Phillip Swanepoel
SWNPHI004

MSc (Med) Bioinformatics

Simulating recombinant sequence data to evaluate and improve computational methods of multiple sequence alignment and recombinant identification

Supervised by Associate Professor Darren Patrick Martin

Division of Computational Biology
Faculty of Health Sciences
University of Cape Town
Anzio Road
7925 Observatory, Cape Town
South Africa

Declaration

I, Phillip Swanepoel (SWNPHI004), hereby declare that the work on which this dissertation is based on my original work (except where acknowledgements indicate otherwise) and that neither the whole work nor any part of it has been, is being, or is to be submitted for another degree in this or any other university. I empower the university to reproduce for the purpose of research either the whole or any portion of the contents in any manner whatsoever.

Date: 21 Dec 2022
Signed: 15 December 2022

Signed by candidate

Acknowledgements

My parents:
Cindy and Rupert, who support me without question.

My supervisor:
Darren, without whom this project would not be possible. It was an absolute privilege to work with someone who shares his profoundly broad knowledge and expertise in such a cheerful, free and human manner. Always there to help when needed.

My peers:
Josh Cullinan, Arné de Klerk, Isaac Abodunrin, Bioinformatics Hons 2022 cohort. Special thanks to Josh who made major contributions to this project, focusing on the machine learning aspect. Also to Isaac and the honours students who helped run the simulated datasets through all the different alignment tools.

Contents

Figures

Tables

Simulating recombinant sequence data to evaluate and improve computational methods of multiple sequence alignment and recombinant identification

Chapter 1: Literature review

1.1) Introduction: Viruses and Recombination

Viruses are infectious agents that can only replicate by infecting living cells. They are able to infect all forms of life and are easily the most abundant biological entity: the total number of viruses vastly exceeding the total number of living cells (Koonin et al., 2006). One could think of viruses as simpler entities than cells in that they do not possess many of the characteristics of living organisms: they do not replicate on their own, they do not produce or store energy and they have no means of translating messenger RNA (mRNA). Yet, despite having very small genomes, viruses exhibit high diversity on a genetic and functional level (Chaitanya, 2019).

Viruses display a large variety of genetic structures, and are commonly classified according to their mechanism for mRNA synthesis, whether their packaged genomes are encoded by DNA or RNA, and whether these packaged genomes are in single or double stranded configurations (Baltimore, 1971). According to this classification, seven groups of viruses emerge: double-stranded DNA, single-stranded DNA, double-stranded RNA, positive sense single-stranded RNA, negative sense single-stranded RNA, single-stranded RNA with an intermediary DNA phase and double-stranded DNA viruses with an intermediary RNA phase. Newer taxonomic classification criteria divide viruses into 15 hierarchical ranks (Gorbalenya et al., 2020).

In addition to wide diversity in the chemical composition of their genomes, the information content encoded within viral genomes is also highly diverse. Viruses frequently encode overlapping genes where one region of a genome can code for multiple genes depending on the reading frame context. This information compression enables virus genomes to, in general, encode more proteins and regulatory elements than similarly sized chromosomal fragments of cellular organisms (Barrell et al., 1976; Scherbakov & Garber, 2000).

Although the life cycles of viruses are as diverse as the structures and information contents their genomes, these life-cycles must accommodate a range of constraints

including error-prone genome replication (which generates mutations), host innate and adaptive antiviral defences (which can both generate mutations and selectively favour their propagation), and inconsistent/absent genome repair and replication proofreading mechanisms (which obstruct mutations being repaired). Viruses therefore tend to have high mutation rates. It is this factor that is the central driver of viral genetic diversity. Although the rates at which mutations accumulate in different viral groups can vary widely even between strains of the same virus species, in general RNA viruses have mutation rates in the order of 10^{-6} to 10^{-4} substitutions per nucleotide site per cell infection (s/n/c), compared to the 10^{-8} to 10^{-6} s/n/c in double stranded DNA viruses (Sanjuán & Domingo-Calap, 2021).

Another key driver of genetic diversity, although not occurring in all types of viruses, is recombination (Pérez-Losada et al., 2015). Recombination is a particularly large contributor to the diversity of positive sense ssRNA viruses and ssDNA viruses (Rawson et al., 2018). When two genetically distinct viruses simultaneously infect the same host cell, interactions between their replication machinery can occur where the viral polymerase of one virus starts replicating its own genome but then switches templates and continues replicating the other virus' genome. This will result in replicated genomes having some fragments of sequence copied from one of the coinfecting viral genomes and other fragments copied from the other co-infecting viral genome. This type of recombination is called template switching or copy-choice recombination (Baron, 1996; Luo & Taylor, 1990). Other types of recombination are also possible. In segmented viruses (viruses that have multiple "chromosomes" all encapsidated within the same virion) or multipartite viruses (viruses with multiple chromosomes each of which is encapsidated in a different virion) recombination can take place by a process called reassortment, where whole chromosomes are exchanged. This recombination mechanism has been extensively studied in influenza virus (Marshall et al., 2013; Sanjuán & Domingo-Calap, 2021). In addition to copy-choice and reassortment mechanisms, recombination in DNA viruses also commonly occurs by strand breakage and rejoining mechanisms: a type of recombination that is shared by cellular organisms and tends to be facilitated by host-encoded DNA double strand break repair pathways (Fleischmann, 1996).

Apart from its general evolutionary role in promoting genetic diversity, by mixing the genetic material of two divergent parental viruses within a recombinant daughter genome, recombination can be evolutionarily important in that it can lead to a sudden and drastic change in the genetic makeup of a daughter genome; changes that, relative to its parents, could profoundly alter the biological characteristics of the daughter such as its transmissibility, host-range, cell tropism, pathogenicity and antigenicity (Fleischmann, 1996). Understanding the causes and consequences of recombination in

viruses is thus vital for both the short-term evolutionary dynamics of viral infections within a single host over days-long periods (which can yield new mutants able to evade host antiviral defences), and the longer scale evolutionary dynamics of viruses within entire ecosystems over thousands (or millions) of years (which can yield entirely new viral species).

One of the ways to understand recombination and its role in the evolution of viruses, is to detect and study traces of past recombination events that are frequently evident within the genome sequences of contemporary viruses. One of the most commonly used tools that is used to do this is recombination detection program, RDP (Martin et al., 2021), which screens multiple sequence alignments (MSAs) of homologous viral genomes to detect and characterise signals of past recombination events.

1.2) Identifying recombinants with computational methods

Many computational analyses of viral evolution rely on the estimation of phylogenetic trees from MSAs. This estimation can be done by various phylogenetic methods, but most rely on the assumption that one phylogeny underlies the evolution of all the sequences in the alignment. Recombination undermines this assumption, since the exchange of blocks of genomic material creates separate regions, within one genome, that have distinct phylogenetic histories (Schierup & Hein, 2000).

Besides RDP, there exists a wide range of methods for detecting past recombination events in MSAs. These methods can be broadly divided into four groups (Posada & Crandall, 2001). (1) Distance methods measure genetic distance (typically pairwise distance) in a sliding window and look for inversion patterns (Weiller, 1998); (2) phylogenetic methods detect recombination by looking for regions of the genome with divergent tree topologies (Hein, 1990); (3) compatibility methods also look for divergent phylogenetic trees, but do so on a site-by-site basis (Jakobsen et al., 1997); and (4) substitution distribution methods examine patterns in nucleotide substitution positioning, testing for significant clustering or divergence from an expected statistical distribution (Sawyer, 1989).

RDP incorporates methods in all four categories into a single meta-method and has become one of the most commonly used tools for detecting and analysing recombination in sequence datasets over the past 20 years. Specifically, RDP runs a suite of these methods to detect recombination events by scanning triplets of sequences. Once a recombination event has been detected and its breakpoint positions estimated, it is still unclear which of the three sequences in the event is the

recombinant. Thus RDP needs to classify one of the sequences in the triplet as the recombinant (not a parent) using a separate suite of computational methods. This triplet will be referred to as a recombinant/parent/parent triplet. While there are some recombination detection methods implemented in RDP that directly identify the recombinant, most are also reliant on this separate computational step for recombinant identification. Recombination detection methods implemented in RDP5 (the latest version of RDP) which also rely on this identification step include: BOOTSCAN (Salminen et al., 1995), GENECONV (Sawyer, 1989), MAXCHI (Smith, 1992), CHIMAERA (David Posada's modified version of MAXCHI, unpublished work) and SISCAN (Gibbs et al., 2000).

Unfortunately, it is still unknown how accurately either RDP or any other recombination detection tools identify recombinant sequences. One approach to determine this accuracy would be to realistically simulate the evolution of recombining sequences in such a way that the recombination history of the sequences was known and then compare the outputs of RDP (or any other recombination detection tool) to this known history.

1.3) Simulating viral sequence data with recombination

There are two major approaches to simulating sequence data with recombination. Thinking about the problem phylogenetically, one can either start with the root of the tree or the terminal branches. Forward in time simulators start with a "seed" or root sequence and evolve a population forwards in time. Backward in time simulators use coalescent theory (Hudson & Kaplan, 1988; Kingman, 1982) to start with a sample of sequences (i.e. those at the tips of terminal branches) and work backwards in time towards a single common ancestor sequence (which represents the root) (Arenas, 2013).

Coalescent based simulations have an advantage in terms of memory efficiency and speed since only a sample of the total population is considered and many of the complexities of evolution are abstracted into models. However, an advantage of forward in time simulators is that they potentially allow the realistic simulation of multiple complex evolutionary processes such as those involving nucleotide insertions, nucleotide deletions and natural selection: it is difficult to achieve this with backwards in time approaches given their relatively higher computational cost (Jariani et al., 2019).

One particularly interesting forward in time simulator is SANTA-SIM (Jariani et al., 2019), a software package written in JAVA (Arnold, Gosling & Holmes, 2005) that simulates evolving sequences as individual agents in discrete time-steps. SANTA-SIM is

ideal for the simulation of RNA viruses, since it can simulate diverse selection pressures and recombination events. An advantage of SANTA-SIM over other simulators is that distinct alleles can evolve under separate, context-dependent selection pressures affected by factors such as population size or the survival times of alleles. SANTA-SIM also allows one to set up custom sampling schemes which can extract summary statistics and sequence alignments at any time in the simulation. Further, it enables the simulation of insertion and deletion mutations (indels) and dynamic population sizes (Jariani et al., 2019). All of these features make SANTA-SIM ideal for the purposes of our study, enabling the simulation of realistic viral sequence alignments that have undergone frequent recombination.

1.4) Aligning sequence data with recombination

Multiple sequence alignment is a first and critical computational step used for many bioinformatics analyses such as recombination detection, phylogenetic reconstruction, motif searches and structure prediction for proteins and RNAs (Chatzou et al., 2016; Gotoh, 1999). MSA methods use a wide variety of algorithmic approaches to infer homologous nucleotides or amino acid residues between a set of evolutionarily related DNA, RNA or protein sequences.

Multiple sequence alignment is an NP-hard problem (no known polynomial time solution) and thus alignment algorithms must rely on heuristics to save computational time (Just, 2001). One of the most commonly used heuristics are progressive alignment algorithms which adds sequences separately into a growing alignment by relying on a simple phylogenetic tree, referred to as the “guide tree”, to determine the orders in which the sequences are added to the alignment (Hogeweg & Hesper, 1984). The estimation of a guide tree is one of the key steps of most progressive alignment algorithms (Chatzou et al., 2016). This tree can be estimated based on pairwise similarities between sequences using a variety of tree-construction methods, most commonly Neighbour Joining (NJ) (Saitou & Nei, 1987) or Unweighted Pair Group Method with Arithmetic Mean (UPGMA) (Murtagh, 1984).

Since progressive alignment algorithms are dependent on this guide tree for the alignment process, and recombination has been shown to negatively impact the estimation of phylogenetic trees (Arenas & Posada, 2010; Scheffler et al., 2006; Schierup & Hein, 2000), it is reasonable to hypothesise that recombination might impact the accuracy of many commonly used alignment tools. One approach to test this would be to measure and compare alignment accuracy between sequence datasets with or without recombination for a specific alignment algorithm.

1.5) Aim

The aim of this study is to implement software capable of simulating realistic viral sequence datasets that undergo frequent recombination to (1) improve the recombinant identification accuracy of RDP and (2) evaluate the impact of recombination on the performance of multiple sequence alignment software.

1.6) Objectives

1. **Implement simulation software**

Find or implement simulation software capable of creating large, realistic sequence datasets that have evolved in the presence of frequent recombination. These sequence datasets need to be globally aligned to (1) save time for downstream analyses and (2) to serve as a reference alignment for comparison and benchmarking.

2. **Simulate realistic recombinant sequence datasets**

This simulation software should then be tested and used to create large numbers of alignments that could then be used to test and refine the accuracy of recombinant identification using machine learning approaches. Key to this objective is learning the parameter setup for the simulator to create diverse and realistic populations of sequences that mimic those commonly encountered when analysing real-world viral genomic sequence datasets. Full characterization of the recombination events is of particular importance and information about these events during the simulation process should be saved for downstream analyses.

3. **Test the accuracy of RDP5 recombinant identification and produce training datasets for machine learning**

These simulated alignments, and their simulated recombination events, should then be analysed by RDP5 to firstly determine the accuracy with which the various recombinant identification methods implemented in RDP5 identify recombinants and, secondly, produce extensive datasets containing the outputs of these recombinant identification methods for each detected recombination event. These datasets can then be used to train a statistical learning model which identifies recombinants from a recombinant/parent/parent sequence triplet.

4. **Train and evaluate machine learning models**

Train and evaluate the recombinant identification accuracy of a variety of machine learning models.

5. **Simulate alignments with realistic indels in the presence and absence of recombination to test the accuracy of multiple sequence alignment programs**

Use the simulation software to simulate sequence datasets without recombination but with realistic indels that could be used to test the accuracy of different multiple sequence alignment methods. Then simulate these datasets with recombination and compare the accuracy of alignment methods in the presence and absence of recombination.

6. **Align datasets with different alignment software**

Take the reference alignments and remove all gap characters. Then re-align both datasets with a wide variety of currently used alignment tools.

7. **Measure and compare software performance**

Using the simulated alignments as a reference, measure the accuracy of the alignment software on both the recombinant and non-recombinant sequence datasets. Statistically compare their performance between the two datasets to test for significant differences in alignment quality. The alignment quality difference will also be compared between the alignment software to validate earlier research and to inform tool choice for future research when dealing with sequences that have undergone frequent recombination.

Chapter 2: Improving the recombinant identification accuracy of recombination detection program

2.1) Abstract

Motivation. Recombination is a central evolutionary process that substantially changes the structure of genomes and shapes their evolutionary trajectory. Recombination detection is thus an important computational step in understanding the evolutionary history of nucleotide sequences, and the accurate identification of recombinant sequences is particularly important in the context of downstream phylogenetics-based sequence analyses. Evaluating recombination detection methods requires the simulation of sequence data, and the training of statistical learning models requires large, realistic datasets. The goal of this study was thus to (1) simulate large, realistic sequence datasets that have evolved in the presence of frequent recombination, and (2) to use these datasets to improve one of the computational steps used in the analysis of recombination by the computer program, recombination detection program 5 (RDP5), specifically: the identification of the recombinant from a recombinant/parent/parent triplet.

Results. To improve the accuracy with which RDP5 identifies recombinant sequences, we simulated the evolution of recombining sequences to produce large datasets that could then be used to train a number of machine learning models to accurately differentiate recombinants from their parental sequences. The artificial intelligence systems created using these models showed a substantial improvement in recombinant identification accuracy over the method currently implemented in RDP5 - with an increase in accuracy of up to 26 percentage points.

Availability and implementation. Our simulation software is a forked version of SANTA-SIM developed in Java. All source code is released and is available at: https://github.com/phillipswanepoel/santa-sim/tree/Recomb_and_align.

2.2) Introduction

2.2.1) Recombination and phylogenetic analyses

Recombination in viruses is an important evolutionary process that drives the generation of genetic diversity and accelerates adaptation. By joining together large blocks of genomes from distinct variants, recombination allows viruses to more efficiently explore fitness landscapes, and can lead to sudden and drastic changes in pathogenicity, transmissibility, cellular tropism, and host range (Lefeuvre & Moriones, 2015). Recombination as a mechanism, as well as its phenotypic consequences, is an important topic for a diverse range of research areas including molecular biology, virology and evolutionary biology (Pérez-Losada et al., 2015). It is also of practical concern to epidemiologists and clinicians, since recombination can lead to the sudden emergence of entirely new strains of viruses: one pertinent example being outbreaks of a recombinant strain of polioviruses in several regions (Combelas et al., 2011).

Besides the importance of recombination as an evolutionary mechanism, it also has implications for the computational study of viral sequences: particularly in the context of phylogenetics. All analytical techniques that rely on the inference of phylogenetic trees that accurately reflect the evolution of a set of organisms will be at least mildly confounded by recombination. For example, the presence of recombination has been shown to negatively influence phylogenetics based methods for ancestral sequence reconstruction (Arenas & Posada, 2010). Creating independent phylogenetic trees for distinct recombinant segments significantly improves the estimation of most recent common ancestors (Arenas & Posada, 2010). Ignoring the effects of recombination leads to inaccurate inferences when estimating substitution rate heterogeneity, when calibrating molecular clocks (Schierup & Hein, 2000) and when detecting evidence of natural selection (Scheffler et al., 2006). Detecting and accounting for recombination events is thus a critical step for downstream phylogenetic based analyses.

2.2.2) Recombination detection and recombinant identification

The suite of recombination detection methods that are implemented in RDP include the original RDP method (Martin & Rybicki, 2000). This method identifies potentially recombinant regions in a multiple sequence alignment (MSA) using average pairwise distances between each unique set of 3 sequences in the MSA. A moving window scans all phylogenetically informative nucleotide sites for regions where one of the three possible sequence couples within the triplet has lower pairwise identity than the other two sequence couples. These potential regions are then tested for statistical significance using a binomial approximation. Due to symmetry in signals, it is still not

clear at this point which of the three sequences is the recombinant and which are the “parentals”; or, more accurately, the sequences most closely related to the hypothesised ancestral parents.

Thus RDP needs to classify one of the sequences in the triplet as the recombinant using a separate computational method. While there are some recombination detection methods implemented in RDP that directly identify the recombinant, most are also reliant on this separate computational step for recombinant identification. Recombination detection methods implemented in RDP5 which also rely on this identification step include: BOOTSCAN (Salminen et al., 1995), GENECONV (Sawyer, 1989), MAXCHI (Smith, 1992), CHIMAERA (David Posada’s modified version of MAXCHI, unpublished work) and SISCAN (Gibbs et al., 2000).

This recombinant identification step in RDP is based on a weighted consensus of various statistical and phylogenetic methods (refer to the RDP5 manual for a summary of these methods; http://web.cbio.uct.ac.za/~darren/RDP5Manual.pdf). The weightings in this consensus are currently based on a heuristic method. Here we utilise simulated recombinant datasets together with a machine learning approach to improve on the accuracy of this weighted consensus method.

2.3) Methods and Materials

2.3.1) Computing global alignments

Although SANTA-SIM is a powerful forward in time simulator, it has two missing features that I considered vital for our study: (1) it does not output a recombination event history, and (2) it does not compute a global alignment. These two features were therefore implemented into a custom forked version of SANTA-SIM.

I was primarily focused on simulating the full evolutionary histories of sets of sequences. I therefore adapted SANTA-SIM to enable it to produce a “perfect” alignment using information from simulated indel and recombination events. A “perfect” alignment being one in which all homologous nucleotides share a column, with minimal gap characters added. I will broadly describe the methodology I have used to do this, but will omit some of the finer details and fine-tuning steps.

Specifically, SANTA-SIM was forked to create a version that both stores information about the recombination events present in a sample, and outputs a global alignment of sequences within the sample. This global alignment saves a lot of downstream computational time, avoiding the need to align thousands of sequence datasets. These alignments can also be used as references to evaluate alignment quality under different simulation setups.

2.3.2) Simulations in the absence of recombination

In the absence of recombination this problem is greatly simplified. Let us take a look at this simplified version of the problem before moving on to the complications caused by recombination. But first, let us define some terminology.

We will refer to insertions and deletions using 2-tuples, where the first element notes the position of the indel event, and the second element the size. If P denotes position, the p’th nucleotide in the sequence indexed from zero, and S denotes size, the number of nucleotides inserted or deleted by the indel event; we will refer to this indel event as (P, S). For example: (0, 5) would be an insertion of 5 nucleotides at the very start of the sequence, and (3, -3) would be the deletion of 3 nucleotides starting at the element with index of 3. It is important to note that P is the position in the sequence at the time of the event, a relative position in time which won’t necessarily be the absolute position in the sequence when it is sampled later.

For global alignment we are strictly interested in the length of the sequences and the relative positions of nucleotides, so with this terminology we can write the evolution of these properties over time as a series of indel operations. Let's look at a concrete example.
If we start with a sequence t = “ACCTTG”, and use t * (P, S) to denote an indel operation on t, the sequence could change over time as follows:

t * (1, -2) = “A~~CC~~TTG” = “ATTG”

t * (1, -2) * (2, 3) = “ATTG” * (2, 3) = “ATXXXTG”,
where X = an inserted nucleotide.

If we have access to this information, namely where and in what order indel operations took place to create some sequence, we can use this information to align it with other sequences that evolved from a shared origin. We can do this by “undoing” the indel event, i.e. performing an operation that will preserve the distances from the start of the seed sequence of homologous nucleotides. For deletions, this inverse operation corresponds to inserting gap characters where the deletion took place. For insertions we delete the inserted characters. Once all the insertions and deletions have been “reversed” this way, we can insert gap characters to cover the insertions. Let us go through another example and align sequence t with itself post indel operations.

2.3.3) Aligning two sequences using indel operation histories

Let s = t * (1, -2) * (2, 3), then we can align t and s as follows. We will consider two sequences aligned if we can find the minimal number of gap characters needed to keep homologous nucleotides in the same vertical position. Let us find this alignment by performing the inverse operations.

	0	1	2	3	4	5	6
s	A	T	X	X	X	T	G
t	A	C	C	T	T	G	

To get an alignment of s with t, we “undo” the operations performed on t. This is done in reverse chronological order. So if we have s = t * (1, -2) * (2, 3), we start by undoing (2, 3), i.e. we perform the inverse operation $(2, 3)^{-1}$. Since this is an insertion, the inverse corresponds to a deletion of size 3 at position 2: (2,-3)

$(2, 3)^{-1} = (2, -3)$

	0	1	2	3	4	5	6
s*	A	T	T	G			
t*	A	C	C	T	T	G	

Now to perform the inverse of (1, -2). So as to “undo” the deletion, we insert gap characters. Two gap characters are inserted at position 1.

$(1, -2)^{-1} = (1, 2)$

	0	1	2	3	4	5	6
s*	A	-	-	T	T	G	
t*	A	C	C	T	T	G	

Now the final step is to re-insert all the insertions that have been removed, adding gap characters to all sequences not affected by the insertion. Since two gap characters have been added to s, the insertion of (2,3) now moves to (4, 3).

(4, 3)

	0	1	2	3	4	5	6	7	8
s*	A	-	-	T	X	X	X	T	G
t*	A	C	C	T	-	-	-	T	G

2.3.4) Computing a global alignment for recombinant sequences

When we compute a global alignment using an indel history, stored indel positions are only accurate relative to the sequence’s state at the time of that event: an insertion of (1, 3) is no longer at position 1 if it is followed by an insertion (0, 10) which shifts its position by 10 nucleotides. After the insertion this newer event it is now at position 11 in the sequence. An indel’s position at the time of the event, has already been shifted by previous events and subsequent indels will shift it even further.

Fortunately we could ignore the complications caused by these constantly shifting positions by dealing with indel operations in reverse chronological order and "undoing" their effects. This ensures the latest event position is always accurate. But once recombination comes into play we can't ignore these complications any longer. Let A and B be two sequences recombining to form a sequence C, where Z is the most recent common ancestor of A and B. Then A and B have both evolved from Z with separate sequences of indel operations.

$A = Z * (P_1, S_1) * (P_2, S_2) * \ldots * (P_n, S_n)$
$B = Z * (X_1, Y_1) * (X_2, Y_2) * \ldots * (X_n, Y_n)$

The stored positions of these indel events, P_n and X_n for example, are both dependent on the positions and sizes of previous indel events during the evolutionary history of a sequence. For P_n and X_n the position stored at the time of the event and the actual current position in the sequence match. This is not the case for earlier events. For example P_2 is the position of the event at the time that it happened, but after events (P_3, S_3), (P_4, S_4), …, (P_n, S_n) the position of P_2 will be a function of the positional shifts caused by these events. If we call the updated, current position of P_2 as C_2, this function counts the positional shifts of subsequent events and can be written as follows.

$$C_2 = P_2 + \sum_{i=3}^{n} \begin{cases} S_i & if P_i <= P_2 \\ 0 & otherwise \end{cases}$$

The same function holds backwards in time such that, the position P_2 is also a function of P_1, but since this indel event has already happened, its shift has already been taken into account. However, it is illustrative to consider what happens to these positions when A and B recombine.

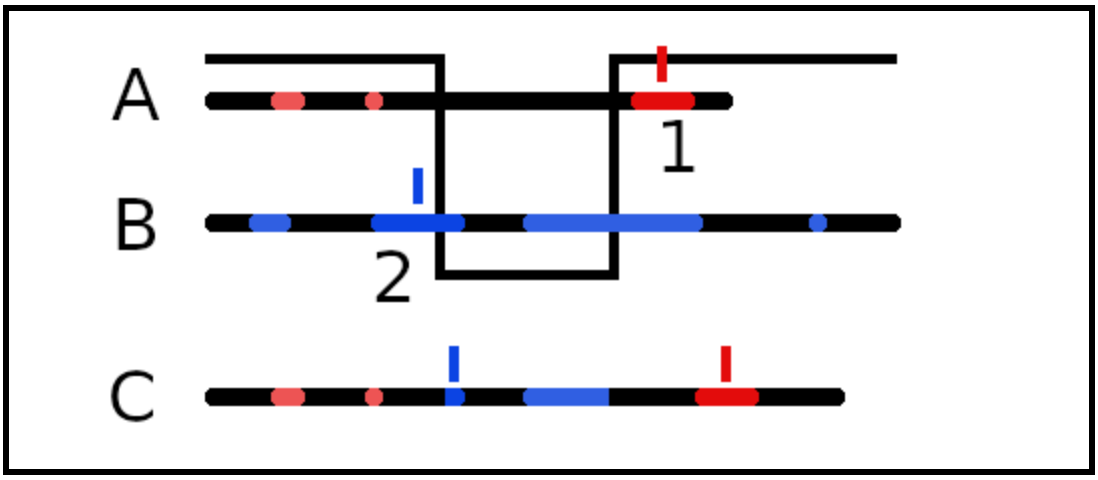

Figure 2.1: Example of two sequences with distinct indel histories recombining. Illustrated example of two sequences, A and B, recombining to form a new sequence. Red regions indicate insertions unique to A, with blue indicating unique insertions for sequence B. Note how sequence C inherits a combination of both A and Bs indels, some of which have also been spliced.

For indel events 1 and 2, pictured in figure 1, let's note how these operations have changed after recombination. Assume that both events are the most recent indel events. Following recombination their positions in the sequence will shift and both indel positions will then be a function of a different set of indel events; namely those that have been inherited by the recombinant, C. For example, event 1's position is now also shifted by the two blue indels inherited from sequence B - before recombination event 1 was only shifted by the two red indels.

In addition, event 2 has been "spliced" and is now smaller. This is a concrete example of why we can not continue "undoing" the latest recombination event to align recombinant sequences like C. The stored positions of the indels in C will no longer be accurate. The actual positions of these indels, which are a function of all the accumulated shifts from other indel events post-recombination, need to be recalculated. This is done by updating their positions as they are at the time of recombination.

2.3.5) Calculating homologous breakpoints

As earlier, let A and B be the two sequences that are recombining, both with distinct indel histories. Recombination with more than one breakpoint can be calculated by repeating the steps for one breakpoint, so let us focus on one breakpoint and call this breakpoint position K.
If we align A and B, then we know which nucleotides are homologous. In doing so we can also calculate homologous breakpoint positions. SANTA-SIM did not have support for homologous breakpoints, so we implement it in this step, calculating the homologous breakpoint as we align the two recombinant sequences. If K is the breakpoint chosen in A, we will find the homologous position of K in B called K'.

STEP 1

Sort indels by position instead of time. This requires all stored indel positions to be updated to their positions at the time of recombination.

$$C_j = P_j + \sum_{i=j+1}^{n} \left\{ \begin{matrix} S_i \ if \ P_i <= P_j \\ 0 \ otherwise \end{matrix} \right\}$$

C_j needs to be calculated for all P_j. But this formula ignores one detail: indels can also interact with each other, for example a deletion could remove part of an insertion or an insertion can bisect another insertion. Thus we need to update indels in more than just position, they also need to be updated after interactions. This is done every time a new indel event happens: a search is done for overlaps with existing indels, and their interactions are resolved. Insertions can be split into two by other insertions. Insertions can be shortened, split or removed entirely by newer deletions. Existing deletions are not modified by subsequent indel events.

STEP 2

Now, given some breakpoint position K in sequence A, we want to find the homologously equivalent nucleotide position, K', in sequence B. This requires calculating how much K has been "shifted" by indel operations in A relative to B.

Given the positionally sorted, updated indels P for sequence A together with position K, we can calculate the shift of K in A as follows:

$$FS_a = \sum_{i=0}^{n} \begin{cases} S_i & \text{if } Pi < K \\ (K - P_i) & \text{if } Pi < K \text{ and } (P_i + S_i >= K) \\ 0 & otherwise \end{cases}$$

The second term here, ($K - P_i$), refers to the situation in which a breakpoint falls inside of an insertion, i.e. only part of the insertion has been inherited and has shifted the breakpoint.

So now we have the "starting" position of K: the position where K would be in A in the absence of the moving effects of indel events. This position is $K - FS_A$. This is the position we use as a starting point to calculate the frameshift of K in B, FS_B. We must use the position $K - FS_A$ and not simply K, since the position of K in A is relative to the indel events in A and is not relevant to the indel events inside of B.

The following is the pseudocode to calculate FS_b

```
L ← K − FSa
FSb ← 0

  for (Pi, Si) in indels do
      if Pi < L then
          if Pi + Si < L then
              FSb ← FSb + Si
              L ← L + Si
          else
              if Pi + Si >= L then
                  FSb ← FSb + (L − Pi)
                  L ← L + (L − Pi)

return FSb
```

The difference from the formula used for A is that the position that indels are being compared to is shifting. P_i is being compared to L, which shifts by S_i. This is due to K in A being a given, static position for the breakpoint. In B we are now interested in calculating where $L = K - FS_a$ would "end up" after being shifted by the indels in B. Thus the positions being compared are not static.

We have calculated FS_b and FS_a and can now calculate the final homologous breakpoint. We have essentially “undone” the indel shifts in sequence A that have shifted the position K at the time of recombination, and then, starting from this indel free position, we have added the shifts caused by the indels in B.

Homologous Breakpoint = $K - FS_a + FS_b$

2.3.6) Aligning recombinant sequences

So now that we have calculated K’, we can form the recombinant sequence together with its newly formed indel event history. We use the respective breakpoints for A and B, and the recombinant sequence inherits indels which fall within the inherited sequence blocks. Some indel events will be spliced by the breakpoint position: indels do not necessarily have to be inherited in full.

Note that for alignment purposes we need to keep track of shared insertion events. Let’s say sequence A has an insertion (3, 6), and the recombinant has inherited a spliced version of that insertion: (3, 4). When aligning these two sequences these two insertions still share homologous nucleotides, and should be treated as such. Thus we add another term to our stored indel tuples, a unique ID which is shared by all indel events with a common evolutionary origin. This unique ID persists even if an indel has been modified by subsequent recombination or indel events. We also store another term, the “shift” or “offset” of the indel event, this is an integer that is updated when an indel is spliced or shortened. This allows us to align indel events to each other later on.

So when we sample recombinant sequences at the end of our simulation, each sequence has its own stored indel history, one which has been modified and updated at each recombination event so that it is still accurate. We can use these indel histories to output a multiple sequence alignment for any sample of sequences.

2.3.7) Outputting an MSA given sampled sequences with stored indel histories

STEP 1:

Sort indels by position for all sampled sequences.

STEP 2:

Remove, or “undo”, all indels from the sequences. This is similar to what we did for non-recombinant sequences, except we manually update all indel positions to be accurate at the sampling time, and so we remove them in reverse positional order instead of reverse chronological order. The reversed positional order ensures that removing indels does not shift the position of indels later in the sequence string. The “remove indel” function returns a string with gap characters added at all deletion positions, and with all inserted nucleotides removed. These insertions are stored in an insertion mapping. The unique insertion IDs are mapped to (index, subsequence, position, size, offset). This mapping informs how the insertions need to be re-added to the MSA in step 3.

Index: A set that stores the indexes of the sequences in the MSA which share this insertion.
Subsequence: The inserted nucleotide sequence. These can be distinct, but all insertions which share an ID will have some homologous nucleotides in common.
Position: The “indel free” position of the insertion (i.e. the position once its shift has been subtracted).
Size: The number of nucleotides in the insertion.
Offset: The number of nucleotides by which this specific variant of the insertion has been “shifted”. For example, if an insertion has its first two nucleotides removed by a subsequent deletion or recombination splicing, this offset value will be 2. This is what enables us to align multiple insertions with a shared evolutionary origin, even if they have been shifted and spliced over time.

STEP 3:

Now we iterate over all unique insertion IDs stored in the insertion mapping, these are the full set of indels present in the MSA. We call the “add insertions” function on the set of sequence strings. The function works as follows:

1. Sort the insertion mapping by the position of the indels. This is to keep their positions consistent with their stored positions.

2. Create two hash maps (integer index -> integer count), gaps_added and insertions_added. These mappings keep track of the accumulated shifts due to added gap characters and insertions for each sequence in the MSA. This is basically just a map of relative movement in 2 dimensions, allowing us to know relative distances between homologous nucleotides for any pair of sequences in the MSA. All columns are aligned at the start, before the insertions have been re-added. So as long as we keep track of gaps and insertions added, we can keep these columns aligned.
3. Iterate over the sorted insertion events in the insertion mapping.
4. Iterate through all sequences affected by a specific insertion event.
5. If the index of a sequence is not in that event, add gaps equal to the maximum insertion size for that insertion event. If a sequence contains that event, add the subsequence of insertion, and also gaps if the sequence has an offset > 0 for that insertion. The position of these added gaps and/or nucleotides are shifted for each sequence by the value of gaps_added and insertions_added for the sequence index key.
6. Update gaps_added and insertions_added for all sequences.
7. Remove gap only columns.

This series of steps yields a globally aligned sequence dataset together with a recombination history.

2.3.8) Simulating datasets for machine learning

I simulated an initial dataset consisting of five distinct XML configurations for a total of 5508 alignments for training a ML model, and an additional 97 larger and more complex alignments for validating trained ML models. The goal here was to create as much evolutionary plausible diversity in parameters as was feasible, given time constraints.

The goal for the validation datasets was to make a more realistic and complex simulation setup. The seed sequence used was a complete reference genome for SARS-CoV-2 (NCBI Reference Sequence: NC_045512.2). For the spike gene and ORF1ab regions of the genome, codons under strong purifying selection were manually specified. About 30% of the sites were removed, since the selection pressure was so strong that surviving recombinants became too rare for the purposes of this dataset.

Table 2.1: Simulation set up for static parameters

Seed sequences were copied from NCBI ref sequences for 5 different viruses. Population size was either dynamic or static. Various fitness functions were used with varying levels of intensity and number of sites under selection. Some datasets were simulated with recombination hot or cold spots, others were simulated with none. Dynamic populations were modelled as having a logistic growth rate where: N = initial population size, R = growth rate and K = carrying population. Length of indels were modelled with a negative binomial distribution, where r = number of successes till length is specified, p = probability of success. Refer to the additional data to see the XML simulation setup files.

	XML1	**XML2**	**XML3**	**XML4**	**XML5**	**Validation dataset**
Sequence Length	3822	9181	5800	19025	10727	29902
Population Type and Size	Dynamic N = 100, R = 1.1, K = 10000	Static N = 4000	Static N = 6000	Dynamic N = 1000, R = 1.2, K = 50000	Static N = 5000	Static N = 3000
Fitness function	Purifying fitness	Exposure dependent	Frequency dependent	Age dependent + Purifying	Purifying	Purifying, based on sites with empirical evidence of selection
Recombination hot and cold spots	Hotspot: 1000-1200 Coldspot: 50-150	Hotspot: 4000-4200 Hotspot: 9000-9100 Coldspot: 6200-6300	None	None	None	None
Indels	None	None	None	None	Yes Rate = 0.035, r = 1 p = 0.4	Yes Rate = 0.025, r = 3 p = 0.3
Virus and region	SARS-CoV-2 Spike	HIV-1	SARS-CoV-2 ORF1ab	Bombali Ebolavirus	Zika Virus	SARS-CoV-2 whole genome

Table 2.2: Simulation runtime parameters
Simulations were varied across all combinations of parameters in this table. Parameter choices were constrained by runtimes and a need to retain these within empirically feasible ranges. Another important factor that was considered for these rates was pairwise sequence similarity in the final sample. The validation datasets had considerably higher recombination rates, starting with a minimum of 0.06.

Generation Count	Recombination Rate	Mutation Rate	Sample Size
2500	0.005	4e-5	50
3250	0.01	8e-5	100
4000	0.02	12e-5	200

2.3.9) Using simulated alignments to generate a machine-learning ready dataset for the identification of recombinant sequences

RDP5 uses a linear combination of 40 metrics calculated on a triplet of sequences to determine which of the 3 is the most likely recombinant, and which are the likely major/minor parents. To train a supervised machine learning model to correctly identify recombinants from a sequence triplet we thus needed the following for a training dataset: A plausible triplet of sequences, the ground truth (i.e. which of these three is actually the recombinant), and a set of metrics calculated on the sequences which the model could use to infer the correct recombinant.

To do this a custom version of RDP5 was developed which used the simulator output to generate the machine learning dataset. The alignment file was used together with the “recombination history” file. This recombination history file lists recombination events and information about them: event breakpoints, ages of events, and which sequences have inherited the events. RDP5 was then used to naively scan the alignment file to detect recombination events following which it matched up detected events with those provided in the recombination history file. Specifically matches to a simulated recombination event were identified as RDP5 detected events (1) where a recombinant carrying evidence of a given simulated event (as indicated in the simulation-derived recombination history file) was among the triplet of sequences used to by RDP5 to detect a recombination event (but not necessarily identified by RDP5 as a recombinant)and (2) where the 95% confidence intervals of the RDP5 detected

recombination breakpoint locations bounded those of the simulated recombination event Thus only events that matched up reasonably well with the actual recombination history were retained for purposes of training ML models. For these matched events the metrics that RDP5 uses for recombinant identification were calculated for each sequence in the parent/parent/recombinant triplet. The final machine learning training dataset thus consisted of a 3-dimensional vector with one binary label to designate the recombinant, and 40 features which were derived from the recombinant identification statistics calculated by RDP5.

2.3.10) Training and evaluating models on dataset

A variety of models were trained for recombinant identification using the training dataset, these included: logistic regression, support vector machines (Cortes & Vapnik, 1995), gradient booster (Friedman, 2001), random forest (Ho, 1995) and a neural network (McCulloch & Pitts, 1943). Machine learning modelling was done in Python using packages from scikit-learn (Pedregosa et al., 2011) and tensorflow (Abadi et al., 2015) for the training of neural networks. The training features were pruned by discarding the three weighted consensus statistics that RDP5 generated for recombinant identification (i.e. the three statistics that RDP5 ultimately uses to identify the recombinants sequence) as well as five other RDP5 calculated metrics which had very low variance across all the training data. Therefore the machine learning models were trained on the remaining 32 features from the potential 40 features. For the neutral network all hyperparameter tuning was also done on a distinct training dataset. These models were then evaluated on a validation dataset. This validation dataset was generated from a completely distinct simulation set up to that used to generate the five training datasets: a setup which was considerably more complex in terms of both indel frequencies and recombination events.

2.4) Results

Machine learning recombinant prediction

RDP's recombinant identification accuracy for the five training datasets ranged between 65% and 75%. This measurement of accuracy was calculated directly from the three weighted consensus statistics that RDP5 output for the subset of simulated recombination events that RDP5 identified and which could be confidently matched with simulated recombination events: statistics that were not included in the 32 used to train machine learning models. Even the simplest ML model, logistic regression, performed considerably better than did RDP5 on all five of the simulation datasets, achieving over 90% recombinant identification accuracy on all of them. On the most complex dataset, RDP only scored 65% accuracy, with the logistic regression scoring 91%.

Table 2.3: Logistic Regression performance on training dataset
Performance metrics for logistic regression model. Model was trained on simulated data and tested on a simulated validation dataset that had a unique combination of parameters as well as added complexity in terms of indel and recombination frequency.

	Precision	**Recall**	**f1-score**	**Support**
Parent	0.95	0.91	0.93	11342
Recombinant	0.83	0.91	0.87	5671
Accuracy			0.91	17013
Macro Avg	0.89	0.91	0.90	17013
Weighted Avg	0.91	0.91	0.91	17013

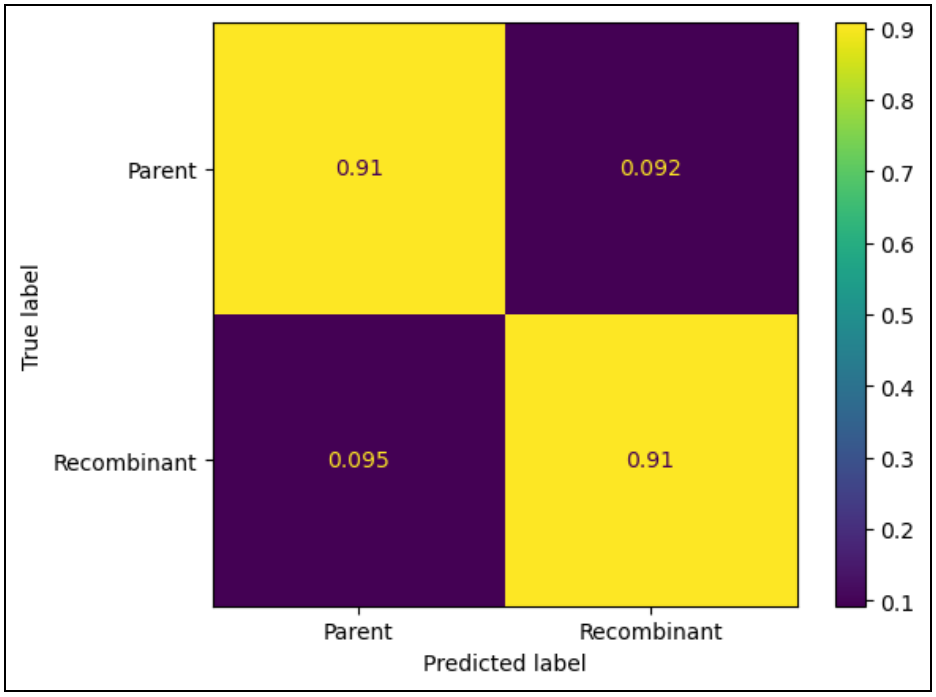

Figure 2.2: Confusion matrix for logistic regression performance on validation dataset. This matrix shows how often the model correctly identified a sequence as a recombinant or a parent, as well as the misclassification rates.

Several different model types were also trained (Gradient boosters, Random Forests, Decision tree, Naive bayes & SVMs), but all performed similarly to the logistic regression model. A simple neural network was also trained. When comparing the logistic regression to the neural network (NN), the NN performs slightly better in terms of precision, but not better in terms of recall, as shown in figure 2.2. Meaning that when it classifies a sequence as a recombinant it is correct more often than the logistic regression, but the recall is identical - they are able to “detect” sequences as recombinants equally well, but the neural network has fewer false positives.

Table 2.4: Neural network performance on validation dataset.

	Precision	Recall	f1-score	Support
0	0.89	0.89	0.89	1418
1	0.9	0.92	0.91	1871
2	0.93	0.92	0.93	2382
Accuracy			0.91	5671
Macro Avg	00.91	0.91	0.91	5671
Weighted Avg	0.91	0.91	0.91	5671

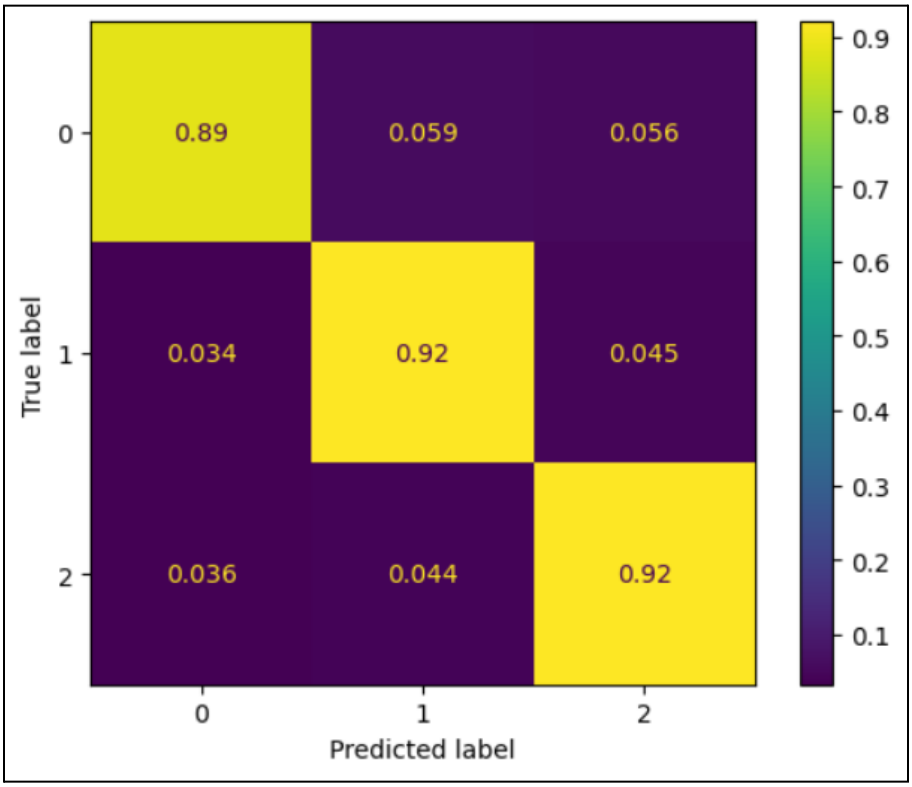

Figure 2.3: Confusion matrix for neural network.
This matrix shows how often the model correctly identified a sequence as a recombinant or a parent, as well as the misclassification rates. 0 refers to the recombinant sequence, with 1 and 2 both being parents.

2.5) Conclusion

While testing and validation on real sequence data has not been performed yet, our simulation results show a significant increase in recombinant identification accuracy for our machine learning models over the method currently implemented in RDP5. All machine learning models showed similar results. This suggests that with this specific set of training features, even a relatively simple model such as a logistic regression is able to reach optimal performance, given the quality of the training data and feature set. The constraint in model performance is not model complexity or the quality of fit, but the input data.

While this is already a promising result, future research could improve recombinant identification even further. Since all models are constrained by the features, there are two approaches that should lead to even better identification. One would be to improve the realism and diversity in simulated alignments. The second would be to implement additional recombinant identification features in RDP5. Implementing additional features is especially promising for neural networks, which are able to synthesise varied and complex data. Examples of such features could be: the sequences of the sequence triplet, phylogenetic trees encoded into a machine learning-ready format, partitioned sequence regions (for example dividing sequences by recombination event breakpoints) or secondary structure information inferred from the sequence data.

Chapter 3: Evaluating the performance of alignment software on recombinant and non-recombinant sequences

3.1) Abstract

Motivation. Multiple sequence alignment is an important first step for a wide variety of downstream computational analyses. To our knowledge no study exists that specifically tests the impact of recombination on alignment quality.

Results. To investigate the impact of recombination on the performance of alignment software we simulated two sequence datasets: one with recombining genomes and one without. A customised simulation program was used to compute an optimal alignment for these datasets. Both datasets were then re-aligned by a range of alignment software. When comparing alignment quality between the different alignment tools our results agree with earlier software comparisons. Although our study had some limitations, the results suggest that, in general, recombination has no substantial impact on the accuracy with which commonly used alignment programs align virus-like sequence datasets. Yet it remains plausible that a future study that controls for indel counts and has a more realistic simulation setup could show a significant impact of recombination on alignment accuracy.

3.2) Introduction

3.2.1) Evaluating alignment software packages

Benchmarking approaches have attempted to objectively evaluate MSA packages by comparing a metric of alignment quality based on standardised inputs. Lantorno et al. (2014) identified four main approaches to MSA benchmarking.

1. Simulating evolution of sequences with known homology
2. Using known consistencies between several alignment techniques
3. Using three-dimensional protein structure encoded by sequence data
4. Knowledge or assumptions about phylogeny of data

This study follows a simulation based approach. Simulating the evolution of a sequence allows one to keep track of indel and recombination events, and thus also the “true” relationships between homologous residue positions. Since the indel events are known, this allows the creation of a “perfect” or “true” reference alignment which can be used to assess alignment quality (Lantorno et al., 2014).

To measure this alignment quality one needs a metric for the similarity between the reference and tested alignments. Two commonly used metrics are the sum-of-pairs (SP) and true column (TC) scores (Lantorno et al., 2014; Shafee & Cooke, 2016). The SP metric counts all pairs of sequences in a MSA which share the same residue in a column when compared to the same column in the reference alignment. The TC score only counts columns which are identical between both alignments. These metrics are both straightforward to calculate, but have the issue of scaling non-linearly with the similarity of a particular site (Shafee & Cooke, 2016).

AlignStat is an R software package which measures similarity between MSAs by aligning them to each other. AlignStat implements a method based on a matrix of equivalency functions to map positional equivalencies between the homologous residues in the compared alignments. One of the metrics it can output is the overall similarity score. This score calculates a weighted average similarity of the two MSAs which is equal to the sum of matched characters over the total number of characters, excluding conserved gap characters (Shafee & Cooke, 2016).

3.2.2) Recombination and its impact on alignment

To my knowledge there have been no published studies which have directly measured or even theoretically discussed the impacts of recombination on alignment quality. While there are numerous benchmarking studies which compare alignment quality between different software packages (Nuin et al., 2006; Sievers & Higgins, 2020; Thompson et al., 1999, 2011), none have made a direct comparison between recombinant and non-recombinant sequence data.

3.3) Methods

Alignments were simulated using the custom forked version of SANTA-SIM described in chapter 2. This version outputs a global alignment calculated from the indel and recombination events in each sequence's evolutionary history. Two hundred alignments were simulated, one hundred with zero recombination events, and one hundred with a recombination rate of 0.05. This means a 5% chance of recombinant replication per replication cycle if a dual infection happened. All the other simulation parameters were identical. Simulation sequences were seeded from a single SARS-Cov-2 isolate (GenBank Accession Number NCO45512.2), taking only the first 1500 nucleotides to keep runtimes low. Refer to additional data for the exact simulation setup used.

All gap characters were removed from these alignments and then these were re-aligned with a selection of different alignment programs. These programs included: Clustal Omega (Sievers & Higgins, 2014), Dialign (Morgenstern et al., 1998), FAMSA (Deorowicz et al., 2016), FMAlign (Liu et al., 2022), KAlign (Lassmann & Sonnhammer, 2005), MAFFT (Katoh et al., 2002), POA (Lee et al., 2002), ProbAlign (Roshan & Livesay, 2006), ProbCons (Do et al., 2005), and StatAlign (Novák et al., 2008). All were run on default settings, with the exception of Dialign which was run separately with the -n or -nt flag enabled.

These re-aligned sequences were then run through AlignStat (Shafee & Cooke, 2016) on default settings. The simulated global alignment was set as the reference alignment. AlignStat returns a score between zero and one based on how closely an alignment matches the reference. This is done by aligning the two MSAs with each other and measuring their distance.

The scores for each alignment were stored in a csv file, and gap characters were counted with a python script for each alignment. Statistical analysis was done in R by fitting a multiple linear regression model using the lm function. See additional data for the scripts and raw data.

3.4) Results

3.4.1) Comparing overall performance of alignment packages

Using AlignStat allows us to measure how closely an alignment generated by a specific program matches with the simulated reference alignment. This is done by aligning two distinct MSAs with each other to determine how well they align conserved nucleotides to the same column. This distance between the MSA and the reference is then outputted as a similarity score which ranges from zero to one.

Table 3.1: Mean AlignStat similarity score for all data
AlignStat similarity score averaged across both recombinant and non-recombinant datasets. ProbAlign, StatAlign, FMAlign and MAFFT were clearly best performing programs, with only a smaller than 0.003 difference in mean scores between the reference alignments and the realigned sequences.

Software	Mean Similarity Score
ProbAlign	0.9575661824
StatAlign	0.9557369903
FMAlign	0.9550714695
MAFFT	0.955001379
ProbCons	0.9482276474
Dialign-n	0.9421328252
KAlign	0.9388519736
FAMSA	0.9303964357
Dialign-nt	0.8814188164
POA	0.8769624675
Clustal Omega	0.8152983566

Comparing our results to earlier benchmarks, the results are similar. In a comprehensive study by Thompson et al, the rankings of the software tested in both our studies was identical (Thompson et al., 2011). Specifically, MAFFT > ProbCons > KAlign > Dialign > Clustal; although in their study they tested clustalw and not Clustal Omega.

Another benchmarking study used the QuanTest2 benchmark which compares alignments according to their accuracy in predicting secondary structures (Sievers & Higgins, 2020). In this study software was tested with a variety of program settings as well as with default settings. When only comparing default settings, the score rankings were identical to our study except for one major exception: ClustalOmega was the best performing software on default settings instead of the worst.

A third study by Nuin et al. used Simprot to simulate reference alignments (Nuin et al., 2006). They also found similar results to mine when comparing alignment accuracy between software packages tested. MAFFT and ProbCons were the two best performing, with Kalign being in the middle of the pack. POA and Dialign were among the worst performing in terms of accuracy, reflecting the results in our study.

3.4.2) Effect of recombination on alignment quality

Surprisingly, when directly comparing the alignment accuracy of recombinant and non-recombinant datasets, median similarity scores between realigned and reference datasets were higher for the recombinant datasets than for the the non-recombinant datasets for all alignment programs tested, as shown in figure 3.1. This was contrary to expectations. Intuitively, recombination leads to a more complex MSA problem since nucleotides are not only mutated by indels, but are also transferred and recombined in blocks: a pattern of evolution that should confound the accuracy of the guide-trees used by most alignment programs.

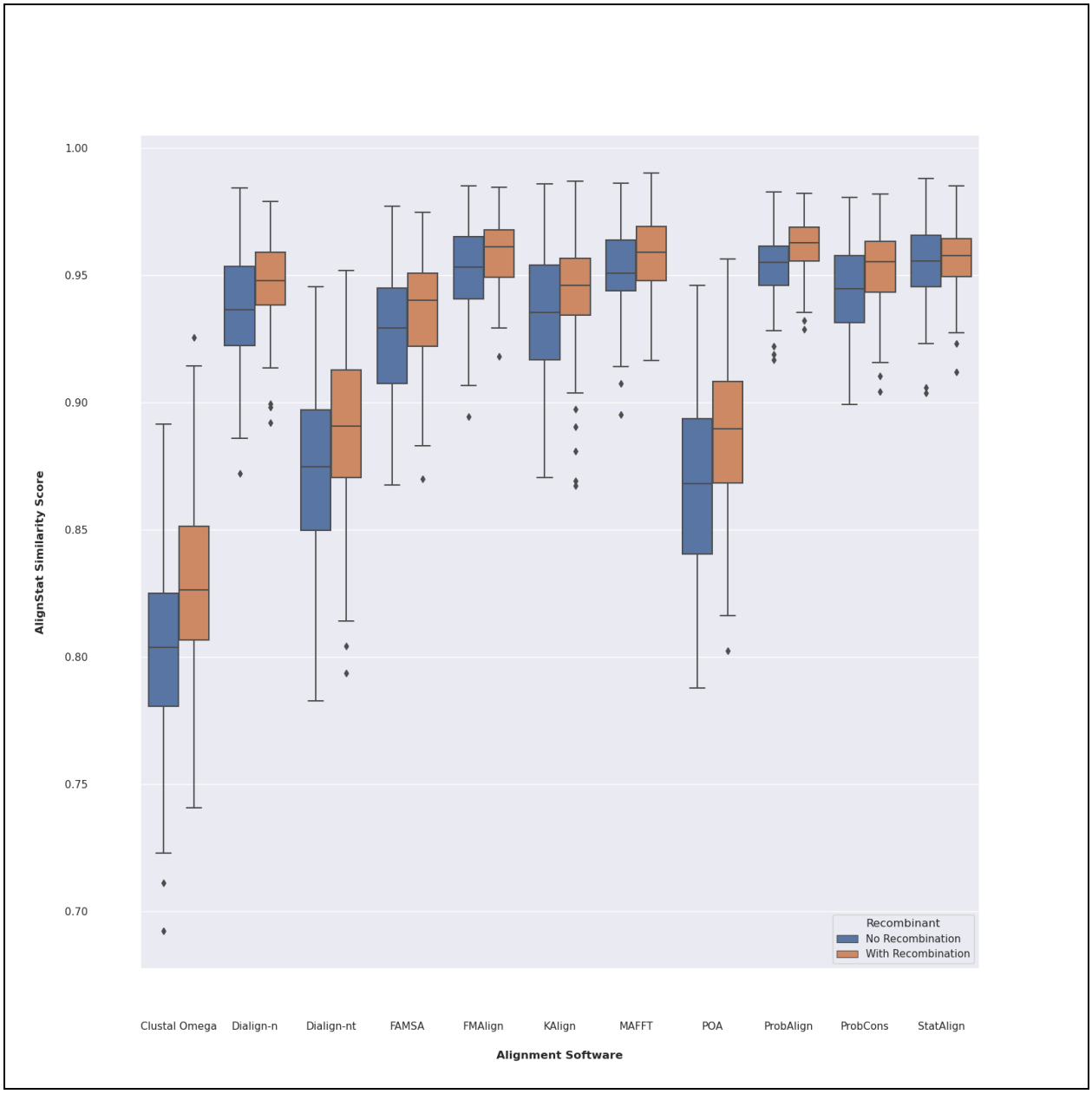

Figure 3.1: AlignStat similarity scores for the non-recombinant and recombinant datasets. Similarity scores for recombinant and non-recombinant sequence datasets for each alignment software. Score is calculated by comparing alignments with the simulated reference. The AlignStat similarity score measures the distance between the two MSAs by aligning them to each other.

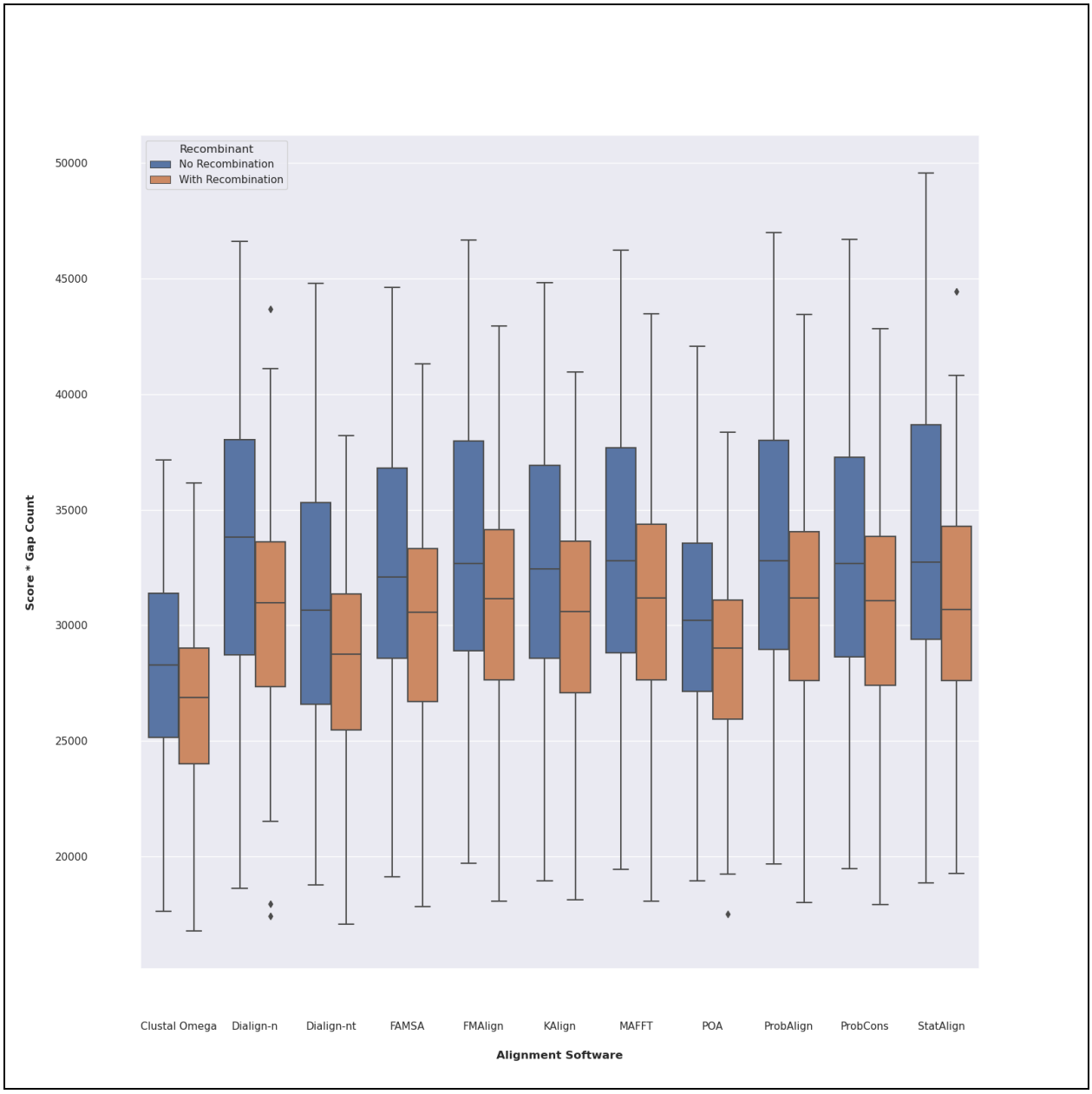

Figure 3.2: Adjusted scores for the non-recombinant and recombinant datasets. Similarity scores for recombinant and non-recombinant sequence datasets for each alignment software. Score is calculated by comparing alignments with the simulated reference. The AlignStat similarity score measures the distance between the two MSAs by aligning them to each other. This score was then adjusted by multiplying each MSA by the total number of gap characters.

A sequence dataset's alignment difficulty is directly proportional to the amount of indels present in the collection of sequences. The true, simulated indel count in the alignment thus has a large influence on the difficulty of the alignment and thus the similarity score. Looking at the reference alignments, the recombinant sequence datasets had an average gap count of 32294.91, whereas the non-recombinant dataset only had an average gap count of 35280.86. The non-recombinant dataset thus contained more indels and was likely more difficult to align. This is a shortcoming of our study and for future research datasets with equalised indel counts would give a more accurate measurement of alignment quality.

This difference in indel count is a direct result of the simulation parameters used. These sequences are simulated in an evolutionary context with a fitness function that selects on chosen codons. In the particular simulation setup we used to generate these alignments, recombination accelerated the removal of indels that don't contribute to fitness, a phenomenon related to Muller's ratchet (Muller, 1964). This is not a general truth for sequences simulated with SANTA-SIM, and is entirely dependent on the exact simulation parameters used, in a neutral evolution context the indel counts for recombinant sequence datasets should be very similar to non-recombinant sequence datasets.

So there is a confounding variable which is shadowing the effect of recombination on the alignment scores - the indel counts. In a crude attempt to control for this variable for visualisation purposes we created an adjusted alignment score which is influenced by the indel count. The results for the adjusted scores are shown in figure 3.2. This adjusted score is calculated simply as score times gap count. This adjusted score only works on the assumption that there is a directly linear relationship between gap count and alignment difficulty. For two alignments with identical scores the one with a higher gap count will have a higher adjusted score, thus accounting for the fact that it was harder to align this more complex dataset with more indels.

A future study with controlled indel counts is needed for a more conclusive result, but using these adjusted scores shows a consistent decrease in scores when recombination is added to the simulation setup, as shown in figure 3.2. This is more in line with intuition and shows there is at least some evidence that recombination negatively impacts alignment accuracy.

3.4.3) Quantifying recombination's effect on alignment scores

We can try and control for gap count by fitting a linear regression and thus controlling for gap count as a variable. We used R (R Core Team, 2022) to test the association between score, gap count and the presence of recombination. Using least squares regression I found a significant negative association of gap count with alignment score ($P < 2 \times 10^{-16}$), with a coefficient of -2.218×10^{-6}. This coefficient might seem small, but this is the effect of a single gap character in alignments that average over 30 000 gap characters. No significant association was found between recombination and alignment score (P = 0.0823, coefficient = 0.003772). We also fit a regression to each individual software to compare the impact recombination had on each alignment program.

Table 3.2: Effect of recombination on alignment scores for different alignment programs. A least squares linear regression was fit with alignment score as the response variable and gap count and recombination as the predictor variables. Recombination is coded as a categorical variable. A negative coefficient corresponds to a lower score in the presence of recombination.

Software	Coefficient	P-Value	Significance Level
MAFFT	7.382e-05	0.968	
Clustal Omega	-0.090358	0.00119	**
Dialign-n	-0.013469	0.0000188	***
Dialign-nt	-0.009865	0.0353	*
FAMSA	-0.001394	0.588	
FMAlign	-0.0018622	0.29	
KAlign	-0.001192	0.669	
POA	-0.007888	0.0156	*
ProbAlign	-0.0036104	0.0159	*
ProbCons	-0.003845	0.0477	*
StatAlign	-0.001046	0.669	

More precise measurement of the impacts of recombination on alignment quality would require a novel technique to normalise indel counts in simulated alignments. Here we have attempted to control for indel count by fitting a regression. Although with this approach it is apparent that recombination likely has only a slight impact on alignment quality, for six of the tested alignment programs there was nevertheless a statistically significant decrease in alignment score in the presence of recombination. It must be stressed, however, that the effect size is very small, with the largest being a 1.35% decrease in score for Dialign with the -n flag. For five of the tested programs no significant reduction in alignment score was found in the presence of recombination, for MAFFT in particular there is strong evidence of no reduction of alignment quality in the presence of recombination, as shown in table 3.2.

3.5) Conclusion

Here we have evaluated the performance of different alignment programs to compare their performance between alignments with no recombination and those with frequent recombination. Although there are difficulties to resolve shortcomings in the simulation methodology that I used, the results suggest that recombination at the levels simulated are likely to have only a marginal impact on alignment quality. This would suggest that one should continue to choose the best alignment software for a given analysis based on metrics which don't account for recombination specifically, even if the dataset that is to be aligned contains evidence of frequent recombination.

For future research, a more conclusive result can be obtained by controlling exactly for indel count in the simulation setup. This would mean having identical amounts of indels for each alignment problem. A longer sequence and more realistic parameter setup would require a significant increase in computational time, but would make the results more applicable to real datasets.

Chapter 4: Conclusions and future research

For this study I implemented new features into an existing sequence simulation tool. These features enable the generation of large scale sequence datasets to evaluate, measure and improve computational methods related to recombination identification and sequence alignment.

We used these sequence datasets to generate thousands of alignments containing sequences that had evolved under recombination. These alignments, together with a history of the recombination events in each sequence's evolutionary history, served as a training dataset. This training dataset was used to train machine learning models to identify the recombinant sequence from triplets of sequences displaying evidence of recombination. When comparing our recombinant identification models to the method currently implemented in RDP, we found significant increases in accuracy. All of the machine learning models exhibited similar outcomes, with the simplest model performing on par with the more complex neural network. This suggests that the constraint on model performance is not model complexity or fit quality, but rather the input data.

These findings are encouraging, and have the potential to immediately improve the recombinant identification accuracy of RDP, but there is scope for even further improvement. Since the model performance seems to be constrained by the data, better training data and cross-validation with RDP should lead to immediate improvements. This could be done by improving the simulation setup, with more complex and well-studied parameter choices. Another approach to improve model performance would be to add additional features, not limiting the models to the metrics currently used by RDP. Examples of such features may include: directly encoded nucleotide sequence data for the sequence triplets (rather than metrics derived from this data which form the bulk of the currently used features), directly encoded phylogenetic trees in a machine learning-friendly format, partitioned sequence regions (e.g., dividing sequences by recombination event breakpoints), or secondary structure information derived from the sequence data. The addition of more complex features such as these will allow more complex machine learning models to more fully leverage their capabilities.

Another sequence dataset was created with a different focus: indels and global alignment. This dataset was relatively simple, consisting of 100 alignments without recombination and 100 datasets with recombination. The goal being to have two sets of alignments with nearly identical properties, except for the presence of recombination. We used these to compare the performance of various alignment programs by comparing the alignment these programs produced to reference alignments calculated

by the simulator. When measuring overall alignment quality for both the recombinant and non-recombinant dataset, our results closely mirrored earlier alignment benchmarks.

Given the complexity level of our simulation and the recombination frequency used, we found only a minor impact of recombination on alignment quality. A limitation of this study that proved difficult to overcome was ensuring that the recombinant and non recombinant dataset alignments contained similar numbers of indels. Although I was unable to conclusively demonstrate that recombination impacts the accuracy of multiple sequence alignment, it remains plausible that this could be demonstrable with more complex simulations involving higher recombination rates and the tuning of selection parameters so that recombinant and non-recombinant datasets do not include significantly different numbers of indels. Other possible limitations of this study included: short seed sequences,and low generation counts during simulation: both of which could be explored in greater detail.

Appendix

Chapter 2: Improving the recombinant identification accuracy of recombination detection program

Simulation data: XML configuration files for different simulation setups and simulated sequence datasets - https://drive.google.com/file/d/1QweuZZXj3BGEj817jEe4UGc-6YTuJLy7/view?usp=sharing

Simulation software - https://github.com/phillipswanepoel/santa-sim/tree/Recomb_and_align

Chapter 3: Evaluating the performance of alignment software on recombinant and non-recombinant sequences

Alignment tool comparison: tables with alignment similarity scores as well as script to generate figures - https://drive.google.com/file/d/1j-RGlaD-1Or8DuPz8EDK5agSIKpbRDuV/view?usp=sharing

Fasta files generated by different alignment tools - https://drive.google.com/drive/folders/1LbSWh40zyK7EM7Wff5cq7tayVO_MP5J-

REFERENCES

Arenas, M. (2013). Computer Programs and Methodologies for the Simulation of DNA Sequence Data with Recombination. *Frontiers in Genetics*, *4*. https://www.frontiersin.org/articles/10.3389/fgene.2013.00009

Arenas, M., & Posada, D. (2010). The Effect of Recombination on the Reconstruction of Ancestral Sequences. *Genetics*, *184*(4), 1133–1139. https://doi.org/10.1534/genetics.109.113423

Arnold, K., Gosling, J., & Holmes, D. (2005). The Java programming language. Addison Wesley Professional.

Baltimore, D. (1971). Expression of animal virus genomes. *Bacteriological Reviews*, *35*(3), 235–241.

Baron, S. (Ed.). (1996). *Medical Microbiology* (4th ed.). University of Texas Medical Branch at Galveston. http://www.ncbi.nlm.nih.gov/books/NBK7627/

Barrell, B. G., Air, G. M., & Hutchison, C. A. (1976). Overlapping genes in bacteriophage phiX174. *Nature*, *264*(5581), 34–41. https://doi.org/10.1038/264034a0

Chaitanya, K. V. (2019). Structure and Organization of Virus Genomes. *Genome and Genomics*, 1–30. https://doi.org/10.1007/978-981-15-0702-1_1

Chatzou, M., Magis, C., Chang, J.-M., Kemena, C., Bussotti, G., Erb, I., & Notredame, C. (2016). Multiple sequence alignment modeling: Methods and applications. *Briefings in Bioinformatics*, *17*(6), 1009–1023. https://doi.org/10.1093/bib/bbv099

Combelas, N., Holmblat, B., Joffret, M.-L., Colbère-Garapin, F., & Delpeyroux, F. (2011). Recombination between poliovirus and coxsackie A viruses of species C: A model of viral genetic plasticity and emergence. *Viruses*, *3*(8), 1460–1484. https://doi.org/10.3390/v3081460

Cortes, C., & Vapnik, V. (1995). Support-vector networks. *Machine Learning*, *20*(3), 273–297. https://doi.org/10.1007/BF00994018

Deorowicz, S., Debudaj-Grabysz, A., & Gudyś, A. (2016). FAMSA: Fast and accurate multiple sequence alignment of huge protein families. *Scientific Reports*, *6*(1), Article 1. https://doi.org/10.1038/srep33964

Do, C. B., Mahabhashyam, M. S. P., Brudno, M., & Batzoglou, S. (2005). ProbCons: Probabilistic consistency-based multiple sequence alignment. *Genome Research*, *15*(2), 330–340. https://doi.org/10.1101/gr.2821705

Fleischmann, W. R. (1996). Viral Genetics. In S. Baron (Ed.), *Medical Microbiology* (4th ed.). University of Texas Medical Branch at Galveston. http://www.ncbi.nlm.nih.gov/books/NBK8439/

Friedman, J. H. (2001). Greedy function approximation: A gradient boosting machine. *The Annals of Statistics*, *29*(5), 1189–1232. https://doi.org/10.1214/aos/1013203451

Gibbs, M. J., Armstrong, J. S., & Gibbs, A. J. (2000). Sister-Scanning: A Monte Carlo procedure for assessing signals in recombinant sequences. *Bioinformatics*, *16*(7), 573–582. https://doi.org/10.1093/bioinformatics/16.7.573

Gotoh, O. (1999). Multiple sequence alignment: Algorithms and applications. *Advances in Biophysics*, *36*, 159–206. https://doi.org/10.1016/s0065-227x(99)80007-0

Hein, J. (1990). Reconstructing evolution of sequences subject to recombination using parsimony. *Mathematical Biosciences*, *98*(2), 185–200. https://doi.org/10.1016/0025-5564(90)90123-g

Ho, T. K. (1995). Random decision forests. *Proceedings of 3rd International Conference on Document Analysis and Recognition*, *1*, 278–282 vol.1. https://doi.org/10.1109/ICDAR.1995.598994

Hogeweg, P., & Hesper, B. (1984). The alignment of sets of sequences and the construction of phyletic trees: An integrated method. *Journal of Molecular Evolution*, *20*(2), 175–186. https://doi.org/10.1007/BF02257378

Hudson, R. R., & Kaplan, N. L. (1988). The coalescent process in models with selection and recombination. *Genetics*, *120*(3), 831–840. https://doi.org/10.1093/genetics/120.3.831

Iantorno, S., Gori, K., Goldman, N., Gil, M., & Dessimoz, C. (2014). Who Watches the Watchmen? An Appraisal of Benchmarks for Multiple Sequence Alignment. In D. J. Russell (Ed.), *Multiple Sequence Alignment Methods* (pp. 59–73). Humana Press. https://doi.org/10.1007/978-1-62703-646-7_4

Jakobsen, I. B., Wilson, S. R., & Easteal, S. (1997). The partition matrix: Exploring variable phylogenetic signals along nucleotide sequence alignments. *Molecular Biology and Evolution*, *14*(5), 474–484. https://doi.org/10.1093/oxfordjournals.molbev.a025784

Jariani, A., Warth, C., Deforche, K., Libin, P., Drummond, A. J., Rambaut, A., Matsen IV, F. A., & Theys, K. (2019). SANTA-SIM: Simulating viral sequence evolution dynamics under selection and recombination. *Virus Evolution*, *5*(1), vez003. https://doi.org/10.1093/ve/vez003

Just, W. (2001). Computational complexity of multiple sequence alignment with SP-score. *Journal of Computational Biology: A Journal of Computational Molecular Cell Biology*, *8*(6), 615–623. https://doi.org/10.1089/106652701753307511

Katoh, K., Misawa, K., Kuma, K., & Miyata, T. (2002). MAFFT: A novel method for rapid multiple sequence alignment based on fast Fourier transform. *Nucleic Acids Research*, *30*(14), 3059–3066. https://doi.org/10.1093/nar/gkf436

Kingman, J. F. C. (1982). The coalescent. *Stochastic Processes and Their Applications*, *13*(3), 235–248. https://doi.org/10.1016/0304-4149(82)90011-4

Koonin, E. V., Senkevich, T. G., & Dolja, V. V. (2006). The ancient Virus World and evolution of cells. *Biology Direct*, *1*, 29. https://doi.org/10.1186/1745-6150-1-29

Lassmann, T., & Sonnhammer, E. L. (2005). Kalign – an accurate and fast multiple sequence alignment algorithm. *BMC Bioinformatics*, *6*(1), 298. https://doi.org/10.1186/1471-2105-6-298

Lee, C., Grasso, C., & Sharlow, M. F. (2002). Multiple sequence alignment using partial order graphs. *Bioinformatics*, *18*(3), 452–464. https://doi.org/10.1093/bioinformatics/18.3.452

Lefeuvre, P., & Moriones, E. (2015). Recombination as a motor of host switches and virus emergence: Geminiviruses as case studies. *Current Opinion in Virology*, *10*, 14–19. https://doi.org/10.1016/j.coviro.2014.12.005

Liu, H., Zou, Q., & Xu, Y. (2022). A novel fast multiple nucleotide sequence alignment method based on FM-index. *Briefings in Bioinformatics*, *23*(1), bbab519. https://doi.org/10.1093/bib/bbab519

Luo, G. X., & Taylor, J. (1990). Template switching by reverse transcriptase during DNA synthesis. *Journal of Virology*, *64*(9), 4321–4328.

Marshall, N., Priyamvada, L., Ende, Z., Steel, J., & Lowen, A. C. (2013). Influenza virus reassortment occurs with high frequency in the absence of segment mismatch. *PLoS Pathogens*, *9*(6), e1003421. https://doi.org/10.1371/journal.ppat.1003421

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Rafal Jozefowicz, Yangqing Jia, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Mike Schuster, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker,

Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas,
Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke,
Yuan Yu, and Xiaoqiang Zheng.
TensorFlow: Large-scale machine learning on heterogeneous systems,
2015. Software available from tensorflow.org.

Martin, D. P., Varsani, A., Roumagnac, P., Botha, G., Maslamoney, S., Schwab, T., Kelz, Z., Kumar, V., & Murrell, B. (2021). RDP5: A computer program for analyzing recombination in, and removing signals of recombination from, nucleotide sequence datasets. *Virus Evolution*, *7*(1), veaa087. https://doi.org/10.1093/ve/veaa087

Martin, D., & Rybicki, E. (2000). RDP: Detection of recombination amongst aligned sequences. *Bioinformatics (Oxford, England)*, *16*(6), 562–563. https://doi.org/10.1093/bioinformatics/16.6.562

McCulloch, W. S., & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, *5*(4), 115–133. https://doi.org/10.1007/BF02478259

Morgenstern, B., Frech, K., Dress, A., & Werner, T. (1998). DIALIGN: Finding local similarities by multiple sequence alignment. *Bioinformatics (Oxford, England)*, *14*(3), 290–294. https://doi.org/10.1093/bioinformatics/14.3.290

Muller, H. J. (1964). The relation of recombination to mutational advance. *Mutation Research/Fundamental and Molecular Mechanisms of Mutagenesis*, *1*(1), 2–9. https://doi.org/10.1016/0027-5107(64)90047-8

Murtagh, F. (1984). Complexities of hierarchic clustering algorithms: state of the art. *Computational Statistics Quarterly, 1(2)*, 101-113.

Novák, Á., Miklós, I., Lyngsø, R., & Hein, J. (2008). StatAlign: An extendable software package for joint Bayesian estimation of alignments and evolutionary trees. *Bioinformatics*, *24*(20), 2403–2404. https://doi.org/10.1093/bioinformatics/btn457

Nuin, P. A., Wang, Z., & Tillier, E. R. (2006). The accuracy of several multiple sequence alignment programs for proteins. *BMC Bioinformatics*, *7*(1), 471. https://doi.org/10.1186/1471-2105-7-471

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, É. (2011). Scikit-learn: Machine Learning in Python. *The Journal of Machine Learning Research*, *12*(null), 2825–2830.

Pérez-Losada, M., Arenas, M., Galán, J. C., Palero, F., & González-Candelas, F. (2015). Recombination in viruses: Mechanisms, methods of study, and evolutionary consequences. *Infection, Genetics and Evolution*, *30*, 296–307. https://doi.org/10.1016/j.meegid.2014.12.022

Posada, D., & Crandall, K. A. (2001). Evaluation of methods for detecting recombination from DNA sequences: Computer simulations. *Proceedings of the National Academy of Sciences of the United States of America*, *98*(24), 13757–13762. https://doi.org/10.1073/pnas.241370698

R Core Team (2022). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. URL https://www.R-project.org/.

Rawson, J. M. O., Nikolaitchik, O. A., Keele, B. F., Pathak, V. K., & Hu, W.-S. (2018). Recombination is required for efficient HIV-1 replication and the maintenance of viral genome integrity. *Nucleic Acids Research*, *46*(20), 10535–10545. https://doi.org/10.1093/nar/gky910

Roshan, U., & Livesay, D. R. (2006). Probalign: Multiple sequence alignment using partition function posterior probabilities. *Bioinformatics*, *22*(22), 2715–2721. https://doi.org/10.1093/bioinformatics/btl472

Saitou, N., & Nei, M. (1987). The neighbor-joining method: A new method for reconstructing phylogenetic trees. *Molecular Biology and Evolution*, *4*(4), 406–425. https://doi.org/10.1093/oxfordjournals.molbev.a040454

Salminen, M. O., Carr, J. K., Burke, D. S., & McCutchan, F. E. (1995). Identification of breakpoints in intergenotypic recombinants of HIV type 1 by bootscanning. *AIDS Research and Human Retroviruses*, *11*(11), 1423–1425. https://doi.org/10.1089/aid.1995.11.1423

Sanjuán, R., & Domingo-Calap, P. (2021). Genetic Diversity and Evolution of Viral Populations. *Encyclopedia of Virology*, 53–61. https://doi.org/10.1016/B978-0-12-809633-8.20958-8

Sawyer, S. (1989). Statistical tests for detecting gene conversion. *Molecular Biology and Evolution*, *6*(5), 526–538. https://doi.org/10.1093/oxfordjournals.molbev.a040567

Scheffler, K., Martin, D. P., & Seoighe, C. (2006). Robust inference of positive selection from recombining coding sequences. *Bioinformatics*, *22*(20), 2493–2499. https://doi.org/10.1093/bioinformatics/btl427

Scherbakov, D. V., & Garber, M. B. (2000). Overlapping genes in bacterial and phage genomes. *Molecular Biology*, *34*(4), 485–495. https://doi.org/10.1007/BF02759558

Schierup, M. H., & Hein, J. (2000). Consequences of Recombination on Traditional Phylogenetic Analysis. *Genetics*, *156*(2), 879–891. https://doi.org/10.1093/genetics/156.2.879

Shafee, T., & Cooke, I. (2016). AlignStat: A web-tool and R package for statistical comparison of alternative multiple sequence alignments. *BMC Bioinformatics*, *17*(1), 434. https://doi.org/10.1186/s12859-016-1300-6

Sievers, F., & Higgins, D. G. (2014). Clustal Omega, accurate alignment of very large numbers of sequences. *Methods in Molecular Biology (Clifton, N.J.)*, *1079*, 105–116. https://doi.org/10.1007/978-1-62703-646-7_6

Sievers, F., & Higgins, D. G. (2020). QuanTest2: Benchmarking multiple sequence alignments using secondary structure prediction. *Bioinformatics*, *36*(1), 90–95. https://doi.org/10.1093/bioinformatics/btz552

Smith, J. M. (1992). Analyzing the mosaic structure of genes. *Journal of Molecular Evolution*, *34*(2), 126–129. https://doi.org/10.1007/BF00182389

Alexander E. Gorbalenya, Mart Krupovic, Arcady Mushegian, Andrew M. Kropinski, Stuart G. Siddell, Arvind Varsani, Michael J. Adams, Andrew J. Davison, Bas E. Dutilh, Balázs Harrach, Robert L. Harrison, Sandra Junglen, Andrew M. Q. King, Nick J. Knowles, Elliot J. Lefkowitz, Max L. Nibert, Luisa Rubino, Sead Sabanadzovic, Hélène Sanfaçon, Peter Simmonds, Peter J. Walker, F. Murilo Zerbini & Jens H. Kuhn. (2020). The new scope of virus taxonomy: Partitioning the virosphere into 15 hierarchical ranks. (2020). *Nature Microbiology*, *5*(5), 668–674. https://doi.org/10.1038/s41564-020-0709-x

Thompson, J. D., Linard, B., Lecompte, O., & Poch, O. (2011). A Comprehensive Benchmark Study of Multiple Sequence Alignment Methods: Current Challenges and Future Perspectives. *PLoS ONE*, *6*(3), e18093. https://doi.org/10.1371/journal.pone.0018093

Thompson, J. D., Plewniak, F., & Poch, O. (1999). BAliBASE: A benchmark alignment database for the evaluation of multiple alignment programs. *Bioinformatics (Oxford, England)*, *15*(1), 87–88. https://doi.org/10.1093/bioinformatics/15.1.87

Weiller, G. F. (1998). Phylogenetic profiles: A graphical method for detecting genetic recombinations in homologous sequences. *Molecular Biology and Evolution*, *15*(3), 326–335. https://doi.org/10.1093/oxfordjournals.molbev.a025929