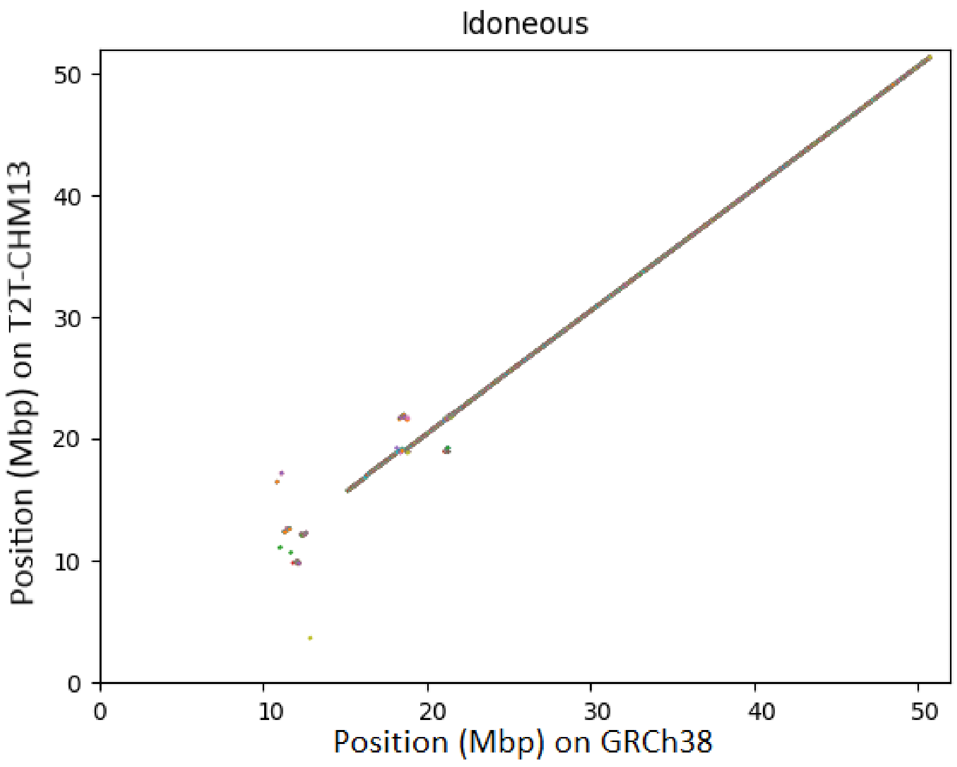

Figure 4.13: Regions of homology found between GRCh38 and T2T-CHM13 using Idoneous

We found almost no matches before the position chr22:15,000,000. We observed that close to this region, there exist tandem repeats that cannot be detected by Idoneous algorithm since repetitive regions do not have sufficient number of blocks. Additionally, before this region, most of the nucleotides (10.5 Mbp of 15 Mbp) are unknown (i.e., N) in the GRCh38 assembly since human chromosome 22 is acrocentric.

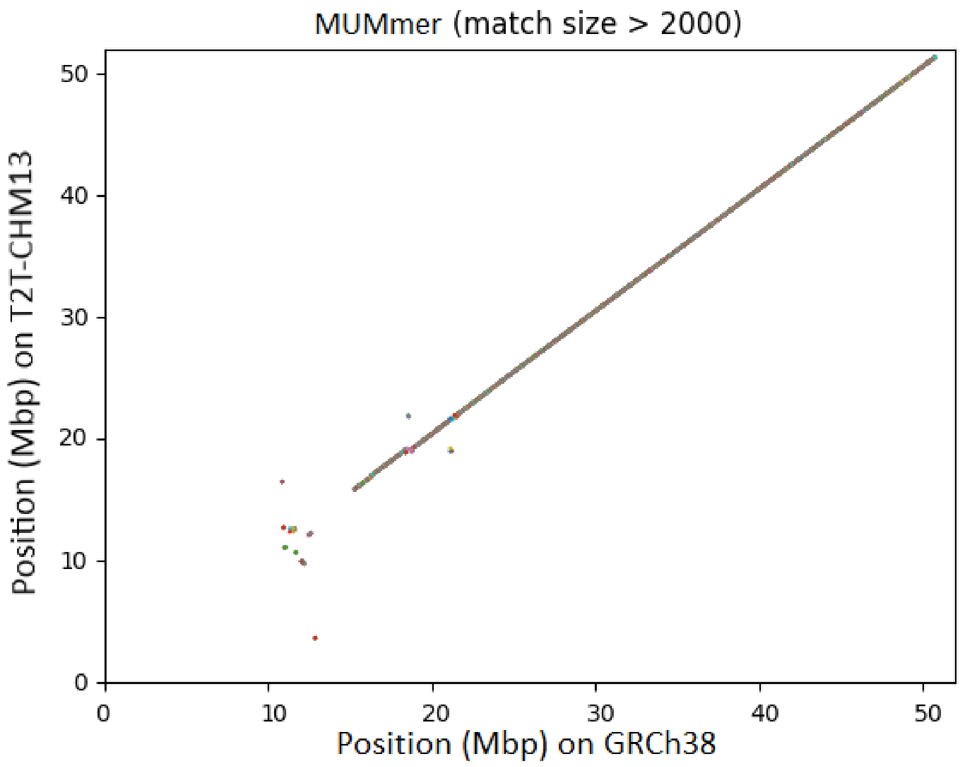

Figure 4.14: Regions of homology found between GRCh38 and T2T-CHM13 using MUMmer

To ensure fairness in comparisons, we applied a filter to eliminate matches from MUMmer that were smaller than 2000, as parameter b in the Idoneous algorithm prevents the detection of such matches.

In the alignment depicted in Figure 4.14, comparable outcomes are observed in comparison to the previous alignment. The algorithm's coverage stands at 62%, indicating the presence of dense gaps between seeds. Notably, Figure 4.14 does not exhibit any significant deviation in terms of gap size when contrasted with the previous alignment.

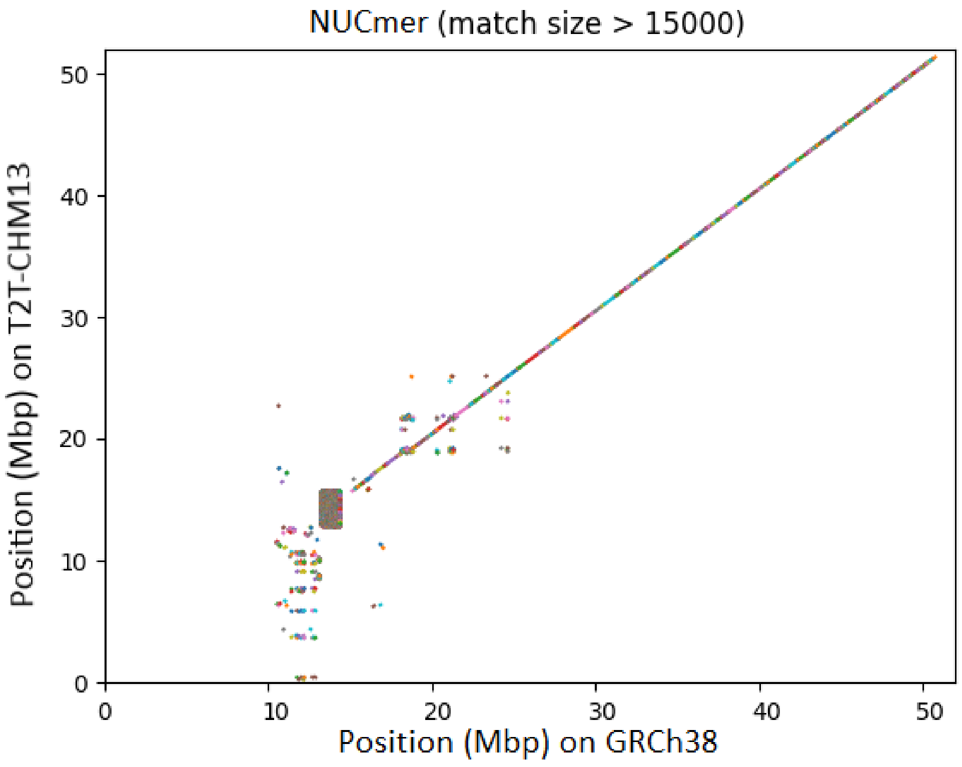

Figure 4.15: Regions of homology found between GRCh38 and T2T-CHM13 using NUCmer

Within the framework of the NUCmer tool, an extra filtering step is implemented to accommodate the presence of a gap-filling procedure, which is inherent to the NUCmer algorithm. This particular procedure mandates the presence of at least two matching regions in order to fill a gap. Specifically, in the Idoneous algorithm, the regions are required to possess a minimum length of $b * (c + 1)$, which corresponds to 6000 in the context of this experiment. Additionally, it is worth noting that the average compression factor of the run-length encoding (RLE) in the data set amounts to approximately 1.4. Consequently, the selection of a length of 15 Kbp can be considered adequately justified.

Figure 4.15 demonstrates an increased number of matching regions. Notably, a distinct match is observed in this algorithm, characterized by a solid rectangle centered around position chr22:14,000,000, which signifies the presence of tandem repeats. Furthermore, additional regions exhibit clustering along either the horizontal or vertical directions, suggesting the existence of structural variants, such as duplications. Remarkably, the algorithm achieves nearly complete alignment, resulting in a coverage of 98%.

4.3.2.2 GRCh38 vs HG002-maternal

To generate the alignment between GRCh38 and the HG002-maternal genome, we used the same parameters as above for the Idoneous algorithm and we observed similar results as the GRCh38 - T2T-CHM13 comparison.

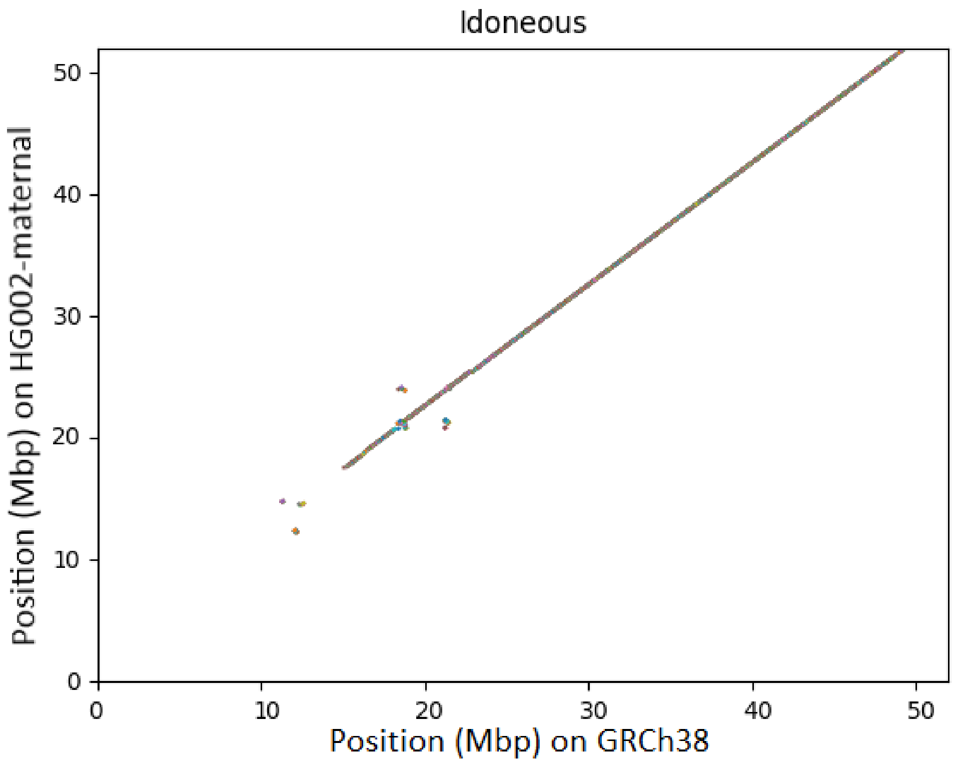

Figure 4.16: Alignment between GRCh38 and HG002-maternal genome in Idoneous

Figure 4.16 reveals the presence of similar variations, such as translocations.

Nevertheless, several notable distinctions arise in this alignment, including a larger coordinate bias and a reduced number of matches compared to the previous alignment. The decrease in matches can be attributed to the presence of unknown regions within this genome, as it has not been fully characterized like the T2T-CHM13 assembly. However, it is worth noting that despite these differences, the experimental results indicate a coverage of 80%. This observation suggests that, in the well-assembled regions, this assembly exhibits a higher degree of similarity compared to the T2T-CHM13 assembly. Note that neither HG002-maternal nor the HG002-paternal genome assemblies can be considered complete, and as of the writing of this thesis they remain as high quality draft assemblies.

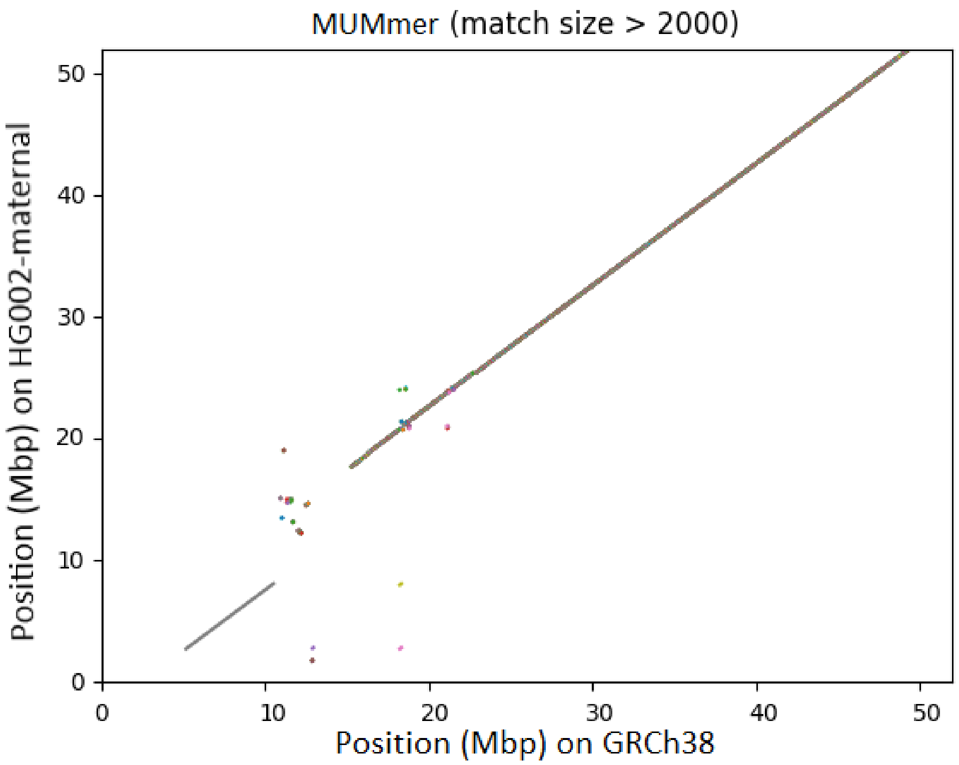

Figure 4.17: Regions of homology found between GRCh38 and HG002-maternal genomes using MUMmer

Similar filtering measures are also applied in this case. When compared to the Idoneous algorithm, this particular alignment exhibits a higher number of matches in entangled regions. However, one notable observation within the alignment is the presence of a long match solely within the gap region, which consists of the character "N". It is important to note that such a match should not be considered valid. Excluding this particular match, the algorithm achieves a coverage of 62%, which is nearly identical to that of the previous data set.

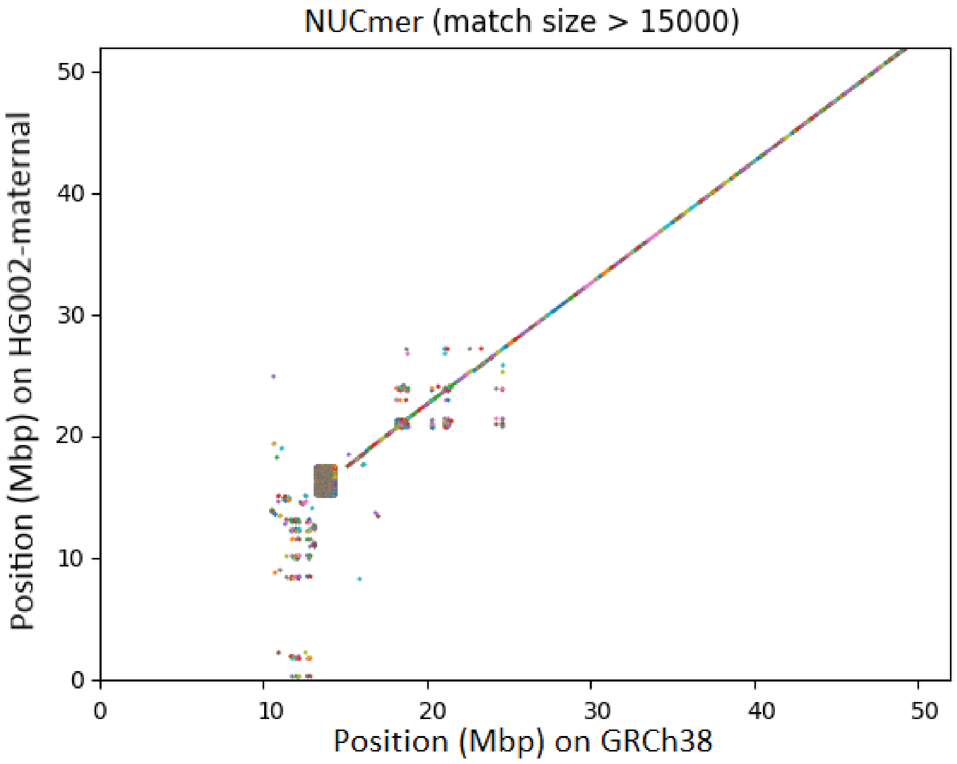

Figure 4.18: Regions of homology found between GRCh38 and HG002-maternal genomes using NUCmer

The alignment generated using NUCmer follows the same filtering protocol. As depicted in Figure 4.18, there are no significant deviations observed when compared to the T2T-CHM13 assembly. The resulting coverage is determined to be 97%.

4.3.2.3 GRCh38 vs HG002-paternal

In the alignment between GRCh38 and HG002-paternal genome, we used same parameters as before for the Idoneous algorithm.

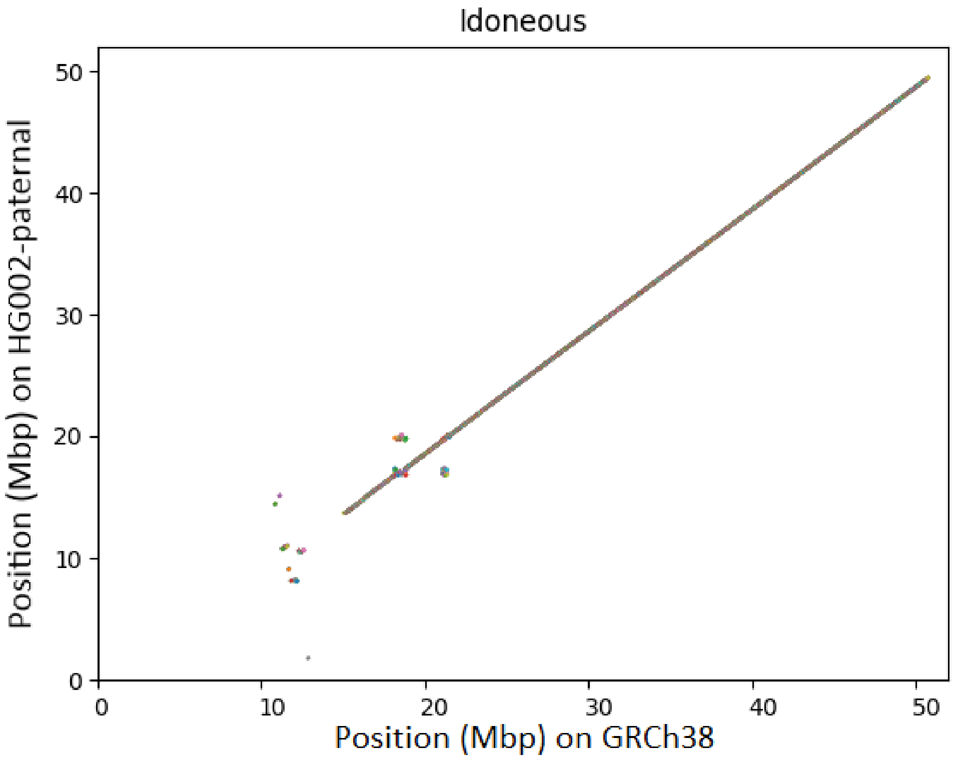

Figure 4.19: Regions of homology found between GRCh38 and HG002-paternal genomes using Idoneous

The alignment conducted using the Idoneous algorithm in this case achieves a coverage of 76%. Figure 4.20 visually demonstrates the similarity of this alignment to the alignments shown above.

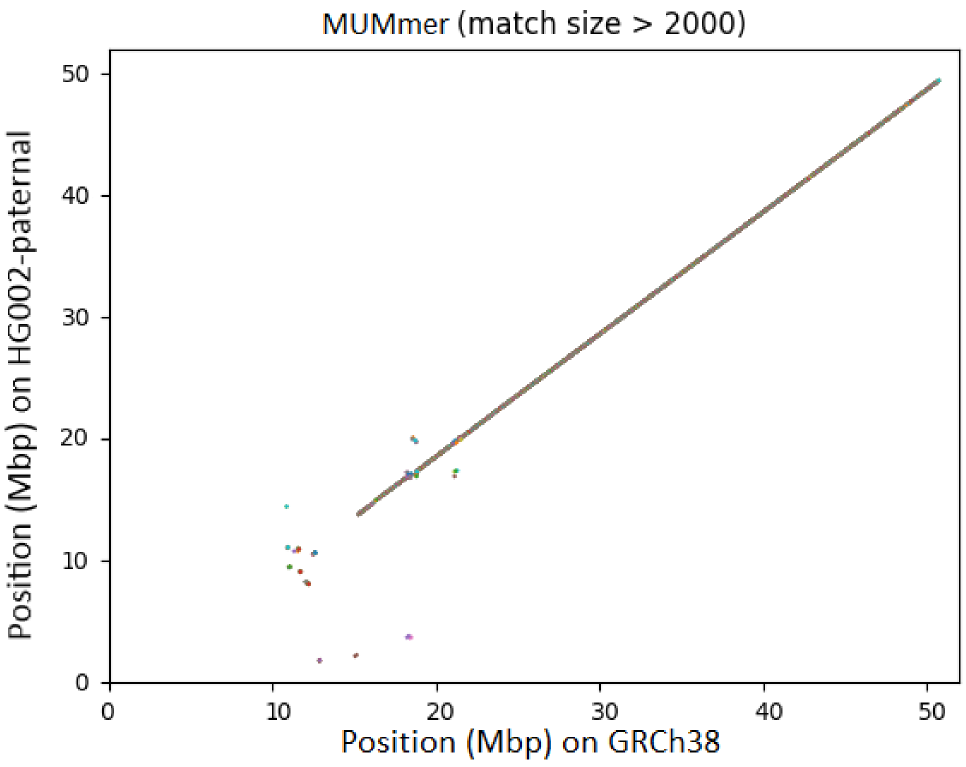

Figure 4.20: Regions of homology found between GRCh38 and HG002-paternal genomes using MUMmer

When considering the filtered matches from MUMmer, we obtained a coverage of 46%. However, Figure 4.20 reveals that despite this lower coverage, the matches are densely distributed throughout the sequence, indicating that minimal information loss occurs.

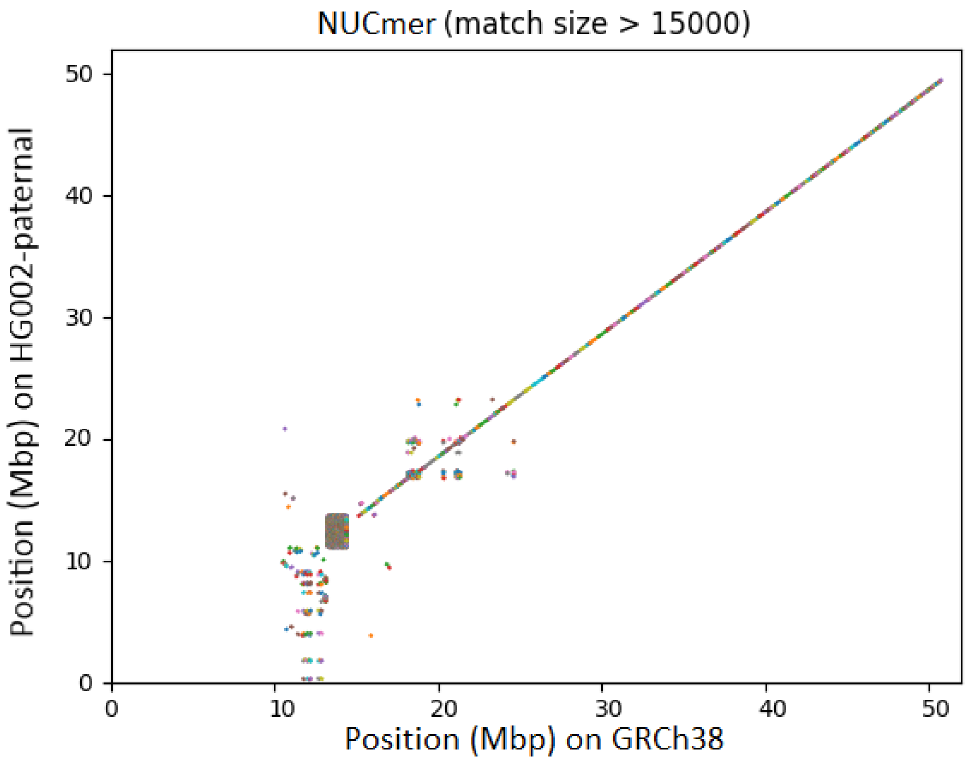

Figure 4.21: Regions of homology found between GRCh38 and HG002-paternal genomes using NUCmer

In contrast, NUCmer exhibits the ability to recover an almost complete alignment, despite the initially lower coverage observed in MUMmer. Figure 4.21 shows closely comparable results to the previous alignments, with a coverage of 98

4.3.2.4 GRCh38 vs panTro6

Despite being the closest relative to humans, the chimpanzee genome exhibits a considerable degree of dissimilarity when compared to the human genome. This substantial divergence poses a challenge for the Idoneous algorithm, leading to relatively suboptimal results.

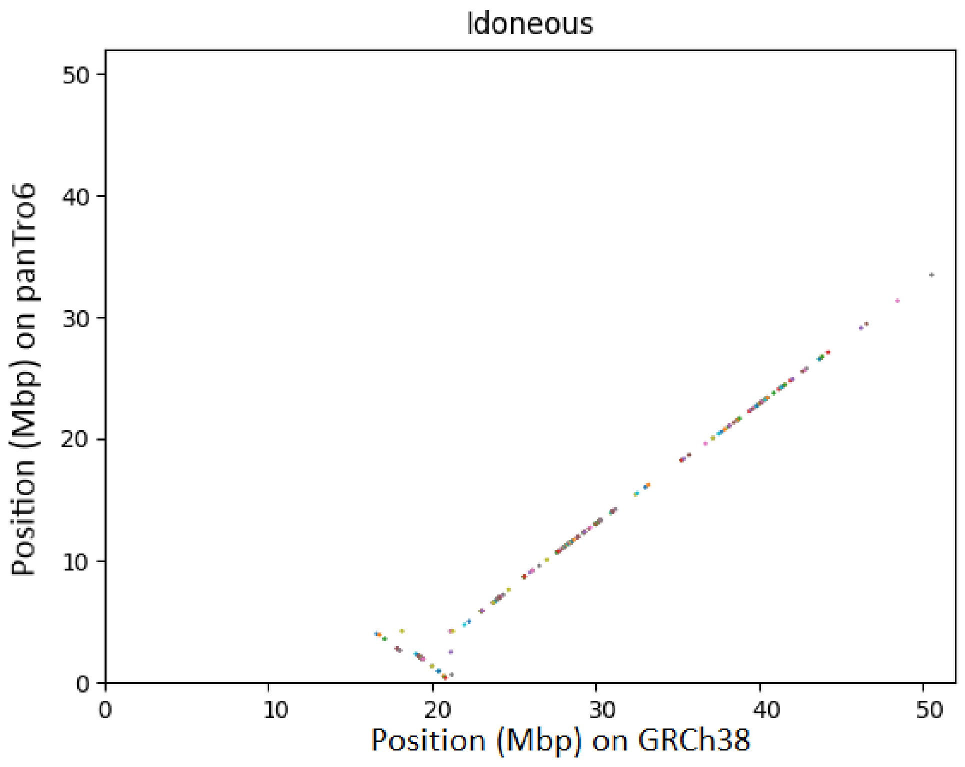

Figure 4.22: Regions of homology found between GRCh38 and panTro6 genomes using Idoneous

Next, we have modified the parameter set to enhance Idoneous algorithm's performance, setting the values as follows: $a = 4, b = 500, c = 2, d = 2000, e = 50$, and $k = 2$. We made these adjustments in response to the increased presence of variations within the sequence. As a result, the algorithm achieves a coverage of 2.2% in this alignment. Figure 4.22 displays the abundance of matches across most regions, while also highlighting a significant inversion at the starting point of the chimpanzee chromosome, indicative of a structural variant when compared to the human genome.

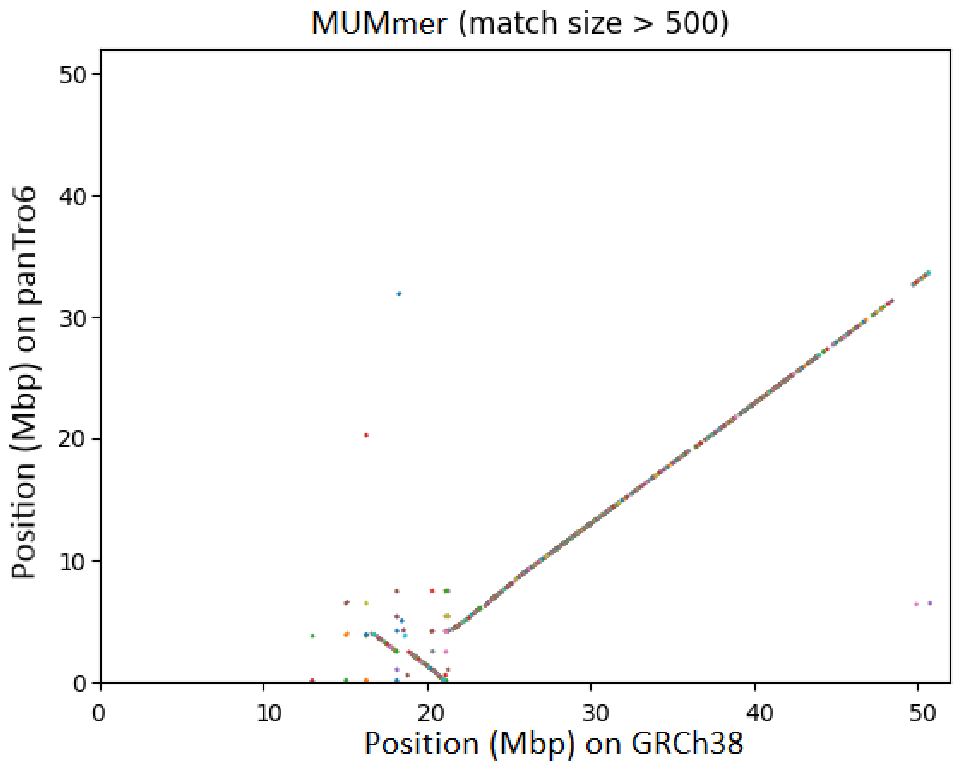

Figure 4.23: Regions of homology found between GRCh38 and panTro6 genomes using MUMmer

Furthermore, we have adjusted the minimum match size threshold in the MUMmer tool for the same reason. MUMmer achieves a coverage of 3.2%, slightly higher than that of the Idoneous algorithm. This discrepancy can be attributed to the smaller seed size employed in MUMmer, which leads to a more evenly distributed pattern of matches. It is worth noting that the majority of matches in MUMmer are smaller than the minimum size requirement of 1500 bp imposed by the Idoneous algorithm. Figure 4.23 demonstrates the consistency in identifying structural variants between the two algorithms.

Lastly, NUCmer yields a coverage of 96%, suggesting a near-complete coverage between the human and chimpanzee genomes. As depicted in Figure 4.24, duplications around the position chr22:20,000,000 and several inversions can be observed.

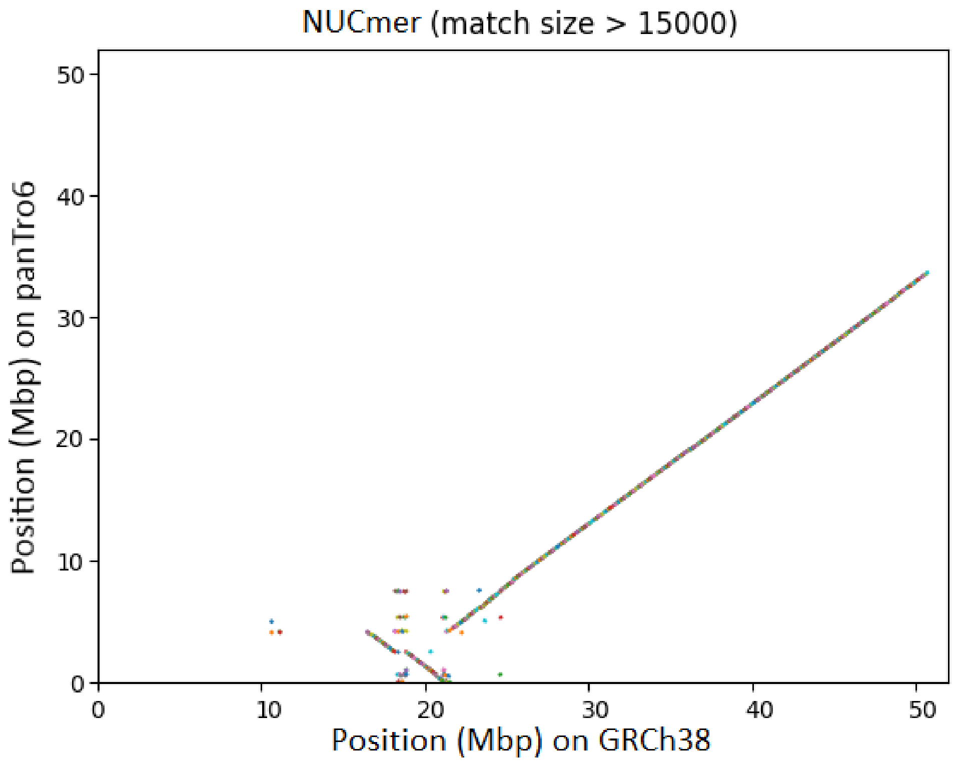

Figure 4.24: Regions of homology found between GRCh38 and panTro6 genomes using NUCmer

4.4 Run Time Results

This section discusses the run time analysis of the Idoneous algorithm using various parameter settings. Additionally, a comparison of the algorithm's run time with that of other tools is conducted across different data sets.

4.4.1 Result for parameters

This section focuses on the analysis of coverage and distance results concerning specific parameters, namely the *division window, block size, chaining threshold,* *division length*, *max indel rate*, and *k-mer size*.

4.4.1.1 Division Window (a)

The run time of the algorithm exhibits a direct proportionality to the length of the matching region. As the division window parameter (a) increases, the length of the matching region grows linearly. Consequently, the algorithm demonstrates a linear run time complexity relative to the parameter a. It is worth noting that although a procedure is implemented to eliminate duplicates, it introduces a larger asymptotic complexity. However, the overall impact on the run time remains minimal due to the procedure's small multiplicative factor and its negligible effect. Figure 4.25 provides empirical evidence supporting the linear trend of the run time with respect to the division window parameter (a).

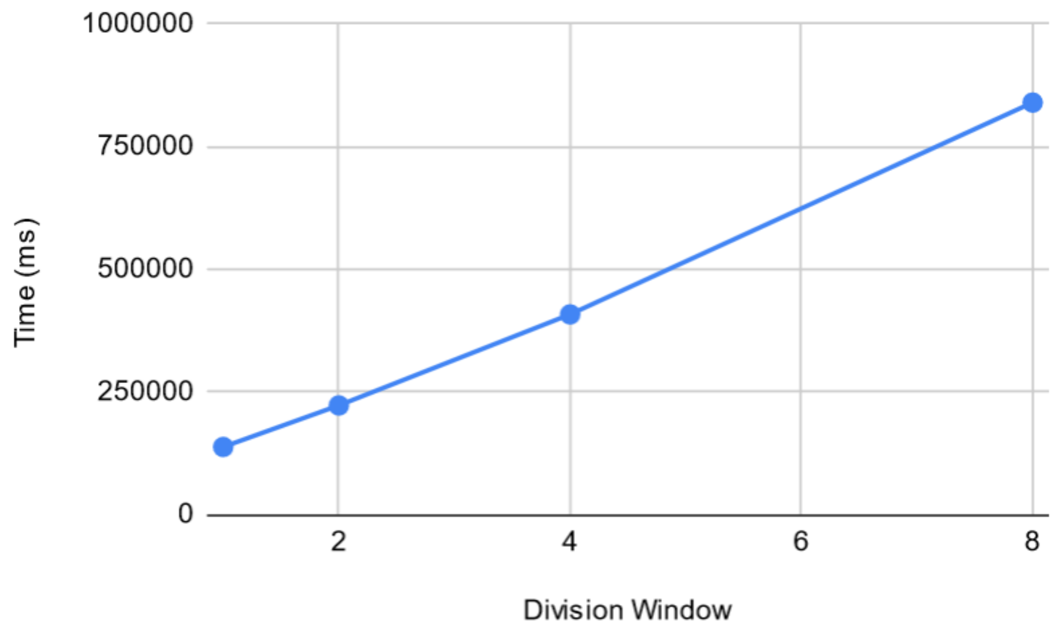

Figure 4.25: Relationship between the block size and time with parameters a = 4, c=2, d=8000, e= 200, and k=2

4.4.1.2 Block Size (b)

When the block size is increased, the length of the matching region decreases, as does the number of matches, assuming the number of blocks in a division remains constant. Consequently, changing the block size has a significant impact on the run time of the algorithm, but in the opposite direction. Figure 4.26 indicates that the time function lies between $O(\frac{1}{\log(b)})$ and $O(\frac{1}{\sqrt{b}})$, suggesting a relationship of this form.

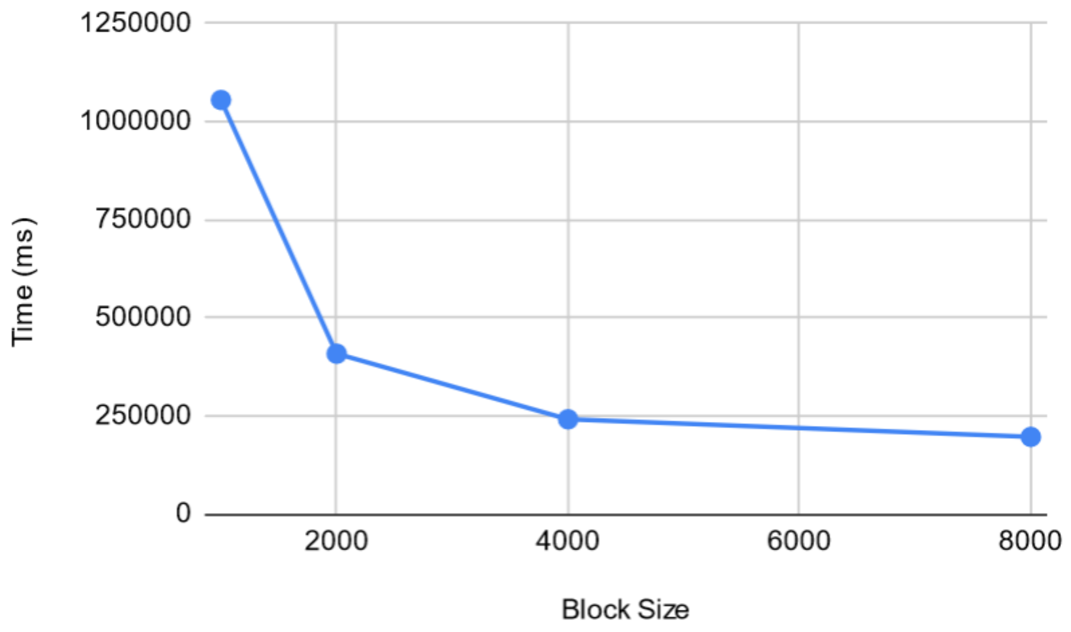

Figure 4.26: Relationship between the block size and time with parameters a = 4, c=2, d=8000, e= 200, and k=2

4.4.1.3 Chaining Threshold (c)

The decrease in run time when the chaining number is increased is primarily attributed to the reduction in the number of matching regions processed by the alignment algorithm. A higher chaining number corresponds to fewer matching regions, resulting in a smaller input size for the alignment algorithm. Consequently, the run time is reduced due to the reduced computational load. This relationship is further supported by the observations depicted in Figure 4.27, where the run time exhibits a similar trend as the coverage rate.

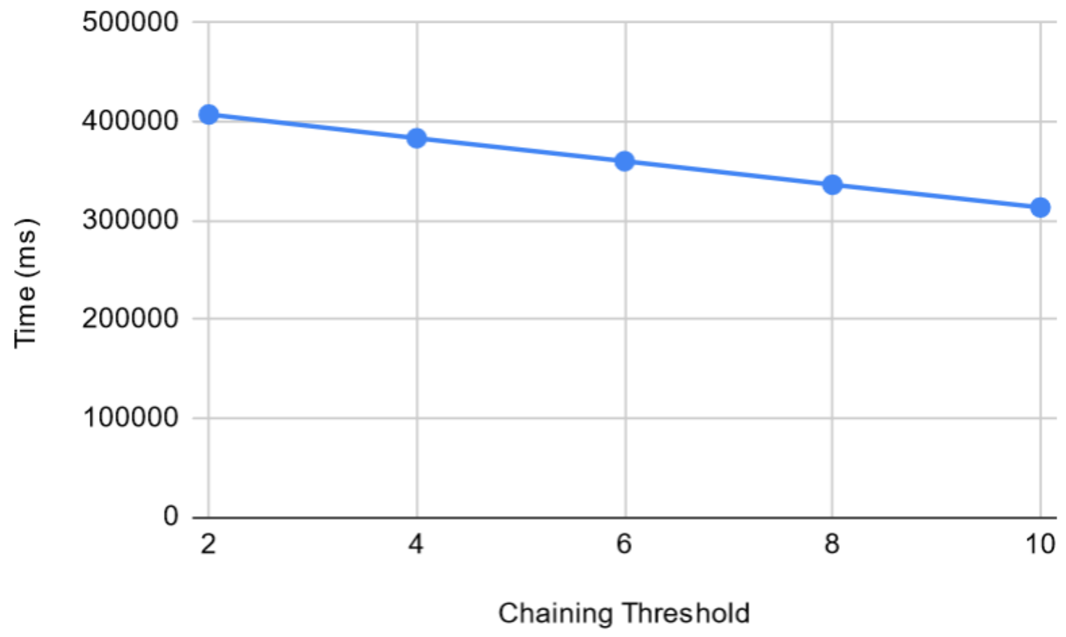

Figure 4.27: Relationship between the chaining number and time with parameters a = 4, b=2000, d=8000, e= 200, and k=2

4.4.1.4 Division Length (d)

A larger division size leads to a decrease in coverage, which is observed previously. However, the trade-off comes in the form of increased computation during tree traversal, resulting in longer processing times for larger divisions. When both factors are considered, it is found that a bigger division size actually takes less time overall, primarily due to the alignment process being more computationally intensive than the tree traversal. This inverse relationship between division size and time is evident in the experimental results presented in Figure 4.28.

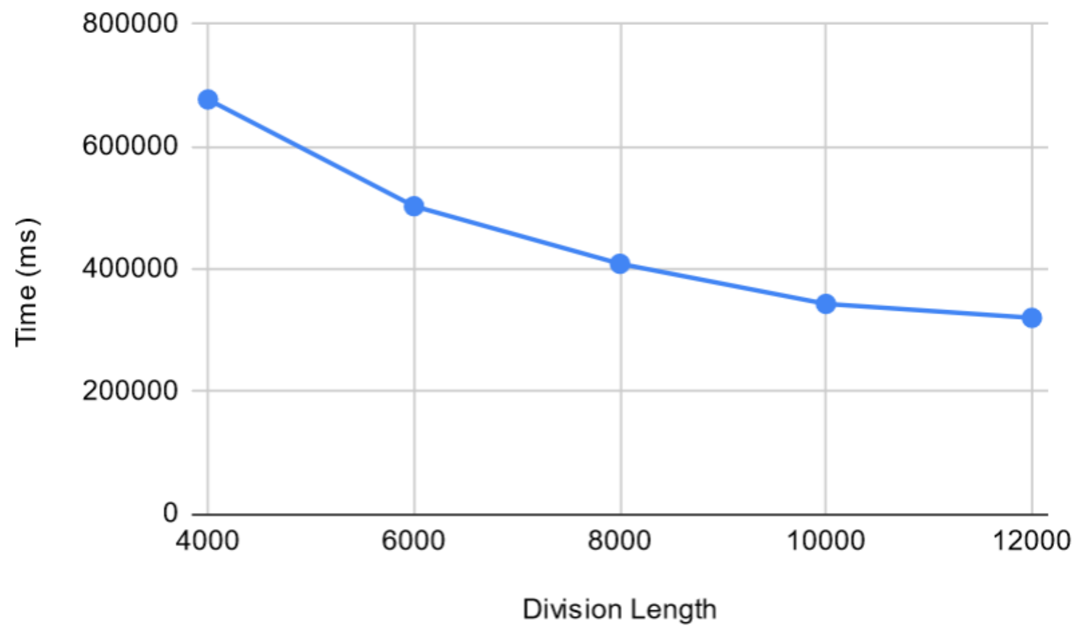

Figure 4.28: Relationship between the division size and time with parameters a = 4, b=2000, c=2, e= 200, and k=2

4.4.1.5 Max Indel Rate (e)

When the parameter *e* has a lower value, it is anticipated that the alignment algorithm will receive some erroneous matches, which will subsequently be pruned and excluded from the coverage. Consequently, in scenarios with a lower e value, the run time increases due to the additional processing required for pruning. However, beyond a certain threshold of the error rate, the alignment algorithm does not prune matches significantly, leading to a similarity between the run time and coverage trends. Figure 4.29 aligns with these theoretical expectations, displaying a similar trend as predicted.

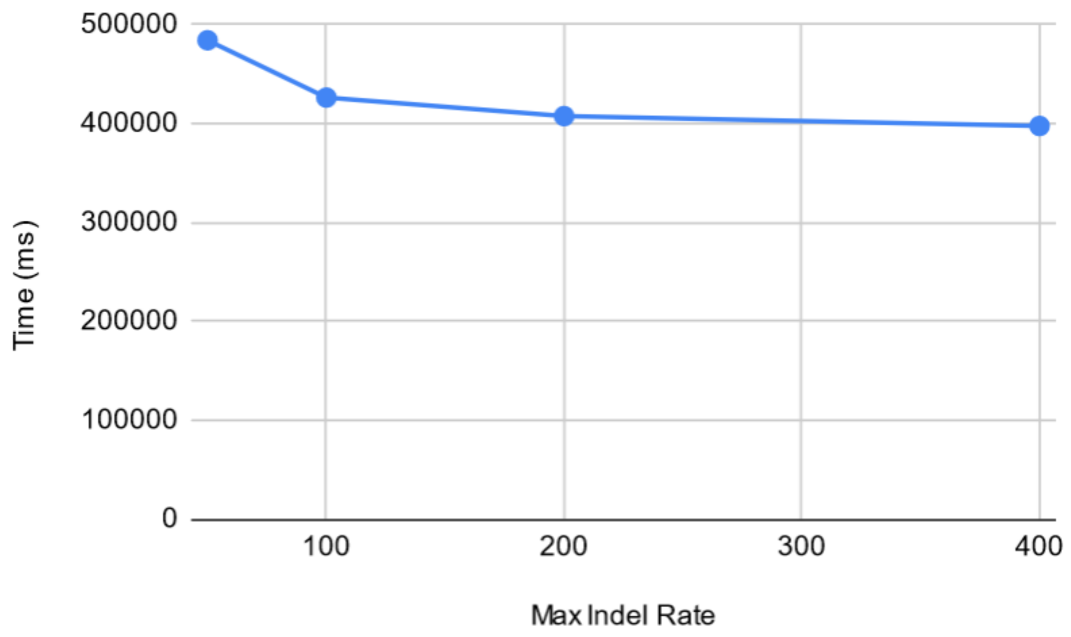

Figure 4.29: Relationship between the error rate and time with parameters a = 4, b=2000, c=2, d= 8000, and k=2

4.4.1.6 K-mer Size (k)

Since the parameter k does not affect the coverage, distance, or any other relevant metrics, it follows that the run time of the algorithm remains unaffected by changes in k. As shown in Figure 4.30, the time remains constant regardless of the value of k, further confirming that the run time does not vary with this parameter.

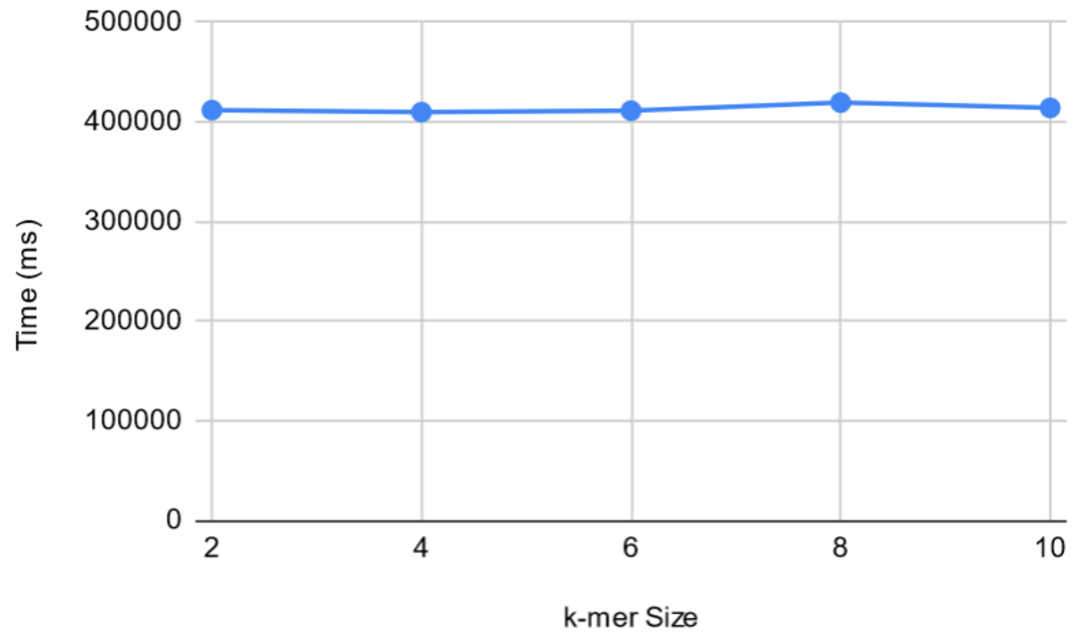

Figure 4.30: Relationship between the k-mer size and time with parameters a = 4, b=2000, c=2, d= 8000, and e=200

4.4.2 Run time Results for Different Data Sets

We show the run times for the above data sets with previous parameters in Table 4.3.

Table 4.3: Time required by each tool to align GRCh38 against different assemblies

Assembly	Idoneous	MUMmer	NUCmer
T2T-CHM13	383.284 s	49.875s	2,425.505 s
HG002-maternal	403.069s	50.261 s	2,031.389 s
HG002-paternal	393.103 s	48.095 s	2,468.722 s
panTro6	2,527.534s	50.252 s	215.374s

When comparing alignments between humans, the Idoneous algorithm falls somewhere in between MUMmer and NUCmer in terms of coverage results. However, when dealing with divergent sequences, the algorithm's performance varies depending on the chosen block size. It can either exhibit a slow run time or a high false negative rate. Despite this variability, the algorithm successfully achieves its objective of generating matching regions with specific lengths, thereby striking a balance between the two extremes presented by alternative alignment methods.

Chapter 5

Discussion & Future work

The goal of this work is to design a method for whole genome alignment with computational efficiency without losing the sensitivity of approximate matches. The approach we used in this thesis provides string approximation with adaptable matching length. We also defined several parameters to determine specificity and sensitivity. Thanks to various parameters, the proposed method is flexible in sensitivity and matching length.

Furthermore, our proposed data structure, which incorporates information from both the lexicographical and spatial domains, as well as the order relationship, can serve as the foundation for a universal reference coordinate system for genomes. This becomes particularly relevant when constructing *de novo* assemblies becomes computationally feasible, resulting in the emergence of numerous reference genomes.

Another potential application of our work is to increase time efficiency in scenarios where an oracle provides information on how two genomes diverge and the extent of their structural variants. Since the algorithm matches seeds with coarse granularity, which can be adjusted using multiple parameters, time efficiency can be enhanced by leveraging such knowledge.

On the other hand, the current version of the algorithm is highly dependent on the positions of homopolymers due to the favored bias of run-length encoding towards them. To mitigate this dependency, a hashing scheme that does not exhibit favoritism towards specific characteristics can be employed. In this approach, k-mers can be hashed, and the order of the hashing values can be utilized to establish a lexicographical order. An Alternating Tree can be constructed using multiple hash functions. While the use of k-mers may reduce the flexibility of matches, this issue can be addressed in future versions by incorporating fuzzy seeds.

There is still room for improvement in compressing the ALFTree. Currently, the tree is accessed sparsely at an approximate ratio of $\frac{k}{b}$ except during the alignment phase. Hence, the algorithm can be modified to employ a lossy compressed version of the ALFTree until the alignment phase, utilizing the original string exclusively for the alignment process. Although the current ALFTree does not contain the actual string but rather pointers, making it space-efficient, accessing memory becomes cumbersome due to the pointers traversing distant points. By using k-mers instead of pointers, the memory access time can be reduced. While this may limit the flexibility of k, it is not a concern as k does not significantly impact sensitivity beyond a certain point.

Lastly, the run time of the algorithm can be further improved through parallelization. Since a significant portion of the algorithm is localized, it can be readily parallelized. Parallelization can be implemented for each interval pair, and duplicates may occur within the $a \cdot b$ neighborhood. Therefore, the main consideration involves sorting the seeds and eliminating duplicates. The map-reduce framework proves to be a suitable method for addressing this issue.

Bibliography

[1] The 1000 Genomes Project Consortium, "A global reference for human genetic variation," *Nature*, vol. 526, pp. 68–74, Sep 2015.

[2] K. Lindblad-Toh, C. M. Wade, T. S. Mikkelsen, E. K. Karlsson, D. B. Jaffe, M. Kamal, M. Clamp, J. L. Chang, E. J. Kulbokas, 3rd, M. C. Zody, E. Mauceli, X. Xie, M. Breen, R. K. Wayne, E. A. Ostrander, C. P. Ponting, F. Galibert, D. R. Smith, P. J. DeJong, E. Kirkness, P. Alvarez, T. Biagi, W. Brockman, J. Butler, C.-W. Chin, A. Cook, J. Cuff, M. J. Daly, D. DeCaprio, S. Gnerre, M. Grabherr, M. Kellis, M. Kleber, C. Bardeleben, L. Goodstadt, A. Heger, C. Hitte, L. Kim, K.-P. Koepfli, H. G. Parker, J. P. Pollinger, S. M. J. Searle, N. B. Sutter, R. Thomas, C. Webber, J. Baldwin, A. Abebe, A. Abouelleil, L. Aftuck, M. Ait-Zahra, T. Aldredge, N. Allen, P. An, S. Anderson, C. Antoine, H. Arachchi, A. Aslam, L. Ayotte, P. Bachantsang, A. Barry, T. Bayul, M. Benamara, A. Berlin, D. Bessette, B. Blitshteyn, T. Bloom, J. Blye, L. Boguslavskiy, C. Bonnet, B. Boukhgalter, A. Brown, P. Cahill, N. Calixte, J. Camarata, Y. Cheshatsang, J. Chu, M. Citroen, A. Collymore, P. Cooke, T. Dawoe, R. Daza, K. Decktor, S. DeGray, N. Dhargay, K. Dooley, K. Dooley, P. Dorje, K. Dorjee, L. Dorris, N. Duffey, A. Dupes, O. Egbiremolen, R. Elong, J. Falk, A. Farina, S. Faro, D. Ferguson, P. Ferreira, S. Fisher, M. FitzGerald, K. Foley, C. Foley, A. Franke, D. Friedrich, D. Gage, M. Garber, G. Gearin, G. Giannoukos, T. Goode, A. Goyette, J. Graham, E. Grandbois, K. Gyaltsen, N. Hafez, D. Hagopian, B. Hagos, J. Hall, C. Healy, R. Hegarty, T. Honan, A. Horn, N. Houde, L. Hughes, L. Hunnicutt, M. Husby, B. Jester, C. Jones, A. Kamat, B. Kanga, C. Kells, D. Khazanovich, A. C. Kieu, P. Kisner, M. Kumar,

K. Lance, T. Landers, M. Lara, W. Lee, J.-P. Leger, N. Lennon, L. Leuper, S. LeVine, J. Liu, X. Liu, Y. Lokyitsang, T. Lokyitsang, A. Lui, J. Macdonald, J. Major, R. Marabella, K. Maru, C. Matthews, S. McDonough, T. Mehta, J. Meldrim, A. Melnikov, L. Meneus, A. Mihalev, T. Mihova, K. Miller, R. Mittelman, V. Mlenga, L. Mulrain, G. Munson, A. Navidi, J. Naylor, T. Nguyen, N. Nguyen, C. Nguyen, T. Nguyen, R. Nicol, N. Norbu, C. Norbu, N. Novod, T. Nyima, P. Olandt, B. O'Neill, K. O'Neill, S. Osman, L. Oyono, C. Patti, D. Perrin, P. Phunkhang, F. Pierre, M. Priest, A. Rachupka, S. Raghuraman, R. Rameau, V. Ray, C. Raymond, F. Rege, C. Rise, J. Rogers, P. Rogov, J. Sahalie, S. Settipalli, T. Sharpe, T. Shea, M. Sheehan, N. Sherpa, J. Shi, D. Shih, J. Sloan, C. Smith, T. Sparrow, J. Stalker, N. Stange-Thomann, S. Stavropoulos, C. Stone, S. Stone, S. Sykes, P. Tchuinga, P. Tenzing, S. Tesfaye, D. Thoulutsang, Y. Thoulutsang, K. Topham, I. Topping, T. Tsamla, H. Vassiliev, V. Venkataraman, A. Vo, T. Wangchuk, T. Wangdi, M. Weiand, J. Wilkinson, A. Wilson, S. Yadav, S. Yang, X. Yang, G. Young, Q. Yu, J. Zainoun, L. Zembek, A. Zimmer, and E. S. Lander, "Genome sequence, comparative analysis and haplotype structure of the domestic dog," *Nature*, vol. 438, pp. 803–819, Dec. 2005.

[3] W. P. Kloosterman, L. C. Francioli, F. Hormozdiari, T. Marschall, J. Y. Hehir-Kwa, A. Abdellaoui, E.-W. Lameijer, M. H. Moed, V. Koval, I. Renkens, M. J. van Roosmalen, P. Arp, L. C. Karssen, B. P. Coe, R. E. Handsaker, E. D. Suchiman, E. Cuppen, D. T. Thung, M. McVey, M. C. Wendl, Genome of Netherlands Consortium, A. Uitterlinden, C. M. van Duijn, M. A. Swertz, C. Wijmenga, G. B. van Ommen, P. E. Slagboom, D. I. Boomsma, A. Schönhuth, E. E. Eichler, P. I. W. de Bakker, K. Ye, and V. Guryev, "Characteristics of de novo structural changes in the human genome," *Genome Res*, vol. 25, pp. 792–801, Apr. 2015.

[4] W. Brandler, D. Antaki, M. Gujral, A. Noor, G. Rosanio, T. Chapman, D. Barrera, G. Lin, D. Malhotra, A. Watts, L. Wong, J. Estabillo, T. Gadomski, O. Hong, K. Fajardo, A. Bhandari, R. Owen, M. Baughn, J. Yuan, T. Solomon, A. Moyzis, M. Maile, S. Sanders, G. Reiner, K. Vaux, C. Strom, K. Zhang, A. Muotri, N. Akshoomoff, S. Leal, K. Pierce, E. Courchesne,

L. Iakoucheva, C. Corsello, and J. Sebat, “Frequency and complexity of de novo structural mutation in autism,” *The American Journal of Human Genetics*, vol. 98, pp. 667–679, Apr 2016.

[5] R. H. Waterson, E. S. Lander, R. K. Wilson, T. C. Sequencing, and A. Consortium, “Initial sequence of the chimpanzee genome and comparison with the human genome,” *Nature*, vol. 437, pp. 69–87, Sep 2005.

[6] S. B. Needleman and C. D. Wunsch, “A general method applicable to the search for similarities in the amino acid sequence of two proteins,” *J Mol Biol*, vol. 48, pp. 443–453, Mar 1970.

[7] T. F. Smith and M. S. Waterman, “Identification of common molecular subsequences,” *J Mol Biol*, vol. 147, pp. 195–197, Mar 1981.

[8] L. Wang and T. Jiang, “On the complexity of multiple sequence alignment,” *Journal of Computational Biology*, vol. 1, no. 4, pp. 337–348, 1994.

[9] J. Prado-Martinez, P. H. Sudmant, J. M. Kidd, H. Li, J. L. Kelley, B. Lorente-Galdos, K. R. Veeramah, A. E. Woerner, T. D. O’Connor, G. Santpere, A. Cagan, C. Theunert, F. Casals, H. Laayouni, K. Munch, A. Hobolth, A. E. Halager, M. Malig, J. Hernandez-Rodriguez, I. Hernando-Herraez, K. Prüfer, M. Pybus, L. Johnstone, M. Lachmann, C. Alkan, D. Twigg, N. Petit, C. Baker, F. Hormozdiari, M. Fernandez-Callejo, M. Dabad, M. L. Wilson, L. Stevison, C. Camprubí, T. Carvalho, A. Ruiz-Herrera, L. Vives, M. Mele, T. Abello, I. Kondova, R. E. Bontrop, A. Pusey, F. Lankester, J. A. Kiyang, R. A. Bergl, E. Lonsdorf, S. Myers, M. Ventura, P. Gagneux, D. Comas, H. Siegismund, J. Blanc, L. Agueda-Calpena, M. Gut, L. Fulton, S. A. Tishkoff, J. C. Mullikin, R. K. Wilson, I. G. Gut, M. K. Gonder, O. A. Ryder, B. H. Hahn, A. Navarro, J. M. Akey, J. Bertranpetit, D. Reich, T. Mailund, M. H. Schierup, C. Hvilsom, A. M. Andrés, J. D. Wall, C. D. Bustamante, M. F. Hammer, E. E. Eichler, and T. Marques-Bonet, “Great ape genetic diversity and population history,” *Nature*, vol. 499, pp. 471–475, Jul 2013.

[10] D. P. Locke, L. W. Hillier, W. C. Warren, K. C. Worley, L. V. Nazareth, D. M. Muzny, S.-P. Yang, Z. Wang, A. T. Chinwalla, P. Minx, M. Mitreva,

L. Cook, K. D. Delehaunty, C. Fronick, H. Schmidt, L. A. Fulton, R. S. Fulton, J. O. Nelson, V. Magrini, C. Pohl, T. A. Graves, C. Markovic, A. Cree, H. H. Dinh, J. Hume, C. L. Kovar, G. R. Fowler, G. Lunter, S. Meader, A. Heger, C. P. Ponting, T. Marques-Bonet, C. Alkan, L. Chen, Z. Cheng, J. M. Kidd, E. E. Eichler, S. White, S. Searle, A. J. Vilella, Y. Chen, P. Flicek, J. Ma, B. Raney, B. Suh, R. Burhans, J. Herrero, D. Haussler, R. Faria, O. Fernando, F. DarrÃ©, D. FarrÃ©, E. Gazave, M. Oliva, A. Navarro, R. Roberto, O. Capozzi, N. Archidiacono, G. D. Valle, S. Purgato, M. Rocchi, M. K. Konkel, J. A. Walker, B. Ullmer, M. A. Batzer, A. F. A. Smit, R. Hubley, C. Casola, D. R. Schrider, M. W. Hahn, V. Quesada, X. S. Puente, G. R. OrdoÃ±ez, C. LÃ³pez-OtÃn, T. Vinar, B. Brejova, A. Ratan, R. S. Harris, W. Miller, C. Kosiol, H. A. Lawson, V. Taliwal, A. L. Martins, A. Siepel, A. Roychoudhury, X. Ma, J. Degenhardt, C. D. Bustamante, R. N. Gutenkunst, T. Mailund, J. Y. Dutheil, A. Hobolth, M. H. Schierup, O. A. Ryder, Y. Yoshinaga, P. J. de Jong, G. M. Weinstock, J. Rogers, E. R. Mardis, R. A. Gibbs, and R. K. Wilson, "Comparative and demographic analysis of orang-utan genomes," *Nature*, vol. 469, pp. 529–533, Jan 2011.

[11] A. Rhie, S. A. McCarthy, O. Fedrigo, J. Damas, G. Formenti, S. Koren, M. Uliano-Silva, W. Chow, A. Fungtammasan, J. Kim, C. Lee, B. J. Ko, M. Chaisson, G. L. Gedman, L. J. Cantin, F. Thibaud-Nissen, L. Haggerty, I. Bista, M. Smith, B. Haase, J. Mountcastle, S. Winkler, S. Paez, J. Howard, S. C. Vernes, T. M. Lama, F. Grutzner, W. C. Warren, C. N. Balakrishnan, D. Burt, J. M. George, M. T. Biegler, D. Iorns, A. Digby, D. Eason, B. Robertson, T. Edwards, M. Wilkinson, G. Turner, A. Meyer, A. F. Kautt, P. Franchini, H. W. Detrich, H. Svardal, M. Wagner, G. J. P. Naylor, M. Pippel, M. Malinsky, M. Mooney, M. Simbirsky, B. T. Hannigan, T. Pesout, M. Houck, A. Misuraca, S. B. Kingan, R. Hall, Z. Kronenberg, I. Sović, C. Dunn, Z. Ning, A. Hastie, J. Lee, S. Selvaraj, R. E. Green, N. H. Putnam, I. Gut, J. Ghurye, E. Garrison, Y. Sims, J. Collins, S. Pelan, J. Torrance, A. Tracey, J. Wood, R. E. Dagnew, D. Guan, S. E. London, D. F. Clayton, C. V. Mello, S. R. Friedrich, P. V. Lovell, E. Osipova, F. O. Al-Ajli, S. Secomandi, H. Kim, C. Theofanopoulou, M. Hiller, Y. Zhou, R. S. Harris,

K. D. Makova, P. Medvedev, J. Hoffman, P. Masterson, K. Clark, F. Martin, K. Howe, P. Flicek, B. P. Walenz, W. Kwak, H. Clawson, M. Diekhans, L. Nassar, B. Paten, R. H. S. Kraus, A. J. Crawford, M. T. P. Gilbert, G. Zhang, B. Venkatesh, R. W. Murphy, K.-P. Koepfli, B. Shapiro, W. E. Johnson, F. Di Palma, T. Marques-Bonet, E. C. Teeling, T. Warnow, J. M. Graves, O. A. Ryder, D. Haussler, S. J. O'Brien, J. Korlach, H. A. Lewin, K. Howe, E. W. Myers, R. Durbin, A. M. Phillippy, and E. D. Jarvis, "Towards complete and error-free genome assemblies of all vertebrate species.," *Nature*, vol. 592, pp. 737–746, Apr. 2021.

[12] S. Cheng, M. Melkonian, S. A. Smith, S. Brockington, J. M. Archibald, P.-M. Delaux, F.-W. Li, B. Melkonian, E. V. Mavrodiev, W. Sun, Y. Fu, H. Yang, D. E. Soltis, S. W. Graham, P. S. Soltis, X. Liu, X. Xu, and G. K.-S. Wong, "10KP: A phylodiverse genome sequencing plan," *GigaScience*, vol. 7, p. giy013, 02 2018.

[13] H. A. Lewin, G. E. Robinson, W. J. Kress, W. J. Baker, J. Coddington, K. A. Crandall, R. Durbin, S. V. Edwards, F. Forest, M. T. P. Gilbert, M. M. Goldstein, I. V. Grigoriev, K. J. Hackett, D. Haussler, E. D. Jarvis, W. E. Johnson, A. Patrinos, S. Richards, J. C. Castilla-Rubio, M.-A. van Sluys, P. S. Soltis, X. Xu, H. Yang, and G. Zhang, "Earth BioGenome Project: Sequencing life for the future of life.," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 115, pp. 4325–4333, Apr. 2018.

[14] N. Nguyen, G. Hickey, D. R. Zerbino, B. Raney, D. Earl, J. Armstrong, W. J. Kent, D. Haussler, and B. Paten, "Building a pan-genome reference for a population," *J Comput Biol*, vol. 22, pp. 387–401, May 2015.

[15] G. Hickey, J. Monlong, J. Ebler, A. M. Novak, J. M. Eizenga, Y. Gao, H. P. R. Consortium, T. Marschall, H. Li, and B. Paten, "Pangenome graph construction from genome alignments with minigraph-cactus.," *Nature biotechnology*, May 2023.

[16] M. D. Atkinson, J.-R. Sack, N. Santoro, and T. Strothotte, “Min-max heaps and generalized priority queues,” *Commun. ACM*, vol. 29, p. 996–1000, oct 1986.

[17] H. Zhang, H. Li, C. Jain, H. Cheng, K. F. Au, H. Li, and S. Aluru, “Real-time mapping of nanopore raw signals.,” *Bioinformatics*, vol. 37, pp. i477–i483, July 2021.

[18] S. C. Sahinalp and U. Vishkin, “Symmetry breaking for suffix tree construction,” in *Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing, 23-25 May 1994, Montréal, Québec, Canada* (F. T. Leighton and M. T. Goodrich, eds.), pp. 300–309, ACM, 1994.

[19] S. C. Sahinalp and U. Vishkin, “Efficient approximate and dynamic matching of patterns using a labeling paradigm,” in *37th Annual Symposium on Foundations of Computer Science, FOCS ’96, Burlington, Vermont, USA, 14-16 October, 1996*, pp. 320–328, IEEE Computer Society, 1996.

[20] M. Lothaire, ed., *Combinatorics on Words*. Cambridge Mathematical Library, Cambridge University Press, 2 ed., 1997.

[21] R. C. Lyndon, “On burnside’s problem,” *Transactions of the American Mathematical Society*, vol. 77, no. 2, pp. 202–215, 1954.

[22] P. Ferragina and G. Manzini, “Opportunistic data structures with applications,” in *Foundations of Computer Science, 2000. Proceedings. 41st Annual Symposium on*, pp. 390–398, 2000.

[23] J. Duval, “Factorizing words over an ordered alphabet,” *J. Algorithms*, vol. 4, no. 4, pp. 363–381, 1983.

[24] M. Šošic and M. Šikic, “Edlib: a C/C++ library for fast, exact sequence alignment using edit distance.,” *Bioinformatics*, vol. 33, pp. 1394–1395, May 2017.

[25] International Human Genome Sequencing Consortium, “Finishing the euchromatic sequence of the human genome.,” *Nature*, vol. 431, pp. 931–945, Oct. 2004.

[26] V. A. Schneider, T. Graves-Lindsay, K. Howe, N. Bouk, H.-C. Chen, P. A. Kitts, T. D. Murphy, K. D. Pruitt, F. Thibaud-Nissen, D. Albracht, R. S. Fulton, M. Kremitzki, V. Magrini, C. Markovic, S. McGrath, K. M. Steinberg, K. Auger, W. Chow, J. Collins, G. Harden, T. Hubbard, S. Pelan, J. T. Simpson, G. Threadgold, J. Torrance, J. M. Wood, L. Clarke, S. Koren, M. Boitano, P. Peluso, H. Li, C.-S. Chin, A. M. Phillippy, R. Durbin, R. K. Wilson, P. Flicek, E. E. Eichler, and D. M. Church, "Evaluation of grch38 and de novo haploid genome assemblies demonstrates the enduring quality of the reference assembly," *Genome Research*, vol. 27, pp. 849–864, 2017.

[27] S. Nurk, S. Koren, A. Rhie, M. Rautiainen, A. V. Bzikadze, A. Mikheenko, M. R. Vollger, N. Altemose, L. Uralsky, A. Gershman, S. Aganezov, S. J. Hoyt, M. Diekhans, G. A. Logsdon, M. Alonge, S. E. Antonarakis, M. Borchers, G. G. Bouffard, S. Y. Brooks, G. V. Caldas, N.-C. Chen, H. Cheng, C.-S. Chin, W. Chow, L. G. de Lima, P. C. Dishuck, R. Durbin, T. Dvorkina, I. T. Fiddes, G. Formenti, R. S. Fulton, A. Fungtammasan, E. Garrison, P. G. S. Grady, T. A. Graves-Lindsay, I. M. Hall, N. F. Hansen, G. A. Hartley, M. Haukness, K. Howe, M. W. Hunkapiller, C. Jain, M. Jain, E. D. Jarvis, P. Kerpedjiev, M. Kirsche, M. Kolmogorov, J. Korlach, M. Kremitzki, H. Li, V. V. Maduro, T. Marschall, A. M. McCartney, J. McDaniel, D. E. Miller, J. C. Mullikin, E. W. Myers, N. D. Olson, B. Paten, P. Peluso, P. A. Pevzner, D. Porubsky, T. Potapova, E. I. Rogaev, J. A. Rosenfeld, S. L. Salzberg, V. A. Schneider, F. J. Sedlazeck, K. Shafin, C. J. Shew, A. Shumate, Y. Sims, A. F. A. Smit, D. C. Soto, I. Sović, J. M. Storer, A. Streets, B. A. Sullivan, F. Thibaud-Nissen, J. Torrance, J. Wagner, B. P. Walenz, A. Wenger, J. M. D. Wood, C. Xiao, S. M. Yan, A. C. Young, S. Zarate, U. Surti, R. C. McCoy, M. Y. Dennis, I. A. Alexandrov, J. L. Gerton, R. J. O'Neill, W. Timp, J. M. Zook, M. C. Schatz, E. E. Eichler, K. H. Miga, and A. M. Phillippy, "The complete sequence of a human genome.," *Science*, vol. 376, pp. 44–53, Apr. 2022.

[28] M. Rautiainen, S. Nurk, B. P. Walenz, G. A. Logsdon, D. Porubsky, A. Rhie, E. E. Eichler, A. M. Phillippy, and S. Koren, "Telomere-to-telomere assembly

of diploid chromosomes with verkko," *Nature Biotechnology*, pp. 1–9, 2023.

Appendix A

Code

Idoneous source code is available at https://github.com/BilkentCompGen/Idoneous