UNIVERSITÀ DEGLI STUDI DI MILANO - BICOCCA

Dipartimento di Informatica, Sistemistica e Comunicazione

PhD Program in Computer Science Cycle XXXVII

COMPUTATIONAL METHODS IN EVOLUTION-AWARE PANGENOMICS FOR GRAPH AND SEQUENCE ANALYSES

Tutor: Prof. Federico Cabitza

Supervisor: Prof. Paola Bonizzoni

Co-Supervisor: Prof. Gianluca Della Vedova

Jorge Eduardo Avila Cartes

892428

Academic Year 2023-2024

To my parents.

Acknowledgments

I would like to express my heartfelt gratitude to my family, my mom, my dad, and my brothers Gustavo and Gonzalo, for all the sacrifices they have made throughout our lives, enabling me to reach this point. Life has had its challenges, but we have enjoyed every moment.

Special thanks to my older brother Gustavo, and my friend Gustavo (not the same person), who have been available to talk whenever I felt discouraged, and pushed me to finish this journey.

To Alessia, Andy, and Adrian, the first people I met during this journey, regardless of the distance that separate us, our friendship is one of the things I value the most.

I want to extend my gratitude to my labmates Simone, Luca, Mattia, Brian, and Davide. This past year has been especially enjoyable during our trips together, where I learned the true Italian way of doing things.

I very special thank to Cata for being my unwavering support; without you, none of this would have been possible. I hope we can continue writing stories on the old continent together.

Lastly, I want to extend my gratitude to my supervisors, Paola and Gianluca, for guiding me in becoming a better researcher. More importantly, I appreciate the quality of individuals you are; your support throughout this journey has exceeded all expectations.

TABLE OF CONTENTS

Chapter 1
Introduction

The notion of a microbial pangenome goes back to 2005 [1], and it is usually considered the first example of the use of the term *pangenome* to describe multiple genomes. Under this definition, the microbial pangenome comprises two fundamental components: the core genome, encompassing the genes that are ubiquitous across all strains within a given microbial population, and the dispensable genomes, which encompass genes that are absent in at least one strain within the population.

Nowadays sequencing organisms is essentially routine, as we have witnessed during the SARS-CoV-2 pandemic, when millions of viral genomes have been sequenced. Indeed, the introduction of Next-Generation Sequencing (NGS) technologies in 2006 made sequencing cheaper and more accessible. Later on, a new sub-area of research in computational biology was consolidated to address the intrinsic challenges introduced by the availability of several genomes, named computational pangenomics. In computational pangenomics, a *pangenome* is a collection of genomic sequences to be analyzed jointly or to be used as a reference [2].

Reference genomes (as a linear string) are widely used for genomic analyses. Typical pipelines consist of mapping a sample of reads against these references, looking for similarities, but more importantly differences, such as single nucleotide polymorphisms (SNPs) and structural variants (SVs). Linear reference genomes often fail to capture the full spectrum of genetic variation. For instance, the human reference genome (GRCh38) does not adequately represent the genetic diversity of African populations [3]. This limitation is even more pronounced in bacterial genomes, where pangenome approaches have proven significantly more effective [4]. On the other hands, pangenome graphs have demonstrated their ability to encompass more comprehensive genetic information accross diverse domains, such as crops [5], bovines [6], and humans [7]. These advancements have important implications for accurately identifying structural variations, particularly when contrasted with conventional linear reference genome assemblies.

With the vast amount of available data, selecting sequences for constructing a variation graph becomes challenging. Evidence suggests that tools developed to handle human genomes can scale to hundreds of sequences, although this demands substantial computational resources [8]. Even with this relatively small subset of sequences, the over representation of variants has been shown to complicate downstream analyses, particularly variant calling. This challenge has led to new approaches in the field, such as extracting subgraphs from the pangenome to make downstream analyses more effective [9]. Ideally one might carefully select a subset of these sequences that can express the vast amount of variability. This raises the question of how to query a large dataset to obtain the most informative sequences within the pangenome, enabling biologists to conduct meaningful analyses.

Pangenomes, either as a graph or as a collection of genomes, inherently capture more variability than a single reference genome. To make the transition from a reference genome as a string to a pangenome graph, it is important to have procedures for the construction of pangenome graphs that are suitable for the application of sequence-to-graph tools [10] [11], such tools should allow on pangenome graphs the same analyses that are routinely done on linear reference genomes. Furthermore, a pangenome as a collection of genomes imposes its own challenges. Ideally, one might be able to work with an entire kingdom at different taxonomy levels, where the ultimate goal is to perform efficient queries to a pangenome with sequences on the order of millions with minimal computational resources, such as on a personal computer. This requires the development and use of succinct data structures for storing and querying pangenomes.

1.1. Construction and comparison of pangenome graphs

In graph pangenomics, we find several graph representations, such as variation graphs, sequence graphs, and De Bruijn Graphs [12]. In this work, we pay special attention to the first one. A variation graphs aim to express a set of input sequences as paths in a graph, where sequences are broken into segments such that each sequence is a concatenation of these segments.

The construction of pangenome graphs has been addressed mainly from heuristics [4, 7, 13–15], and their comparison has relied upon downstream analyses instead of the graph itself [6, 7, 16]. The establishment of an optimal representation of genomics sequences as a graph has been discussed only to extend the good properties known for indexing strings to these graphs [17].

As the first contribution of this thesis, we present an approach to construct variation graphs starting from a multiple sequence alignment (MSA), leveraging the notion of maximal blocks, called `pangeblocks`. The MSA naturally highlights similarities and differences in a set of genomic sequences, and blocks capture a subset of sequences in an interval of columns sharing a substring in the MSA. `pangeblocks` is an Integer Linear Programming (ILP) approach that finds a tiling of the MSA using blocks. The construction is guided by several objective function criteria that aim to force the desired properties of the final graph, using the most natural criteria, like the number of nodes, the length of node labels, and others intended to ensure good properties of the graph for downstream analyses, like optimizing the number of seeds for sequence-to-graph tools. The nature of our approach restricts our analyses to relatively short genomic sequences, like genes, plasmids, or viral sequences like SARS-CoV-2, whose genomes are around 30 Kbp.

In the second contribution of this thesis, we explore the alignment between variation graphs under the realm of Riemannian Geometry, specifically by modeling pangenome graphs as discrete manifolds and proposing a mathematical model based on an ILP formulation for assessing isomorphism between manifolds by leveraging the Ricci-Flow algorithm on discrete manifolds [18, 19] that might help solve the isomorphism on variation graphs.

1.2. Exploring viral and bacterial pangenomes with Deep Learning

Pangenomics is becoming more relevant than ever with the explosive increment in genomic data, particularly in the study of viral and bacterial genomes. By the end of 2024, the number of SARS-CoV-2 genome sequences overcomes the 20 millions[1], and in the field of bacteria, the largest publicly available dataset was released, with almost 2 million assemblies, spanning more than 10000 species [20], and is in continuos expansion.

Since 1990, BLAST [21] has been the primary tool for querying large genomic datasets. However, with the explosion of genomic data especially for viruses and bacteria, its capacity in terms of computational resources and speed is now exceeded. To address these limitations, new methods have been developed for bacteria [22, 23], that use k-mer-based indexing to improve query efficiency. While the ultimate goal is to perform these analyses on personal computers [22], k-mer-based indexes still demand substantial memory. For example, two recent approaches have managed to index the 2MM bacterial dataset using 248 GB [20] and 4.26 TB [23], enabling read-level queries to expand BLAST's capabilities.

This large volume of data brings additional challenges. Addressing the quality of data is not a trivial task, and manual curation is still needed to build reliable databases of reference sequences, both in the quality of the assembly and the correct assignment of the taxonomy label. High-quality databases are crucial for taxonomy classification. GISAID has a large team of specialists to help curate SARS-CoV-2 genomes submitted. In the case of bacterial applications, several tools rely on manually curated databases for taxonomy classification purposes [24–28], while others are more flexible and allow the user to provide their dataset [29,

[1]https://gisaid.org/

30]. Ideally one would like to rely on all the data available, but manual curation becomes almost impossible at this scale.

Machine Learning tools have been proposed to speed up classification and avoid the dependency on index databases, thus decreasing computational resources needed to solve taxonomy level assignment [31–33]. Nonetheless, the need for reliable methods to alleviate manual curation of large datasets is imminent.

The second contribution of this thesis combines the best of indexes and deep learning approaches. We exploit an encoding called the Chaos Game Representation of DNA (CGR) [34], on top of which is a k-mer-based representation of a sequence, known as the Frequency Matrix of the CGR (FCGR) [35]. The FCGR is a matrix storing k-mer occurrences of a DNA sequence, and when visualized as an image it shows fractal patterns.

First, we explore well-known deep-learning architectures for image based classification of SARS-CoV-2 sequences. Secondly, we develop architectures for exploiting the FCGR, and propose an embedding-based index for the largest bacterial dataset, comprising around 1.8MM assemblies, allowing data curation, fast queries at the assembly level, and accurate taxonomic classification at species and genus levels. Most notably, the index for this dataset uses roughly 1GB of disk space, and queries require less than 5 GB of RAM, building bridges for analyses of high-volume genomic datasets with low computational resources.

1.3. Outline

This thesis is organized as follows. In Chapter 2 we introduce the basic concepts necessary to understand the content of the thesis. We introduce the notion of pangenome graphs, present the Chaos Game Representation of DNA, and the basics notions of Machine Learning and Deep Learning.

In Chapter 3 we present the first contribution of this thesis, a variation graph builder from MSA, namely `pangeblocks`, describing its algorithmic approach, and experimental evaluation. The contributions of this chapter have been recently published in BMC Bioinformatics [36].

In Chapter 4 we present an exploration to align pangenome graphs with tools from Riemannian geometry, we explain how a pangenome graph is modeled as a discrete manifold, and explore the isomorphism between variation graphs.

In Chapters 5 and 6 we present two contributions in the realm of deep learning. The first one, `CouGaR`, for the classification of SARS-CoV-2 sequences, and the second one, `PANSPACE`, for indexing, querying, and assessing the quality of bacterial assemblies. `CouGaR` has been published in Gigascience in 2022 [37], while the work on `PANSPACE` is ready for submission to a journal.

Finally, in Chapter 7 we outline potential directions for future research based on the findings of this work and provide a comprehensive list of publications and contributions developed during my PhD.

Chapter 2
Preliminaries

In this chapter, we introduce the essential concepts required to understand the core topics of this thesis. We begin by defining pangenome graphs, with special emphasis on Variation graphs, the main object of study in Chapters 3 and 4. Then, we dive into the Chaos Game Representation of DNA (CGR), central to our deep-learning-based approaches in Chapters 5 and 6. Finally, we introduce the foundational concepts and core techniques in machine learning and deep learning that are utilized throughout this thesis.

2.1. Pangenome graphs

Computational pangenomics highlights the transition from working with reference genomes as strings to pangenome graphs, where the natural representation of a pangenome is a directed graph [12]. In a pangenome graph nodes are labeled with strings that correspond to portions of the input sequences, and edges are added such that any input sequence can be spelled by a path in this graph, by concatenating the labels of its nodes.

Pangenome graphs are referred to as Variation graphs when paths are labeled, and as Sequence graphs when these paths are not explicit, or in other words when all paths are assumed to be valid. Hence, a Sequence graph can be seen as a particular case of a Variation graph.

The construction of pangenome graphs has been assessed by several approaches that deal with the input sequences in different ways, for example, starting from a Multiple Sequence Alignment (MSA) (`make_prg` [4], `vg` [13], `founderblockgraph` [17]), using progressive alignment (`minigraph` [14], `minigraph-cactus` [7]), all-vs-all pairwise alignment (`pggb` [15]), and exploiting the Variant Calling Format (VCF) (`vg` [13]). As we mentioned earlier, pangenome graphs are directed and, while most tools create acyclic graphs, some of them allow cycles, which in theory can lead to more compact graph representations. Currently, `make_prg` and `founderblockgraph` are the only tools that guarantee the construction of an acyclic graph.

Notice that some approaches, such as `minigraph` do not ensure a lossless representation of the input sequences since they do not care about small variations between sequences.

In what follows, we define the main object of study, the *Variation Graph* (for a tutorial on Computational Graph Pangenomics, we refer the reader to [12]). Sequences used to create these graphs are strings over the alphabet $\Sigma = \{A, C, G, T\}$, but in some cases, when null calls are allowed, denoted by the character N, we deal with the alphabet $\Sigma_N = \{A, C, G, T, N\}$, and the following definition is equally valid for both alphabets (and the RNA alphabet $\{A, C, G, U\}$ as well).

Definition 2.1 [Variation Graph] A Variation graph $G = \langle V, A, W \rangle$ is a directed graph whose vertices are labeled by nonempty strings, with $\lambda : V \mapsto \Sigma^+$ being the labeling function, and where A denotes the set of arcs and W denotes a nonempty set of distinguished walks.

A *Sequence graph* $G = \langle V, A \rangle$ is a graph obtained from a variation graph by ignoring the set W of distinguished walks, in other words, where all walks are allowed. Let G be a variation graph and let $w = \langle v_1, \dots, v_l \rangle$ be a walk in G. Then the *label* of the walk w is the concatenation $\lambda(w) = \lambda(v_1) \cdots \lambda(v_l)$ of the labels of the vertices of the walk. Given a string g, then G *expresses* g if there is a walk $w \in W$ such that the label of the walk w is exactly g, that is $\lambda(w) = g$.

An example of a variation graph built from the sequences $\bigcirc = AACCGA$, $\square = AAACGAT$, $\Diamond = GAACGAT$, $\triangle = CAACGAT$, and $\bigtriangledown = AATCCGGAA$ is shown in Figure 2.1, where each sequence is a path in this graph. The paths correspond to sequences of nodes following the figures on top of them. Each sequence can be recovered by concatenating the node labels in each path.

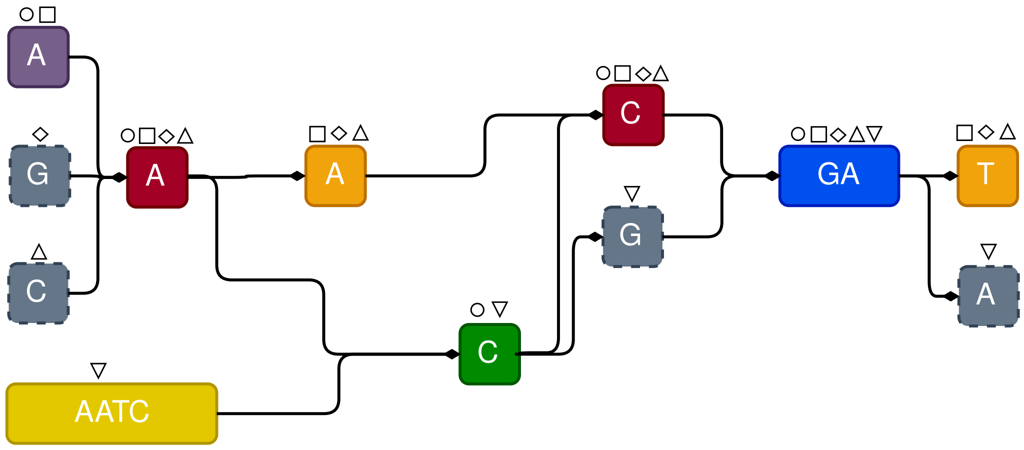


Figure 2.1: An example of a Variation graph $G = \langle V, A, W \rangle$ built from the sequences $\bigcirc = AACCGA$, $\square = AAACGAT$, $\Diamond = GAACGAT$, $\triangle = CAACGAT$, and $\bigtriangledown = AATCCGGAA$. The 12 colored rounded squares form the set of vertices V, the set of arcs A (13 in total) is represented by the black curved lines ending in a diamond shape between vertices, and the set of distinguished walks W is obtained by traversing the vertices with the 5 different shapes $\bigcirc, \square, \Diamond, \triangle$ and $\bigtriangledown$, spelling each sequence.

2.2. Chaos Game Representation of DNA

The Chaos Game Representation of DNA (CGR) was proposed in 1990 [34], and is a direct application of the Chaos Game, an iterative algorithm used to create fractals by randomly plotting points within a predefined geometric shape (*e.g.* a triangle or a square), following specific contraction rules to produce self-similar patterns.

In the CGR, these points are plotted within a square, where each vertex of the square is assigned a character in the DNA alphabet $\Sigma = \{A, C, G, T\}$, and instead of a random selection of points, these are guided by the characters in a DNA sequence. A new point is plotted in the square by choosing a vertex of the square and moving half of the distance (the contraction rule) from the current point in the direction of the chosen vertex.

The CGR encoding has been used in several bioinformatics applications, and extended to the aminoacid alphabet as well. For a more extensive list of application we refer the reader to [38].

In what follows, we define the CGR of sequence s over the alphabet Σ embedded in the square $[-1,1]^2 \subset \mathbb{R}^2$.

Definition 2.2 [Chaos Game Representation of DNA (CGR)] Given a sequence $s = s_1 \cdots s_n \in \Sigma^+$, the CGR encoding of s is the 2-D representation of the ordered pair (x_n, y_n) which is defined iteratively as follows:

$$(x_i, y_i) = \frac{1}{2}\Big((x_{i-1}, y_{i-1}) + g(s_i)\Big), \text{ if } i \geq 1 \tag{2.1}$$

where $(x_0, y_0) = (0,0)$ and,

$$g(s_i) = \begin{cases} (1,1) & s_i = A \\ (-1,1) & s_i = C \\ (-1,-1) & s_i = G \\ (1,-1) & s_i = T \end{cases} \tag{2.2}$$

The CGR encoding of a sequence is used to create visual representation by plotting all intermediate points in the iterative process.

An example of the CGR encoding for the sequence $ACGGTG$ is shown in Figure 2.2, where points are shown as white dots, and the encoded characters that guide the movements in each iteration are colored.

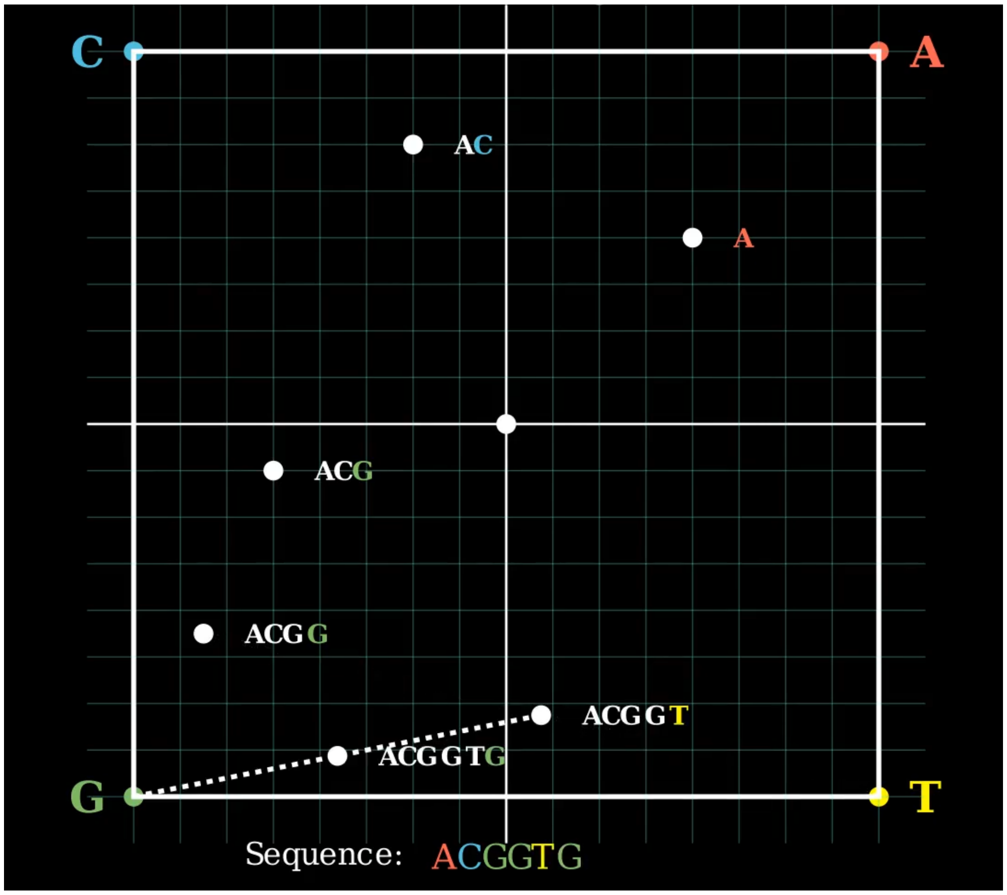


Figure 2.2: **Example of the CGR encoding** for the sequence *ACGGTG*. Each white circle correspond to the $\mathbb{R}^2$ coordinate encoding a prefix of the sequence. The last character of the prefix encoded in each iteration is colored, and correspond to the same character assigned to the corner of the sub-quadrant it is located. The dashed line exemplifies how the encoding moves half of the distance from the current coordinate in the direction of the vertex assigned to the next character in the sequence.

Notice that as per the definition above, the CGR encoding is defined over the DNA alphabet Σ, but in real settings we can have null calls (represented by the character N), *i.e.* we deal with sequences in the extended alphabet Σ_N, a situation that the CGR encoding cannot handle. However, to obtain a visual representation that might be obtained with the CGR in this scenario, there is an alternative, called the Frequency matrix of the CGR (FCGR), which encodes all strings of length k in the sequence, called k-mers, in a matrix that is the result of a subdivision of the square used by the CGR in 4^k small squares (with 4^k the total number of k-mers), and where the values assigned to each small square correspond to the number of ocurrences of the k-mer assigned to such square by the CGR encoding. As shown in [35, 39] the CGR visual representation of a sequence is well approximated by the FCGR, where k-mers containing an N are simply excluded, this offers a good approximation to the CGR visual representation, because the CGR encoding of a sequence is well approximated by the CGR encoding of its suffix (a k-mer).

Definition 2.3 [Frequency matrix of CGR (FCGR)] Let $s = s_1 \cdots s_n \in \Sigma_N^+$ be a sequence, and let k be an integer. Then the Frequency matrix of CGR, in short FCGR, of the sequence s is a $2^k \times 2^k$ 2-D matrix $F = (a_{i,j}), 1 \leq i, j \leq 2^k, i, j \in \mathbb{N}$, where for each k-mer $b \in \{A, C, G, T\}^k$, we have an element $a_{i,j}$ in the matrix F, that is equal to the number of occurrences of b as a substring of s. The position (i, j) of such element is computed as follows:

$$i = 2^k - \lceil 2^{k-1}(y+1) \rceil + 1 \tag{2.3}$$

$$j = \lceil 2^{k-1}(x+1) \rceil \tag{2.4}$$

where (x, y) is the CGR encoding for the k-mer b.

For example, following the definition above, for the 3-mer ACG, its CGR encoding is given by $(-5/8, -1/8)$, then the position in the FCGR matrix corresponds to the element $a_{i,j}$, where

$$\begin{aligned} i &= 2^k - \lceil 2^{k-1}(y+1) \rceil + 1 \\ &= 8 - \lceil 4(-1/8+1) \rceil + 1 \\ &= 8 - \lceil 7/2 \rceil + 1 \\ &= 8 - 4 + 1 \\ &= 5 \end{aligned}$$

and

$$
\begin{aligned}
j &= \lceil 2^{k-1}(x+1) \rceil \\
j &= \lceil 4(-5/8+1) \rceil \\
j &= \lceil 3/2 \rceil \\
j &= 2
\end{aligned}
$$

As we can see in Figure 2.3, the position of ACG in the matrix is the element $a_{(5,2)}$.

It is immediate to obtain a 2-D image from an FCGR matrix; it suffices to rescale all its values to the interval $[0, 255]$ (8 bits). Examples of these images for five different species are shown in Figure 2.4, and the ordering that k-mers adopt in the representation is shown in Figure 2.3, for $k \in \{1, 2, 3\}$.

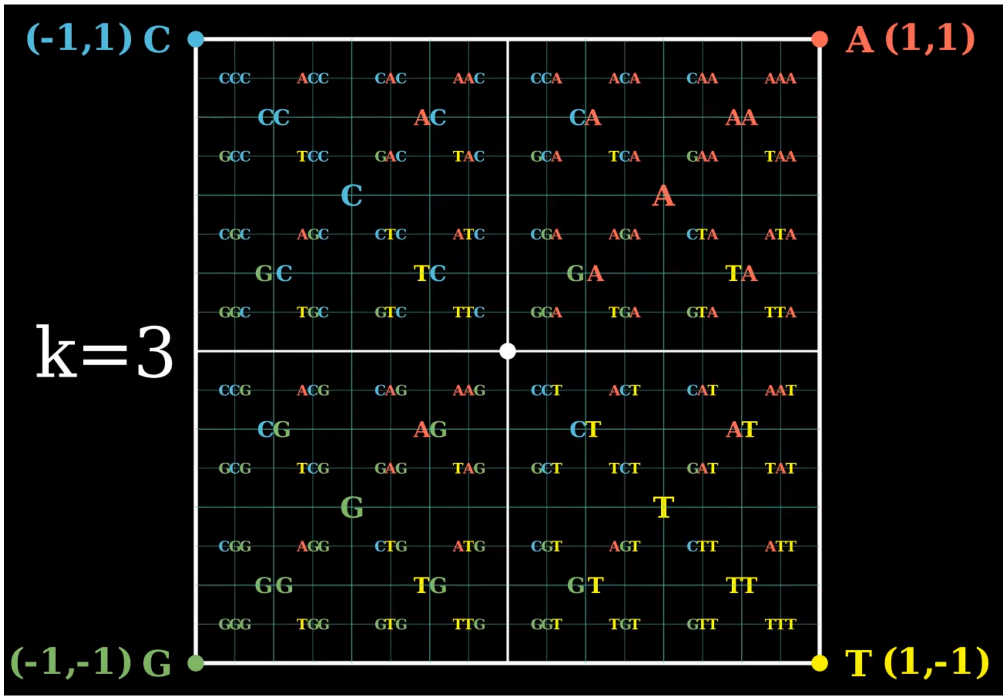

(-1,1) C
A (1,1)
(-1,-1) G
T (1,-1)
k=3
CCC ACC CAC AAC CCA ACA CAA AAA
CC AC CA AA
GCC TCC GAC TAC GCA TCA GAA TAA
C A
CGC AGC CTC ATC CGA AGA CTA ATA
GC TC GA TA
GGC TGC GTC TTC GGA TGA GTA TTA
CCG ACG CAG AAG CCT ACT CAT AAT
CG AG CT AT
GCG TCG GAG TAG GCT TCT GAT TAT
G T
CGG AGG CTG ATG CGT AGT CTT ATT
GG TG GT TT
GGG TGG GTG TTG GGT TGT GTT TTT

Figure 2.3: **Position of each k-mer** (for $k \in \{1, 2, 3\}$) in the square $[-1, 1]^2 \subset \mathbb{R}^2$ given by the CGR encoding. Each nucleotide is assigned to a vertex of the square to guide the encoding of k-mers. A subdivision of one level of a sub-square (*i.e.* from k to $k + 1$) corresponds to adding the letters A, C, G, T as prefixes to the k-mer encoded in the center of such sub-square, following the orientation of the vertices.

Notice that the set of k-mers that share a suffix of length l are encoded by the CGR encoding inside a sub-square of size $2^{k-l} \times 2^{k-l}$ in the FCGR matrix.

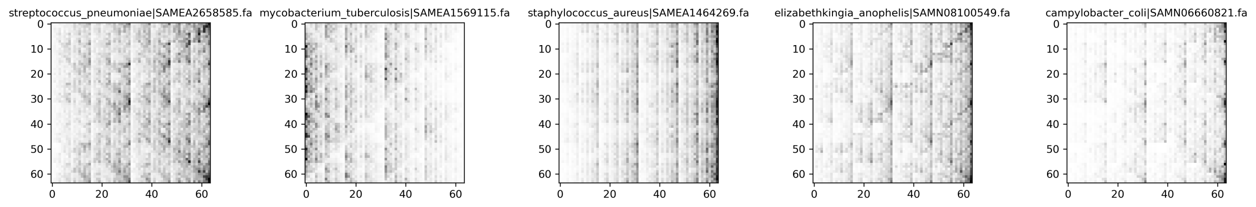


Figure 2.4: **FCGR of bacterial isolates** Examples of five bacterial assemblies, species names, and sample ID identifiers are in the title of each image. Each image corresponds to the visualization of the FCGR in 8-bits (k-mer frequencies rescaled to the interval $[0, 255]$) computed from 6-mers, and it has dimensions $2^6 \times 2^6$, one pixel or position for each 6-mer. White color in a pixel means that the k-mer encoded has the minimum value of frequency found in the assembly. Black color represents k-mers with the maximum frequency in the assembly. Gray colors correspond to k-mer frequencies lying between the minimum and maximum values.

2.3. Machine Learning and Deep Learning

2.3.1. Supervised and Unsupervised Learning

Supervised and unsupervised learning are two fundamental paradigms in machine learning, each serving different purposes in data analysis and model building [40].

Supervised Learning is a framework where models are trained on a labeled dataset, meaning that each input has a corresponding output label. The primary goal of supervised learning is to learn a mapping function that predicts output labels accurately for new, unseen data. This approach is widely used in applications like classification and regression, where labeled data is available, allowing models to learn from direct feedback.

In contrast, Unsupervised Learning involves training models on an unlabeled dataset, where no explicit output labels are provided. Here, the objective is to identify patterns, groupings, or underlying structures within the data. Techniques like clustering, dimensionality reduction, and density estimation are common in unsupervised learning. These methods are often used in exploratory data analysis, where the task is to uncover hidden relationships within the data without prior labels.

2.3.2. Metric Learning

Metric Learning is a type of machine learning that focuses on learning a distance function tailored to specific tasks [41]. Rather than using standard distance measures like Euclidean or cosine distances, metric learning aims to learn a custom metric that optimally separates classes or clusters data points according to specific criteria. This approach is particularly useful in scenarios where traditional distance measures are insufficient for capturing the nuanced relationships between data points.

In metric learning, a model is trained to map data into a transformed space, where similar data points are closer together and dissimilar points are farther apart. One common application is face recognition, where a learned metric can ensure that different images of the same person are close in the embedding space, while images of different people are far apart [42]. Metric learning can be supervised (with labels indicating similarity) or unsupervised (based on structural properties within the data), and methods such as triplet loss [42] and contrastive loss [43] are commonly used to train these models effectively.

2.3.3. Classification metrics

Class-specific metrics

In evaluating the performance of our model, we consider three key metrics: precision, recall, and the F1-score. Precision measures the proportion of correctly predicted positive instances among all instances predicted as positive, while recall quantifies the ability of the model to correctly identify all positive instances from the dataset. The F1-score provides a harmonic mean of precision and recall, offering a balanced metric when both false positives and false negatives are of concern. These metrics are critical in understanding the trade-offs between precision and recall, especially in imbalanced datasets.

1. **Precision** $= \dfrac{TP}{TP + FP}$
2. **Recall** $= \dfrac{TP}{TP + FN}$
3. **F1-score** $= 2 \cdot \dfrac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$

where True Positives (TP) represent instances where the model correctly identifies positive cases. False Positives (FP) occur when the model incorrectly classifies negative instances as positive, and False Negatives (FN) arise when the model fails to identify positive instances.

In multi-class classification, we can report an aggregated value for these metrics with the *macro average* and the *weighted average*. The macro average computes the metric independently for each class and then takes the unweighted mean, treating all classes equally regardless of their frequency. In contrast, the weighted average accounts for the class imbalance by weighting each class's contribution to the overall metric according to its frequency in the dataset. The macro average is useful for evaluating performance across all classes, while the weighted average gives a more balanced view in the presence of imbalanced class distributions.

Global metrics

Given a classification problem on S samples and N classes, the corresponding confusion matrix $C = (c_{ij}), i, j \in [1, N]$ is a square matrix where each entry $c_{i,j}$ is the number of elements that belong to the true class i and were classified in the class j, and the sum of the entries in C is exactly S. For global metrics, we define accuracy and the Matthews Correlation Coefficient (MCC).

1. The **Accuracy** of the model is defined as the proportion of the corrected classified samples over the total number of samples, this value ranges between 0 and 1, where 0 means that all samples were erroneously classified, while a value of 1 means a perfect classification. It can be defined in terms of the entries of the confusion matrix as follows:
$$acc = \frac{\displaystyle\sum_{k=1}^{N} c_{kk}}{S} \tag{2.5}$$

2. The **Matthews Correlation Coefficient** (MCC), proposed in [44] as a binary classification metric, was generalized to the multi-class case in 2004 [45], and it can be defined in terms of the confusion matrix as follows (see [46] for details):
$$MCC = \frac{cp \times S - \displaystyle\sum_{k=1}^{N} p_k \times t_k}{\sqrt{(S^2 - \displaystyle\sum_{k=1}^{N} p_k^2) \times (S^2 - \displaystyle\sum_{k=1}^{N} t_k^2)}} \tag{2.6}$$
where $cp = \sum_{k=1}^{N} c_{kk}$ is the total number of samples correctly predicted, $t_k = \sum_{i=1}^{N} c_{ik}$ is the number of times class k was truly occurred, and $p_k = \sum_{j=1}^{N} c_{kj}$ is the number of times class k was predicted. MCC lives in the range $[-1, 1]$, where 1 is perfect classification, -1 is the opposite, and 0 means that the confusion matrix is all zeros but for one single column, or when all entries are equal $c_{ij} = K \in \mathbb{N}$ [46].

2.3.4. Clustering metrics

In order to assess the quality of the class separability given by the CNN, we evaluate the embeddings of the last layer (the one used to perform the classification) in the network with three clustering evaluation measures. These embeddings are the output from the final layer of the network for each FCGR.

1. **Silhouette Coefficient** [47] Given an embedding v belonging to a cluster A, the silhouette coefficient $s(v)$ of v compares the mean intra-cluster distance in A (a) with the mean nearest-cluster distance for v (b), that is, the closest cluster to v different from A.

$$s(v) = \frac{a-b}{\max\{a,b\}} \tag{2.7}$$

where $a = \frac{1}{|A|}\sum_{w\in A, w\neq v}^{|A|} d(v,w)$ and $b = \min_{B\neq A}\frac{1}{|B|}\sum_{w\in B}^{|B|, v\in A} d(v,w)$.

The value of $s(v)$ ranges between -1 (wrongly assigned) and 1 (perfect separability). For a cluster A, the mean silhouette coefficient of A is computed as the average of $s(v)$ over all embeddings $v \in A$.

2. **Calinski-Harabasz Score** [48] Given a set of embeddings E of size n_E that has been clustered into k clusters, the Calinski-Harabasz Score s, also known as the Variance Ratio Criterion, is defined as the ratio of the between-clusters dispersion and the inter-cluster dispersion for all clusters (the dispersion of a group of n points is measured by the sum of the squared distances of the points from their centroid).

$$s = \frac{tr(B_k)}{tr(W_k)}\frac{n_E - k}{k-1} \tag{2.8}$$

where $tr(B_k)$ is the trace of the between-cluster dispersion matrix and $tr(W_k)$ is the trace (the sum of all elements in the diagonal of W_k) of the within-cluster dispersion matrix, defined as follow:

$$W_k = \sum_{q=1}^{k}\sum_{v\in C_q}(v-c_q)(v-c_q)^T \tag{2.9}$$

$$B_k = \sum_{q=1}^{k} n_q(c_q - c_E)(c_q - c_E)^T \tag{2.10}$$

where C_q is the set of embeddings in the cluster q, c_q is the centroid of the cluster q, c_E is the centroid of E and $n_q = |C_q|$. The higher the score s means that the clusters are dense and well separated.

3. **Generalized Discrimination Value (GDV)** [49] Given a set of N D–dimensional embeddings $\{x_1, \ldots, x_N\}$, with $x_n = (x_{n,1}, \ldots, x_{n,D})$ and a set of L classes $\{C_1, \ldots, C_L\}$, where each x_n is assigned to one of the L distinct classes. Consider their z–scored points $(s_1, \ldots, s_N)$, with $s_i = (s_{i,1}, \ldots, s_{i,D})$, where $s_{n,d} = \dfrac{1}{2}\frac{x_{n,d}-\mu_d}{\sigma_d}$. Here $\mu_d = \frac{1}{N}\sum_{n=1}^{N} x_{n,d}$ denotes the mean, and $\sigma_d = \sqrt{\frac{1}{N}\sum_{n=1}^{N}(x_{n,d}-\mu_d)^2}$ the standard deviation of dimension d. Using the re-scaled data points $s_n = (s_{n,1}, \ldots, s_{n,D})$, the Generalized Discrimination Value Δ is calculated from the mean intra–class and inter–class distances as follows:

$$\Delta = \frac{1}{\sqrt{D}}\left[\frac{1}{L}\sum_{l=1}^{L} d_{intra}(C_l) - \frac{2}{L(L-1)}\sum_{l=1}^{L-1}\sum_{m=l+1}^{L} d_{inter}(C_l, C_m)\right] \tag{2.11}$$

where the mean intra–class for each class C_l is defined as

$$d_{intra}(C_l) = \frac{2}{N_l(N_l-1)}\sum_{i=1}^{N_l-1}\sum_{j=i+1}^{N_l} d(s_i^{(l)}, s_j^{(l)}) \tag{2.12}$$

and the mean inter–class for each pair of classes C_l and C_m is defined as follows,

$$d_{inter}(C_l, C_m) = \frac{1}{\sqrt{D}}\Big[\frac{1}{N_l N_m}\sum_{i=1}^{N_l}\sum_{j=1}^{N_m} d(s_i^{(l)}, s_j^{(l)})\Big] \tag{2.13}$$

here N_k correspond to the number of points in class k, and $s_i^{(k)}$ is the ith point of class k. The quantity $d(a, b)$ is the distance between a and b, for our case, we considered the Euclidean distance. The value Δ range between -1 (perfect separability) and 0 (wrongly assigned),

2.3.5. Convolutional layers and Batch normalization

A convolutional layer is fundamental in Convolutional Neural Networks (CNNs) [50] and has transformed the way deep learning models process visual and spatial data. This layer applies a series of filters or kernels—small matrices that move (convolve) across the input data—to detect and learn spatial patterns. By capturing these local patterns in the input, such as edges, textures, or shapes, convolutional layers can form hierarchical representations where earlier layers detect low-level features, and deeper layers capture higher-level patterns (e.g., objects or specific textures).

Each convolutional layer has a set of hyperparameters, such as the filter size, stride, and padding—that impact the size and resolution of the output feature maps. In conjunction with other layers (like pooling and fully connected layers), convolutional layers enable CNNs to perform exceptionally well on image classification, object detection, and segmentation tasks.

Batch Normalization is a technique commonly used in conjunction with convolutional layers to improve the stability and efficiency of deep neural network training [51]. Introduced to address issues related to internal covariate shift, batch normalization normalizes the activations of each layer by adjusting and scaling the inputs for each mini-batch. This normalization maintains a consistent distribution of activations throughout the network, which helps stabilize the learning process and enables the use of higher learning rates. In convolutional neural networks (CNN), batch normalization is applied after the convolutional operation and before the activation function. This step has been shown to reduce training time and improve generalization, making it a standard component in many CNN architectures.

2.3.6. Cross validation

Cross-validation is a crucial technique in machine learning for model evaluation and validation, particularly useful in avoiding overfitting and ensuring that model performance is generalizable across unseen data [52]. The most commonly used method, k-fold cross-validation, involves dividing the dataset into k subsets or folds. In each iteration, one fold is used as the validation set while the remaining k-1 folds serve as the training set. This process repeats k times, ensuring that each data point is used for both training and validation. Repeated k-fold cross-validation is an extension of the standard k-fold cross-validation approach that increases the reliability of model performance estimates by further reducing variance due to data partitioning. In repeated k-fold cross-validation, the k-fold cross-validation procedure is performed multiple times with different random splits of the data into folds in each repetition. The average of the performance metrics across these folds provides a more robust estimate of the model's generalization performance, as opposed to relying on a single train-test split, which could be biased. Cross-validation is especially advantageous when working with limited datasets, as it maximizes data utilization for both training and evaluation.

2.3.7. Feature importance

Feature importance methods in machine learning and deep learning aim to quantify the contribution of input features to a model's predictions. These methods are crucial for interpreting complex, often opaque, neural networks, particularly in high-stakes domains like healthcare, finance, and autonomous systems. Traditional techniques, such as permutation importance [53], rely on evaluating the impact of feature perturbations on model performance. More recent approaches, including gradient-based methods like Integrated Gradients [54] and saliency maps [55], leverage the internal structure of deep learning models to provide more fine-grained attributions. Additional methods, such as Lime [56], Grad-

CAM [57], and DeepLIFT [58], have further enriched the toolkit for model interpretability. These tools bridge the gap between predictive accuracy and interpretability, enabling researchers and practitioners to trust and validate deep learning models.

Chapter 3

Customized construction of pangenome graphs via maximal blocks

The construction of a pangenome graph is a fundamental task in pangenomics. A natural theoretical question is how to formalize the computational problem of building an optimal pangenome graph, making explicit the underlying optimization criterion and the set of feasible solutions. Current approaches build a pangenome graph with some heuristics, without assuming some explicit optimization criteria. Thus it is unclear how a specific optimization criterion affects the graph topology and downstream analysis like read mapping and variant calling.

Our contribution

- By leveraging the notion of maximal block in a Multiple Sequence Alignment (MSA), we reframe the pangenome graph construction problem as an exact cover problem on blocks called *Minimum Weighted Block Cover* (MWBC). Then we propose an Integer Linear Programming (ILP) formulation for the MWBC problem that allows us to study the most natural objective functions for building a graph.

- We provide an implementation of the ILP approach for solving the MWBC and we evaluate it on SARS-CoV-2 complete genomes, showing how different objective functions lead to pangenome graphs that have different properties, hinting that the specific downstream task can drive the graph construction phase.

Our tool, `pangeblocks`, is open source and available at https://github.com/AlgoLab/pangeblocks.

3.1. Introduction

The natural representation of a pangenome is a directed graph [12], and computational pangenomics highlights the transition from working with reference genomes as strings to pangenome graphs.

The need for richer data structures to represent pangenomes is clear. However, the pursuit of establishing an unambiguous notion of *optimal* pangenome graph is still unsettled. For example, pangenome graphs can be built with different tools (`vg` [13], `minigraph` [14], `pggb` [15], `minigraph-cactus` [7], `make_prg` [4], `founderblockgraph` [17]), but the comparison of their results has always relied on specific downstream analyses [6, 7, 16], while we would like to have a quality measure that is not overly dependent on downstream analysis.

It is noteworthy that computational tools diverge in several critical attributes, including scalability, the ability to represent exactly the original genomes, the presence of cycles in the graph, and the proficiency in showcasing well-documented genetic variants. In particular, a critical aspect to consider in the construction of a pangenome graph is whether the graph is suitable for tasks such as mapping reads under the seed-and-extend paradigm. In fact, when most of the vertices have a label that is shorter than the seed length, obtaining a high quality alignment becomes harder [10, 13, 59, 60].

We propose a formal framework to build a variation graph [12]) that (1) is acyclic and (2) represents perfectly the input genomes, from a Multiple Sequence Alignment (MSA) of the input genomes. More precisely, we frame the pangenome construction problem as finding an optimal tiling of the MSA into blocks, where each block is the set of cells of the MSA defined by a set of rows and an interval of consecutive columns, where all rows of the block are labeled by the same string. Each block becomes a vertex of the pangenome graph and we add arcs to the graph to connect blocks that are adjacent in the MSA. This guarantees that each input sequence corresponds to a path of an acyclic graph. More precisely, we introduce an optimization problem, the Minimum Weight Block Cover problem, in short MWBC, which, given a set of blocks, finds an optimal subset of blocks that cover the MSA. The MWBC problem is a special case of a more general optimization problem, called the General Minimum Weight Block Cover problem (G-MWBC) that asks for a set of blocks that cover the MSA, where the set is optimum w.r.t. a weighted function defined over the solution.

Other approaches, based on Elastic Founder Graphs (EFG) [17, 61, 62] use the notion of block as a portion of the MSA resulting from a vertical segmentation of it. Under this definition, a block is essentially a subMSA instance, where several strings might belong to a block, while our definition of block is based on maximal blocks and relates to a unique string. The EFG approach aims to create indexable graphs for linear time pattern matching queries and propose several optimization criteria for the segmentation of the MSA for enhancing the efficiency and expressiveness of the graph for queries.

Currently, `make_prg` and `founderblockgraph` are the only tools that guarantee the construction of an acyclic graph. The former builds a sequence graph, which is a graph whose paths are not distinguished. On the other hand, the second one creates a variation graph with all sequences explicitly expressed as a path in the graph, but both express all input sequences in the final graph. On the other hand, `vg` can remove cycles from a variation graph, but only at a later step: its main heuristics are tailored for the general case.

The MWBC problem is similar to some problems that have been studied in the literature, such as the Weighted Rectangle Cover [63] or the Maximum Weighted Submatrix Coverage [64] problems. Unfortunately, none of those problems are on matrices containing characters and where the order of the rows is irrelevant, therefore MWBC does not inherit their hardness results. On the other hand, the similarities are sufficiently strong that we conjecture that MWBC is NP-hard.

We propose an ILP approach for solving MWBC, based on the classical ILP formulation of Exact Set Cover [65]. To avoid including all possible blocks of an MSA in the formulation, first we identify its maximal blocks, a notion introduced in the context of haplotyping, since they can be computed in linear time [66].

More precisely, we use an extension, called Wild-pBWT[1] [67], that works on an arbitrary alphabet, with this approach we can compute blocks directly over an MSA over the alphabet $\Sigma_N = \{A, C, G, T, N\}$ extended with the special symbol $-$ called *indel*. Hence characters N and $-$ are treated equally as $ACGT$ for computing blocks.

Moreover, we restrict our attention to blocks that are obtained by decomposing overlapping maximal blocks. We will describe two decomposition strategies; a slower strategy that produces more blocks, but is limited to smaller instances, and a faster strategy that can be applied to larger MSAs. The result of the decomposition phase is a set of blocks that is an instance of MWBC.

We propose and analyze five different objective functions for the ILP formulation, each tailored for a different goal. Those objective functions aim to highlight some measurable properties that the graph should exhibit, such as the length of the labels, the number of paths traversing the vertices, and the total size of the graph. These are the first elementary properties that can be measured or optimized when constructing the graph. On the other hand, an open question is how can we computationally encode in the graph construction some properties that are biological meaningfully, such as how to represent similarities or differences between the sequences or some biological events.

We assess experimentally the impact on the final pangenome graph of the different strategies used to produce the set of blocks that is an instance of the MWBC problem. Then we measure the impact of the objective functions on an MSA built from 50 complete SARS-

[1] github.com/AlgoLab/Wild-pBWT

CoV-2 genomes — we have obtained similar results on 20 and 100 complete SARS-CoV-2 genomes.

Our results show that a strategy that produces a larger set of blocks improves significantly the optimal values reached by each objective function at the expense of an increase in the computation time for solving the MWBC problem.

Most notably, with `pangeblocks` we were able to construct variation graphs where the number of potential seeds is much larger than the graphs computed by `pggb`— this is a property that helps tools based on seeding approaches like GraphAligner [10]. We also notice that graphs built with `vg` are closer (w.r.t. the properties measured) to the graphs built with `pangeblocks` when minimizing the length of the graph. On the other hand, compared to `pggb`, `pangeblocks` is able to produce graphs with a smaller number of nodes in general, and in particular has significantly fewer nodes that are used by only a smaller percentage of the input genome sequences. In conclusion, the experimental analysis shows that pangenome graph measures may significantly change with the use of objective functions in building an optimal graph under such functions.

3.2. Preliminaries

Since our algorithm starts from a *multiple sequence alignment* (MSA), we need to provide a formal definition of it, while referring to the reader to [68] for a more detailed exposition.

Definition 3.1 [Expansion of a sequence] Given a string $s = s_1 s_2 \cdots s_l$ over the alphabet Σ_N, an *expansion* t of s is obtained by possibly inserting $-$ symbols into s. A maximal substring of t containing only $-$ symbols is called a gap.

Definition 3.2 [Multiple Sequence Alignment (MSA)] Let $\mathcal{S} = \{s_1, \ldots, s_k\}$ be a set of k strings. Then an MSA of $\mathcal{S}$ is a set $\mathcal{T} = \{t_1, \ldots, t_k\}$ such that each t_i is an expansion of s_i, all t_i have the same length n (also called the *length* of the MSA) and for all j with $1 \leq j \leq n$, then there exists a t_i such that $t_i[j] \neq -$.

We will use the example of Figure 3.1 as a running example to introduce the technical aspects of this work.

A variation graph (see Definition 2.1) is a representation of an MSA [69], but such a representation is not unique. Indeed there can exist more than one variation graph expressing the sequences in an MSA. Conversely, from each variation graph we can obtain several MSAs, *e.g.* depending on where indels are inserted. However, it is not immediate to obtain those alignments, since a graph might contain cycles that must be broken to obtain the MSA.

In this work we explore a connection between MSAs and variation graphs based on the notion of *block*, where a block is a portion of the MSA correponding to the same string and is associated with a vertex of the variation graph.

In what follows, given a set $\mathcal{S} = \{s_1, \ldots, s_k\}$ of sequences and a subset $K \subset \{1, \ldots, k\}$ of indexes, $\mathcal{S}|_K$ is defined as the set $\{s_i : i \in K\}$. We can now introduce the definition of block.

	1	2	3	4	5	6	7	8	9
○	A	A	$-$	$-$	C	C	G	A	$-$
□	A	A	A	$-$	$-$	C	G	A	T
◇	G	A	A	$-$	$-$	C	G	A	T
△	C	A	A	$-$	$-$	C	G	A	T
▽	A	A	T	C	C	G	G	A	A

Figure 3.1: Example of MSA on the sequences ○ $= AACCGA$, □ $= AAACGAT$, ◇ $= GAACGAT$, △ $= CAACGAT$, and ▽ $= AATCCGGAA$.

Definition 3.3 [Block] Let $\mathcal{T} = \{t_1, \ldots, t_k\}$ be an MSA of length n. Then a *block* is a triple (K, b, e) with $K \subseteq \{1, \ldots, k\}$, $K \neq \emptyset$ and $1 \leq b \leq e \leq n$, such that $t_i[b : e] = t_j[b : e]$ for all $i, j \in K$.

The *label* of the block (K, b, e) is the string $t_l[b : e]$, for any $l \in K$.

Informally, a block is a collection K of rows of the MSA and an interval of columns between b and e such that all rows of K in that interval are the same sequence.

In this paper, each vertex of the variation graph corresponds to a block, and the label of the vertex is exactly the label of the block. Moreover, the vertices of the graph correspond to non-overlapping blocks of the MSA. While the number of blocks is exponential in the number of sequences, maximal blocks can be computed in linear time, where a block (K, b, e) is maximal if enlarging K, decreasing b, or increasing e does not result in a block.

Definition 3.4 [Maximal block] Let $\mathcal{T} = \{t_1, \ldots, t_k\}$ be an MSA of length n, and let (K, b, e) be a block of $\mathcal{T}$. Then (K, b, e) is a *maximal block* if:

1. for each $h \in \{1, \ldots, k\} \setminus K$, $t_h[b : e] \neq t_k[b : e]$ for any $k \in K$ (row-maximality),
2. $b = 1$ or $t_i[b - 1] \neq t_j[b - 1]$ for some $t_i, t_j \in \mathcal{T}|_K$ (left-maximality), and
3. $e = n$ or $t_i[e + 1] \neq t_j[e + 1]$ for some $t_i, t_j \in \mathcal{T}|_K$ (right-maximality).

Finally, we say that two blocks are overlapping if they share some entry of the MSA.

Definition 3.5 [Overlapping blocks] Two blocks (K_1, b_1, e_1) and (K_2, b_2, e_2) *overlap* if (1) $K_1 \cap K_2 \neq \emptyset$ and (2) there exists an integer i such that $b_1 \leq i \leq e_1$ and $b_2 \leq i \leq e_2$.

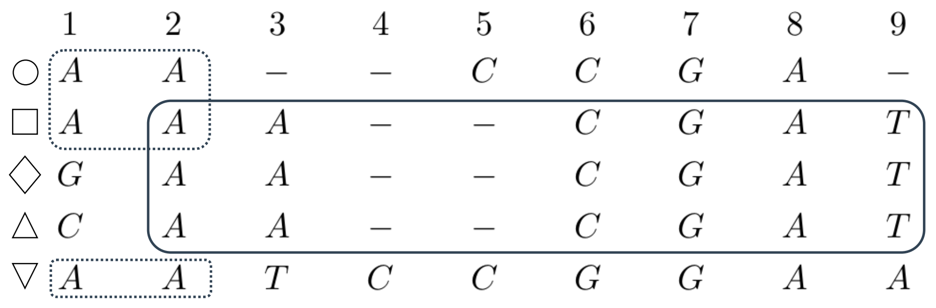


Figure 3.2: Two overlapping maximal blocks whose sets of rows are not nested. On the left, the (dashed line) block $(\{\bigcirc, \square, \bigtriangledown\}, 1, 2)$, on the right the (solid line) block $(\{\square, \Diamond, \bigtriangleup\}, 2, 9)$

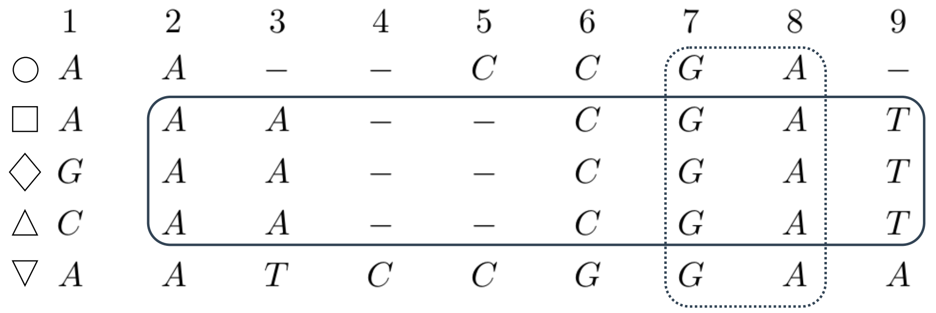


Figure 3.3: Two overlapping maximal blocks with nested sets of rows Starting from the left, we have the red block $(\{\square, \Diamond, \bigtriangleup\}, 2, 9)$ and the blue block $(\{\bigcirc, \square, \Diamond, \bigtriangleup, \bigtriangledown\}, 7, 8)$. The blue block is a *vertical block* because it contains all rows of the MSA.

3.2.1. The Minimum Weight Block Cover problem

We formalize the variation graph construction as computing a block cover of the MSA.

Definition 3.6 [Block Cover] Let $\mathcal{T} = \{t_1, \ldots, t_k\}$ be an MSA of length n over k sequences, and let $\mathcal{B}$ be a set of blocks of $\mathcal{T}$. Then $\mathcal{B}$ is a *cover* of the MSA $\mathcal{T}$ if, for each $1 \leq i \leq k$ and $1 \leq j \leq n$, there exists at least a block (K, b, e) such that $i \in K$ and $b \leq j \leq e$. Moreover, if all blocks in $\mathcal{B}$ are non-overlapping, then $\mathcal{B}$ is an *exact cover* of the MSA.

The natural computational problem that we formalize is the Minimum Weight Block Cover, where the instance is an MSA and we want to compute an exact cover of the MSA using blocks that minimize some objective function.

Notice that we are not specifying the objective function here, since different biological settings will result in different objecting functions. In fact, one of the advantages of our approach is that we can easily consider different objective functions, resulting in the following (meta)problem.

Problem 3.1 [General Minimum Weight Block Cover (G-MWBC)] The instance of the G-MWBC problem is an MSA $\mathcal{T}$. Then a feasible solution is an exact cover $\mathcal{C}$ of $\mathcal{T}$. We denote $f()$ as the objective function. Then $\mathcal{C}$ is an optimal solution if $\mathcal{C}$ minimizes the objective function.

Our strategy to solve G-MWBC will mainly consist of selecting a set of blocks $\mathcal{B}$ and looking for a subset of $\mathcal{B}$ that is an exact cover of the MSA. This is formalized with the following computational problem, where the set of possible blocks is given as part of the instance.

Problem 3.2 [Minimum Weight Block Cover given a set of blocks (MWBC)] The instance of the MWBC problem is an MSA $\mathcal{T}$ and a set $\mathcal{B}$ of blocks of the MSA. Then a feasible solution, if it exists, is an exact cover $\mathcal{C} \subseteq \mathcal{B}$ of $\mathcal{T}$. We denote $f()$ as the objective function. Then $\mathcal{C}$ is an optimal solution if $\mathcal{C}$ minimizes the objective function.

As already observed, we conjecture that G-MWBC and MWBC are NP-hard even though they are restricted versions of the (weighted) exact set cover. In fact, we can easily encode an instance of MWBC as an instance of minimum weight exact set cover, and G-MWBC is the special case of MWBC where $\mathcal{B}$ is the set of all possible blocks of the MSA.

3.2.2. Building a variation graph from a Block Cover

Once we have an exact cover $\mathcal{C}$ of the MSA (*e.g.* Figure 3.4), we compute a variation graph whose vertices are exactly the blocks of the cover, and two blocks (K_1, b_1, e_1) and

(K_2, b_2, e_2) are an arc of the graph iff $e_1 = b_2 - 1$, and $K_1 \cap K_2 \neq \emptyset$, that is those blocks correspond to consecutive columns of the MSA, and share at least one row. Observe that the variation graph we build is a directed acyclic graph.

The label of a vertex is the label of the corresponding block, as shown in Figure 3.5.

Observe that, by construction, each row of the MSA is the concatenation of the labels of blocks corresponding to a path of the graph. This means that the graph we have obtained is a lossless representation of the input sequences of the MSA.

Although the graph represents each input sequence without any loss of information, it might require some post-processing steps to obtain a simpler graph that is able to represent the same set of sequences. First, we remove all nodes whose label consists of indels only — when such a node is removed, an arc is added between its predecessors and successors, to preserve the property that the graph expresses all the input sequences. Then we remove indels from all labels.

Finally, we collapse each non-branching subpath into a single node. Notice that the initial graph construction requires time linear in the MSA size, while the final step consists of a

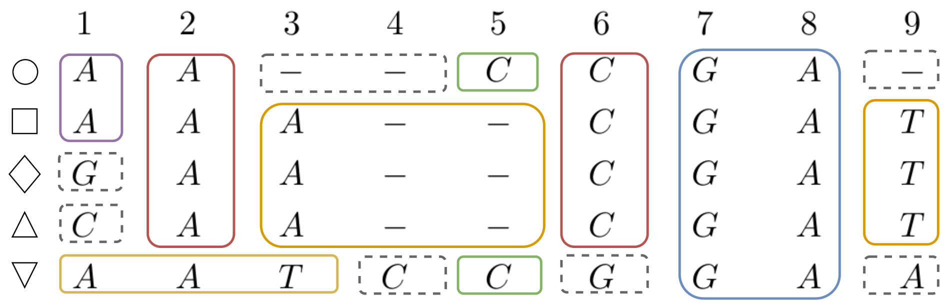


Figure 3.4: Exact cover of the MSA of Figure 3.1. Blocks with a solid line contain at least 2 rows. Blocks with dashed lines contain only 1 row — they are one-row or one-character blocks. The same colors will be used for the associated variation graph (see Figure 3.5 and Figure 3.6).

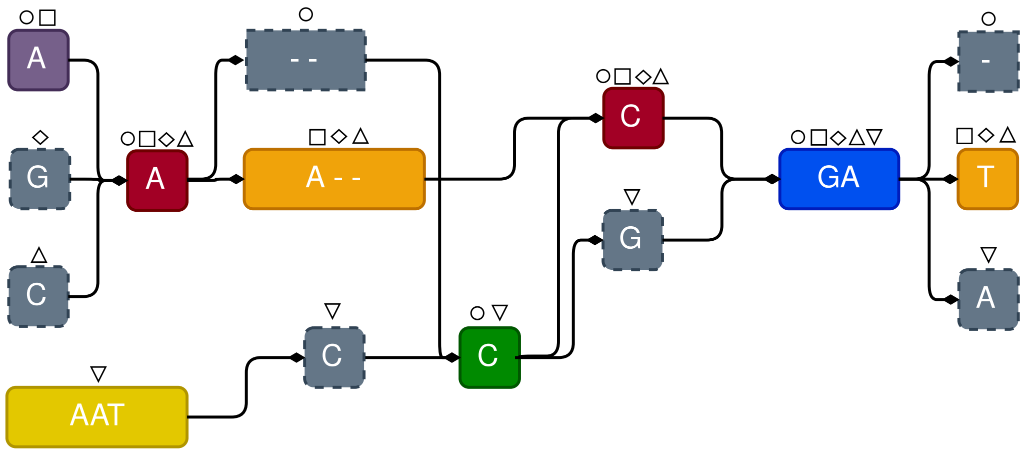


Figure 3.5: The Variation graph obtained from the Exact Cover of Figure 3.4, by defining a node for each block, and including arcs between consecutive blocks sharing at least one row in the MSA.

graph traversal that updates the graph. The creation of nodes and arcs from consecutive blocks is illustrated in Figure 3.4 and Figure 3.5.

3.3. Methods

3.3.1. Solving the Minimum Weight Block Cover problem

Since the number of blocks of an MSA $\mathcal{T}$ can be exponential in the MSA size, it is infeasible to use a direct ILP formulation of the MWBC problem to obtain an optimal solution. Therefore we must describe some approaches that reduce the running time, based on restricting the set of possible blocks, that is solving an instance of MWBC, and forcing some blocks to be part of the solution.

We first look at blocks spanning all rows, called *vertical* blocks (see Figure 3.11). We introduce an additional parameter α: all maximal vertical blocks spanning at least α columns will be a block in our Exact Cover. The final effect is that each long substring that is shared by all rows of the MSA results in a single vertex of the variation graph, and all paths traverse such vertex. This heuristic will greatly reduce the running time, since we can split the MSA

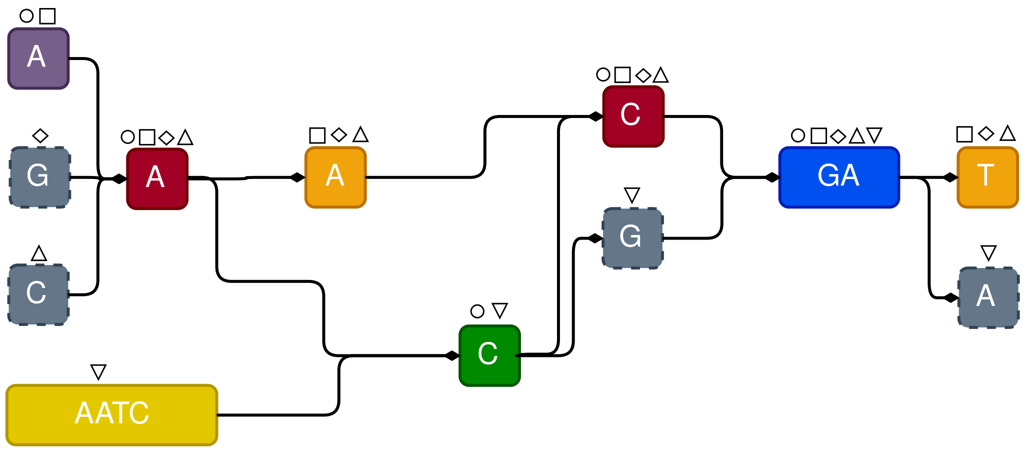


Figure 3.6: The result of postprocessing on the variation graph of Figure 3.5: indels are removed from node labels, and nodes labeled only by indels are removed. Non-branching paths are collapsed to a single node.

into smaller subMSAs, one for each portion delimited by two vertical blocks. For example, if a vertical block consists of columns 10–20 of the MSA and another vertical block consists of columns 57-81, and then we extract and solve the subMSA corresponding to the columns 21-56. Those subMSAs are then solved independently (if possible, in parallel) to obtain an exact cover of the initial ILP.

Even restricting to subMSA might result in a huge number of blocks, therefore we need to compute a set of blocks that is sufficiently small to be manageable but allows us to find an almost-optimal exact cover. To obtain such a set, we start from the set $\mathcal{B}$ of all maximal blocks. Then a procedure, we called *decomposition*, is applied to each pair of overlapping maximal blocks to create some smaller blocks that are added to $\mathcal{B}$ — we will detail in Section 3.3.3 the decomposition procedures.

Finally, we add some blocks to guarantee the existence of an exact cover in the set of blocks $\mathcal{B}$, we will detail this procedure in Section 3.3.4.

3.3.2. Computing maximal blocks

As observed before, to obtain the instance $\mathcal{B}$ for the MWBC problem, we first compute the set of *maximal* blocks, since their number is linear in the MSA size and they can be computed in linear time [66] via the Positional Burrows-Wheeler Transform (PBWT). In particular, in [66] the notion of maximal perfect haplotype blocks has been introduced in the framework of genome-wide selection. Maximal perfect haplotype blocks correspond to the notion of maximal block over a binary alphabet with the requirement that at the block has at least two rows. In our context, we used an implementation that extends the maximal haplotype block notion to the DNA alphabet extended with the indel symbol.

Still, an exact cover consisting only of maximal blocks might not exist, while we need to apply the ILP approach on a set $\mathcal{B}$ of blocks that contains a feasible solution of the MWBC problem.

3.3.3. Decomposition of maximal blocks into non-overlapping blocks

First, we will show that two overlapping maximal blocks can involve two nested intervals of columns if and only if the two sets of rows involved are one included in the other.

Lemma 3.1 *Let (K_1, b_1, e_1) and (K_2, b_2, e_2) be two overlapping maximal blocks. Then $b_2 \leq b_1 \leq e_1 \leq e_2$ if and only if $K_2 \subset K_1$.*

Proof: ($\Rightarrow$) Assume to the contrary that $\exists r \in K_2 \setminus K_1$. Since $b_2 \leq b_1 \leq e_1 \leq e_2$, this implies that $r[b_1 : e_1]$ is equal to the label of the block (K_1, b_1, e_1), hence contradicting the maximality of (K_1, b_1, e_1).

($\Leftarrow$) Assume now that $K_2 \subset K_1$. Since the two maximal blocks are overlapping, $b_2 \leq e_1 \leq e_2$ or $b_2 \leq b_1 \leq e_2$. Assume that $b_2 \leq e_1 \leq e_2$ (the other case is symmetrical), and assume to the contrary that $b_1 < b_2$. Since $K_2 \subset K_1$, also (K_2, b_1, e_2) is a block, contradicting the maximality of (K_2, b_2, e_2). $\square$

The coverage of an MSA can be split into two parts: those positions covered by maximal blocks and the remaining ones. In the latter case, there exists a unique option to cover the positions, that consists in creating one-row blocks. We can now focus our attention on finding the coverage of the regions covered by maximal blocks.

The first notion of decomposition called *row-maximal decomposition*, is stated in the following definition which builds on Lemma 3.1 (see Figure 3.7).

Definition 3.7 [Row-maximal decomposition of maximal blocks] Given two overlapping blocks $l_1 = (K_1, b_1, e_1)$ and $l_2 = (K_2, b_2, e_2)$ with $b_1 \leq b_2$, the result of their *row-maximal decomposition* consists of the following blocks:

$$(K_1, b_1, b_2 - 1) \tag{3.1}$$

$$(K_2, e_1 + 1, e_2) \text{ if } e_1 < e_2 \tag{3.2}$$

$$(K_1, e_2 + 1, e_1) \text{ if } e_2 < e_1. \tag{3.3}$$

Notice that more than one condition might be true in Definition 3.7. In that case, the result consists of all blocks whose conditions hold.

The case when $b_2 < 1$ is symmetrical to the one of Definition 3.7.

A second decomposition, called *complete decomposition*, of two overlapping maximal blocks is described next (see Figure 3.8). This second decomposition returns a superset of the blocks computed with a row-maximal decomposition, hence resulting in a larger set of blocks. Just as for Definition 3.7, we describe only the case when $b_1 \leq b_2$, since the other case is symmetrical.

Definition 3.8 [A complete decomposition of maximal blocks] Given two overlapping blocks $l_1 = (K_1, b_1, e_1)$ and $l_2 = (K_2, b_2, e_2)$ with $b_1 \leq b_2$, the result of their *complete decomposition* consists of the union of the result of their row-maximal decomposition, and of the following blocks:

$$(K_1 \cap K_2, b_1, e_1), (K_1 \cap K_2, b_2, e_2) \text{ if } K_1 \cap K_2 \neq \emptyset \tag{3.4}$$

$$(K_1 \setminus K_2, b_1, e_1), (K_1 \setminus K_2, b_2, e_2) \text{ if } K_1 \setminus K_2 \neq \emptyset \tag{3.5}$$

$$(K_2 \setminus K_1, b_1, e_1), (K_2 \setminus K_1, b_2, e_2) \text{ if } K_2 \setminus K_1 \neq \emptyset \tag{3.6}$$

$$(K_1 \cap K_2, b_2, e_1) \text{ if } K_1 \cap K_2 \neq \emptyset \text{ and } b_2 < e_1 \tag{3.7}$$

$$(K_1 \setminus K_2, b_2, e_1) \text{ if } K_1 \setminus K_2 \neq \emptyset \text{ and } b_2 < e_1 \tag{3.8}$$

$$(K_2 \setminus K_1, b_2, e_1) \text{ if } K_2 \setminus K_1 \neq \emptyset \text{ and } b_2 < e_1 \tag{3.9}$$

Notice that more than one of the above conditions might be true. In that case, the result consists of all blocks whose conditions hold.

For each pair of overlapping maximal blocks, we compute their decomposition and the result is added to the set of blocks. In Section 3.4 we will discuss when to use the row-maximal or complete decomposition.

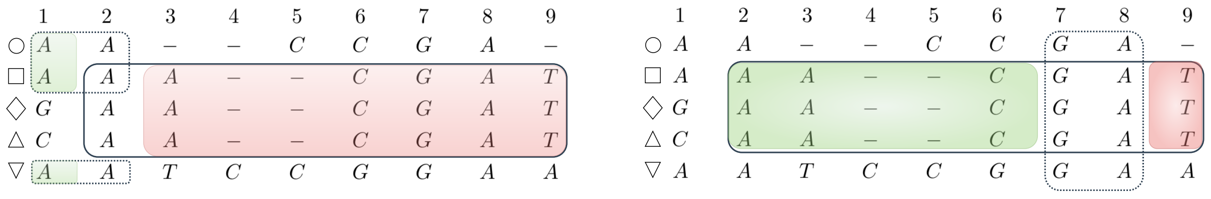


Figure 3.7: The additional blocks obtained by the *row-maximal decomposition* on the overlapping maximal blocks of Figure 3.2 (on the left) and of Figure 3.3 (on the right).

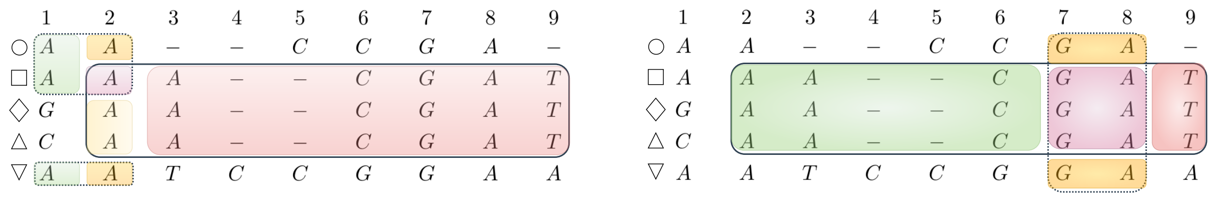


Figure 3.8: The additional blocks obtained by the *complete decomposition* on the overlapping maximal blocks of Figure 3.2 (on the left) and of Figure 3.3 (on the right). Additional blocks are colored.

3.3.4. Adding short blocks

Maximal blocks involving only one row — blocks (K, b, e) with $|K| = 1$ — correspond to vertices of the variation graph encoding an entire genome, without highlighting the common portions of the genomes we want to represent in a pangenome. For this reason, they are undesirable and we remove them from the set $\mathcal{B}$ of blocks. On the other hand, their removal does not guarantee that $\mathcal{B}$ contains an exact cover. To overcome this problem, we add to $\mathcal{B}$ all blocks of the form $(\{r\}, b, e)$ where $[b, e]$ is a maximal interval in the row r not covered by any maximal block spanning at least two rows. We refer to these blocks as *one-row blocks.*

Finally, we add to $\mathcal{B}$ all blocks of the form (K_σ, b, b) for each $1 \leq b \leq n$ and for each character σ, where K_σ is the set of rows that have the character σ in column b. Notice that, for each column b there are at most $|\Sigma|+1$ such blocks, since the character σ can be the indel. We refer to these blocks as *one-character blocks.* The addition of one-row and one-character blocks guarantees that the set $\mathcal{B}$ contains a feasible solution of MWBC.

3.3.5. An ILP for the Minimum Weight Block Cover problem

We can now describe the ILP for solving the MWBC that forms the main ingredient of our approach. We recall that the set of blocks $\mathcal{B}$ fed to the solver consists of the following blocks: i) the set of maximal blocks, ii) the result of the decomposition of the maximal blocks described in Section 3.3.3 (which is either the row-maximal or the complete decomposition),

iii) the one-row blocks, and finally iv) the one-character blocks described in Section 3.3.4. Our implementation also removes duplicate blocks, but that is not relevant to our discussion.

The binary variables we use are $C[K, b, e]$ for each block (K, b, e), where $C[K, b, e] = 1$ iff the block (K, b, e) belongs to the solution. The main constraints in our ILP are the usual ones used to encode an exact cover:

$$\sum_{(K,b,e)\in\mathcal{B}:r\in K,b\leq c\leq e} C[K, b, e] = 1 \quad \forall 1 \leq r \leq k, 1 \leq c \leq n \tag{3.10}$$

The above constraints express the condition that exactly one block in the solution covers each position (r, c) of the MSA.

We have not introduced the objective function of our ILP approach, since different biological or computational goals can be better modeled with different objective functions: one of the advantages of an ILP approach is that we do not rely on a specific objective function, but we can leave it as a decision that the final user will take.

We recall that each block in the solution will be later transformed into a node of a Variation Graph. We are going to show five possible objective functions, the first three will be a subject of our experimental assessment in Section 3.4.

We will denote by $\gamma(K, b, e)$ the label of the blocks (K, b, e) without considering indels. Clearly, the length of $\gamma(K, b, e)$, denoted as $|\gamma(K, b, e)|$, is at most $e - b + 1$ and it is equal to $e - b + 1$ when the string in the block does not contain indels.

1. Minimize the number of blocks. The objective function is called *blocks* and it is defined as

$$\min \sum_{(K,b,e)\in\mathcal{B}} C[K, b, e]. \tag{3.11}$$

2. Minimize the number of vertices of the variation graph, penalizing vertices whose labels are shorter than a certain threshold q. Notice that read mappers based on seeding, such as GraphAligner [10], can benefit from graphs with longer node labels, where it is more likely to find a seed. The objective function is called *weighted* and it is defined as

$$\min \sum_{(K,b,e)\in\mathcal{B}} P(|\gamma(K,b,e)|)C[K,b,e] \tag{3.12}$$

where the penalization function P is defined as

$$P(a) = \begin{cases} \Delta & \text{if } a \leq q \\ 1 & \text{otherwise} \end{cases}$$

and $\Delta > 1$ is the penalization parameter for these shorter labels.

3. Minimize the number of vertices of the variation graphs, penalizing vertices that are not used by at least p input sequences, that is we prefer nodes with high coverage. The rationale is that some variant calling pipelines (such as minigraph-cactus or `vg` prune) start by removing vertices with low coverage. The objective function is called *depth* and it is defined as

$$\min \sum_{(K,b,e)\in\mathcal{B}} P(|K|)C[K,b,e] \tag{3.13}$$

where the penalization function P is

$$P(a) = \begin{cases} \Delta & \text{if } a \leq p \\ 1 & \text{otherwise} \end{cases}$$

with $\Delta > 1$ the penalization parameter.

4. Minimize the total length of the labels of the nodes of the variation graph, excluding indels. The objective function is called *strings* and it is defined as

$$\min \sum_{(K,b,e)\in\mathcal{B}} |\gamma(K,b,e)| C[K,b,e]. \tag{3.14}$$

5. Minimize the total length of the labels of the variation graphs, penalizing vertices that are shared by fewer sequences. The objective function is called *penalized strings* and it is defined as

$$\min \sum_{(K,b,e)\in\mathcal{B}} \frac{|\gamma(K,b,e)|}{|K|} C[K,b,e]. \tag{3.15}$$

3.4. Results

We randomly select 50 complete SARS-CoV-2 genome sequences from ENA and downloaded them with ENA TOOLS[2], we then create an MSA using MAFFT (version 7.525) with default parameters. This MSA has 29903 columns.

Our analysis is divided into 3 parts. The first part examines the scalability of the decomposition strategy, by determining the time and memory needed to apply the *row-maximal* and the *complete* decomposition on the input MSAs. Secondly, we analyze the effect of varying the parameter α, that is the minimum length of forced vertical blocks. More precisely, we report the values of the objective function *blocks* for different values of α, using both decompositions. Finally, we study the effect of choosing different objective functions on the resulting variation graph. The graphs are evaluated on the number of vertices, the length of the graph labels, the number of potential seeds it contains, and the number of shared nodes they have. In this last part, we have compared the variation graphs computed by

[2]https://github.com/enasequence/enaBrowserTools

`pangeblocks` with those created with `vg`, `make_prg`, `founderblockgraph` (starting from the same MSA), and `pggb` (using the same set of sequences).

While our approach requires more computational resources, the experiments show that `pangeblocks` can be run on a personal computer for real instances.

3.4.1. Assessing the decomposition strategies

We start by investigating the impact of the decomposition strategies on the computational resources required to run `pangeblocks`. Since the blocks obtained with the complete decomposition (Definition 3.8) include those obtained with the row-maximal decomposition (Definition 3.7), selecting the complete decomposition leads to a larger ILP instance.

We ran `pangeblocks` varying the number of columns, ranging from 100 to 3000 (incrementing by 100). For each number of columns, we have extracted 10 random MSAs from the complete MSA on 50 SARS-CoV-2 genomes. We ran all those experiments on the *blocks* objective function, since its the choice should not affect the memory and time, and without any forced vertical block (this is imposed by setting α equal to the length of the MSA).

RAM usage and user time are comparable when using both decompositions in instances up to a few hundred columns. After that point, the complete decomposition requires more time and memory than the row-maximal decomposition (see Figure 3.9). The complete decomposition failed since it used too much RAM (more than 90GB) on approximately 18% of the instances — only successful runs are displayed in the Figure.

Notice that the row-maximal decomposition never exceeded 12 GB of RAM and completed its execution in less than 10 minutes.

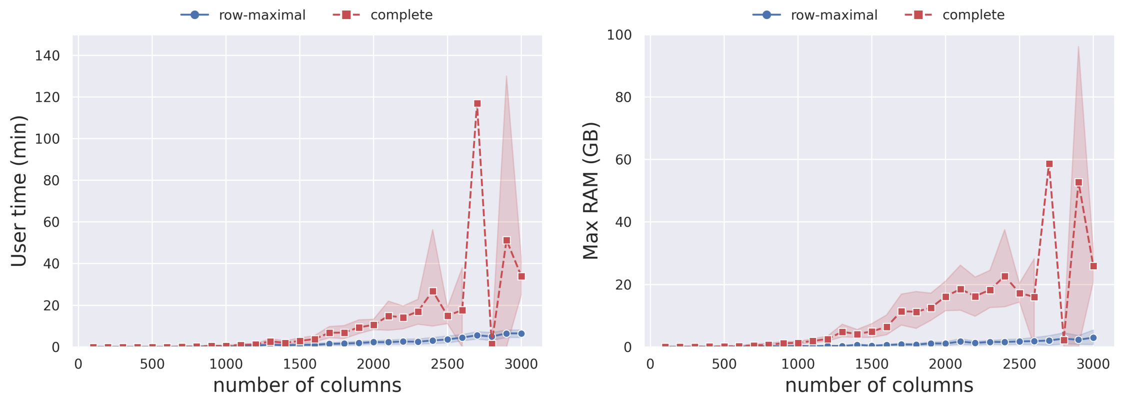


Figure 3.9: Time (on the left) and peak RAM usage (on the right) needed to solve the ILP, as a function of the MSA length and the choice of decomposition. For each MSA length, ranging from 100 to 3000, incremented by 100, we have extracted 10 random MSAs with that length from the MSA of 50 SARS-CoV-2 complete genomes. The red dashed line represents the complete decomposition, and the blue line represents the row-maximal decomposition. The objective function *blocks* was used in this experiment.

3.4.2. Assessing the choice of α

We recall that the parameter α is the minimum number of columns of all forced vertical blocks, that is all vertical blocks spanning at least α columns are forced to be part of the solution. We explore if vertical blocks are chosen by `pangeblocks` with different objective functions.

We focus on those α values where the longest subMSA (between two forced vertical blocks) has more columns than the ones with smaller values of α, we refer to those values of α as *breakpoints*. Breakpoints are especially relevant since each subMSA can be solved independently and in parallel, therefore the computational resources needed are affected mostly by the size of the largest subMSA (*i.e.* the one that changes at a breakpoint). Breakpoints are shown in Figure 3.10 and can be identified as the values of α where the blue curve increases its value. The effect of choosing different values of α on the entire MSA can be seen in Figure 3.11, where black rectangles represent the forced vertical blocks, white areas are the subMSA with the largest number of columns, and the gray rectangles are all other subMSAs.

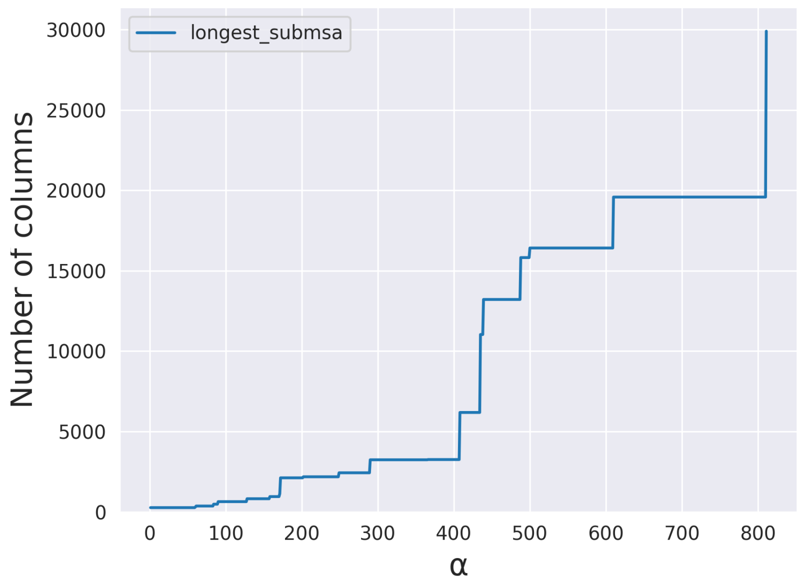


Figure 3.10: Longest (columns) subMSA obtained for different values of α in the SARS-CoV-2 MSA with 50 sequences and 29903 columns . The number of columns (y-axis) shows the longest subMSA given that all vertical blocks with length at least α (x-axis) are fixed. The curve shows how increasing the value of α implies an increase of the number of columns in the longest subMSA. In particular, we call *breakpoints* of α those values of α that determines a change in length of the subMSA. This plot can help to decide a reasonable value of α given the available resources. A picture of the MSA given these breakpoints is shown in Figure 3.11.

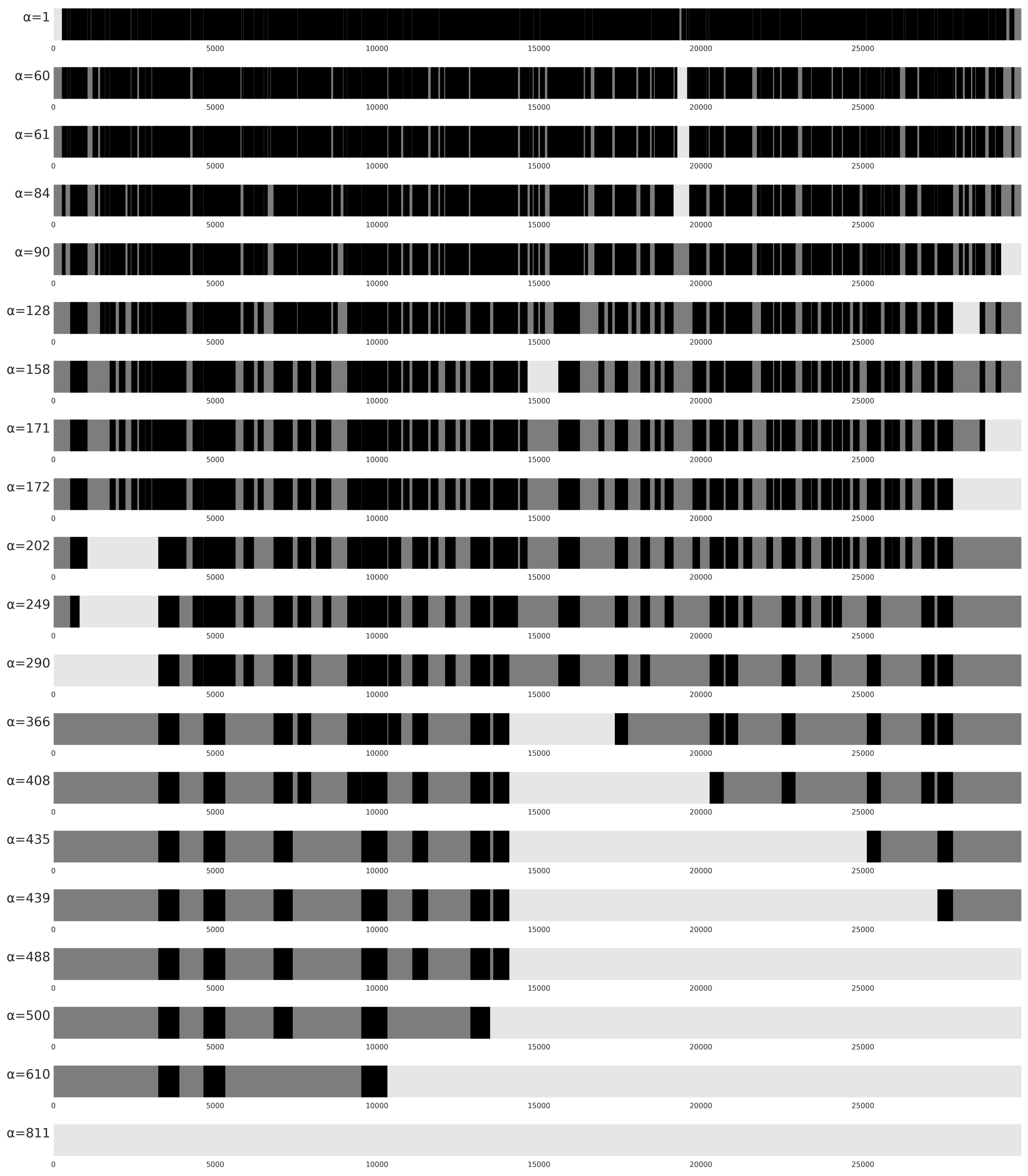


Figure 3.11: A view of SARS-CoV-2 MSA with 50 sequences and 29903 columns for different values of α (see label to the left), from $\alpha = 1$ to $\alpha = 881$ (from top to bottom). Forced vertical blocks (those with the number of columns less or equal to α) are shown in **black**. SubMSAs in between vertical blocks (or at the beginning, or the end of the MSA) are in gray. The longest subMSA is shown in light gray. Values of α correspond to *breakpoints* shown in Figure 3.10, where the longest subMSA when fixing α increases in length.

Notice that we have decided to focus on the number of columns of the largest subMSA instead of the time and memory used, since Section 3.4.1 shows that the complete decomposition can manage at most a few thousand columns using at most 90 GB of RAM. On the other hand, Figure 3.10 represents all possible number of columns of a subMSA, hence it does not have such restriction. Moreover, each subMSA is solved independently, therefore we can actually use the number of columns of the largest subMSA as a proxy of the computational resources needed.

Now we can analyze the effect of the choice of α on the number of vertices of the resulting graph — results are shown in Figure 3.12 — when we use the *blocks* objective function. Since we need to run `pangeblocks` on the resulting subMSAs, we have been able to consider only values of α smaller than 158.

We expected larger values of α to correspond to fewer vertices, but this is what we observe for the complete decomposition only, and even in that case it is not a monotonous decrease. This phenomenon is due to the fact the maximal blocks and their decomposition is computed independently for each subMSA, therefore the set of blocks fed to the ILP for larger values α is not a superset of the set for smaller values. Moreover, the phase where a solution of MWBC is used to compute a variation graph introduces an additional uncertainty in the number of vertices of the final graph.

We observe in Figure 3.12 that the row-maximal decomposition reach a plateau, or even begin to worsen, for moderate values of α — around 90 for the row-maximal decomposition — hinting that the large computational resources needed for larger values of α are not actually necessary in this case. In the case of the complete decomposition, due to computational resources, we are limited to consider up to $\alpha = 158$, nevertheless, in smaller instances (See Supplementary material) we observe the same behavior described for the row-maximal decomposition.

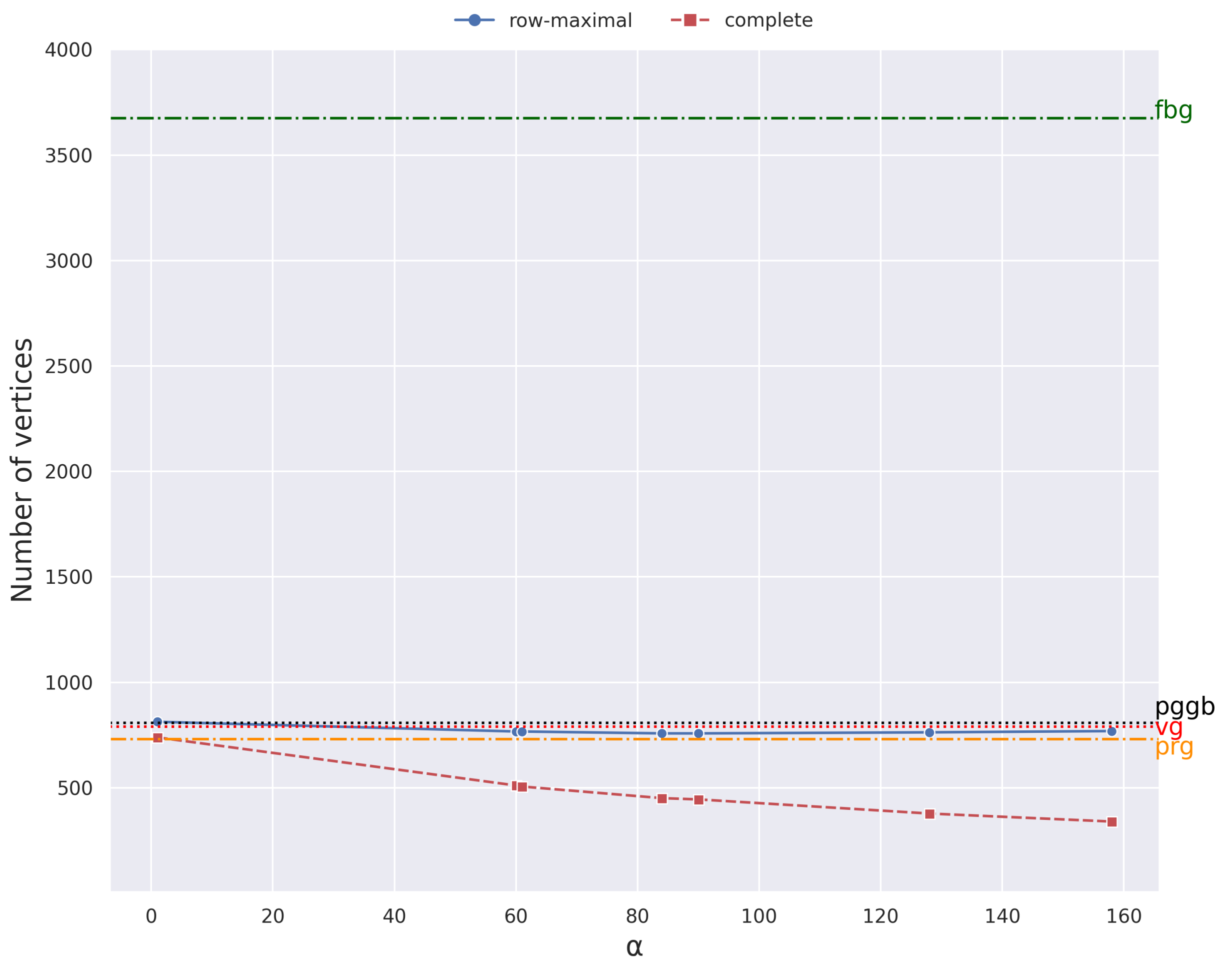


Figure 3.12: **Number of vertices**. The plot is the results of running `pangeblocks`, `pggb`, `vg`, `make_prg` and `founderblockgraph` on the 50 SARS-CoV-2 genomes instance, using the row-maximal (blue dot) and the complete (red square) decomposition. The y-axis is the number of vertices of the variation graphs (after post-processing), and the x-axis corresponds to various values of α (from 1 to 158, see Figure 3.11). The number of nodes for other tools is shown by horizontal lines. The black dotted line corresponds to `pggb`, the red dashed line corresponds to `vg`, and the orange dash-dot line corresponds to `make_prg`, and the green dash-dot line corresponds to `founderblockgraph`.

It is especially noticeable that both decompositions are able to obtain graphs with fewer vertices than `vg` and `pggb` as shown in Figure 3.12. This is partially surprising since `vg` and `pggb` can build variation graphs that contain cycles (which helps in reducing the size of the graph), while `pangeblocks` constructs only acyclic graphs. `make_prg` creates a graph with the number of nodes in between both decompositions, while `founderblockgraph` graph has more than 3× the number of nodes than all other tools. The row-maximal decomposition results in essentially the same number of vertices as `vg` and about 10% fewer vertices than `pggb`. On the other hand, the more computationally expensive complete decomposition obtains 55-60% fewer vertices than `vg`.

By choosing α we are fixing vertical blocks with equal or more columns than α, this imposes finding a suboptimal solution of the MSA by solving an ILP for each resulting subMSA not covered by the fixed vertical blocks. Vertical blocks are nodes in the graph traversed

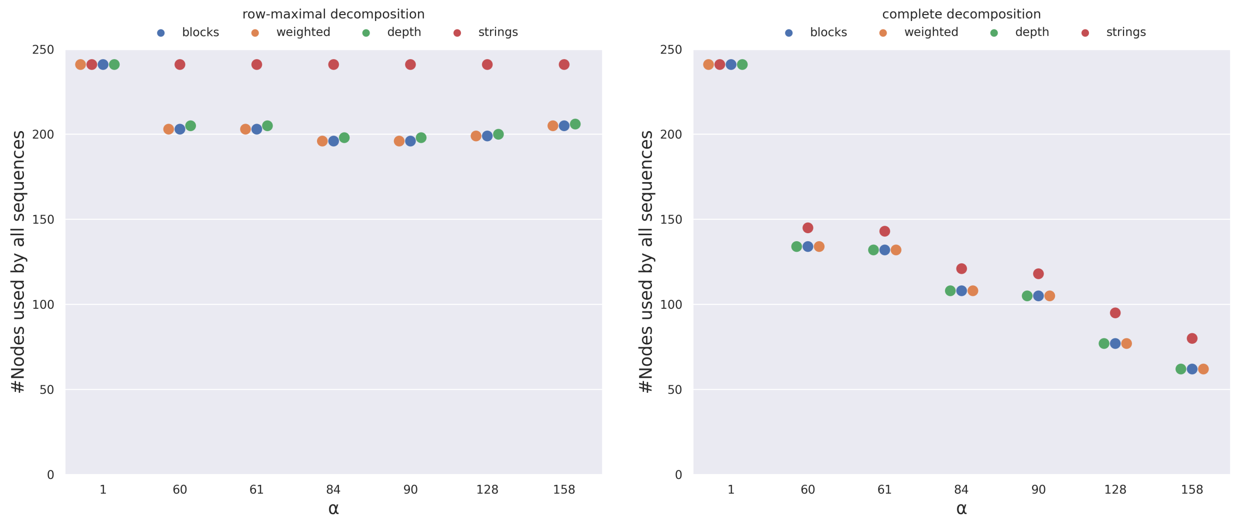


Figure 3.13: Number of vertices of the variation graph that are traversed by all sequences as a function of α and the choice of decomposition, on the MSA of 50 SARS-CoV-2 complete genomes. For the complete decomposition (right), the objective functions *blocks*, *weighted*, and *depth* coincide in the use of vertical blocks (y-axis), and decrease when increasing α, the same tendency for *strings* with higher usage of vertical blocks. In the row-maximal decomposition (left), the tendency is to decrease the use but with lower fluctuation due to the smaller search space. The *strings* objective function does not change with α, meaning that vertical blocks are always preferred.

by all sequences. We can see in Figure 3.13 (right) that with the complete decomposition, the *blocks*, *weighted*, and *depth* objective functions lead to fewer nodes that are shared by all sequences when α is increased. With larger values of α, the ILP can discard (unforced) vertical blocks to find a better solution that contains larger blocks, i.e. those that cover more cells in the MSA. In other words, both the number of blocks and the total length of labels of the graph decrease with larger values of α, as expected since the space of the feasible solution is larger.

The left plot of Figure 3.13 is on row-maximal decomposition, where there is no substantial difference among the objective functions. When choosing the *depth* objective function, the procedure prefers to pick blocks traversed by a larger number of sequences. Among those, all vertical blocks fit this criteria, therefore they are likely to be retained even for larger values of α, hence the final solution computed is largely unaffected by the choice of α.

This is likely to be the result of the smaller search space created by this decomposition, which is heavily constrained and does not allow the choice of the objective function to affect the selection of the blocks covering the MSA. On the other hand, larger values of α and the associated larger search space lead to fewer nodes overall, including those shared by all nodes. Finally, comparing the two plots of Figure 3.13 we can observe that the two decompositions lead to a completely different behavior of the ILP when choosing the different objective functions, except for $\alpha = 1$, when all vertical blocks are forced to be in the solution.

3.4.3. Assessing the objective functions

In this section, we assess the effect of the objective functions on some properties of the resulting variation graph. More precisely, we analyze the blocks, weighted, and depth objective functions, since the strings and the penalized strings objective functions result in graphs that have a simplistic structure. The optimal solution with the strings objective function consists of one-character blocks, therefore demanding the nontrivial graph optimization steps to the post-processing phase.

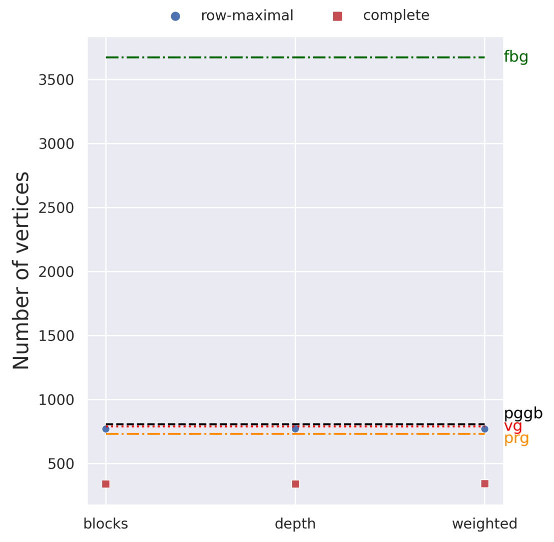
Figure 3.14 (a) shows the total number of vertices of the variation graphs (after post-processing) obtained with the different objective functions, with $\alpha = 168$. In all three settings, the number of vertices is smaller than those achieved by `pggb` and `vg`, even though the difference with `vg` is negligible when using the row-maximal decomposition. Choosing the *blocks* or the *weighted* objective functions produces graphs with the fewest vertices.

When choosing the weighted objective function, we have run `pangeblocks` with the parameter q equal to 20 (all blocks spanning fewer than q columns have cost $\Delta = 1000$ instead of 1). We present only the results with $\Delta = 1000$, since smaller values of Δ result in graphs that are very close to those obtained with $\Delta = 1$, that is without penalization. When choosing the depth objective function, we have run `pangeblocks` with the parameter p equal to 0.11 (all blocks that are traversed by fewer than a p fraction of the input sequences have cost $\Delta = 1000$ instead of 1). When choosing the blocks objective function, no parameter is needed.

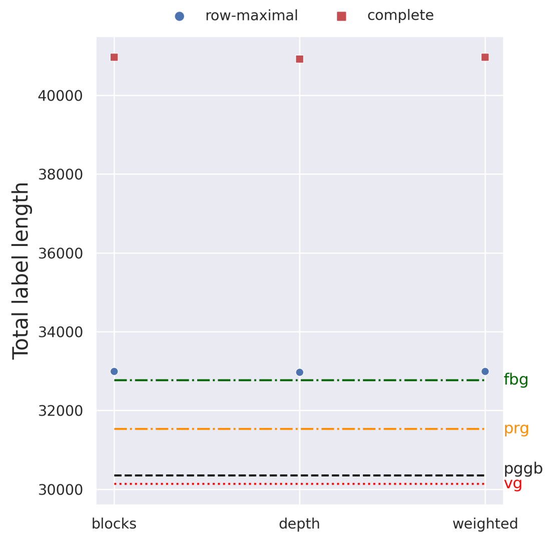
Notice that choosing the *weighted* objective function results in a graph with negligible differences from the one created by choosing the *blocks* objective function.

We recall that the objective function determines the block cover, while the y axis of the figure is computed on the result of the post-processing over the block cover, nodes that are labeled only by indels are removed. The *weighted* objective function is designed to prefer nodes with longer labels, by penalizing those who contain too many indels, see Eq. 3.12, which leads to the selection of fewer, but longer (w.r.t. their label without indels) nodes, as we can see in Figure 3.14 (b).

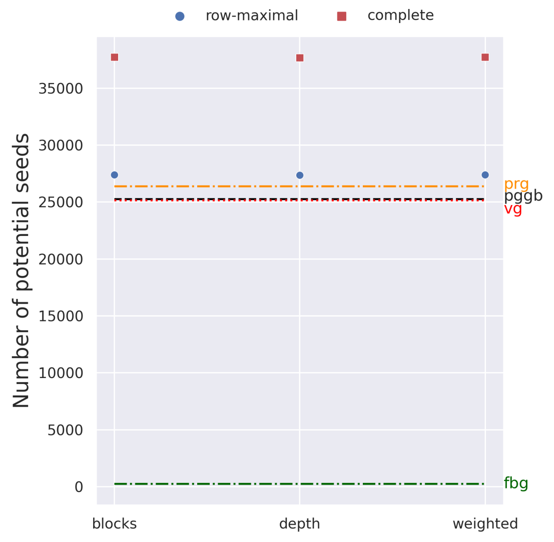
Figure 3.14 (b) shows the total number of characters of the graph labels (after post-processing) obtained with the different objective functions, with $\alpha = 158$. In all three



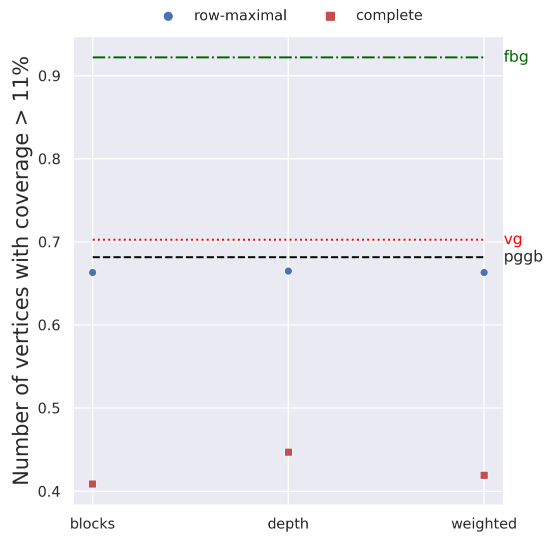
(a) Number of vertices



(b) Length of the graph



(c) Number of seeds



(d) Node depth

Figure 3.14: Four metrics were measured over pan-genome graphs created with five construction tools on the 50 SARS-CoV-2 genome instance. We report (a) the number of vertices of each graph, (b) the total length of the graph, (c) the number of potential seeds (k-mers) of length 20, and (d) the number of vertices used by at least the 11% of the input sequences. For `pangeblocks` we report the results of three objective functions: *blocks*, *depth*, and *weighted* (horizontal axis) The metrics for other tools are shown by horizontal lines. The black dotted line corresponds to `pggb`, the red dashed line corresponds to `vg`, the orange dash-dot line corresponds to `make_prg`, and the green dash-dot line corresponds to `founderblockgraph`. Since `make_prg` creates a sequence graph, node depth cannot be computed for it.

settings, the total number of characters obtained by `pangeblocks` is larger than that achieved by `pggb` and `vg`, even though the differences are marginal when using the row-maximal decomposition.

Figure 3.14 (c) shows the total number of potential seeds (that is the 20-mers that are substring of at least a vertex label) of the variation graphs, obtained with the different objective functions, with $\alpha = 158$.

When using the complete decomposition, we obtain a number of seeds that is larger than those achieved by `pggb` and `vg`. When choosing the row-maximal decomposition, the results obtained by `pangeblocks` are marginally larger than both `vg` and `pggb`. The *weighted* objective function produces a graph with more seeds, thus the graph can be potentially more informative if the task is aligning reads to the graph. Figure 3.14 (d) shows the fraction of vertices traversed by at least 11% of the genome sequences, obtained with the different objective functions, with $\alpha = 158$. When using the row-maximal decomposition, the results are slightly smaller than `pggb`. `vg` on the other hand has the highest number of these nodes. In the case of `pangeblocks`, we notice that choosing the *depth* objective function results in a higher number of vertices with this coverage, for both decompositions, as was expected, since this objective function was built for this purpose.

In general, we can observe that `founderblockgraph` creates bigger graphs w.r.t. the other tools, both in number of nodes and length of the graph, with the particularity of very short nodes, which translates into a small number of potential seeds. `make_prg` creates graphs with fewer nodes than `vg`, `pggb`, and `founderblockgraph`, but with longer nodes, showing a higher number of potential seeds than the other tools.

3.5. Discussion

We have presented a computational problem, Minimum Weight Block Cover (MWBC), that formalizes the problem of constructing a variation graph from an MSA. We have proposed an ILP approach for solving the MWBC problem and constructing a graph and we implemented it as a tool, called `pangeblocks`.

We have experimentally shown that `pangeblocks` can scale to genome-scale instances. More precisely, our experimental analysis has been run on an MSA on 50 complete SARS-CoV-2 genome sequences (approximately 30kbp). We have also shown how the ILP formulation is sufficiently flexible to accommodate different goals. In fact, by choosing an objective function, the decomposition strategy, or the minimum size of a forced vertical block, we are able to obtain variation graphs that have different characteristics, for example we can obtain graphs whose number of vertices ranges from 800 to 1400. Notice that all graphs computed by `pangeblocks` are complete representations of the input MSA, that is each input genome corresponds to the concatenation of the labels if a path of the graph.

We have compared `pangeblocks` with the two most widely used tools for constructing variation graphs, `vg` and `pggb`, showing that we can obtain graphs that are similar to theirs. We have to remind that our ILP approach is more flexible, therefore allowing to tailor the graph construction to specific need. At the same time, those possibilities affect the computational resources needed by `pangeblocks`.

We observe that the use of ILP and optimization criteria in the construction of the variation graphs allows to include biological constraints. In particular, we have introduced the parameter α to bound the length of the so called vertical blocks, i.e. those nodes of the graph corresponding to consecutive MSA columns where the input sequences are all identical. While we have introduced α for computational reasons, setting its value might have a possible biological motivation, since it can influence the regions where recombinations are more likely.

There are some directions for future research. First of all, there is a large gap in both the computational resources needed and the resulting graphs, between the complete and the

Row-maximal decompositions. Therefore we think that different decomposition strategies can be relevant. Moreover, we would like to keep most of the flexibility allowed by the ILP approach, but avoiding its computational cost. This requires restricting the possible parameters to a subset of those that are currently modeled by the ILP, and designing a combinatorial approach that is able to quickly compute a solution on those parameters.

3.6. Availability of data and materials

All sequences used in this manuscript were downloaded from [3]ENA. The datasets generated and/or analysed during the current study are available in the Zenodo repository doi.org/10.5281/zenodo.12187814.

[3]https://www.ebi.ac.uk/ena

Chapter 4

Comparison of pangenome graphs with Ricci-Flow

A pangenome graph is a data structure used to represent a collection of genomic sequences, with each sequence depicted as a path within the graph. Regardless of the increasing development of graph builders, the comparison of different graph representations has been always based on some downstream analysis; like variant calling or read alignment. Even though metrics for graph comparison exist, to the best of our knowledge no prior work has extended any of these approaches to work on Pangenome Graphs.

The comparison of Pangenome Graphs imposes a challenge from the combinatorial point of view since paths and labels need to be considered, and the graph isomorphism problem is not known to be solvable in polynomial time yet. We propose to study the comparison of two pangenome graphs using Riemannian geometry, in particular, with the use of Ricci-Flow which was proposed in the 1980s.

In this ongoing work, we extend previous work for the comparison of undirected and unlabeled graphs to variation graphs. We present a framework where information regarding nodes, edges, and paths can be considered, and explore how to align variation graphs modeled as discrete manifolds.

A proof of concept of the work introduced here was materialized in the tool `PanRicci`, open source and available at https://github.com/jorgeavilacartes/panricci.

Our contribution

- We propose a framework for modeling variation graphs as discrete manifolds, enabling manipulation with the discrete Ollivier Ricci-Flow algorithm. Specifically, we show how path and label information in a variation graph can define probability distributions over its nodes.
- We provide formal definitions of isomorphism for both variation graphs and discrete manifolds.
- Our method uses Integer Linear Programming to align variation graphs by defining relative node representations. These representations are derived from weights obtained after applying discrete Ricci-Flow to the corresponding discrete manifold.

4.1. Introduction

Pangenome graphs can be built with different tools (`vg` [13], `minigraph` [14], `pggb` [15], `minigraph-cactus` [7], `make_prg` [4], `founderblockgraph` [17]), but the comparison of their results has always relied on specific downstream analyses [6, 7, 16], while we would like to have a quality measure that is not overly dependent on downstream analysis, but on the graphs themselves.

A motivation to pursue a geometrical direction to align pangenome graphs based on [70], where the problem of covering alignment between graphs was investigated. Their work concludes that comparing two pangenome representations is NP-hard if the notion of covering alignment they propose is accepted. Comparing pangenome graphs presents a combinatorial challenge, as both paths and labels must be taken into account.

In our approach, we explore a novel direction using Riemannian geometry. Specifically, we leverage the concept of manifolds, which are smooth, curved spaces that generalize the idea of surfaces in higher dimensions. Ricci-Flow, introduced by Hamilton in the 1980s [71], is a process that smooths out the curvature of a manifold over time, providing a powerful tool in Riemannian geometry. This area of mathematics gained significant attention in the early 2000s when Perelman published a series of groundbreaking papers [72–74], claiming to have solved the Poincaré Conjecture, one of the Millennium Prize Problems. However, it took nearly four years for the mathematical community to fully understand and validate his results.

Subsequently, Ollivier extended these concepts to metric spaces [75, 76], and in [77], the Ollivier-Ricci curvature was adapted to graphs. This adaptation has led to numerous applications in graph theory, such as approximate graph alignment [18], community detection in networks [78], and the study of over-squashing and bottlenecks in the training of graph neural networks [79].

We propose studying the comparison of two variation graphs using the discrete Ollivier Ricci-Flow [18], along with an extension of Ricci-Flow to discrete manifolds. Our approach begins by exploring the isomorphism between variation graphs within this framework, and we introduce formal definitions of alignment between manifolds to establish a connection with alignment in variation graphs.

4.2. Preliminaries

First, we introduce the Wasserstein distance, a metric for probability distributions, along with the concepts of discrete manifolds, Ricci curvature, and discrete Ollivier Ricci-Flow. Finally, we provide definitions of the isomorphism between graphs and variation graphs.

4.2.1. Wasserstein distance

The Wasserstein distance (also known as the optimal transport distance) has been extensively studied for comparing probability distributions on Riemannian manifolds [80].

Here we define the Wasserstein for two discrete probability distributions.

Definition 4.1 [Wasserstein Distance] Given two discrete probability distributions $f_1 = (a_1, \cdots, a_m)$ and $f_2 = (b_1, \cdots, b_n)$, the p-Wasserstein Distance between f_1 and f_2 is defined as $\mathcal{W}(f_1, f_2) = (\langle C, T \rangle)^{1/p}$, where $C = (c_{ij})$ assigns a cost for each pair (i, j), and $T = (T_{ij})$ is the optimal transport plan obtained by the solution of the following model,

$$\begin{aligned} \text{minimize} \quad & \sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}T_{ij} \\ \text{subject to} \quad & \sum_{i=1}^{m} T_{ij} = a_j \quad , j = 1, \ldots, n \\ & \sum_{j=1}^{m} T_{ij} = b_i \quad , i = 1, \ldots, n \\ & \qquad\qquad T_{ij} \geq 0 \end{aligned}$$

4.2.2. Discrete manifolds and Curvature

The main object of study in this framework is the manifold. In particular, we will model a variation graph (see Definition 2.1) as a discrete manifold, which we formalize in the following definition.

Definition 4.2 [Discrete manifold] A discrete manifold is a tuple (X, d) with X a set of elements, and $d : X \times X \mapsto \mathbb{R}^+$ a metric. Each element of X is doted by a probability distribution over its one-step proximity neighbors, *i.e.* over elements that can be reached across one geodesics defined by the metric d (analogous to traversing one edge in a graph).

To better understand the concept of curvature, consider the following analogy. Suppose you need to move an object from one point to another along the shortest possible path, known as a geodesic. In this scenario, the curvature of the surface indirectly quantifies the effort required to transport the object along the geodesic.

Consider the difference between moving an object across various terrains. On a flat surface, where curvature is zero, the energy required to move the object is consistent regardless of the direction you choose —– whether moving straight, left, or right. This is because a flat surface implies that the shortest path between two points remains constant and the effort is uniform. In Riemannian geometry, a surface with zero curvature is one in which distances and the corresponding effort are unchanged across different directions.

In contrast, if the object is moved uphill on a surface with negative curvature, the effort increases due to the steeper incline. The path is more challenging to traverse, and more energy is expended in the process. Similarly, if moving downhill on a surface with positive curvature, the path might appear easier, but additional factors, such as gravity or the nature of the terrain may increase the effort required to maintain control.

In essence, curvature tells you how terrain affects the effort to move an object along a geodesic. On a flat surface, the effort is constant; on a curved surface, the effort changes depending on the type of curvature.

Curvature in a discrete manifold (X, d), is defined for a pair of elements $u, v \in X$, and is denoted by $\kappa_{(u,v)}$. It measures the deviation of the manifold from being flat.

$$W(m_u, m_v) = (1 - \kappa_{(u,v)})d(u, v), \tag{4.1}$$

where m_u,m_v are probability distributions defined over the elements u and v, respectively, and $W(\cdot,\cdot)$ is the Wasserstein distance with costs defined by the shortest paths between pairs of nodes in the metric d.

In the next section, we pay special attention to these probability distributions, an essential part of the curvature. We will show how to manipulate them to model the information present in variation graphs.

For simplicity, we will consider a discrete manifold (X, d) as a weighted graph $G = \langle V, A, w \rangle$, where for each node $u \in V$ we have a probability distribution m_u. We assume we have a bijection between nodes of G and the elements in X, $\Theta : V \mapsto X$ such that the weight function $w : V \times V \mapsto \mathbb{R}^+$ is defined by the metric d as $w(u, v) = d(\Theta(u), \Theta(v))$. Furthermore, the weighted graph is assumed to be a single connected component. Since d is a metric, for any existing edge $(u, v) \in A$, the weight of such edge, $w(u, v)$, is the shortest path between those nodes. This will be relevant since the curvature in a discrete manifold is computed only for existing edges.

From equation Eq. 4.1 we can define the curvature as follows.

Definition 4.3 [Discrete Ollivier-Ricci Curvature [18]] The discrete Ollivier-Ricci Curvature of an edge (u, v) is defined by

$$\kappa_{(u,v)} = 1 - \frac{W(m_u, m_v)}{d(u, v)} \tag{4.2}$$

where m_u,m_v are probability distributions defined over the nodes u and v, respectively, and $W(\cdot,\cdot)$ is the Wasserstein distance. While $d(u, v)$ is the distance between nodes u and v given by the weight of the arc $(u, v) \in A$.

The overall curvature of the graph can be defined as the average curvature of its edges:

$$\kappa = \frac{1}{|E|} \sum_{(u,v)\in E} \kappa_{(u,v)} \tag{4.3}$$

Given a discrete manifold, we can apply the following Ricci-Flow algorithm to evolve its metric.

4.2.3. Probability measures for nodes

Probability measures are the key to compute the curvature of an edge, as shown in Definition 4.3. In [18], given $0 < \alpha < 1$, the probability measure m_α^u of a node u is defined as follows,

$$m_\alpha^u(v) = \begin{cases} \alpha & , v = u \\ (1-\alpha)/|\pi(u)| & , v \in \pi(u) \\ 0 & , \sim \end{cases} \tag{4.4}$$

In [78] another probability measure extending definition (Eq. 4.4) was proposed, to consider nodes distant from the proximal neighbors, but we stick with the former definition.

Let us observe that (Eq. 4.4) can be generalized by,

$$m_\alpha^u(v) = \begin{cases} \alpha & , v = u \\ (1-\alpha) f_u(v) & , v \in \pi(u) \\ 0 & , \sim \end{cases} \tag{4.5}$$

where the mapping $f_u : \pi(u) \mapsto [0,1]$ satisfies $\sum_{v\in\pi(u)} f(v) = 1$.

Notice that in equation (Eq. 4.4), $f_u(v) = 1/|\pi(u)|$ for all nodes $u \in V$, which is not suitable for modeling information like paths and labels.

4.2.4. Ricci-Flow

Manifolds preserve the intrinsic relationships between their elements but possess a degree of freedom in their metric. The Ricci-Flow algorithm provides a mechanism to iteratively modify this metric, smoothing the manifold's curvature while maintaining its underlying structure.

Definition 4.4 [Discrete Graph Ricci-Flow algorithm [18]] The discrete Ricci-Flow is an iterative process applied to the edges of a discrete manifold $G = \langle V, A, w \rangle$, where in each iteration t, the weight of each edge $(u,v) \in A$, denoted by $w^{(t)}_{(u,v)}$ is updated until the overall curvature of the graph is constant and equal to 0. The updating process is defined as follows

$$w^{(t+1)}_{(u,v)} = w^{(t)}_{(u,v)} - \kappa^{(t)}_{(u,v)} \cdot w^{(t)}_{(u,v)} \tag{4.6}$$

where weights for t=0 are initialized randomly ($w^{(0)}_{(u,v)} > 0$).

Notice that the initialization of the weights must preserve d as a metric, *i.e.* , the shortest path between two connected nodes must be the weight of the edge itself, otherwise, the triangle inequality breaks.

In [19] a *normalized* version of the Ricci-Flow is proposed, where in each iteration, the sum of the weights of the graph is ensured to be equal to 1. This prevents the graph from collapsing to a point since all edges have a positive weight.

Definition 4.5 [Discrete Normalized Graph Ricci-Flow algorithm [19]]

$$w^{(t+1)}_{(u,v)} = w^{(t)}_{(u,v)} - \kappa^{(t)}_{(u,v)} \cdot w^{(t)}_{(u,v)} + w^{(t)}_{(u,v)} \sum_{(i,j)\in A} \kappa^{(t)}_{(i,j)} w^{(t)}_{(i,j)} \tag{4.7}$$

In the case of networks with a large number of edges, the edge weight can be infinitesimal under the constraint that all weights sum up to 1. To avoid this, a small modification of definition 4.5 was proposed, where the sum over the edges is defined by $\sigma > 1$. In addition, a step size $\varepsilon > 0$ is also introduced.

Definition 4.6 [σ-weighted-sum of Discrete Normalized Graph Ricci-Flow algorithm [19]]

$$w^{(t+1)}(u,v) = w^{(t)}(u,v) - \varepsilon \cdot \left(\kappa^{(t)}_{(u,v)} - \frac{1}{\sigma} \sum_{(i,j)\in E} \kappa^{(t)}_{(i,j)} w^{(t)}(i,j)\right) \cdot w^{(t)}(u,v) \tag{4.8}$$

With this modified normalization, the calculation is guaranteed within the computer precision regardless of networks with many edges [19].

Finally, when (any of) the Ricci-Flow algorithm is applied to a manifold, the resulting metric is defined as the Ricci-Flow metric.

Definition 4.7 [Ricci-Flow Metric [18]] After Ricci-Flow is applied to a manifold (X, d), the (updated) metric d—determined by the shortest paths between nodes based on the weights when the overall curvature is 0— is called the Ricci-Flow metric.

4.2.5. Isomorphism on graphs

We first define the isomorphism on unlabeled unweighted graphs.

Definition 4.8 [Isomorphism between graphs] Given $G_1 = \langle V_1, A_1\rangle$ and $G_2 = \langle V_2, A_2\rangle$ two graphs. We say that G_1 and G_2 are isomorphic if there exists a bijection $\phi : V_1 \mapsto V_2$ such that $(u,v) \in A_1$ if and only if $(\phi(u), \phi(v)) \in A_2$.

And we extend this definition to variation graphs, where we include labels and paths.

Definition 4.9 [Isomorphism between variation graphs] Given $G_1 = \langle V_1, A_1, W_1 \rangle$ and $G_2 = \langle V_2, A_2, W_2 \rangle$ two variation graphs, with $\lambda_1 : V_1 \mapsto \Sigma^+$, and $\lambda_2 : V_2 \mapsto \Sigma^+$ labelling functions. If there exists a bijection $\phi : V_1 \mapsto V_2$ such that

1. $(u, v) \in A_1$ if and only if $(\phi(u), \phi(v)) \in A_2$.
2. $u_1, \ldots, u_n \in W_1$ if and only if $\phi(u_1), \ldots, \phi(u_n) \in W_2$.
3. $\lambda_1(u) = \lambda_2(\phi(u))$ for all $u \in V_1$.

4.3. Methods

Our approach consists of modeling a variation graph as a manifold and applying Ricci-Flow to obtain its metric when the overall curvature is 0. Then we use the updated metric to define relative vector representations of each element of the manifold w.r.t. chosen reference elements.

In this section, we start by showing how to model a variation graph as a manifold by defining probability measures for nodes from a variation graph that fits into this framework. Then we define the relative vector representations of nodes that depend on its metrics and relate it to the alignment problem between variation graphs. Finally, we propose an Integer Linear programming approach to solve the mapping between discrete manifolds and focus on how to solve the isomorphism between variation graphs.

4.3.1. Probability measures for nodes in variation graphs

Given a variation graph $G = \langle V, A, W \rangle$, with labeling function $\lambda : V \mapsto \Sigma^+$, and $u \in V$, let us denote by $\pi_{out}(u) = \{v : (u, v) \in E\}$, and by $\pi_{in}(u) = \{v : (v, u) \in A\}$, and by $\pi(u) = \pi_{in}(u) \cup \pi_{out}(u)$, notice that $u \notin \pi(u)$ (if we do not allow bucles).

In a variation graph, paths are present in the graph, and traverse nodes. The proportion of paths traversing a node is called *node depth*, formally defined as follows.

Definition 4.10 [Node depth] Given a Variation Graph $G = \langle V, A, W \rangle$, the node depth of $u \in V$, is defined as the number of paths using the node u,

$$N_d(u) = |\{p \in W : u \in p\}|$$

Given a node $v \in \pi(u)$, we can incorporate the paths in the definition of f_u (see Equation 4.5) using the node depth as follows.

Definition 4.11 [Probability measure for a node in a Variation Graph with paths] Given a Variation Graph $G : \langle V, E, W \rangle$, and given a node $v \in V$, with $\pi(v)$ the neighborhood of v, with $0 < \alpha < 1$.

$$m_\alpha^u(v) = \begin{cases} \alpha & , v = u \\ (1-\alpha)\dfrac{N_d(v)}{\sum_{w \in \pi(u)} N_d(w)} & , v \in \pi(u) \\ 0 & , \sim \end{cases} \tag{4.9}$$

Notice that we weighted each node depth by the total sum of them (in the neighborhood of u), this is necessary to obtain a mapping that sums up to 1. Following the same reasoning previously described, we can replace the node depth in Eq. 4.9 by an arbitrary function over the neighbors of u, $g_u : \pi(u) \mapsto \mathbb{R}^+$

To include node labels in the probability distribution of the node u, we propose to extract the labels of the subpaths traversing a node in a subgraph around u, such a subgraph is formally defined as a locally induced variation graph.

Definition 4.12 [Locally Induced Variation Graph] Given a variation graph $G = \langle V, A, W\rangle$, and a node $u \in V$, we will define a locally induced variation graph as the variation graph around the node u and its neighbors, formally $G_u = \langle \{u\} \cup \pi(u), A_u, W_u\rangle$, where A_u is the set of edges in E with u as one endpoint, and W_u is the set of sub-paths of W traversing u.

Given a node u, we use the locally induced variation graph $G_u = \langle \{u\} \cup \Pi(u), A_u, W_u\rangle$, to use a k-mer based definition of g_u.

Definition 4.13 [Probability measure for a node in a Variation Graph with labels] Given a Variation Graph $G : \langle V, E, W\rangle$, and given a node $v \in V$, with $\pi(v)$ the neighborhood of v, with $0 < \alpha < 1$.

$$m_\alpha^u(v) = \begin{cases} \alpha & , v = u \\ (1-\alpha)\dfrac{g_u(v)}{\sum_{w\in\pi(u)} g_u(w)} & , v \in \pi(u) \\ 0 & , \sim \end{cases} \tag{4.10}$$

where g_u is define as follows

$$g_u(v) = \frac{\sum_{m\in K_v \cap K_u} (F^v(m) + F^u(m))}{2\sum_{m\in K_u} F^u(m)} \tag{4.11}$$

K_v correspond to the set of k-mers computed from the set of paths (strings) in the induced variation graph of node v (analogous for K_u), and $F^v(m)$ correspond to the frequency of k-mer m in the set K_v (analogous for $F^u(m)$).

4.3.2. Variation graph as a discrete manifold

To model a variation graph $G = \langle V, A, W \rangle$ as a discrete manifold (X, d), we simply define each node in V as an element of X, and define probability distributions for each node as described in Section 4.3.1.

To make explicit the situation where we have a manifold obtained from a variation graph after Ricci-Flow, *i.e.* provided with a Ricci-Flow metric d, we define it as a discrete manifold variation graph as follows.

Definition 4.14 [Discrete manifold variation graph] Given a variation graph $G = \langle V, A, W \rangle$ with labeling function $\lambda : V \mapsto \Sigma^+$, we will denote by $\mathcal{M}(G)$ the discrete manifold (X, d), where X corresponds to the set of nodes V, and d is obtained by applying (one of the) discrete Ricci-Flow to G.

We do not explicitly mention the distribution of nodes in the previous definition, since when applying (normalized) discrete Ricci-Flow the resulting metric d captures this information.

4.3.3. Alignment between discrete manifolds variation graphs

Now that we have definitions of discrete manifolds, we are interested in finding a way to align them, *i.e.* to find a correspondence between their elements. We start by defining the mapping between discrete manifolds using their metrics.

Definition 4.15 [Mapping between discrete manifolds] Given $\mathcal{M}_1 = (X_1, d_1)$ and $\mathcal{M}_2 = (X_2, d_2)$ two discrete manifolds, and without loss of generality let us assume that $|X_1| \leq |X_2|$, a mapping between $\mathcal{M}_1$ and $\mathcal{M}_2$ is a partial injection $\phi : X_1 \mapsto X_2$ such that the following expression is minimized,

$$\sum_{u,v \in X_1} |d_1(u,v) - d_2(\phi(u), \phi(v))| \tag{4.12}$$

Notice that in our situation, where variation graphs are modeled as manifolds, the mapping ϕ corresponds to a mapping between nodes of the graphs involved in the alignment.

Given that with the Ricci-Flow metric we have the distance for each pair of nodes, following the idea proposed in [18], we propose to define *relative distance representations* for each node w.r.t. chosen anchors. These anchors can be existing nodes in the graph or dummy ones added for convenience. In the case of variation graphs, since we have paths, we include dummy source and sink nodes to serve as the starting and ending points for all paths, respectively.

Definition 4.16 [Relative distance representations of nodes in a variation graph] Given a Variation Graph $G = \langle V, A, W \rangle$ with discrete manifold variation graph $\mathcal{M}(G) = (X, d)$ (see Definition 4.14). We include two auxiliary elements to X, namely source s, and sink t nodes, such that all paths in W start in s, and all end in t. We extend the metric d with $d(s, u) = 0$, for all u starting node of a path in W, and $d(v, t) = 0$, for all v ending node of a path in W. Then, for each node x in V, we define its relative representation as follows,

$$\vec{x} = [d(s, x), d(x, t)] \tag{4.13}$$

Our main goal is to align variation graphs, to do so, we model them as discrete manifolds (see Definition 4.14). To find ϕ for the problem defined in Definition 4.15, we solve a matching problem between relative representations as in [18]. A key element of this approach is the use of the source and sink nodes (see Definition 4.16). When aligning two discrete manifolds (X_1, d_1) and (X_2, d_2), with source nodes $s_1 \in X_1, s_2 \in X_2$, and sink nodes $t_1 \in X_1$ and $t_2 \in X_2$, we assume that these nodes correspond under the alignment ϕ. Specifically, the source nodes are aligned such that $\phi(s_1) = s_2$, and the sink nodes are aligned such that $\phi(t_1) = t_2$.

4.3.3.1. On variation graphs with the same number of nodes

We start by considering two manifolds with the same number of elements. Given $\{\vec{u}_1, \cdots, \vec{u}_n\}$ and $\{\vec{v}_1, \cdots, \vec{v}_n\}$ relatives representations of the elements of two discrete manifold variation graphs, $\mathcal{M}(G_1) = (X_1, d_1)$ and $\mathcal{M}(G_2) = (X_2, d_2)$, respectively.

We define the binary decision variable

$$x_{ij} = \begin{cases} 1, & \text{if } \vec{u}_i \text{ is aligned with } \vec{v}_j \\ 0, & \sim \end{cases} \tag{4.14}$$

The cost of aligning the relative representations $\vec{u}_i$ and $\vec{v}_j$ is defined by the Euclidean distance between them.

$$c_{ij} = \|\vec{u}_i - \vec{v}_j\|_2 \tag{4.15}$$

The goal is to minimize the global alignment cost between each pair of elements, *i.e.*

$$\begin{aligned} \text{minimize} \quad & \sum_{j=1}^{n} c_{ij} x_{ij} \\ \text{subject to} \quad & \sum_{i=1}^{n} x_{ij} = 1, j = 1, \dots, n \\ & \sum_{j=1}^{n} x_{ij} = 1, i = 1, \dots, n \\ & \qquad\qquad x_{ij} \in \{0, 1\} \end{aligned} \tag{4.16}$$

The solution of this problem is a bijection $\phi : [1, n] \mapsto [1, n]$, such that $\phi(i) = j$ if and only if $\vec{u}_i$ match with $\vec{v}_j$.

We claim that considering relative representations as defined in Definition 4.16 suffice to check if two manifold variation graphs are isomorphic.

The following lemma shows that when the former alignment problem has an optimal cost alignment value of 0, these relative representations allow us to prove that the metrics of the aligned manifolds are the same.

Lemma 4.1 *Given $\mathcal{M}(G_1) = (X_1, d_1)$ and $\mathcal{M}(G_2) = (X_2, d_2)$ discrete manifolds variation graph with $|X_1|= |X_2|$. If there exists a bijective mapping $\phi : [1, n] \mapsto [1, n]$, such that $\|\vec{u}_i - \vec{v}_{\phi(i)}\|_2 = 0, \forall i \in [1, n]$, where $\vec{u}_i$ and $\vec{v}_{\phi(i)}$ are the relative representations of the elements $u_i \in X_1$ and $v_{\phi(i)} \in X_2$, respectively. Then $\forall u_i, u_j \in X_1$, $d_1(u_i, u_j) = d_2(v_{\phi(i)}, v_{\phi(j)})$*

Proof: Given $\vec{u}_i, \vec{u}_j$ relatives representations of u_i and u_j in X_1, and $\vec{v}_{\phi(i)}, \vec{v}_{\phi(j)}$ relative representations of $v_{\phi(i)}$ and $v_{\phi(j)}$ in X_2. Let us suppose that,

$$d_1(u_i, u_j) \neq d_2(v_{\phi(i)}, v_{\phi(j)}) \tag{4.17}$$

Notice that the bijection ϕ over the relative representation induces a bijection $\psi : X_1 \mapsto X_2$, where $\psi(u_k) = v_{\phi(k)}$, and which we extend with $s \mapsto \psi(s)$ and $t \mapsto \psi(t)$, for source and sink nodes in X_1 (s and t) and X_2 ($\psi(s)$ and $\psi(t)$). Then Eq. 4.17 can be rewritten as,

$$d_1(u_i, u_j) \neq d_2(\psi(u_i)), \psi(u_j)) \tag{4.18}$$

Since $\|\vec{u}_i - \vec{v}_{\phi(i)}\|_2 = 0$, using the definition of the relative representations in Equation 4.13 and ψ, we obtain that

$$\begin{cases} d_1(s, u_i) = d_2(\psi(s), \psi(u_i)) \\ d_1(u_i, t) = d_2(\psi(u_i), \psi(t)) \end{cases} \tag{4.19}$$

analogously, since $\|\vec{u}_j - \vec{v}_{\phi(j)}\|_2 = 0$,

$$\begin{cases} d_1(s, u_j) = d_2(\psi(s), \psi(u_j)) \\ d_1(u_j, t) = d_2(\psi(u_j), \psi(t)) \end{cases} \tag{4.20}$$

Let us consider the closed path in $\mathcal{M}(G_1)$,

$$p_1 : s, u_i, t, u_j, s$$

and the closed path in $\mathcal{M}(G_2)$

$$p_1^{'} : \psi(s), \psi(u_i), \psi(t), \psi(u_j), \psi(s).$$

The length of p_1 is given by,

$$L_1 = d_1(s, u_i) + d_1(u_i, t) + d_1(t, u_j) + d_1(u_j, s) \tag{4.21}$$

and the length of $p_1^{'}$ by,

$$L_1^{'} = d_2(\psi(s), \psi(u_i)) + d_2(\psi(u_i), \psi(t)) + d_2(\psi(t), \psi(u_j)) + d_2(\psi(u_j), \psi(s)) \tag{4.22}$$

L_1 can also be computed by summing up the lengths of the closed paths $p_2 : s, u_i, u_j, s$ and $p_3 : u_i, t, u_j, u_i$, and subtracting twice the length of u_i, u_j. Let us define by L_2 and L_3, the lengths of p_2 and p_3, respectively. Then, we can equivalently express L_1 as follows,

$$L_1 = L_2 + L_3 - 2d_1(u_i, u_j) \tag{4.23}$$

Analogously, $L_1^{'}$ can be computed by

$$L_1^{'} = L_2^{'} + L_3^{'} - 2d_2(\psi(u_i), \psi(u_j)) \tag{4.24}$$

where $L_2^{'}$ and $L_3^{'}$ is the length of the closed path $p_2^{'} : \psi(s), \psi(u_i), \psi(u_j), \psi(s)$ and $p_3^{'} : \psi(u_i), \psi(t), \psi(u_j), \psi(u_i)$, respectively.

By Equations 4.19 and 4.20, $L_1 = L'_1$, $L_2 = L'_2$ and $L_3 = L'_3$, then from Equations 4.23 and 4.24,

$$d_1(u_i, u_j) = d_2(\psi(u_i), \psi(u_j)) \tag{4.25}$$

which is a contradiction with our assumption in Eq. 4.18. □

Notice that Lemma 4.1 is merely geometric and the relative representations play a crucial role in determining that the metrics of the two manifolds are the same, where knowing the correspondence for the source and sink nodes are fundamental in the alignment. Another observation is that since the metrics of the manifolds are the same, we also have a correspondence on the weights between the edges.

Recall that for each element in a manifold, we have a probability distribution. In the following Lemma we show that two non-isomorphic manifold variation graphs cannot have the same metric.

Lemma 4.2 *If $\mathcal{M}(G_1) = (X_1, d_1)$ and $\mathcal{M}(G_2) = (X_2, d_2)$ are two non-isomorphic manifold variation graphs (with curvature $\kappa = 0$), then $d_1 \neq d_2$.*

Proof: Let us suppose that $|X_1| = |X_2|$, with d_1 and d_2 defined over the same number of edges, with isomorphic underlying graphs G_1 and G_2 (without considering distributions over nodes), respectively. Consider the bijective mapping $\psi : X_1 \mapsto X_2$ such that (u, v) arc in G_1 if and only if $(\psi(u), \psi(v))$ arc in G_2.

Since $\mathcal{M}(G_1)$ and $\mathcal{M}(G_2)$ are non-isomorphic, let us assume that there exist $x_1 \in X_1$ and $x_2 \in X_2$ such that their probability distributions, f_1 and f_2, respectively, are different, *i.e.* $W(f_1, f_2) > 0$ (see Definition 4.1).

Consider $y \in \pi(x_1)$ and $\psi(y) \in \pi(x_2)$, with $x_2 = \psi(x_1)$, where the distributions defined over y and $\psi(y)$ are equal, *i.e.* $f_y = f_{\psi(y)}$, and suppose that $d_1 = d_2$. In both cases, $\kappa_{(x_1,y)} = 0$

and $\kappa_{(x_2,\psi(y))} = 0$,

$$1 - \frac{W(f_1, f_y)}{d_1(x_1, y)} = 1 - \frac{W(f_2, f_{\psi(y)})}{d_2(\psi(x_1), \psi(y))} \tag{4.26}$$

since $d_1 = d_2$, we obtain

$$W(f_1, f_y) = W(f_2, f_{\psi(y)}) \tag{4.27}$$

let us define by T^1 the optimal transport plan between f_1 and f_y, and by T^2 the one between f_2 and f_y $(= f_{\psi(y)})$, by Equation 4.27

$$T^1 = T^2 \tag{4.28}$$

given $f_1 = (a_1^1, \cdots, a_n^1)$ and $f_2 = (a_1^2, \cdots, a_n^2)$, since $f_1 \neq f_2$, exists $l \in \{1, \cdots, n\}$ such that $a_l^1 \neq a_l^2$, then

$$\sum_{j=1}^{n} T_{lj}^1 \neq \sum_{j=1}^{n} T_{lj}^2 \tag{4.29}$$

which is a contradiction with Equation 4.28. $\square$

A take-home message of the above proof is that the choice of node distribution is crucial. For example, if we have a linear graph with a single bubble (two nodes) in the middle, such that two possible walks between the source and sink nodes are possible, each one traversed by 2 paths, when considering the probability measure with weights given by the node depth Equation 4.9, the distributions of both nodes in the bubble are the same, hence their Wasserstein distance is 0. This might lead to the wrong alignment between two variation graphs with the same underlying structure w.r.t. nodes and paths, since their relative represenations might be the same if node labels are not considered. Hence, is crucial to define a proper definition of probability distribution for nodes that can capture all the relevant information. We conjecture that including k-mer counts might avoid the described situation. Equation 4.11 might be a way to properly model the information of paths and labels. In this case, the distributions of nodes with different labels, traversed by the same number of paths, might result to be different (because the set of subpaths traversing each node differs in at least one k-mer), but we have no proof yet about this.

One way to bypass this situation is to modify the formulation of Equation 4.16 and include node labels as part of the cost, as follows.

$$c_{ij} = \|\vec{u}_i - \vec{v}_j\|_2 + \Delta(\lambda(u_i), \lambda(v_j)) \tag{4.30}$$

where, $\lambda(u_i)$ and $\lambda(v_j)$ are the labels of nodes u_i and v_j, respectively.

$$\Delta(a, b) = \begin{cases} \delta, & \text{if } a = b \\ 0, & \text{otherwise.} \end{cases} \tag{4.31}$$

and $\delta > 0$ is the penalization cost.

4.3.3.2. On the isomorphism between variation graphs

The main result in [81] is the existence and uniqueness theorem for solutions to a continuous-time normalized Ricci-Flow. This means that given a manifold (X, d), after applying this version of the Ricci-Flow the updated metric d will always be the same (which theoretically is not guaranteed for other versions of the discrete Ricci-Flow).

In this scenario, given two discrete manifold variation graphs, there will exist a bijective mapping between their corresponding relative representations satisfying Lemma 4.1. Since their metrics are the same, by Lemma 4.2, we conclude that the manifolds are isomorphic.

The last ingredient to find the correct mapping between the nodes is to solve the alignment problem in Equation 4.16. To ensure that the assignment is correct (*i.e.* to avoid nodes with the same relative representation and different labels being assigned incorrectly) we can use the costs from Equation 4.30.

4.3.3.3. On variation graphs with different number of nodes

The formulation of aligning two pangenome graphs with different numbers of nodes requires several considerations, in particular the size of the graphs, which should lead to the definitions of global (graph-vs-graph) and local (subgraph-vs-graph) alignments. Regarding the approach described in the previous section, when comparing graphs with a substantial difference in the number of nodes, the current definition of relative representation (see Definition 4.16) might not capture relevant information for the comparison of the resulting manifolds.

Here we relax the formulation given in Equation 4.16 for a global alignment and consider the same relative representations given in Definition 4.16.

Given $\{\vec{u}_1, \cdots, \vec{u}_m\}$ and $\{\vec{v}_1, \cdots, \vec{v}_n\}$ relatives representations of the elements of two discrete manifold variation graphs ($m < n$), $\mathcal{M}(G_1) = (X_1, d_1)$ and $\mathcal{M}(G_2) = (X_2, d_2)$, respectively.

We define the binary decision variable

$$x_{ij} = \begin{cases} 1, & \text{if } \vec{u}_i \text{ is aligned with } \vec{v}_j \\ 0, & \sim \end{cases} \tag{4.32}$$

with the same cost alignment as in Equation 4.15.

The goal is to minimize the alignment cost, *i.e.*

$$\begin{aligned} \text{minimize} \quad & \sum_{j=1}^{n} c_{ij} x_{ij} \\ \text{subject to} \quad & \sum_{i=1}^{n} x_{ij} = 1, j = 1, \ldots, m \\ & \sum_{j=1}^{n} x_{ij} \leq 1, i = 1, \ldots, n \\ & \qquad\qquad x_{ij} \in \{0, 1\} \end{aligned} \tag{4.33}$$

The solution of this problem is a partial injection $\phi : [1, m] \mapsto [1, n]$, such that $\phi(i) = j$ if and only if $\vec{u}_i$ is aligned with $\vec{v}_j$.

We refer to the function ϕ as *the alignment* between $\mathcal{M}(G_1)$ and $\mathcal{M}(G_2)$ (or G_1 and G_2).

Even though the same formulation could be considered for the local alignment, we believe it requires other considerations regarding the relative representation defined in Equation 4.13, Since these representations involve introducing dummy source and sink nodes, they may vary considerably between graphs of different sizes, subsequently affecting the alignment costs. Addressing this is considered a direction for future work.

4.4. Discussions

In this work, we introduce a novel framework that models variation graphs as discrete manifolds, utilizing the principles of discrete Ollivier Ricci-Flow, particularly by defining probability distributions over nodes based on paths and labels. We believe that this approach paves the way for exploring the underlying geometry of variation graphs in a structured, mathematically rigorous way, through the lens of the curvature.

We offer both a novel theoretical viewpointfor the alignment and analysis of variation graphs. Future work could expand on these foundations by exploring additional discrete manifold techniques and notions of curvature, potentially enhancing the precision and scalability of variation graph analyses in genomics and other fields.

We propose an Integer Linear Programming (ILP) approach to model alignment tasks between variation graphs when modeled as discrete manifolds, by defining relative node representations based on weights derived from applying discrete Ricci-Flow, our ILP model provides a practical method for graph alignment, enabling theoretically grounded alignment processes.

Some theoretical work must be done on the analysis and definition of probability distributions for elements in a manifold. The goal is to ensure that the relative representations for the nodes in a variation graph are unique for each element. This would allow us to solve the alignment problem entirely based on the relative representations, instead of considering additional information to penalize decision variables in the ILP formulation.

Chapter 5

Fast and accurate clade assignment of Sars-CoV-2 sequences via Deep Learning

Since the beginning of the COVID-19 pandemic, there has been an explosion of sequencing of the SARS-CoV-2 virus, making it the most widely sequenced virus in history. Several databases and tools have been created to keep track of genome sequences and variants of the virus, most notably the GISAID platform hosts millions of complete genome sequences, and it is continuously expanding every day. A challenging task is the development of fast and accurate tools that can distinguish between the different SARS-CoV-2 variants and assign them to a clade, *i.e.* species evolutionarily related to a common ancestor.

In this chapter, we leverage the Frequency Chaos Game Representation (FCGR) and Convolutional Neural Networks (CNNs) to develop an original method that learns how to classify genome sequences that we implement into `CouGaR`, a tool for the clade assignment problem on SARS-CoV-2 sequences. On a testing subset of the GISAID, `CouGaR` achieves an 96.29% overall accuracy, while a similar tool, `Covidex`, obtained a $77,12\%$ overall accuracy. As far as we know, our method is the first to use Deep Learning and FCGR for intra-species classification. Furthermore, by using some feature importance methods `CouGaR` allows to identify k-mers that match SARS-CoV-2 marker variants.

`CouGaR` is open source and available at https://github.com/AlgoLab/CouGaR-g.

Our contribution

- By combining FCGR and CNNs, we develop a method that achieves a better accuracy than Covidex (which is based on Random Forest) for clade assignment of SARS-CoV-2 genome sequences, thanks to our training on a much larger dataset, with comparable running times.

- Our method implemented in CouGaR-g is able to detect k-mers that capture relevant biological information that distinguishes the clades, known as marker variants.

5.1. Introduction

The global coordination in combating the COVID-19 pandemic has led to the sequencing of one of the largest amounts of viral genomic data ever produced. All this data is stored in publicly available archives, such as the European Nucleotide Archive (ENA) and GISAID [82], currently having more than 20 million sequenced genomes, classified in *variants*, *clades*, and *lineages*.

The SARS-CoV-2 virus has evolved since its discovery, and the currently available phylogenies describing its evolutionary history [83] show more than 2000 different genomes, divided into lineages.

Since the phylogeny is fairly stable and the main (existing) *lineages*, i.e. the lines of descent, have been identified, a natural and interesting problem is to quickly find, given a sequence, the *clade* to which it belongs, i.e. a group of descendants sharing a common ancestor [84]. Fast and efficient solutions to the clade assignment problem would help in tracking current and evolving strains and it is crucial for the surveillance of the pathogen.

This classification problem has been attacked with machine learning approaches [85–87] using the Spike protein amino acid sequence to drive the classification step, and using k-mer (substrings of length k) frequencies from complete genomes [88].

Subtyping Sars-Cov-2 sequences has been addressed in the literature with bioinformatics pipelines that require the alignment to a reference genome [83] [89], and also with machine learning approaches aiming to skip the alignment step [90]. Furthermore, an early approach to construct phylogenetic trees within SARS-CoV-2 strains and closely related species was proposed in [91] using FCGR as embedding for a Hierarchical Agglomerative Clustering. Similarly, FCGR was explored along with other techniques as embedding for the identification of homologies between different known and emerging viruses in [92].

5.2. Preliminaries

Let us define by $\Sigma = \{A, C, G, T\}$ the DNA alphabet, and the extended DNA alphabet by $\Sigma_N = \{A, C, G, T, N\}$, where N indicates a null call. We are given a phylogeny over the possible viral strains, partitioned into classes: each class c of such partition $\mathcal{L}$ is a *clade* of the tree, where a clade is a group of related organisms descended from a common ancestor.

Consider a collection S of complete genome assemblies over the alphabet Σ_N, and a set of $\mathcal{L}$ clades. We assume that there exists a labeling function $\lambda : \mathcal{S} \mapsto \mathcal{L}$ that assigns a clade in $\mathcal{L}$ to each assembly $s \in \mathcal{S}$. The problem asks for a mapping $f : S \mapsto \mathcal{L}$ that assigns each string to its clade.

To solve this problem, we propose a supervised learning model based on Convolutional Neural Networks (CNN) [93], using FCGR as inputs.

5.3. Methods

We use the k-th order FCGR representation for each sequence. In order to obtain this representation, we need to count the k-mers in each sequence and to put those frequencies in the FCGR based on the CGR encoding.

Before feeding the FCGR to the model, we rescale its elements to values between 0 and 1 for stability of the learning process. To do so, we divide each FCGR element-wise by the maximum value in the FCGR. It is worth mentioning that other preprocessing steps were taken into consideration but were ultimately excluded because found empirically worse.

Due to the huge amount of data available for most of the clades, we undersampled at most 20.000 sequences per clade to perform our experiments, nevertheless, for some clades only a portion of it was available (see table representativity). In order to overcome the unbalance in the undersampled dataset, we decided to use a weighted binary cross-entropy loss function (instead of oversampling the underrepresented classes), where the cost associated with a class c is inversely proportional to its representativity in the training set.

5.3.1. Model architecture

We choose a residual neural network, ResNet50 [94] as our CNN, adapted for k-th order FCGR, i.e. with input size equal to $(2^k \times 2^k \times 1)$, and output size equal to the number of clades: $|\mathcal{L}|$, with softmax activation function in the last layer and categorical crossentropy as loss function, since we want to assign only one clade to each DNA sequence.

5.3.2. Model training

Each model was set to be trained for 50 epochs with a batch size of 32 using Adam optimizer [95] with learning rate 0.001 (the default parameters in keras). The validation loss was monitored after each epoch to save the best trained weights, reducing the learning rate with a patience of 8 epochs and a factor of 0.1, and by an early stopping in case the metrics do not improve after 12 epochs.

5.3.3. Model evaluation

To assess the performance of our trained model, we perform a classification evaluation of the predictions and also a clustering evaluation for the embeddings in order to evaluate the class separability. The reported metrics are based on a Repeated 5-fold Cross Validation.

We report global (accuracy and Matthews correlation coefficient) and class-specific metrics (precision, recall, and F1-score) for the trained models.

5.3.4. Identification of relevant k-mers

After we train a model, we can perform feature importance methods to analyze the impact of each element of the FCGR in our prediction. We selected Saliency Maps [55] and Shap Values [96]. Saliency Maps calculate the gradient of the loss function for a specific desired class concerning the input (FCGR) elements, the gradients are rescaled to $[0,1]$, where elements with values closer to 1 represent the more influential features for the input FCGR over the predicted class. Shap (Shapley Additive Explanations) Values is a game theoretic approach to explain the output of any machine learning model. It aims to explain the influence of each feature compared to the average model output over the dataset the model was trained on, it outputs positive and negative values, where positive values push the prediction higher, and negative values push the prediction lower. Using the most relevant features from both methods over the FCGR, we aim to identify the most relevant k-mers for the classification of each clade.

5.3.5. Data Selection

The dataset for this experiment was downloaded from GISAID. By the time of our access to GISAID [1] there were around 10 million sequences.

To undersample the available data, we first dropped all the rows in the metadata without information in the columns `Virus name`, `Collection Date`, `Submission Date`, `clade`, `Host` and `Is complete?`, then we built a `fasta_id` identifier from the metadata as a concatenation of the columns `Virus name`, `Collection Date` and `Submission Date`.

[1] April 04, 2022. https://www.gisaid.org/

For each clade, we randomly selected $20,000$ sequences considering only those rows where the `Host` column has value "Human" — clades L, V, and S have less than $20,000$ sequences available, in these cases all sequences have been selected.

As a result of the above procedure, we obtained $191,456$ sequences among the 11 GISAID clades (S, L, G, V, GR, GH, GV, GK, GRY, O, and GRA) over the 12 available, we excluded the clade GKA from our study since there were only 81 sequences reported in the metadata. The undersampled dataset was randomly split into train, validation and test sets in $80:10:10$ proportion, preserving the same proportion of clades (labels) in each set. The distribution of the clades over the datasets is given in Table 5.1

Clade	Train	Val	Test	Total	Available
S	14,298	1,788	1,788	17,874	17,874
L	5,154	644	644	6,442	6,442
G	15,999	2,000	2,000	20,000	408,552
V	5,713	714	714	7,141	7,141
GR	16,000	2,000	2,000	20,000	625,662
GH	16,000	2,000	2,000	20,000	547,792
GV	16,000	2,000	2,000	20,000	182,248
GK	16,000	2,000	2,000	20,000	4,170,758
GRY	16,000	2,000	2,000	20,000	944,876
O	16,000	2,000	2,000	20,000	55,400
GRA	16,000	2,000	2,000	20,000	2,833,863
	153,164	19,146	19,146	191,456	9,800,608

Table 5.1: Distribution of the number of sequences selected for train, validation and test sets by each clade. The final dataset for the 11 clades was split in a $80:10:10$ proportion for train, validation, and test sets.

5.4. Results

In this section, we present the experimental setup, clustering, and classification metrics. We train one model for each $k \in \{6,7,8\}$ and we complement the study of the accuracy of each model (compared against Covidex [88]) with an analysis of the most relevant k-mers for the classification of each clade using Saliency Maps and Shap.

For this experiment we choose $k \in \{6, 7, 8\}$ and sequences from 11 GISAID clades: S, L, G, V, GR, GH, GV, GK, GRY, O, and GRA.

5.4.1. Model training

We show the accuracy (average of the 5 Repeated fold cross validation) of the train and validation sets for $k = 8$ in Figure 5.1. For $k = 6$ and $k = 7$ the training is more unstable for the first epochs, but it behaves similar to $k = 8$ in the later epochs, i.e. training and validation metrics are similar.

The architecture used in this experiment is the same for all k (ResNet50 [94]), we only changed the input size. Originally, this architecture was designed for inputs of size $(224 \times 224 \times 3)$, which led us to the assumption that this architecture could be more suitable for $k = 8$. Notice that our sequences are $\approx 29,000bp$ long, which means that our input FCGR for $k = 8$ is very sparse, since from an n-long sequence we can count $n - k + 1$ k-mers, it means that (in the case where all k-mers are different) we have at most $29,000$ k-mers, at least 55% of the elements of the FCGR are 0 for $k = 8$. In Table 5.2 a comparison of the number of features for each k and the training time per epoch in our experiments is detailed.

k-mer	Dimensions	Features	Size (GiB)	Time per epoch (min)
6	(64,64)	4,096	6.6	4:05
7	(128,128)	16,384	24.1	8:21
8	(256,256)	65,536	94.2	24:50

Table 5.2: For each k, the dimension of the FCGR, its number of features (4^k), the amount of memory required to store the selected dataset of $191,456$ sequences as FCGR, and the average training time per epoch are reported in the table. The number of features and the space increase exponentially w.r.t k.

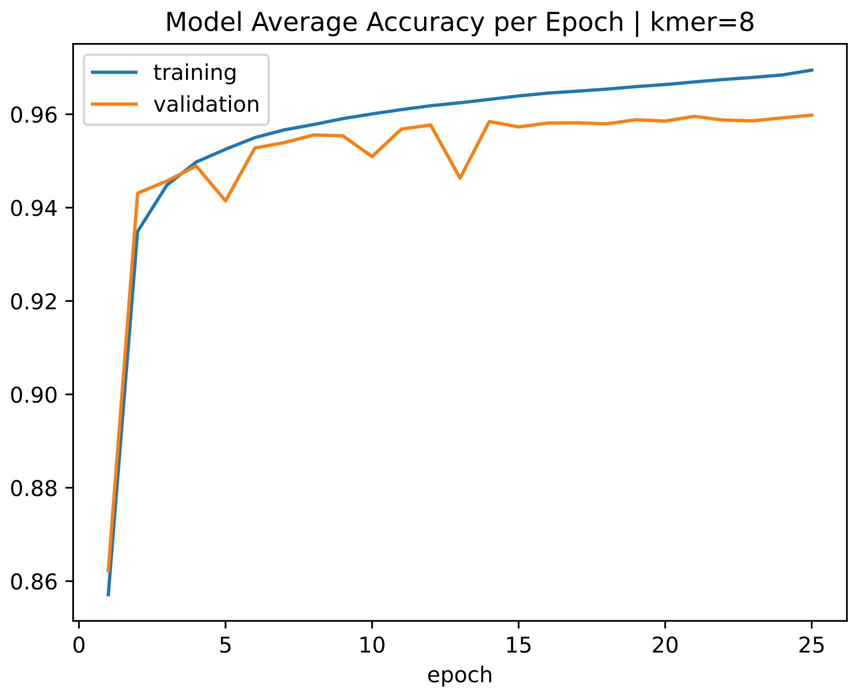


Figure 5.1: Average accuracy in the training and validation sets for our model with $k = 8$. The best model (final weights) is set as the one with the lowest validation loss, achieved at epochs 24 ± 5 for $k = 8$ (from a 5 RepeatedFold cross validation process). All models were trained for 50 epochs using an early stopping of 12 epochs based on the validation loss (hence, not all of them ran for 50 epochs).

5.4.2. Classification results

After each model is trained the precision and recall for the test set are computed for each clade using the best trained weights (lowest loss in the validation set), achieved at epochs 24 ± 5, 27 ± 8, and 20 ± 3 for $k = 6$, $k = 7$, and $k = 8$, respectively. In our case, we assign each sequence to the clade with highest score. Precision, recall and F1-score are shown in Table 5.4.

Precision and recall are very similar among all the trained models, with small improvements when k increases, 5 out of 11 clades have F1-score greater than 99% in our best model ($k = 8$). Most notable differences in the performance can be seen in clades GR and GRY, which present the lowest (and under 90%)reported recall and precision in each model, respectively. Moreover, from the confusion matrices (see Fig. 5.2) we can see that misclassified sequences that belong to clades GR and GRY, are confused between them, this can be explained since clade GRY is originated from clade GR. For the other clades, most of the misclassified sequences are predicted as (or belong to) clade G, that is the former one. Clade O exhibits the second lowest recall, where the misclassified sequences are assigned predominantly to clades G, GH, GK, and GRY.

k-mer	Accuracy	MCC
6	0.953714 ± 0.001589	0.948792 ± 0.001813
7	0.959856 ± 0.001740	0.955566 ± 0.001910
8	**0.962175 ± 0.002829**	**0.958211 ± 0.003141**

Table 5.3: Accuracy and Matthews Correlation Coefficient in the set for each of our models. Each metric ($\mu \pm \sigma$) is reported by its average (μ) and standard deviation (σ) from a 5 RepeatedFold cross validation process. For both, accuracy and MCC the model increases with the value of k. Going from k=6 to k=8 increases accuracy in 0.84%, and MCC in 0.94%. In **bold** the highest value of each metric.

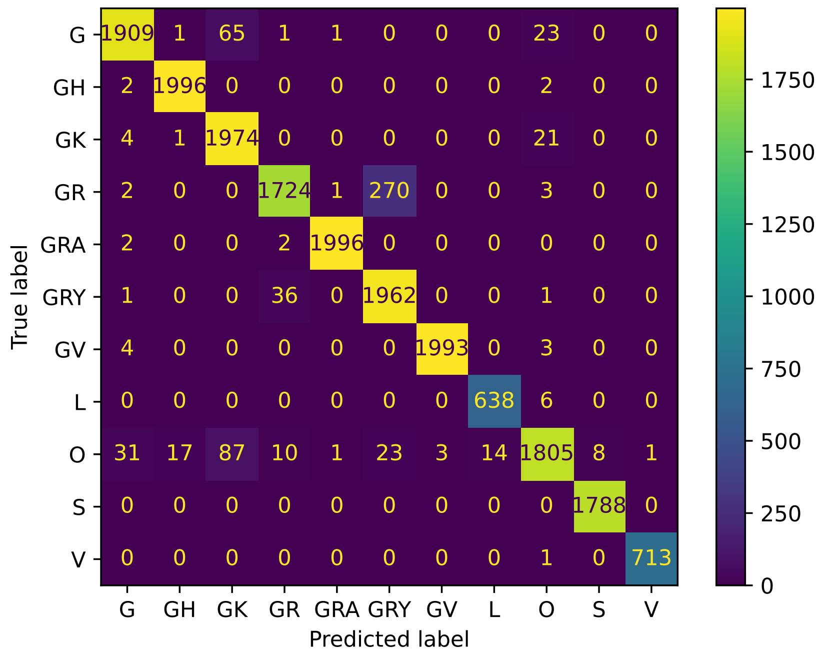


Figure 5.2: Confusion matrix for the test set for one of the trained models with $k = 8$ (from a 5 RepeatedFold cross validation process). All the models are able to correctly classify more than 98% of the sequences for all clades except for G, GR, GRY, and O. Most of the incorrectly classified sequences of GR and GRY are confused between them, which makes sense since they are evolutionarily related. For the G clade, the incorrectly classified sequences are shared between clades GK and O. For the clade O, the incorrectly classified sequences are predominantly assigned to clades G, GH, GK, and GRY.

k-mer	6			7			8		
Clade	Precision	Recall	F1score	Precision	Recall	F1score	Precision	Recall	F1score
S	99.4 ± 0.2	99.6 ± 0.2	99.5 ± 0.1	99.7 ± 0.1	99.6 ± 0.1	99.7 ± 0.1	99.8 ± 0.2	99.7 ± 0.3	**99.8 ± 0.1**
L	98.3 ± 0.4	97.7 ± 0.6	98.0 ± 0.3	98.7 ± 0.4	99.0 ± 0.2	**98.9 ± 0.2**	98.0 ± 0.4	99.5 ± 0.3	98.7 ± 0.2
G	95.8 ± 0.7	94.7 ± 0.8	95.2 ± 0.4	97.1 ± 0.4	95.1 ± 0.4	96.1 ± 0.3	97.3 ± 0.8	95.9 ± 0.7	**96.6 ± 0.1**
V	99.1 ± 0.4	99.2 ± 0.4	99.1 ± 0.2	99.5 ± 0.5	99.4 ± 0.3	99.4 ± 0.3	99.6 ± 0.5	99.6 ± 0.2	**99.6 ± 0.3**
GR	91.7 ± 2.0	85.9 ± 1.7	88.7 ± 0.4	92.4 ± 1.3	87.5 ± 1.4	89.8 ± 0.2	93.9 ± 2.3	86.6 ± 1.7	**90.1 ± 0.7**
GH	98.6 ± 0.3	99.5 ± 0.1	99.0 ± 0.2	98.9 ± 0.2	99.7 ± 0.1	**99.3 ± 0.1**	98.8 ± 0.2	99.8 ± 0.1	**99.3 ± 0.1**
GV	99.5 ± 0.3	99.6 ± 0.2	99.5 ± 0.1	99.7 ± 0.1	99.6 ± 0.1	99.7 ± 0.1	99.6 ± 0.1	99.8 ± 0.1	**99.7 ± 0.0**
GK	91.8 ± 0.4	97.6 ± 0.5	94.6 ± 0.2	92.2 ± 0.5	97.7 ± 0.2	94.9 ± 0.4	92.7 ± 0.2	98.1 ± 0.8	**95.3 ± 0.4**
GRY	86.4 ± 1.3	93.2 ± 2.4	89.7 ± 0.6	87.8 ± 1.1	93.9 ± 1.4	90.7 ± 0.3	87.1 ± 1.5	95.5 ± 2.0	**91.0 ± 0.7**
O	94.3 ± 0.8	86.8 ± 0.2	90.4 ± 0.4	95.0 ± 0.6	89.2 ± 0.8	92.1 ± 0.6	96.7 ± 0.4	88.8 ± 1.2	**92.6 ± 0.7**
GRA	99.7 ± 0.1	99.8 ± 0.1	99.8 ± 0.1	99.9 ± 0.1	99.7 ± 0.1	**99.8 ± 0.0**	99.8 ± 0.1	99.8 ± 0.1	99.8 ± 0.1

Table 5.4: Precision, recall, and F1-score. Each of our models is represented by length of the the k-mers used to generate the FCGR. Two clades, GR and GRY present deviations greater than 1% in their precision and recall for all values of k. In **bold** the highest F1score for each clade and k. Each metric ($\mu \pm \sigma$) is reported by its average (μ) and standard deviation (σ) from a 5 RepeatedFold cross validation process.

5.4.3. Clustering results

We evaluate the embeddings of the last layer of each trained model using the Silhouette Coefficient, Calinski-Harabasz score and Generalized Discrimination Value (GDV). These results are shown in Table 5.5. We can observe that the model for $k = 9$ is the best one among all metrics, however, all trained models exhibit a very similar separability based on Silhouette and GDV.

5.4.4. Comparison with the literature

We compare our results against Covidex [88], a tool that classifies Sars-CoV-2 sequences into three nomenclatures: GISAID, Nexstrain and Pango lineages. Using a different model for each task, all based on Random Forest and 6-mers as input. The reported accuracy are $97,77\%$, $99,52\%$ and $96,56\%$ for GISAID, Nextstrain and Pango models, respectively. They also trained the models using 7-mers, but they claim that it only produced slightly better results in terms of accuracy but with more than doubling the computation time [88].

The input for Covidex is a vector with the normalized counting of the frequencies for all 4^k k-mers. Our input, the FCGR also considers all k-mers but in a bi-dimensional matrix. The

k-mer	Silhouette	Calinski-Harabasz	GDV
6	0.939 ± 0.007	145,879.214 ± 18,132.428	-0.712 ± 0.003
7	0.948 ± 0.003	163,926.767 ± 8,638.391	-0.717 ± 0.002
8	0.948 ± 0.006	174,736.086 ± 22,554.904	-0.718 ± 0.003

Table 5.5: Clustering metrics for our trained models. Each metric is computed using the output of each model and the predicted clade (that is, the clade that achieves the highest score by our model) in the test set. Each model is represented by the length of the k-mers used to generate the FCGR. For the Silhouette score, the closest to 1 the better. For the Calinski-Harabasz score, larger values are better. For the GDV score, the closest to -1, the better. All models exhibit comparable separability of the clusters. Each metric ($\mu \pm \sigma$) is reported by the average (μ) and standard deviation (σ) in the from a 5 RepeatedFold cross validation process.

main difference between both approaches is the model behind it, while Covidex uses Random Forest to perform the classification, we take advantage of the CNNs and use a 2-dimensional input, the FCGR. Notice that using the FCGR with any other classical Machine Learning method implies to convert the FCGR into a vector, and hence, the loss of the 2-dimensional structure.

Since our model is trained using GISAID clades, we only compare to those results. In Covidex, they used 10 clades: S, L, G, V, GR, GH, GV, GK, GRY and O. In our case, we included GRA since there were enough available sequences by the time of our experiments, but this is not considered in the comparison.

For Covidex, the model for the GISAID nomenclature was trained with $66,126$ sequences and tested on $13,230$. Since Covidex is made available as an user app for any SARS-Cov2 sequence, we used the app over our test dataset to compare the results. We tested Covidex on our test dataset of $17,146$ sequences (excluding the 2000 sequences from GRA clade). Achieving a $77,12\%$ of accuracy, more than a 18% lower than all our trained models and $20,65\%$ lower than their reported accuracy. The reported precision, recall and F1-score, as well as the test results over our selected dataset can be seen in Table 5.6. We found that the reported F1-score of Covidex is quite distant for the one we obtained in our test dataset for clades L (-8.4%), G (-15.8%), GR (-42.4%), GK (-10.9%), GRY (-19.3%) and O (-28.8%), while for clades S (-0.8%), V (-2.9%), GH (-2.5%) and GV (-0.9%), we can observe a decrement on the reported F1-score ranging from 0.8% to 2.9%. Our models (see Table 5.4) exhibit better performance than Covidex in all clades and metrics on our test set, with similar results only on clades S and GV. We did not perform an extensive comparison of the running times since both tools classify a genome sequence in less than a second (on $k = 8$, our tool took 0.15 seconds in average).

	Report			Test		
Clade	Prec.	Rec.	F1score	Prec.	Rec.	F1score
S	0.998	1	0.999	**0.988**	**0.995**	**0.991**
L	0.997	1	0.999	0.859	**0.979**	0.915
G	0.993	0.984	0.989	0.811	0.852	0.831
V	1	1	1	0.958	**0.985**	**0.971**
GR	0.945	0.915	0.930	0.379	0.760	0.506
GH	0.995	0.999	0.997	0.957	**0.987**	**0.972**
GV	0.996	0.999	0.997	**0.980**	**0.995**	**0.988**
GK	0.977	0.995	0.986	0.925	0.833	0.877
GRY	0.920	0.961	0.940	0.732	0.763	0.747
O	0.994	0.959	0.976	0.722	0.658	0.688

Table 5.6: Precision, recall, and F1-score for Covidex. The Report part is taken from the Supplementary material of [88]. The Test part has the precision, recall, and F1-score obtained by Covidex on our test set, restricted to the 10 clades (17,146 sequences) analyzed in [88]. We found significant differences between Covidex and our trained models in the Test metrics (see Table 5.4). In particular, the most notorious differences w.r.t F1-score, ranging from 8.4%–42.4% are found for clades L (−8.4%), G (−15.8%), GR (−42.4%), GK (−10.9%), GRY (−19.3%) and O (−28.8%), while for clades S (−0.8%), V (−2.9%), GH (−2.5%) and GV (−0.9%), we can observe a decrement on the reported F1-score ranging from 0.8%–2.9%. Metrics in **bold** in the Test part are those which **did not decrease** more than 3% w.r.t the reported metrics.

5.4.5. Relevant k–mers for the classification of each clade

The purpose of this experiment is to study if a set of the most relevant k-mers (based on feature importance methods) are informative enough to a SVM to perform similarly than the trained CNNs (that uses FCGR as input, and hence all the 4^k possible k-mers).

Using Saliency Maps and Shap Values, we can evaluate the contribution of each element of a FCGR in the classification, for each model. From each one of these feature attribution methods we can obtain an ordered list of all k-mers. For each clade, we use the centroid FCGR of all correctly classified sequences in the test set, then we use each centroid FCGR to identify the most relevant k-mers for each clade and then train a SVM using the N most relevant k-mers (for different values of $N \in \{1, 2, 3, 4, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50\}$) and their respective frequencies as input.

The same training and test sets used for the CNNs were used for the SVM. The results of the accuracy in the test set for the different values of N are shown in Figures 5.6 and 5.5. We can observe that k-mers identified by Saliency Maps are more informative than those identified by Shap Values, since for $N = 20$, we obtain similar accuracy in the test set for $k = 6, 7, 8$ compared to CNN (96–97%), while in the case of Shap Values, this accuracy is only achieved by $k = 7$ with $N = 35$. Notice that using $N = 20$, we are considering a small number of all possible k-mers (0.49% for $k = 6$, 0.12% for $k = 7$ and 0.03% for $k = 8$).

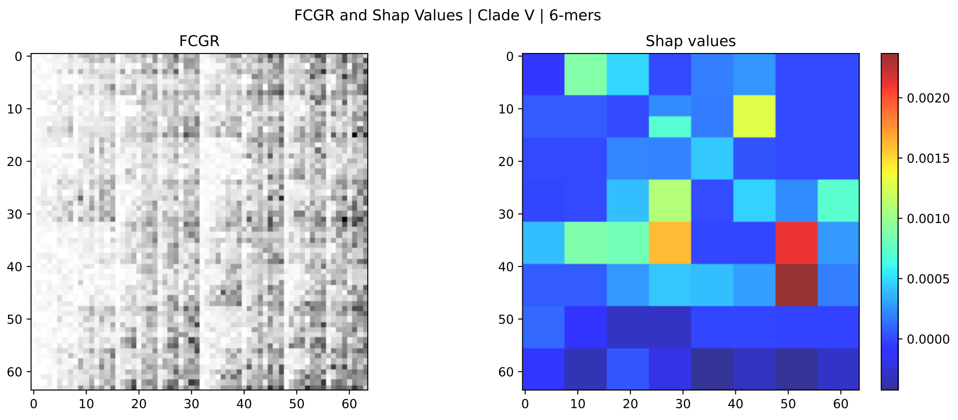


Figure 5.3: FCGR image (left) and Shap Values (right) of the centroid FCGR for the clade V ($k = 6$). The FCGR image is obtained rescaling the frequencies in the FCGR to a gray-scale range of 8 bits ([0,255]), an inversion of colors is performed to visualize higher values as black squares and lower values as white. Shap Values represent the importance of the features in the FCGR, the higher the value (red) the more important is the feature. Each feature (pixel) in the FCGR corresponds to a k-mer.

5.4.6. Matching relevant k-mers to mutations

Using the reference genome employed by GISAID (EPI_ISL_402124) [2] and the list of marker variants [3] for each GISAID clade with respect to this reference, we evaluated how

[2]https://www.gisaid.org/resources/hcov-19-reference-sequence/

[3]https://www.gisaid.org/resources/statements-clarifications/clade-and-lineage-nomenclature-aids-in-genomic-epidemiology-of-active-hcov-19-viruses/

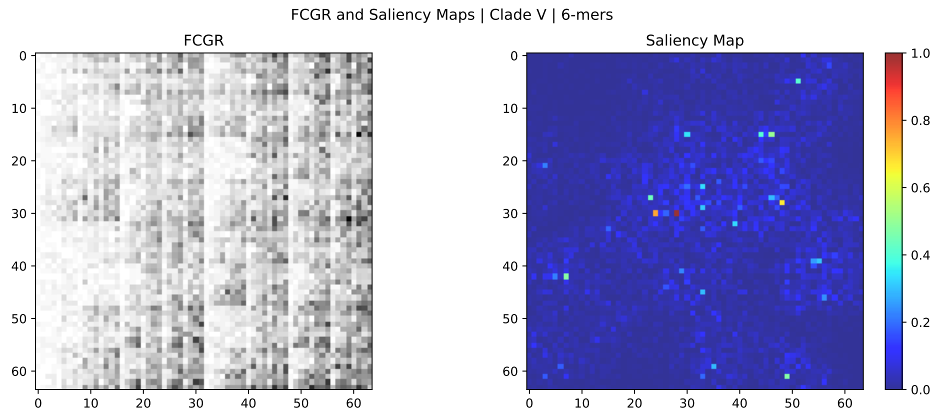


Figure 5.4: FCGR image (left) and Saliency Map (right) of the centroid FCGR for the clade V ($k = 6$). The FCGR image is obtained rescaling the frequencies in the FCGR to a gray-scale range of 8 bits ([0,255]), an inversion of colors is performed to visualize higher values as black squares and lower values as white. Saliency Map represent the importance of the features in the FCGR, the higher the value (red) the more important is the feature. Each feature (pixel) in the FCGR and Saliency Map corresponds to a k-mer.

many k-mers among the 50 chosen ones by Saliency Maps and Shap Values matched any of the reported marker variants. A summary is shown in Table 5.7.

The results shown that the most relevant k-mers selected using Saliency Maps match several of the reported marker variants (46 matches for $k = 6$, 51 for $k = 7$, and 11 for $k = 8$). On the other hand, the ones chosen by Shap Values barely match with the mutation (3 for $k = 6$), suggesting that Saliency Maps could provide a richer explainability of the model from a biological perspective.

k-mer	Saliency Maps	Shap Values
6	46	3
7	51	0
8	11	0

Table 5.7: Summary of matches between the 50 most relevant k-mers (from Saliency Maps and Shap Values) and the list of marker variants reported by GISAID for each clade. The k-mers obtained by Saliency Maps are able to match several mutations and the matches decrease when k increases, but the ones from Shap Values only reported 3 matches, for $k = 6$.

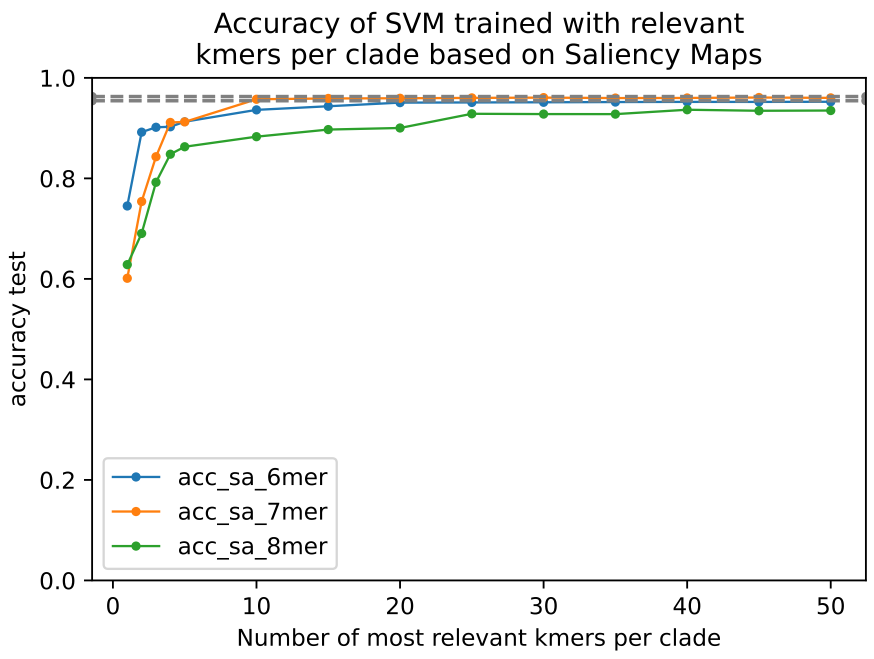


Figure 5.5: Accuracy of test set for SVM trained models using only the most N relevant k-mers for each clade ($N \in \{1, 2, 3, 4, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50\}$). The relevant k-mers are selected using Saliency Maps on the centroid of the correctly classified FCGR for each clade and model. The same train and test datasets used for the trained CNNs are used for the SVM. The SVM trained with 20 most relevant k-mers identified by Saliency Map, for $k \in \{6, 7\}$ achieves an accuracy in the test set ($\approx 96\%$) that is in the range of the minimum and maximum accuracies (see Table 5.3) obtained by our trained CNNs (the gray dashed band represents the minimum and maximum accuracy for the trained CNNs).

5.5. Discussion

In this work we have shown that FCGR can be used to classify DNA sequences. Most notably, we have used FCGR to assign SARS-CoV-2 genome sequences to its GISAID strain by running a CNN on $191,456$ genome sequences (80% training set, 10% validation set, and 10% test set). In particular, the 8-th order FCGR achieved a test accuracy of 96.22%. The majority of misclassified sequences are shared between two strongly related strains, GR and GRY (GR is a close ancestor of GRY).

We decided to exclude transfer learning from our experiments after trying this approach without success on 8-mers. For this trial we used pre-trained weights from the Imagenet

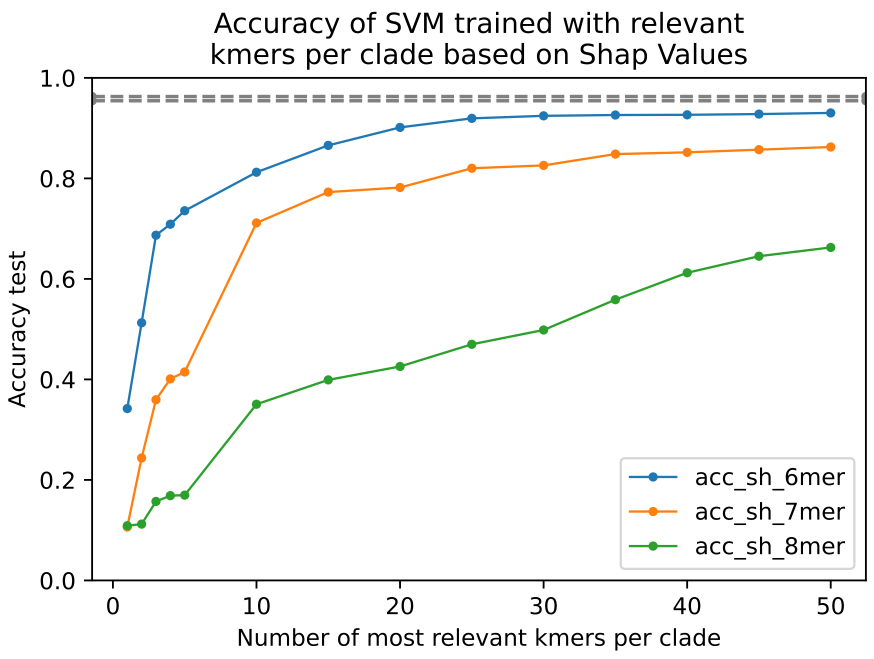


Figure 5.6: Accuracy of test set for SVM trained model using only the most N relevant k-mers for each clade ($N \in \{1, 2, 3, 4, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50\}$). The relevant k-mers are selected using Shap Values on the centroid of the correctly classified FCGR for each clade and model. The same train and test datasets used for the trained CNNs are used for the SVM. The SVM trained with the 30 most relevant (or more) 6-mers identified by Shap Values, achieves the closest accuracy (92, 44%) to the ones obtained by our trained models (see Table 5.3). When k increases, the accuracy always decreases (for the same number of relevant k-mers), which can be explained since when k increases the total number of possible k-mers increases exponentially.

dataset using ResNet50 architecture, where the backbone weights were frozen, and three dense layers were included at the top of it for the classification.

We have assessed the influence of the length k of the substrings (k-mers) used to build the FCGR, showing that values between 6 and 8 lead to very similar results, with less than 1% of difference in both accuracy and MCC on the same test set. However, when increasing the value of k, the training time for the model and the memory required to save the FCGRs increases exponentially. For $k = 6$ each epoch required 4:05 minutes and 6.6GB of memory, while for $k = 8$ it required 24:50 hour and 94.2GB. However, FCGRs show a fractal structures; this suggests that we might couple increasing k with using only a portion of the FCGR.

We compare our results with Covidex, a Random Forest based tool that classify sequences on GISAID clades based on k-mers frequencies. Under the same test set, our results show that our models outperform Covidex in all clades and reported metrics (accuracy, precision, recall and F1-score). Moreover, we found that the reported precision, recall and F1-score of Covidex are quite different for all clades but S and GV in our test set, exhibiting a decreasing in the F1-score metric up to 42.4%.

We have used Saliency Maps and Shap to identify relevant k-mers, looking for matches with the marker variants reported for each strain. Using the k-mers obtained by Saliency Map, we found 46, 51, and 11 matches for $k = 6, 7, 8$, respectively. While, for the k-mers identified by Shap, only 3 matches were found for $k = 6$. A possible direction for future works is to explore other existing methods (*e.g.* Lime [56], GradCAM [57], DeepLIFT [58]) that might be suitable in explaining the decisions of the model.

Classifying genome sequences introducing the assembly bias includes more factors to take care of, since any classification depends on the specific assembly pipeline that has been used. To lessen this possible problem, we should study a related problem, where we classify read samples instead of fully assembled genomes. This new problem is more complex, since different regions of the viral genome can have different coverage — hence impacting the frequencies — and reads needs to be cleaned from both errors and contamination artifacts.

We did not perform an extensive comparison of the running times since both tools classify a genome sequence in less than a second.

5.6. Availability of data and materials

The list of FASTA sequences and metadata used for the SARS-CoV-2 experiment can be downloaded from gisaid.org after creating an account and accepting the *Terms of Use.* The data used in this study was downloaded on April 4, 2022. Trained models and results of our experiments can be downloaded from https://zenodo.org/record/7185290g.

Chapter 6

Taxonomic aware querying and indexing of bacterial pangenomes via Deep Learning

In bacteria, species identification is of great importance in areas such as agriculture, food processing, and healthcare. The continually growing genomics databases, especially with the increased focus on investigating new bacterial genomes in clinical microbiology, have exceeded the capabilities of conventional tools for basic search and query procedures like BLAST. In addition, more complex tasks are needed, such as rapid identification and classification of species from draft assemblies. In particular, these two tasks face two common challenges: species mislabeling and outliers, i.e. cases where the assembly contains sequences that do not fit well within any known taxonomy category.

We propose `PANSPACE`, a framework for indexing genomic sequences in a continuous n-dimensional space, referred to as *Embedding space*. The FCGRs are used as feature representations of the sequences and then mapped to the Embedding space ($n \ll 4^k$) by leveraging a Convolutional Neural Network (CNN) designed to exploit the k-mer ordering in the FCGR with metric learning for clustering based on species labels, and optionally the use of Autoencoders when no labels are available. The embedding representations are indexed with FAISS [97], an index for high-dimensional vectors, proven to scale to 1 billion embeddings. Embeddings allow the application of computational methods for data curation: the detection of outliers, and the evaluation of correct labels.

`PANSPACE` is open source and available at https://github.com/pg-space/panspace.

Our contribution

- We describe a method for rapidly identifying the species associated with an isolated (draft) assembly, using a database of labeled assemblies for species identification.
- We propose a carefully designed Convolutional Neural Network that exploits the FCGR representation and utilizes metric learning to learn compressed vector representations of (draft) assemblies in an n-dimensional space.
- We use Confident Learning to identify outliers and mislabeled assemblies, facilitating database curation without manual intervention. Autoencoders can optionally handle cases without known labels in the detection of outliers.
- We index the largest publicly available database of bacterial draft assemblies [20], using approximately 1 GB of disk space.
- Our experiments reveal a strong correlation between ANI distances of (draft) assemblies and the Euclidean distances of their embeddings.

6.1. Introduction

Classifying DNA in terms of the species from which it was extracted is one of the most fundamental tasks in biology. In addition to its obvious relevance, the prominence of the task is documented by the fact that BLAST [21] is still one of the most popular, if not the best known bioinformatics tools developed so far. The principle of BLAST is to align sequenced DNA against databases storing large collections of annotated genomes. This principle is well-justified because alignment scores positively correlate with the average nucleotide identity (ANI) of the two matched sequences.

For example, it was shown that an ANI of 95-96% or greater corresponds to DNA-DNA hybridization (DDH) of 70% or greater, where DDH of 70% or greater is the ultimate indication that the two sequences are evolutionarily closely related. However, because DDH can only be determined through costly and time-consuming laboratory experiments, DDH was replaced by ANI early on in computational biology, which is statistically well justified by the strong positive correlation between DDH and ANI that has been observed in innumerous experiments. `fastANI` is a tool that accurately approximates ANI values for (draft) genomes with 20-100% completeness, in the identity range of 80-100% [98].

In the meantime, databases have been filling up with DNA sequences, whose numbers now rather range in the millions than in the thousands. By 2024, GenBank and RefSeq has more than 2.3 million bacterial genomes [23], and ENA more than 1.9 million [20]. So, querying DNA sequence databases with alignment-based search tools is no longer feasible: runtimes inherent to the computation of alignments, even if guided by smart heuristics such as with BLAST, become excessive when hundreds of thousands, or even millions of alignments have to be computed. Novel alignment-free approaches that accurately classify query sequences are urgently sought for.

The delimitation of boundaries for defining a species is a fundamental task in taxonomy. In the bacterial world, defining species is of great importance in the food industry and healthcare settings [99], [100], [101], [102].

The field of bacterial taxonomy was significantly advanced by the seminal work of Woese and colleagues [103, 104]. He was the first to construct a phylogenetic tree using only the *16S rRNA gene*, which is also known as the *subunit rRNA gene*. The 16S rRNA gene is used since it is highly conserved and is ubiquitous in prokaryotes (bacteria and archaea) [105].

Most computational species identification frameworks tools utilize the 16S rRNA gene [26, `TYGS`] due to its efficiency in conducting pairwise comparisons since it is short ($\approx$ 1485 bp long) and highly variable between species. Thus, it is well suited to serve as a marker to distinguish between different species. However, a prerequisite for this approach is the identification of the gene in the assembly using RNAmmer [106] which can be missing in some (draft) assemblies. Some tools [27, `fIDBAC`] integrate 16S rRNA gene identification with k-mer-based approaches to address this limitation. In contrast, other tools [25, `GAMBIT`] exclusively rely on k-mers, while additional tools prioritize species delimitation only using ANI [107]. Is it noteworthy that all species identification tools rely on a fixed database of known assemblies.

The Type (Strain) Genome Server (`TYGS`)[1] [26] database contains a comprehensive collection of 20,196 microbial type-strain genomes[2], and is updated weekly by searching GenBank [111, 112]. It incorporates quality checks to reject poor quality assemblies, and incorrectly identified species. This is done by rejecting assemblies > 500 contigs and matching the known 16S rRNA gene sequence of a type strain of the same species or subspecies respectively. TYGS is offered as web server only.

In `fIDBAC` [27] the authors curated the database by collecting bacterial genome assemblies with meta-information such as strain, culture collection, clone, and comments from NCBI. In addition, CheckM(v1.0.18) [113] was used to evaluate the completeness and contamination of each genome. Assemblies with completeness smaller than 90%, or with contamination smaller than 5% were removed. Furthermore, the LTP [114] database was used to map 16S rRNA genes for consistency. Assemblies with any disagreement were excluded by the authors. Subsequently, the authors also investigated wrongly labeled genomes, and among species containing more than one type of genome, they found and removed 20 genomes that were incorrectly labeled, as assessed by ANI values and using strain information from LPSN[3]

[1]https://tygs.dsmz.de/

[2]data accession: January 12, 2024. The initial release of TYGS used only the 16S gene to determine species. The latest release of TYGS [108] also incorporates MASH [109] to determine inter-genomic distances, in cases where the 16S gene is missing. However, as noted in the latest release, the 16S gene is still preferred. The TYGS database, which contains genome sequences with pre-computed G+C content, and uses taxonomy databases that list all the names of prokaryotes, which have been validated and published according to the International Code of Nomenclature of Prokaryotes [110]. The number of species is not mentioned

[3]https://bacterio.net/

[114] or articles in the International Journal of Systematic and Evolutionary Microbiology[4]. This way, the authors of the study curated a database of 12,783 bacterial genome assemblies, covering 9,827 species and 2,448 genera. A four-step method is used to perform the identification. The first step in this process is the extraction of 16S sequences from queries, which are BLASTed against the LTP database. The second step consists of matching k-mers from the query genome against the k-mers in the curated database using KmerFinder [115].

In `GAMBIT` [25] a curated database of 48,224 bacterial genomes is used from the NCBI RefSeq database spanning 1,414 species and 454 unique genera. To perform species identification, an approximate Jaccard similarity between the query and a subset of the k-mers of the genomes in the database is computed.

Recently, `GSearch` [116] was published, a genome search tool that combines MinHash-like or HyperLogLog sketching algorithms for genomic distance estimation and Hierarchical Navigable Small World graphs (HNSW) for finding nearest neighbors. A subset of k-mers and its ocurrences is selected as a feature representation of each assembly, and using a weighted Jaccard-like similarity (Jaccard similarity only accounts for presence and absence of k-mers) they create an index based on HNSW. An advantage of k-mer hashing methods is that there exist approximations for computing the ANI distance between assemblies based on the Jaccard distance.

Aiming to replace BLAST, in `MOF` [22] the authors utilized a bacterial dataset that included 661k genomes, comprising all bacterial genome isolates up to November 2018 at the European Nucleotide Archive (ENA) [117]. The dataset is not manually curated in any manner but relies on its uniformly assembled construction and criteria for defining high-quality assemblies. The identification of species is not explicitly defined as one of their main tasks, since it focuses on exposing the advantages of phylogenetic compression to deal efficiently

[4]https://www.microbiologyresearch.org

with large genomics datasets. However, species can be identified by a majority vote based on a query.

Most recently, the largest bacterial dataset was released [20], comprising all bacterial genomes uniformly assembled, available at ENA (up to May 2023), where the same approach as in `MOF`, this time refer as to `Phylign`, was used to index the dataset with an index size of 248GB.

In addition to `Phylign`, another tool called `LexicMap` [23] was proposed for handling large bacterial datasets, reporting an index of 2.46TB for the same dataset. Both `Phylign` and `LexicMap` aim to overcome the limitations of BLAST for querying large genomic datasets, hence proposing a pipeline consisting of a query (at the level of reads or contigs) and posterior alignment against assemblies, while `Phylign` leverages minimap2 for this task, `LexicMap` makes use of the wavefront alignment algorithm.

Although, out of the scope of this work, several tools are specifically designed for read classification in metagenomics, where the goal is to bin reads in a sample based on the species they come from. Examples of these tools are `Kraken` [118], and `Kraken2` [24], `Centrifuge` [28], and `Centrifuger` [29], all of them work with short reads, while `Centrifuger` can work with long reads as well. `Taxor` [119] is another tool that only handles long reads. These tools rely on databases of taxonomically labeled sequences.

Machine Learning tools have been proposed to speed up classification and avoid the dependency on index databases, thus decreasing computational resources needed to solve taxonomy-level assignments at the level of reads. Most notably, `HiTaxon` [31] and `Taxometer` [32] for bacterial taxonomy classification, `DeepMicrobes` [33] restricted to gut microbiota. `EukRep` [120] and `Tiara` [121] are the most general tools at the Domain taxonomy level, and `MycoAI` [122] targets fungal sequences. While not at their main purpose, the last three tools have been shown to work with bacterial data.

While small databases allow for manageable datasets for manual curation and require few computational resources, they lack variability, which is a concern, especially in bacterial applications due to their high mutation nature. This variability is naturally present in the pangenome, but considering such datasets makes manual curation impractical, and current tools have been demonstrated to require significant computational resources, which limits their accessibility and practical use to the broader research community. In contrast, classification methods based on machine learning have the advantage of being faster and generalizable to unseen data (if well trained), at the expense of restricting its utility to mere classification, not allowing the user to find similar sequences for downstream analyses like traditional methods that index databases.

6.2. Preliminaries

Let us define by $\Sigma = \{A, C, G, T\}$ the DNA alphabet, and the extended DNA alphabet by $\Sigma_N = \{A, C, G, T, N\}$, where N indicates a null call. Consider a collection $\mathcal{S}$ of (possibly draft) assemblies, where each assembly $s \in \mathcal{S}$ could be a set of more than one (contig) sequence over the alphabet Σ_N, and a set $\mathcal{L}$ of labels. We assume that there exists a labeling function $\lambda : \mathcal{S} \mapsto \mathcal{L}$ that assigns a species in $\mathcal{L}$ to each assembly $s \in \mathcal{S}$.

We aim to learn a function $f : \mathcal{S}_k \mapsto \mathbb{R}^n$ (for $n \ll 4^k$) that reduces the dimensionality of the k-mer distribution of an assembly, where $\mathcal{S}_k$ represents the space of k-mer distribution for the set $\mathcal{S}$ of assemblies, and the n-dimensional representation is known as the *embedding*. The embedding representation of the assemblies is then used to create an index that can be query based on similarity given by their Euclidean distance. We can approximate f by choosing a mapping $E : \mathcal{S}_k \mapsto \mathbb{R}^n$, from now on the *encoder*. The construction of this encoder is the main goal of our approach. Furthermore, we exploit the embedding representation for the identification of outliers and mislabeled assemblies using Confident Learning [123, 124].

6.2.1. In-distribution and out-of-distribution samples

In machine learning, understanding the concepts of outliers, in-distribution samples, and out-of-distribution (OOD) samples is crucial for developing robust models. In-distribution samples are those that conform to the patterns learned during training [125], while outliers are data points that significantly deviate from these patterns, potentially indicating anomalies [124]. OOD samples, on the other hand, originate from different distributions than the training data, posing challenges for model reliability. Effectively identifying and handling these categories is essential for ensuring that models can generalize well and maintain accuracy when faced with novel or unexpected inputs during deployment.

1. **Outlier**. An outlier is a data point that significantly differs from the majority of the data in a dataset. In the context of machine learning and OOD detection, an outlier is typically an instance that does not conform to the expected patterns or distributions of the training data. Outliers can indicate anomalies or novel instances that the model has not been trained on.

2. **In-distribution sample**. An in-distribution sample refers to data points that belong to the same distribution as the training data used to train a model. These samples are expected to conform to the patterns and characteristics learned by the model during training.

3. **Out-of-distribution sample (OOD)**. An out-of-distribution sample is a data point that comes from a different distribution than the training data. These samples do not conform to the patterns learned by the model and are considered to be outside the scope of what the model was trained to recognize. OOD samples are important for testing the robustness and generalization capabilities of machine learning models, as they help assess how well the model can identify instances that it has not encountered during training.

6.2.2. Indexes

An index is a data structure that allows for efficient retrieval of information within a large dataset. Just as a book index provides page numbers for specific topics, an index in a database or data structure enables quick access to particular records, reducing the need for a full scan of the data. Indexes are essential for performing queries efficiently, especially as the volume of data grows. By organizing data for faster retrieval, indexes make it possible to perform search, filter, and sort operations more quickly.

In bioinformatics, indexing is crucial for managing large genomic datasets, particularly for tasks involving k-mers, which are often used to represent and compare genomic sequences, where each sequence is broken down into these k-mers (and possibly its occurrences) to capture its unique features. Indexes for k-mers enable a rapid search for similarity, alignment, and comparison of genomic sequences by allowing efficient access to these substrings without scanning entire genomes.

Several indexing structures are specifically designed for k-mer-based bioinformatics tasks. Hash tables and suffix trees are widely used [68], providing a way to quickly locate all occurrences of a k-mer in a genome or across multiple genomes. More recent approaches, like Burrows-Wheeler Transform (BWT) and FM-indexes [126], and most recently, Compact Bit-Sliced Signature (COBS) index [127] offer compressed indexing techniques that are memory-efficient and highly effective for large-scale sequence alignment tasks. These indexes enable rapid searching and alignment, making them foundational for genome assembly, variant calling, and other large-scale bioinformatics analyses.

In machine learning and data analysis, a dense vector is a representation of data in a continuous, high-dimensional space where most elements have non-zero values. Dense vectors are widely used to represent items in tasks like natural language processing, image retrieval, and recommendation systems. Each vector encodes features of an item, such as the semantic content of a text document or the pixel patterns in an image, allowing for similarity-based retrieval (*e.g.* finding similar documents or images).

Indexing dense vectors is more challenging than indexing traditional tabular data, as it requires efficient similarity search in high-dimensional spaces. Techniques such as approximate nearest neighbor (ANN) search algorithms, like Locality-Sensitive Hashing (LSH) [128] and Hierarchical Navigable Small World (HNSW) [129], are commonly used. These methods create indexes that facilitate quick similarity searches, enabling applications such as image recognition or text-based recommendation to locate similar vectors (and thus similar items) without exhaustively comparing all pairs.

6.2.3. Metric learning and the triplet loss

In metric learning, in the context of supervised learning, the goal is to embed representations of the input data learned such that embeddings sharing a label are closer in Euclidean distance than embeddings with different labels.

The triple loss for a triplet (x_i^a, x_i^p, x_i^n) is defined as follows,

$$\mathcal{L}_{\text{triplet}} = \min\left(0, \|f(x_i^a) - f(x_i^p)\|_2^2 - \|f(x_i^a) - f(x_i^n)\|_2^2 + \alpha\right) \tag{6.1}$$

where x_i^a is the anchor sample, x_i^p is the positive sample (same label as the anchor), x_i^n is the negative sample (different label from the anchor), $f(\cdot)$ is the embedding function that maps inputs into the feature space, $\|\cdot\|_2$ is the Euclidean (L2) distance, α is the margin enforced between positive and negative pairs, usually defined in $[0.1, 0.5]$. In our case, $f(\cdot)$ denotes the encoder, and the pair (x_i^a, x_i^p) are two FCGRs from the same species, while x_i^n corresponds to an FCGR of a different species.

Notice that by the definition of $\mathcal{L}_{\text{triplet}}$ not all triplets might contribute to the learning process, those triplets (x_i^a, x_i^p, x_i^n) such that

$$\|f(x_i^a) - f(x_i^p)\|_2^2 + \alpha \leq \|f(x_i^a) - f(x_i^n)\|_2^2 \tag{6.2}$$

already satisfied the desired separability condition, hence $\mathcal{L}_{\text{triplet}} = 0$, making meaningless the computation of the gradient and consequently not influencing the update of weights in the neural network.

As explained in [42], choosing triplets is crucial to use the triplet loss effectively, and a triplet of interest (for which $\mathcal{L}_{\text{triplet}} \neq 0$) can be categorized in two classes: *semi-hard* and *hard* triplets.

In this context, given an anchor x_i^a, a hard-positive is computed as the $\arg\max_{x_i^p} \|x_i^a - x_i^p\|$, while a hard-negative is computed as the $\arg\min_{x_i^n} \|x_i^a - x_i^n\|$. These are the most difficult examples to separate that violate the desired condition from Equation 6.2. Since computing $\arg\max$ and $\arg\min$ is computationally expensive over the entire dataset (*e.g.* for a dataset of N elements, the number possible triplets is $O(N^3)$), in practice, these *hard* triplets are chosen from a mini-batch (the subset of data used in each iteration of an epoch). A drawback of selecting the hardest negatives is that they can lead to bad local minima early on in training [42].

On the other hand, a triplet (x_i^a, x_i^p, x_i^n) is called *semi-hard* when

$$\|f(x_i^a) - f(x_i^p)\|_2^2 < \|f(x_i^a) - f(x_i^n)\|_2^2 < \alpha \tag{6.3}$$

so even when x_i^p is closer to the anchor than x_i^n, it still lies inside the margin α.

The suggested procedure when using the triplet loss is to start training with semi-hard triplets, and then focus on hard triplets, but we need to be careful since hard triplets might be composed of outliers or mislabeled samples and could worsen the results.

6.2.4. Confident Learning

Mislabeled assemblies are identified using Confident Learning (CL) [123], a model agnostic framework to identify mislabeled data. Let us recall that each assembly in the dataset $\mathcal{S}$ is dotted with a label in $\mathcal{L}$ given by the function $\lambda : \mathcal{S} \mapsto \mathcal{L}$. In CL the assumption is that λ is a noisy function, *i.e.* it may contain errors w.r.t. the real (uncorrupted and unknown) labels. A classifier $\mathbf{\Theta}$, which outputs probabilities for each assembly $s \in \mathcal{S}$ and label $l \in \mathcal{L}$ in the dataset is needed to identify these errors, and gives confidence thresholds. The probabilities used by CL must be from out-of-sample data, *i.e.* on assemblies the model was not trained on. Here is where cross-validation plays a crucial role in analyzing the entire dataset.

In the CL framework, the identification of mislabeled data relies on the computation of the *confident joint*, denoted by $C_{\tilde{y},y^*}$, where $\tilde{y}$ represents the noisy observed labels, and y^* the real (uncorrupted) ones. The goal is to estimate the joint probability distribution between $\tilde{y}$ and y^*, denoted by $Q_{\tilde{y},y^*}$. Since the real labels y^* are unknown, the joint probability distribution must be estimated from $C_{\tilde{y},y^*}$.

The confident joint $C_{\tilde{y},y^*}$ is similar to the confusion matrix, in the sense that it counts occurrences between predicted and true labels, with the difference that it takes into account the probabilities of each prediction to filter what is counted. Its main purpose is to estimate the set $X_{\tilde{y}=i,y^*=j}$, that is the set of $s \in \mathcal{S}$ labeled as $\lambda(s) = \tilde{y} = i$ with large enough probability $\hat{p}(\tilde{y} = j; s; \mathbf{\Theta})$ to likely belong to class $y^* = j$. Here $\hat{p}(\tilde{y} = j; s; \mathbf{\Theta})$ is the probability that s is labeled as j given by the model $\mathbf{\Theta}$ being used.

Formally, the confidence threshold t_j for a class j is defined as

$$t_j = \frac{1}{|X_{\tilde{y}=j}|} \sum_{x \in X_{\tilde{y}=j}} \hat{p}(\tilde{y} = j; s; \mathbf{\Theta}) \tag{6.4}$$

where $|X_{\tilde{y}=j}|$ is the number of assemblies $s \in \mathcal{S}$ with label $\lambda(s) = \tilde{y} = j$. Intuitively, if the model is over-confident in predicting a class j, then t_j will be proportionally larger, and vice-versa. These thresholds help to handle class imbalance.

With these thresholds, an element (i, j) of the confident joint, $C_{\tilde{y},y^*}(i, j)$ is computed as $|\hat{X}_{\tilde{y}=i,y^*=j}|$, the number of assemblies in that set (which estimates $X_{\tilde{y}=i,y^*=j}$), and is formally defined as follows,

$$\hat{X}_{\tilde{y}=i,y^*=j} = \{s \in \mathcal{S} : \tilde{y} = i, \hat{p}(\tilde{y} = j; s; \mathbf{\Theta}) \geq t_j, j = \underset{l \in [|\mathcal{L}|]:\hat{p}(\tilde{y}=l;s;\mathbf{\Theta}) \geq t_l}{\arg\max} \hat{p}(\tilde{y} = l; s; \mathbf{\Theta})\} \tag{6.5}$$

Finally, the joint probability distribution $Q_{\tilde{y},y^*}$ can be estimated from the confident joint by $\hat{Q}_{\tilde{y},y^*}$, where each element (i, j) is defined as follows,

$$\hat{Q}_{\tilde{y}=i,y^*=j} = \frac{\frac{C_{\tilde{y}=i,y^*=j}}{\sum_{j \in [|\mathcal{L}|]} C_{\tilde{y}=i,y^*=j}} \cdot |X_{\tilde{y}=i}|}{\sum_{i \in [|\mathcal{L}|], j \in [|\mathcal{L}|]} \left(\frac{C_{\tilde{y}=i,y^*=j}}{\sum_{k \in [|\mathcal{L}|]} C_{\tilde{y}=i,y^*=k}} \cdot |X_{\tilde{y}=i}| \right)} \tag{6.6}$$

$\hat{Q}_{\tilde{y}=i,y^*=j}$ is the estimated joint probability for the observed label $\tilde{y} = i$ and the true label $y* = j$.

The numerator ensures that the total of each row in $\hat{Q}_{\tilde{y},y^*}$ equals the total number of examples we have for each label. While the denominator calibrates it such that the sum of all probabilities equals 1.

Finally, based on the estimations of $C_{\tilde{y}=i,y^*=j}$ and $Q_{\tilde{y}=i,y^*=j}$, several rank and pruning methods can be used to flag label issues. We refer the reader to Section 3.2 in [123] for more details. We used the prune-by-noise rate method.

6.3. Methods

In this section, we describe a convolutional (and deconvolutional) layer specifically designed to exploit the FCGR representation: ConvFCGR (and DeconvFCGR), and two architectures for supervised and unsupervised learning on top of these layers, the `CNNFCGR` and `AutoencoderFCGR`, respectively. Following with details on how they are trained. We describe the bacterial dataset we use for our experiments, how their labels are assigned, and link its importance with our data curation procedure. Finally, we describe how to create and query the embedding-based index of assemblies for fast query and accurate classification at the species taxonomy level.

6.3.1. Convolutional Neural network for the Frequency matrix of the Chaos Game Representation of DNA

A convolutional layer is a layer of a deep neural network in which a convolutional filter passes along an input matrix. In the case when the FCGR is the input matrix, this filter corresponds to a 2-dimensional matrix and applies a linear operation between subsets of the input matrix and the filter. The convolutional layer can be characterized by the *kernel size* (numbers of rows and columns of the convolutional filter), the *stride* (number of rows/columns to move left, right/up, down to apply the convolutional filter in the input matrix), and the *number of filters*, which correspond to additional matrices applied to the input matrix independently (in-depth), usually refer to as channels.

Given k, in the FCGR representation, k-mers sharing the length l suffix ($l < k$) are contained in a sub-square of dimension $2^{k-l} \times 2^{k-l}$ in the FCGR. In Figure 2.3 we can illustrate this with an example with $k = 3$ and $l = 2$. In the top-left, the sub-square with center CC contains the 3-mers ACC, CCC, GCC, and TCC, all sharing the same suffix CC of length 2. This can be generalized as the following result.

Proposition 6.1 When a convolutional layer with kernel size 2^{k-l} and stride 2^{k-l} is applied to the FCGR matrix, the convolutional filter only processes k-mer occurrences of subsets of k-mers sharing a length l suffix.

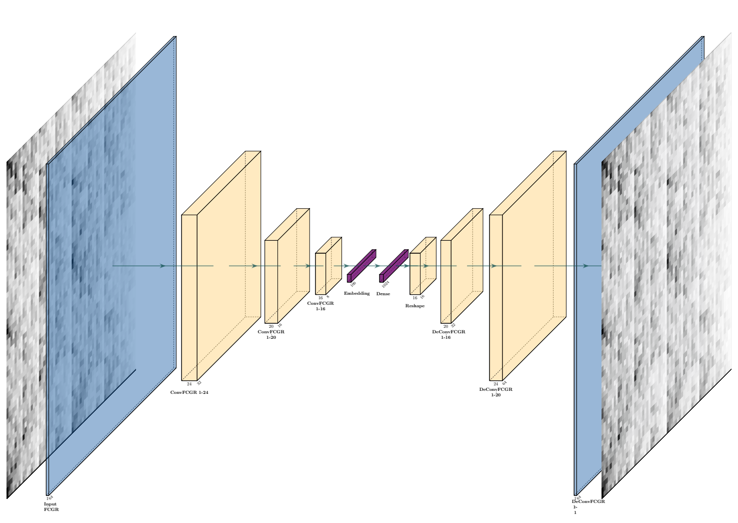
We propose two (strongly related) convolutional network architectures for exploiting the FCGR representation under unsupervised (no labels) and supervised metric learning (with labels) scenarios. We will use these two modes when classifying bacterial species.

For unsupervised learning, we propose the `AutoencoderFCGR` architecture (see Figure 6.1a). Where the encoder is designed using *ConvFCGR* layers, a convolutional layer [50], and the decoder is designed using *DeConvFCGR* layers, a deconvolution layer [130] (see Figure 6.2). These proposed layers exploit the CGR encoding by analyzing k-mers sharing the length $(k - l)$ suffix, for $0 < l < k$.

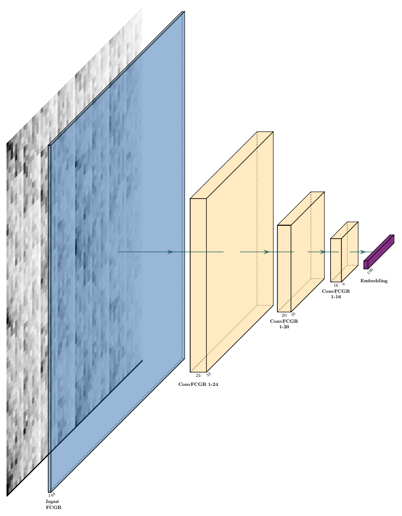
When labels are available, we can apply supervised learning. More specifically, under the umbrella of supervised metric learning, we propose the `CNNFCGR` (see Figure 6.1b), a convolutional neural network defined as the encoder part of the `AutoencoderFCGR`, and whose output is an n-dimensional vector, the so-called embedding. Unlike traditional supervised learning approaches designed for mere classification into a fixed number of classes, metric learning focuses on learning distance metrics or similarity measures tailored to specific tasks, such that similar instances are closer in the embedding space, and dissimilar ones are further apart, clustering the input data in the n-dimensional space. This is achieved using ad-hoc loss functions, most notably contrastive loss [43], or triplet loss [42] by directly optimizing the embeddings.

6.3.1.1. *ConvFCGR and DeConvFCGR layers*

The *ConvFCGR* layer of level l corresponds to a convolutional layer with kernel size 2^l, stride 2^l, and no padding. Hence, for an input of size $2^k \times 2^k$, the *ConvFCGR* of level



(a) `AutoencoderFCGR` $k = 6$



(b) `CNNFCGR` $k = 6$

Figure 6.1: The input of the `AutoencoderFCGR` is the $2^6 \times 2^6 \times 1$ dimensional tensor, representing the FCGR of a (draft) assembly. The **architecture** of the autoencoder is composed of an encoder and a decoder. The encoder (left side) is a concatenation of three *ConvFCGR* layers of level 1, and the number of filters $24, 20$, and 16, respectively. The output of the encoder is an L2 normalized 128-dimensional vector (**Embedding**), represented by a dense layer with 128 neurons. The decoder (right side) receives as input the embedding and is followed by three *DeConvFCGR* of level 1, and the number of filters $16, 20$ and 24, respectively, the output of the decoder is the reconstructed FCGR ($2^6 \times 2^6 \times 1$). The `CNNFCGR` is the encoder part of the `AutoencoderFCGR`. The ReLU activation function follows all *ConvFCGR* layers, and batch normalization in the filter axis. The same applies to the *DeConvFCGR* layers, except for the last one, for which we do not apply batch normalization.

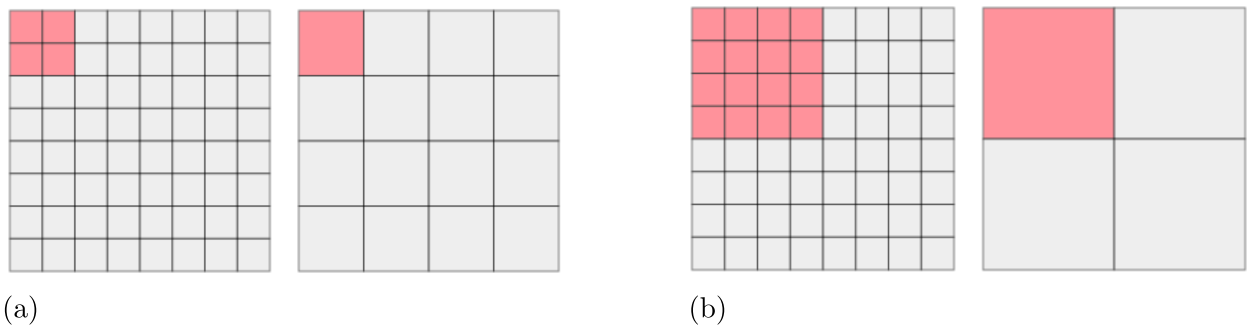


Figure 6.2: **ConvFCGR layer**. Gray grids correspond to the input and output of the layers. Red grids represent the position where a filter of a layer is applied. In (a) a $ConvFCGR(1, 1)$ is applied to the input (FCGR) tensor of size $2^3 \times 2^3$, level $l = 1$ correspond to a layer with stride and kernel equal to 2, *i.e.* the output is a tensor of size $2^2 \times 2^2$. In (b) $ConvFCGR(2, 1)$ is applied to the same input. This time, since $l = 2$, the layer uses 16 positions of the tensor, and outputs 1.

l outputs a tensor of dimension $2^{(k-l)} \times 2^{(k-l)}$. The $DeConvFCGR$ of level l represents the inverse operation of $ConvFCGR$ of level l, *i.e.* for an input of size $2^{(k-l)} \times 2^{(k-l)}$, the $ConvFCGR$ of level l outputs a tensor of dimension $2^k \times 2^k$ (filters were intentionally excluded here for the ease of explanation since they add information in-depth).

Given l levels and f filters, the convolutional FCGR and deconvolutional FCGR layers are denoted by $ConvFCGR(l, f)$, and $DeConvFCGR(l, f)$, respectively. In Fig 6.2 are illustrated $ConvFCGR(1, 1)$ and $ConvFCGR(2, 1)$ on an input of dimension $2^3 \times 2^3$. The same example illustrates how $DeConvFCGR$, with the same parameters (l, f) works.

6.3.1.2. Encoder

Given an input tensor of dimensions $2^k \times 2^k \times 1$ (FCGR from k-mers with depth channel included) the encoder is defined as a concatenation of $ConvFCGR(1, 4l)$ layers, $l \in \{k, k-1, \ldots, 4\}$ (in that order) each one followed by the ReLU activation function [131] and a batch normalization in the channel axis. The number of layers is equal to $k - 3$, so given an input of dimension $(2^k \times 2^k \times 1)$, it outputs a $(2^4 \times 2^4 \times 16)$, *i.e.* 1024 features.

This is then flattened and fed to a dense layer of n neurons, with an L2 normalization, so embeddings have a norm equal to 1, and Euclidean distances between embeddings lie in $[0, 2]$. Notice that the choice of stopping at $k = 4$ is to consistently decrease the number of dimensions of the feature space up to the embedding layer so that n can be defined up to $n = 1000$ to follow the described desired criteria. We choose $n = 128$ for our experiments.

6.3.1.3. Decoder

The decoder first maps the n-dimensional input to a dense layer of 1024 neurons, followed up by a reshape to $(2^4 \times 2^4, 16)$ and a concatenation of $DeConvFCGR(1, 4l)$ layers, $l \in \{4, \ldots, k-1\}$, each one followed by the ReLU activation function and a batch normalization in the channel axis, finally a $DeConvFCGR(1, 1)$ with activation function ReLU, to output the $2^k \times 2^k \times 1$ tensor with values in $[0, 1]$.

6.3.2. Autoencoder training

The `AutoencoderFCGR` is designed to reconstruct the FCGR of assemblies. It receives as input the FCGR matrix and outputs a matrix of the same dimensions. We use the encoder described in Section 6.3.1.2 and decoder from Section 6.3.1.3. The mean squared error (MSE) is used as loss function [40], and is defined as follows,

$$\mathcal{L}_{\text{MSE}} = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2 \tag{6.7}$$

where N is the number of data points, y_i is the true value for the i-th data point, $\hat{y}_i$ is the predicted value for the i-th data point. In our case, $N = 4^k$ is the total number of k-mers in the FCGR, while y_i is the (rescaled) frequency of the i-th k-mer.

We set the training of the `AutoencoderFCGR` to 100 epochs, monitoring the validation loss with a patience early stopping (number of epochs the training continues if the validation loss does not improve) of 50 epochs, and a patience learning rate (number of epochs the optimizer continues using the same learning rate if the validation loss does not improve) of 30 epochs with a factor of 0.1. A random split of 80 : 10 : 10 was used for training, validation, and test sets.

6.3.3. Metric Learning training

When dealing with a dataset with a large number of labels, the generation of mini-batches to train the models requires special considerations. For example, in the 661k bacterial dataset we find 2336 species belonging to high-quality assemblies, while in the 2MM bacterial dataset, the number of species reported is 10542. Recall that a mini-batch consists of a subset of the dataset randomly sampled. When dealing with the triplet loss, we must provide a way to subsample the dataset for creating (either semi-hard or hard) triplets. For this purpose, we use a mini-batch generator that randomly selects a portion of the labels (species), and then selects a fixed number of FCGRs for each label, thus, each time we train the `CNNFCGR`, the triplets are created from a balanced dataset with a constrained number of possible anchors.

We set the training of the `CNNFCGR` to 100 epochs, with a patience early stopping of 30 epochs, and a patience learning rate of 20 epochs with a factor of 0.1, both monitoring the validation loss. We selected 64 species per mini-batch and multiplied by a factor of 10 the number of mini-batches generated for training and validation steps, *i.e.* the number of mini-batches corresponds to 10× the ratio between the FCGRs in the training set (as well in the validation set) divided by the mini-batch size, thus increasing the number of triplets seen during training.

Training, validation, and test sets were built as follows, for each species with more than 13 samples in the dataset, we randomly split them into training, validation, and test sets in proportion 80 : 10 : 10. This decision is in line with how we query the index (created with training and validation sets) with the test set since we search for the 11 closest neighbors for classification purposes. As we will see in the Section 6.3.7 and Section 6.3.8.

6.3.4. Large bacterial datasets

Our goal is to index and query large bacterial datasets, and to evaluate their quality under our approach. We consider the 661K [117] and the 2MM [20] bacterial datasets. The former contains all the bacterial data from the European Nucleotide Archive (ENA) available up to 2018, while the latter was updated with the bacterial data up to 2023. They were both assembled with a unified pipeline from short reads but differ in how a "high-quality" assembly was defined. The main difference lies in how the species label is assigned and trusted. In the 661K dataset, the label is defined as the most abundant species in the sample (with a Kraken [24]Bracken [132] pipeline) if its abundance exceeds 90%. In contrast, in the 2MM dataset, the label is assigned to the most abundant species from isolated data (with sylph [30]), with the abundance threshold set to 99%. The number of assemblies and species for both datasets is reported in Table 6.1.

	High quality		All	
Dataset	Assemblies	Species	Assemblies	Species
661K	639981	2336	661405	6997
2MM	1857792	10542	1932812	11445*

Table 6.1: **Bacterial datasets.**
* The total number of species was obtained by counting all unique values in the column label from the file SAMPLE2SPECIES2FILE.TSV.GZ from https://ftp.ebi.ac.uk/pub/databases/AllTheBacteria/Releases/0.2/metadata/, since only the number of species of high-quality assemblies was reported in the manuscript.

6.3.5. Outlier detection

To find outliers we need to define two sets of assemblies (labels are not needed here); $\mathcal{X}$, an in-distribution dataset, and $\mathcal{Y}$, the set to be evaluated. We identify outliers using the embedding representations of the FCGRs after training a model.

The set $\mathcal{X}$ is defined by the embeddings of the assemblies in the training and validation set, while the set $\mathcal{Y}$ is defined by the embeddings of the assemblies in the test set.

To identify outliers, we adopt the method from [124]. For each embedding in $\mathcal{X}$, the average distance to its N (we use $N = 10$) closest neighbors in $\mathcal{X}$ is calculated, and a distance threshold d_α (we use $\alpha = 99$) is defined as the α-th percentile of the averages of those distances. Then, for each embedding in the set $\mathcal{Y}$ the average distance to the neighbors of the N closets in $\mathcal{X}$ is calculated, if this distance is greater than d_α, the associated assembly is flagged as an outlier.

Examples of FCGRs for in-distribution assemblies, and outliers for 5 different species are shown in Figure 6.4.

6.3.6. Verification of mislabeled assemblies with ANI

To flag possible mislabeled assemblies with Confident Learning, on top of the embeddings obtained with the `AutoencoderFCGR`, we trained a 3-layer dense neural network, with 64, 32, and $|\mathcal{L}|$ neurons, respectively, with the ReLU activation function for internal layers, and the Sigmoid activation function in the output, to obtain probabilities for each label in $\mathcal{L}$, and the categorical cross-entropy loss function.

Given an assembly $s \in \mathcal{S}$ with label $\lambda(s) = X$ and predicted label Y that is flagged as misclassified, we compute the average nucleotide identity (ANI) between s and the reference sequences of labels X and Y (when possible, not all species have a known reference). If the

ANI distance between s and the reference sequence of Y exceed the 95.0 threshold required to classify them as the same species, and is greater than the ANI distance between s and the reference sequence of X, we redefine $\lambda(s) = Y$. Reference sequences were downloaded from NCBI [5], and `fastANI` [98] was used to compute ANI values.

Once outliers are identified and flagged mislabeled assemblies are verified from the initial dataset and, we can optionally modify the initial dataset, and proceed with the construction of the index on top of the embeddings of the curated dataset. But not doing this does not impede to proceed to the creation of the index.

6.3.7. Creating the Index

We recall that we aim to learn a function $f : \mathcal{S}_k \mapsto \mathbb{R}^n$ (for $n \ll 4^k$) that reduces the dimensionality of the k-mer distribution of an assembly by encoding some relevant information for its reconstruction, where $\mathcal{S}_k$ represents the space of k-mer distribution for the set $\mathcal{S}$ of assemblies. We can approximate f by choosing a mapping $E : \mathcal{S}_k \mapsto \mathbb{R}^n$, from now on refer as the *encoder*, that could be either the encoder of the `AutoencoderFCGR` or directly the `CNNFCGR` (see Figure 6.1).

The index $\mathcal{I}$ has three components: (1) a FAISS index $\mathcal{F}$ [97] storing the embeddings of each assembly in $\mathcal{S}$, (2) the encoder E, and (3) the labeling function λ.

With the encoder E, we create the embedding representation for each assembly $s \in \mathcal{S}$.

These embeddings are indexed using FAISS [97], an easy-to-use index for dense vectors, extremely compressed (less than 250 MB for half a million embeddings of dimension 100), and fast to query. To facilitate associating embeddings in $\mathcal{F}$ to the labels (species) by the λ function, we store a list where the i-th label corresponds to the i-th embedding in $\mathcal{F}$.

[5]accession July 30th, 2024. https://www.ncbi.nlm.nih.gov/datasets/genome/?taxon=2&reference_only=true

To compare or query other assemblies against the set $\mathcal{S}$, we need to map them to the embedding space, so we need to preserve the mapping E as part of the index $\mathcal{I}$.

6.3.8. Querying the Index

The process to go from a (possibly draft) assembly to the embedding space where the indexed assemblies lie involves (1) counting k-mers from the assembly, (2) creating the FCGR matrix, and (3) using the mapping E to get an embedding in $\mathbb{R}^n$, as illustrated in Fig 6.3.

A query to the index $\mathcal{I}$ is defined as a (draft) assembly s_q (in fasta format, and which might not belong to $\mathcal{S}$), and the query results correspond to a list with the N closest embeddings in the FAISS index $\mathcal{F}$ and their labels given by the labeling function λ. We assign the label L to s_q, where L corresponds to the most common label in the query result. Additionally, we define a confidence threshold for the query as the ratio between the frequency of L in the query results and N.

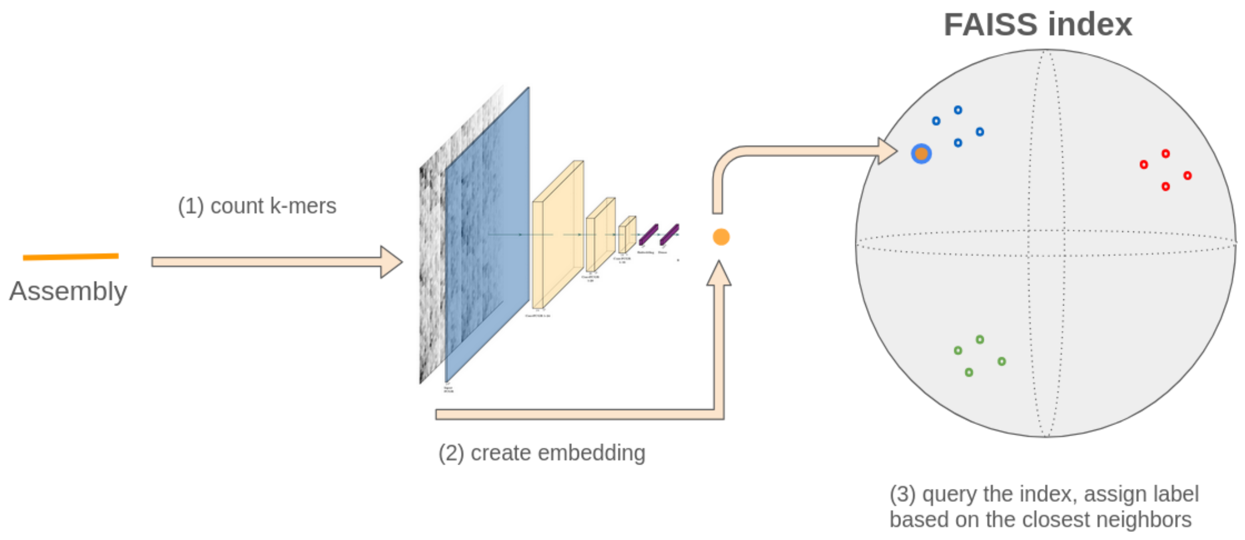

FAISS index
(1) count k-mers
Assembly
(2) create embedding
(3) query the index, assign label based on the closest neighbors

Figure 6.3: **Query index**. Given a query (draft) assembly, we start by (1) counting k-mers and creating its FCGR representation. Then (2) the FCGR is fed to the Encoder to produce the embedding representation. Finally, (3) we query the FAISS index to retrieve its N-closest neighbors and assign a label to the query assembly based on the majority of species in the query result.

6.4. Results

We have three main goals in these experiments. The first one is to assess the utility of the data curation procedure, the second one is to evaluate how accurately we retrieve sequences of the same species given a query assembly, and the third one, is to index the largest bacterial dataset available in the literature, the 2MM bacterial dataset (see Table 6.1)

We have described two architectures to handle the FCGR, the `AutoencoderFCGR`, and the `CNNFCGR`. For the first goal related to data curation, we use the `AutoencoderFCGR` and the 661K bacterial dataset. Let us recall that labels information is not required for training the `AutoencoderFCGR`, nor the identification of outliers. We chose the 661K dataset for this experiment since the criteria for defining high-quality assemblies is less restrictive than for the 2MM dataset, in addition, the assignment of labels was obtained from the sample of reads, while in the 2MM was from the isolated data (see Section 6.3.4), so we expect it to be the more noisy one w.r.t. outliers and labels.

For the other experiments, we have decided to use the `CNNFCGR` and the 2MM bacterial dataset, since we are interested not only in building the index but in how accurate is to classify and group assemblies from the same species in an imbalanced dataset. Additionally, we include correlation analysis between the Euclidean distances of embeddings produced by `PANSPACE`, and distances produced by `fastANI` and `GAMBIT` for a subset of assemblies. In all experiments, we report results using 6-mers.

6.4.1. Assessing data curation

The first step of data curation consists of training the `AutoencoderFCGR`, where training and validation sets are indexed for each fold, and the test set is used for the query. We identify outliers in each fold's test set and then report all of them so that the entire dataset is analyzed. Mislabeled assemblies are also identified in each fold's test set, but then validated by computing the ANI distance w.r.t. the reference genomes of their original label, and the presumably correct label.

We report in Table 6.2, the user time and maximum RAM usage in the 4 main steps when creating the index: training of the autoencoder, indexing embeddings, querying the index, and data curation. All metrics are reported from a 5-fold cross-validation experiment on the 661K dataset. Each fold required on average an estimated 1:25 hr for $k = 6$, 4:35 hr for $k = 7$, and 14:59 hr for $k = 8$, where most of the time is spent in the training phase. On average, the maximum RAM usage for $k = 6$ is 5.71 GB (data curation step), for $k = 7$ 10.51 GB (train step), and $k = 8$ is 11.52 GB (train step). Computational resources do not differ significantly in the query and data curation steps since all of it is defined at the level of embedding, while training and indexing depend on the size of k.

	User time (s)			
k-mer	Train	Index	Query	Data Curation
6	1212.5 ± 146.2	59.7 ± 1.2	827.6 ± 10.7	3029.0 ± 98.6
7	12538.6 ± 1328.2	86.9 ± 0.9	840.0 ± 9.6	3044.3 ± 110.5
8	50019.2 ± 11584.8	183.0 ± 2.1	838.9 ± 10.4	2922.7 ± 99.6

	Max RAM (GB)			
k-mer	Train	Index	Query	Data Curation
6	4.60 ± 0.18	2.39 ± 0.01	2.91 ± 0.01	5.71 ± 0.00
7	10.51 ± 0.22	1.97 ± 0.00	2.50 ± 0.00	5.72 ± 0.00
8	11.52 ± 0.19	2.43 ± 0.00	2.91 ± 0.02	5.64 ± 0.00

Table 6.2: **5-fold cross-validation** User time and RAM usage for the main steps when creating the index. Training of the autoencoder (Train), index creation (Index), querying the index (Query), and outliers detection and mislabeled assemblies identification (Data Curation). All values are reported as the mean and standard deviation ($\mu \pm \sigma$) from a 5-fold validation experiment. The time to count k-mers is not considered in the metrics reported, it is assumed that the input is the FCGR format (matrix stored as .npy files).

Outliers Outliers are defined as assemblies that significantly differ from the other assemblies in the dataset and are computed as described in Section 6.3.5. Examples of FCGRs from non-outliers and outliers for 5 different species are shown in Figure 6.4.

In the 661K dataset, we found 7707 outliers using $k = 8$, that span 1675 species. We show the top 20 species with outliers in Table 6.3. We noticed that the most represented species in the dataset (Salmonella enterica) is not the one with the most outliers, it is preceded by Escherichia coli, and Staphylococcus aureus, and that for species with relative high representativity (more than 5000 assemblies in the dataset), we find less than 1% of their assemblies flagged as outliers. Additionally, in all these cases, most of the outliers correspond to assemblies that fall outside of the high-quality group (obtained using the column `high_quality` in the file `File4_QC_characterisation_661K.txt` from https://doi.org/10.6084/m9.figshare.16437939.v1).

On the opposite side, we find species with a very high proportion of outliers, most notably, Faecalibacterium prausnitzii, Prevotella intermedia, and Muribaculum intestinal, for which more than 80% of their sequences were flagged as outliers. All these species contribute less than 100 assemblies each to the dataset. In some cases a considerable number of assemblies tagged as high-quality are par the outliers, we suspect this is due to their under-representativity in the dataset (except for Enterococcus faecium with a high representativity in the dataset, but half of their outliers are high-quality assemblies), and the training method we selected (`AutoencoderFCGR` with a random selection of data during training). This approach may be affected by imbalanced datasets, as under-represented data points have less influence on the learning process.

	Species	Outliers	HQ	not HQ	Assemblies	% Outliers
1	Escherichia coli	266	21	245	88749	0.30
2	Staphylococcus aureus	191	25	166	48418	0.39
3	Salmonella enterica	176	12	164	181871	0.10
4	Mycobacterium tuberculosis	116	3	113	48727	0.24
5	Streptococcus pneumoniae	102	12	90	51517	0.20
6	Clostridioides difficile	86	5	81	13713	0.63
7	Chlamydia trachomatis	80	10	70	1201	6.66
8	Faecalibacterium prausnitzii	79	41	38	84	94.05
9	Prevotella intermedia	78	49	29	90	86.67
10	Roseburia hominis	76	29	47	109	69.72
11	Bacillus subtilis	72	27	45	645	11.16
12	Klebsiella pneumoniae	56	9	47	13621	0.41
13	Enterococcus faecium	53	26	27	8635	0.61
14	Streptococcus agalactiae	51	7	44	10302	0.50
15	Lactobacillus plantarum	49	20	29	158	31.01
16	Pseudomonas aeruginosa	47	1	46	6371	0.74
17	Acinetobacter baumannii	46	11	35	5162	0.89
18	Campylobacter jejuni	45	2	43	28498	0.16
19	Streptococcus pyogenes	44	18	26	16830	0.26
20	Muribaculum intestinale	44	9	35	52	84.62
	Total	1757	337	1420	524753	0.33

Table 6.3: **Top 20 species with outliers.** For each species, we show the number of outliers detected in the 661K dataset, disaggregated into high-quality assemblies (HQ) and no-high-quality ones (not HQ), the total number of assemblies per species, and the proportion of outliers w.r.t. to the number of assemblies in the dataset. The results correspond to the models using $k = 8$ with the `AutoencoderFCGR` architecture.

Mislabeled assemblies This experiment aims to determine whether assemblies identified by CL as having label issues (see Section 6.2.4) can be validated using ANI distance analysis. Let us recall that the rule to state that two assemblies correspond to the same species is that the ANI value between their assemblies is greater or equal to 95%. In ad-

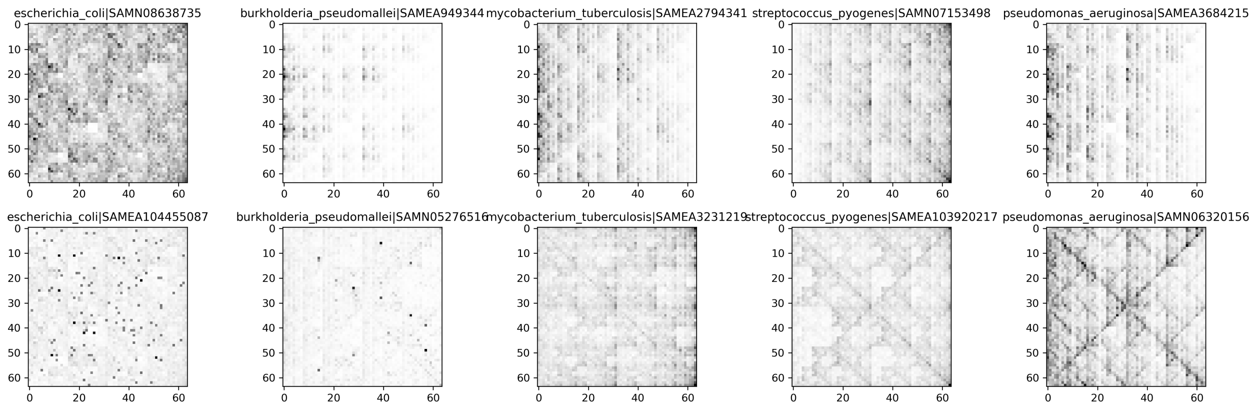


Figure 6.4: **Examples of outliers through the lens of the FCGR** Examples of five bacterial assemblies, species names, and sample ID identifiers are in the title of each image. Each image corresponds to the visualization of the FCGR in 8-bits (k-mer frequencies rescaled to the interval $[0, 255]$) computed from 6-mers, and it has dimensions $2^6 \times 2^6$, one pixel or position for each 6-mer. White color in a pixel means that the k-mer encoded has the minimum value of frequency found in the assembly. Black color represents k-mers with the maximum frequency in the assembly. Gray colors correspond to k-mer frequencies lying between the minimum and maximum values. The top row corresponds to FCGRs of assemblies considered not outliers, while the bottom row corresponds to FCGRs flagged as outliers from the same species.

dition, we recall that the label assigned to each assembly in the 661K dataset is the most abundant species in its sample of reads (see Section 6.3.4).

For each assembly flagged as with label issues, we compute the ANI distance between the assembly against (1) the reference sequence of its label, denoted by ANI_1, and (2) the reference sequence of the second most abundant species in its sample of reads, denoted by ANI_2.

The second most abundant species was obtained from https://doi.org/10.6084/m9.figshare.16437939.v1 (`File1_full_krakenbracken.txt`). Bacterial reference sequences were downloaded from NCBI (we obtained a total of 1709 reference sequences for the analysis).

We categorize the label of each assembly are as follows:

- If $ANI_1 < 95$ and $ANI_2 < 95$, the label of the assembly is categorized as *unreliable*.

- If $ANI_1 \geq 95$ and $ANI_2 \geq 95$, the label of the assembly is categorized as *doubt*.
- If $ANI_1 < 95 \leq ANI_2$, the label of the assembly is categorized as *change label*.
- If $ANI_1 \geq 95 > ANI_2$, we keep the original label, categorized as *unchanged*.

On 8642 label issues assessed with the described procedure, we obtained that 756 assemblies fit in the category *change label*, 1924 in the category *doubt*, 2340 are in the category *unreliable*, and 3622 with *unchaged* label. Most of the label issues in the category *change label* correspond to the species Enterobacter cloacae, and according to the ANI distance, the assemblies should be labeled as Enterobacter hormaechei.

	Original label	Change to	Assemblies
1	Enterobacter cloacae	Enterobacter hormaechei	476
2	Bacillus cereus	Bacillus anthracis	73
3	Burkholderia dolosa	Burkholderia cenocepacia	51
4	Mycobacterium tuberculosis	Mycobacterium ulcerans	9
5	Staphylococcus pasteuri	Staphylococcus warneri	9
6	Klebsiella oxytoca	Klebsiella michiganensis	7
7	Escherichia coli	Klebsiella quasipneumoniae	7
8	Acinetobacter calcoaceticus	Acinetobacter baumannii	6
9	Salmonella enterica	Escherichia coli	6
10	Klebsiella pneumonia	Klebsiella quasipneumoniae	5
11	Escherichia coli	Salmonella enterica	5
12	Listeria monocytogenes	Salmonella enterica	5
13	Lactococcus garvieae	Enterococcus faecalis	5
14	Campylobacter lari	Campylobacter jejuni	4
15	Bacillus thuringiensis	Bacillus anthracis	4
16	Xanthomonas citri	Xanthomonas phaseoli	4
17	Stenotrophomonas maltophilia	Pseudomonas aeruginosa	3
18	Bacillus cereus	Streptococcus pneumonia	3
19	Streptococcus oralis	Streptococcus pneumonia	3
20	Campylobacter coli	Campylobacter jejuni	3

Table 6.4: **Top 20 pairs of species that should change labels.** The first column (Original label) corresponds to the original label of assemblies in the 661K that were flagged as having label issues with confident learning, the second column (Change to) corresponds to the suggested species label validated with ANI distance, for which the assembly is closer in ANI distance to the reference sequence of the second column than to the reference of its original label. The last column corresponds to the number of assemblies that fit into this scenario.

6.4.2. Indexing the bacterial dataset

We recall that the index produced with `PANSPACE` is composed of the FAISS index which stores all embeddings of the indexed assemblies, the trained model, namely the *Encoder*, which serves to map assemblies to the embedding space and performs queries, and the list of labels for each assembly in the index.

We report in Table 6.5 the size of each component of the PANSPACE Index. Creating the index from FCGRs took less than 8 minutes with maximum RAM usage of 4.5 GB. The encoder corresponds to the CNNFCGR architecture for 6-mers.

	FAISS Index	Encoder	Labels
Size	899 (MB)	567 (KB)	68 (MB)

Table 6.5: PANSPACE **Index**. The FAISS Index for the high-quality assemblies from the 2MM bacterial dataset (1.841.109 assemblies). The Encoder is the trained model CNNFCGR for $k = 6$.

Furthermore, for testing purposes, we built the index for PANSPACE using 6-mers with embedding size of 128 and 256 considering only training and validation sets (90% of the high-quality data). The same dataset was used to create the index with the recent tool GSearch, using the default parameters: 16-mers for the size of the sketch, using the ProbMinHash algorithm.

For GSearch, using 24 cores, the maximum RAM usage was 156.8 GB, with 605 hrs of user time (elapsed time of 24:52 hrs). The size of the index is 75 GB.

For PANSPACE, the maximum RAM usage was 2.3 GB in both cases. With embedding size 128 the user time was 20:43 hrs (elapsed time 14:24), and with embedding size 256 the user time was 29:54 (elapsed time 20:44). With embedding size 128, the index uses 817 MB of disk space, and the encoder 567 KB. With embedding size 256, the index uses 1.6 GB, and the encoder 1.1 MB. The labels (same list for both) uses 62 MB.

6.4.3. Quality of queries to the index

This experiment aims to assess the quality of queries against the index built on top of the train and validation sets from the 2MM dataset.

The dataset was split into training, validation, and test sets, in a proportion of 80:10:10. The test set is used to query the index built on top of the 90% of the dataset used in the training phase, as described in the previous section.

Given a query assembly, we retrieve the n=11 assemblies and their labels for which their embeddings are closer (in Euclidean distance) to the embedding of the query. The label assigned to the query is the most common label among the labels in the query result, if the query has two different labels with the same frequency (*e.g.* 5 of species X, and 5 of species Y), we label it as *unrealiable* for the effects of this experiment.

For this experiment (and the ones that follow) we consider only high-quality assemblies, and the subset of species for which there are at least 13 assemblies so that at least 1 assembly can be part of the test set based on the split of the dataset and how we defined the classification based on the query results (11 assemblies are retrieved).

Precision, recall, and F1-score for the 30 more represented species in the dataset are reported in Table 6.6. We can observe that all F1-scores reported among these species are above 0.97, except for Campylobacter coli reporting an F1-score of 0.9. These are notable results considering how imbalanced is the dataset, suggesting that the metric learning with the triplet loss successfully handles this problem.

Species	Precision	Recall	F1-score	Assemblies
Salmonella enterica	1	1	1	53324
Escherichia coli	1	1	1	31136
Mycobacterium tuberculosis	1	1	1	13215
Staphylococcus aureus	0.999	1	1	10366
Streptococcus pneumoniae	0.992	1	0.996	9508
Campylobacterd jejuni	0.939	1	0.968	8041
Listeria monocytogenes	0.999	1	1	5907
Klebsiella pneumoniae	0.983	1	0.992	5665
Neisseria gonorrhoeae	1	1	1	4358
Neisseria meningitidis	1	1	1	3449
Streptococcus pyogenes	1	1	1	3173
Campylobacterd coli	1	0.823	0.903	2958
Enterococcusb faecium	0.970	1	0.985	2747
Clostridioides difficile	0.999	1	0.999	2617
Pseudomonas aeruginosa	1	1	1	2486
Streptococcus agalactiae	1	1	1	2286
Acinetobacter baumannii	0.992	0.999	0.996	1430
Haemophilus influenzae	0.999	1	1	1082
Vibrio cholerae	1	1	1	1058
Enterococcus faecalis	1	1	1	794
Bordetella pertussis	1	1	1	715
Vibrio parahaemolyticus	1	1	1	713
Enterobacter hormaechei	0.997	0.998	0.998	619
Staphylococcus epidermidis	1	0.995	0.997	569
Mycobacterium abscessus	0.998	1	0.999	532
Legionella pneumophila	0.959	1	0.979	519
Burkholderia mallei	0.988	1	0.994	509
Salmonella houtenae	0.995	0.995	0.995	366
Staphylococcus pseudintermedius	0.981	1	0.991	365
Salmonella diarizonae	1	0.972	0.986	357

Table 6.6: **Classification metrics 30 most represented species** We show the 30 species more represented in 2MM dataset. Precision, recall, and F1-score are reported for each species. The number of assemblies in the test set is reported in the last column. The predicted label is computed as the most frequent label of the query result (of size 11).

The weighted average and macro average of the precision, recall, and F1-score are reported at the level of species and at the level of genus for `PANSPACE` and `GSearch` in Table 6.7 and Table 6.8, respectively. For `PANSPACE` at the level of species, the macro average F1-score is 0.87, and at the level of genus, is 0.99, suggesting that misclassified assemblies are not random but taxonomically consistent. We also report results for the new tool `GSearch` for comparison, which present better results at the level of species, but we recall that its index uses 16-mers and is 75× larger than ours.

		Precision	Recall	F1-score
`PANSPACE`	Weighted Average	0.992	0.992	0.991
	Macro Average	0.895	0.871	0.874
`GSearch`	Weighted Average	1	1	1
	Macro Average	0.974	0.972	0.971

Table 6.7: Average Precision, Recall, and F1-score at species level for the test set (184,512 assemblies). For `PANSPACE` using 6-mers and a model with embedding size 128, and `GSearch` with 16-mers for the size of the sketch.

		Precision	Recall	F1-score
`PANSPACE`	Weighted Average	1	1	1
	Macro Average	0.992	0.989	0.990
`GSearch`	Weighted Average	1	1	1
	Macro Average	0.981	0.981	0.980

Table 6.8: Average Precision, Recall, and F1-score at genus level for the entire dataset. The total number of assemblies in the test set is in the last column

6.4.4. Distance correlation with ANI

In this experiment, we aim to assess how the Euclidean distances between the embedding representation of assemblies correlate to the ANI values between the assemblies. We also include `GAMBIT` in our analyses (it computes a Jaccard-based distance between assemblies based on k-mers), since has been shown to correlate with ANI.

We first compare `fastANI`, `PANSPACE`, and `GAMBIT` distances, by randomly selecting 1000 assemblies from the 30 species. We compute all-vs-all distances for all of them. Still, since `fastANI` only reports ANI values close to 80% (in our case, the minimum ANI value reported was 73.4%), we restrict our analysis to 59.593.343 pairwise distances. In Table 6.9 we compute the Spearman correlation between distances computed with `fastANI`, `GAMBIT`, and `PANSPACE`. Results show that both, `PANSPACE` and `GAMBIT` strongly correlate (Spearman correlation < -0.7) with ANI distance.

These values suggest that as the `PANSPACE` distance decreases (indicating greater genomic similarity), the ANI values tend to increase, which is consistent with the expected relationship between these two measures of genomic relatedness. The same reasoning applies to `GAMBIT` distances compared to ANI.

	`fastANI-PANSPACE`	`fastANI-GAMBIT`	`PANSPACE-GAMBIT`
Spearman correlation	-0.745	-0.966	0.756

Table 6.9: **Spearman correlation between distances** computed by `fastANI`, `GAMBIT` and `PANSPACE`. A random selection of 1000 sequences for the 30 species with more assemblies was selected. An all vs all pairwise distance was computed. We restricted our analysis to the 59.593.343 pairwise distances successfully returned by `fastANI` (the minimum ANI value reported was 73.4). All values suggest a strong correlation between the three distance measures, In all cases, p-values were 0.0 (most likely due to large sample size, or computational precision).

Secondly, we explore how assemblies cluster themselves by using `PANSPACE` and `GAMBIT` distances. We randomly selected one assembly for each of the 30 species we selected to compare distances and create a cluster map for each tool. In Figure 6.5 we show the cluster map for `GAMBIT` and in Figure 6.6 the cluster map for `PANSPACE`. We observe that groups of species tend to be clustered by genus level.

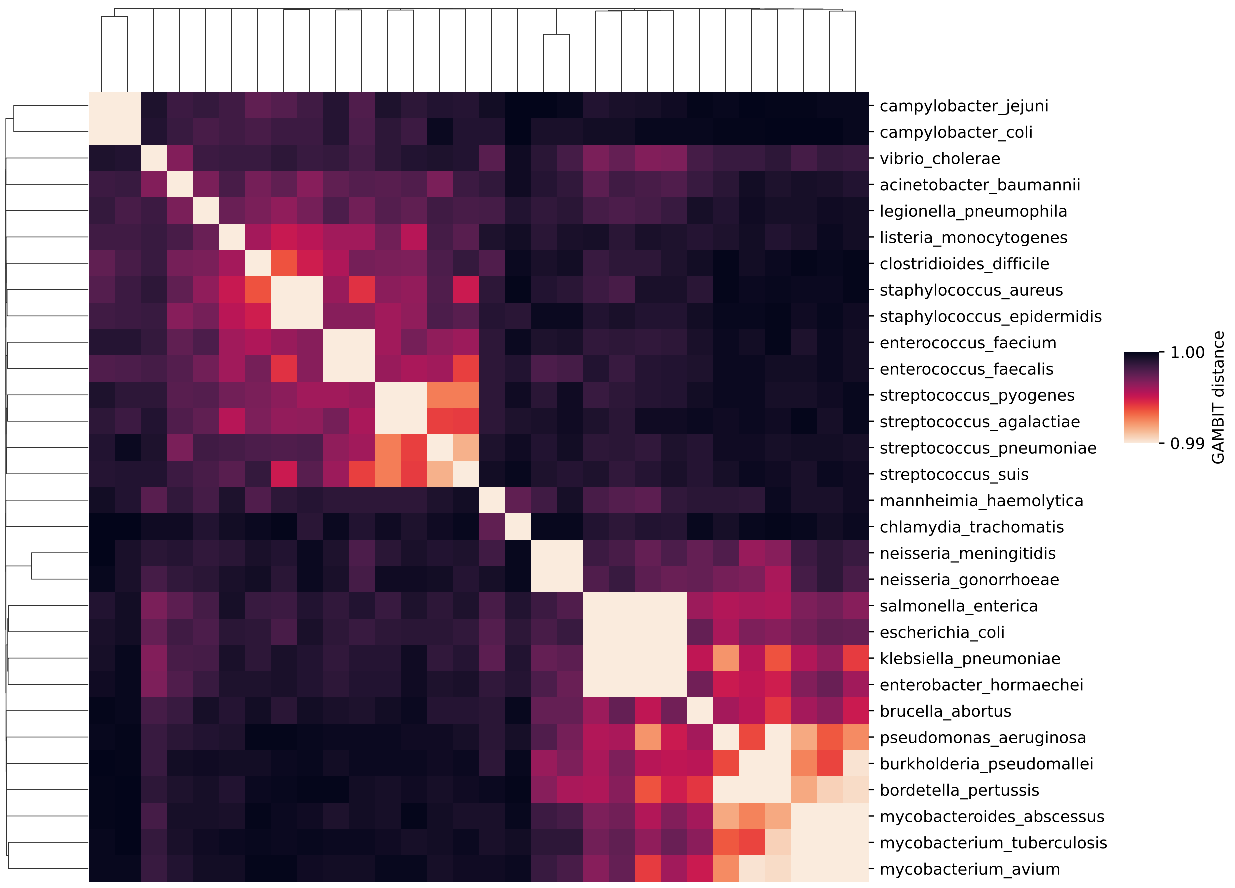


Figure 6.5: **Cluster map** GAMBIT. We randomly selected one assembly of 30 species in the 2MM dataset. We can observe that species tend to be clustered by genus level, most notably (from top to bottom) Mycobacterium, Neisseria, Campylobacter, and Staphylococcus. GAMBIT distances are in the range $[0, 1]$, we chose different ranges for colors for visualization purposes.

6.4.5. Discussion

In this work, we have shown PANSPACE, a deep-learning-based tool with two architectures, AutoencoderFCGR and CNNFCGR, for unsupervised and supervised learning, respectively, built on top of convolutional layers that exploits the FCGR representation of assemblies, leading to small models that are accurate and efficient to train.

By using embedding representations of assemblies given by the output of these models, we have shown a pipeline for the identification of outliers and mislabeled assemblies.

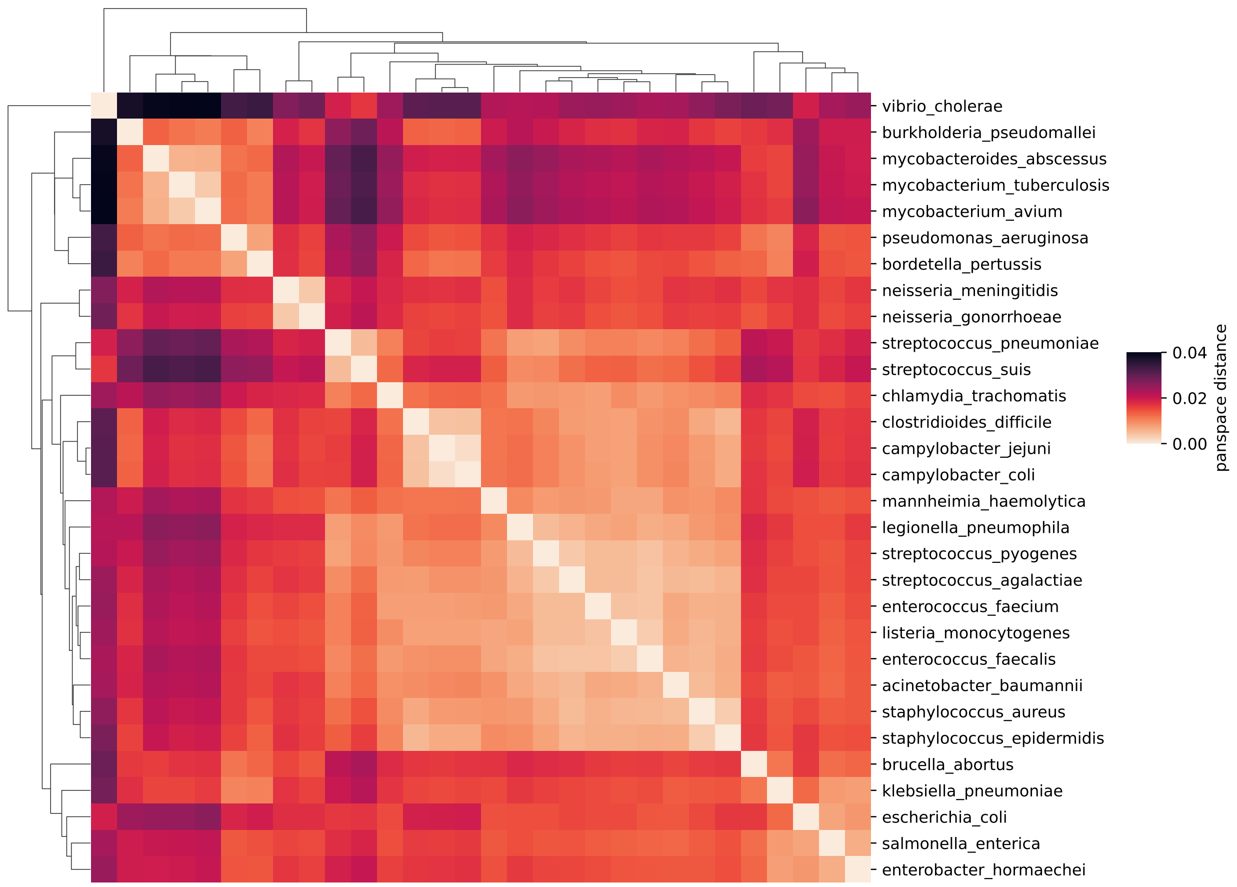


Figure 6.6: **Cluster map** `PANSPACE`. We randomly selected one assembly of 30 species in the 2MM dataset. We can observe that species tend to be clustered by genus level, most notably (from top to bottom) Mycobacterium, Neisseria, Campylobacter, and Staphylococcus, except for Streptococcus genus which is split into two groups. `PANSPACE` distances are in the range $[0, 2]$, we chose different ranges for colors for visualization purposes.

More importantly, the embedding representations of assemblies serve as the basis for defining an extremely compressed and efficient index to handle large bacterial datasets. The Euclidean distance between embeddings of assemblies correlates with ANI values between the assemblies. Our index, comprising more than 1.8 million assemblies is roughly 1 GB of disk space and requires less than 5 GB of RAM to be queried and trained.

We have shown that we can tackle an imbalanced dataset with the use of metric learning and the triplet loss, leading to a taxonomy-aware classification model at the level of genus and species. We believe that results can be improved by defining better strategies for mini-batch selection when training the models, and by cleaning the dataset, *i.e.* removing outliers, and fixing label issues by applying the data curation pipeline with models trained with metric learning. Additionally, we can explore other metric learning approaches like proxy-based losses, which have been shown to excel in this framework compared to pair-based losses (the ones used here). Incrementing the dimension of the FCGR (by selecting larger k-mers) and the embedding size would also help to improve results, but at the expense of incrementing the index and encoder sizes.

To the best of our knowledge, this is the first approach that deals with genomic sequences in a continuous space.

6.5. Availability of data and materials

For experiments on bacterial data, results and experiments can be downloaded from https://zenodo.org/records/14004182, and the codes to replicate the experiments at https://github.com/pg-space/panspace-paper.

Chapter 7
Future work

In this thesis, we present a novel approach for constructing customized variation graphs from a multiple sequence alignment (MSA), utilizing the concept of maximal blocks. Although the ILP formulation limits our analysis to smaller instances, it enables the exploration of various objective functions to guide graph construction based on different criteria. A potential future direction is to bypass the MSA by extending our ILP formulation to work with Maximal Exact Matches (MEMs) computed directly from the input sequences, similar to the use of maximal blocks derived from the MSA.

We have explored the study of variation graphs as discrete manifolds, proposing the basis for addressing the alignment between variation graphs from a geometrical point of view. Experimental evaluation of our method remains as one of the priorities as future work. A natural direction for our approach is to model other pangenome graph representations as manifolds (*e.g.* de Bruijn Graphs, colored de Bruijn Graphs), by defining suitable probability distributions that encapsulate the information within these graph structures.

Deep learning combined with the FCGR representation of assemblies has proven to be effective at the pangenome scale, successfully analyzing viral and bacterial sequences. We anticipate that this approach could be extended to all kingdoms of life. Additionally, through metric learning, we demonstrate that imbalanced genomic datasets can be managed effectively and that embedding representations of assemblies offer a promising path for creating compact, fast-to-query indexes. A logical next step would be to adapt our approach to function at the read level, particularly in metagenomic contexts.

7.1. List of Publications The following publications were developed during the PhD.

1. Avila Cartes, J., Anand, S., Ciccolella, S., Bonizzoni, P., and Della Vedova, G. (2023). Accurate and fast clade assignment via deep learning and frequency chaos game representation. GigaScience, 12, giac119.

2. Avila Cartes, J., Bonizzoni, P., Ciccolella, S., Della Vedova, G., Denti, L., Didelot, X.,Monti, D.C. and Pirola, Y.(2024). RecGraph: recombination-aware alignment of sequences to variation graphs. Bioinformatics, 40(5), btae292.

3. Bonizzoni, P., Avila Cartes, J. E., Ciccolella, S., Della Vedova, G., and Denti, L. (2024). PangeBlocks: customized construction of pangenome graphs via maximal blocks. bioRxiv, 2024-09.

BIBLIOGRAPHY

[1] H. Tettelin, V. Masignani, M. J. Cieslewicz, C. Donati, D. Medini, N. L. Ward et al., *Genome analysis of multiple pathogenic isolates of Streptococcus agalactiae: Implications for the microbial "pan-genome"*, *Proceedings of the National Academy of Sciences* **102** (Sept., 2005) 13950–13955. 1

[2] T. C. P.-G. Consortium, *Computational pan-genomics: status, promises and challenges*, *Briefings in bioinformatics* **19** (2016) 118–135. 1

[3] R. M. Sherman, J. Forman, V. Antonescu, D. Puiu, M. Daya, N. Rafaels et al., *Assembly of a pan-genome from deep sequencing of 910 humans of african descent*, *Nature genetics* **51** (2019) 30–35. 1

[4] R. M. Colquhoun, M. B. Hall, L. Lima, L. W. Roberts, K. M. Malone, M. Hunt et al., *Pandora: nucleotide-resolution bacterial pan-genomics with reference graphs*, *Genome biology* **22** (2021) 1–30. 1, 3, 7, 24, 56

[5] S. F. Zanini, P. E. Bayer, R. Wells, R. J. Snowdon, J. Batley, R. K. Varshney et al., *Pangenomics in crop improvement—from coding structural variations to finding regulatory variants with pangenome graphs*, *The Plant Genome* **15** (2022) e20177. 2

[6] A. S. Leonard, D. Crysnanto, X. M. Mapel, M. Bhati and H. Pausch, *Graph construction method impacts variation representation and analyses in a bovine super-pangenome*, *Genome Biology* **24** (2023) 124. 2, 3, 24, 56

[7] G. Hickey, J. Monlong, J. Ebler, A. Novak, J. M. Eizenga, Y. Gao et al., *Pangenome graph construction from genome alignment with minigraph-cactus*, *bioRxiv* (2022) 2022–10. 2, 3, 7, 24, 56

[8] F. Andreace, P. Lechat, Y. Dufresne and R. Chikhi, *Comparing methods for constructing and representing human pangenome graphs*, *Genome biology* **24** (2023) 274. 2

[9] J. Sirén, P. Eskandar, M. T. Ungaro, G. Hickey, J. M. Eizenga, A. M. Novak et al., *Personalized pangenome references*, *Nature Methods* (2024) 1–7. 2

[10] M. Rautiainen and T. Marschall, *Graphaligner: rapid and versatile sequence-to-graph alignment*, *Genome biology* **21** (2020) 253. 2, 24, 27, 40

[11] J. Avila Cartes, P. Bonizzoni, S. Ciccolella, G. Della Vedova, L. Denti, X. Didelot et al., *Recgraph: recombination-aware alignment of sequences to variation graphs*, *Bioinformatics* **40** (2024) btae292. 2

[12] J. A. Baaijens, P. Bonizzoni, C. Boucher, G. Della Vedova, Y. Pirola, R. Rizzi et al., *Computational graph pangenomics: a tutorial on data structures and their applications*, *Natural Computing* (2022) 1–28. 3, 7, 8, 24

[13] E. Garrison, J. Sirén, A. M. Novak, G. Hickey, J. M. Eizenga, E. T. Dawson et al., *Variation graph toolkit improves read mapping by representing genetic variation in the reference*, *Nature biotechnology* **36** (2018) 875–879. 3, 7, 8, 24, 56

[14] H. Li, X. Feng and C. Chu, *The design and construction of reference pangenome graphs with minigraph*, *Genome Biology* **21** (Dec., 2020) 265. 3, 7, 24, 56

[15] E. Garrison, A. Guarracino, S. Heumos, F. Villani, Z. Bao, L. Tattini et al., *Building pangenome graphs*, *bioRxiv* (2023) 2023–04. 3, 7, 24, 56

[16] D. Crysnanto, A. Leonard and H. Pausch, *Comparison of methods for building pangenome graphs*, in *Proceedings of 12th World Congress on Genetics Applied to Livestock Production (WCGALP) Technical and species orientated innovations in animal breeding, and contribution of genetics to solving societal challenges*, pp. 1066–1069, Wageningen Academic Publishers, 2022. 3, 24, 56

[17] N. Rizzo, M. Equi, T. Norri and V. Mäkinen, *Elastic founder graphs improved and enhanced*, *Theoretical Computer Science* **982** (2024) 114269. 3, 7, 24, 25, 56

[18] C.-C. Ni, Y.-Y. Lin, J. Gao and X. Gu, *Network alignment by discrete ollivier-ricci flow*, in *Graph Drawing and Network Visualization: 26th International Symposium, GD 2018, Barcelona, Spain, September 26-28, 2018, Proceedings*, pp. 447–462, Springer, 2018. 4, 57, 60, 61, 62, 63, 68

[19] X. Lai, S. Bai and Y. Lin, *Normalized discrete ricci flow used in community detection*, *Physica A: Statistical Mechanics and its Applications* **597** (2022) 127251. 4, 62, 63

[20] M. Hunt, L. Lima, W. Shen, J. Lees and Z. Iqbal, *Allthebacteria-all bacterial genomes assembled, available and searchable*, *bioRxiv* (2024) 2024–03. 4, 99, 100, 103, 117

[21] S. F. Altschul, W. Gish, W. Miller, E. W. Myers and D. J. Lipman, *Basic local alignment search tool*, *Journal of molecular biology* **215** (1990) 403–410. 4, 99

[22] K. Břinda, L. Lima, S. Pignotti, N. Quinones-Olvera, K. Salikhov, R. Chikhi et al., *Efficient and robust search of microbial genomes via phylogenetic compression*, *bioRxiv* (2023) 2023–04. 4, 102

[23] W. Shen and Z. Iqbal, *Lexicmap: efficient sequence alignment against millions of prokaryotic genomes*, *bioRxiv* (2024) 2024–08. 4, 100, 103

[24] D. E. Wood, J. Lu and B. Langmead, *Improved metagenomic analysis with kraken 2*, *Genome biology* **20** (2019) 1–13. 4, 103, 117

[25] J. Lumpe, L. Gumbleton, A. Gorzalski, K. Libuit, V. Varghese, T. Lloyd et al., *Gambit (genomic approximation method for bacterial identification and tracking): A methodology to rapidly leverage whole genome sequencing of bacterial isolates for clinical identification*, *Plos one* **18** (2023) e0277575. 4, 101, 102

[26] J. P. Meier-Kolthoff and M. Göker, *Tygs is an automated high-throughput platform for state-of-the-art genome-based taxonomy*, *Nature communications* **10** (2019) 2182. 4, 100, 101

[27] Q. Liang, C. Liu, R. Xu, M. Song, Z. Zhou, H. Li et al., *fidbac: a platform for fast bacterial genome identification and typing*, *Frontiers in Microbiology* **12** (2021) 723577. 4, 101

[28] D. Kim, L. Song, F. P. Breitwieser and S. L. Salzberg, *Centrifuge: rapid and sensitive classification of metagenomic sequences*, *Genome research* **26** (2016) 1721–1729. 4, 103

[29] L. Song and B. Langmead, *Centrifuger: lossless compression of microbial genomes for efficient and accurate metagenomic sequence classification*, *Genome Biology* **25** (2024) 106. 4, 103

[30] J. Shaw and Y. W. Yu, *Rapid species-level metagenome profiling and containment estimation with sylph*, *Nature Biotechnology* (2024) 1–12. 4, 117

[31] B. Verma and J. Parkinson, *Hitaxon: a hierarchical ensemble framework for taxonomic classification of short reads*, *Bioinformatics Advances* **4** (2024) vbae016. 5, 103

[32] S. Kutuzova, M. Nielsen, P. Piera, J. N. Nissen and S. Rasmussen, *Taxometer: Improving taxonomic classification of metagenomics contigs*, *Nature Communications* **15** (2024) 8357. 5, 103

[33] Q. Liang, P. W. Bible, Y. Liu, B. Zou and L. Wei, *Deepmicrobes: taxonomic classification for metagenomics with deep learning*, *NAR Genomics and Bioinformatics* **2** (2020) lqaa009. 5, 103

[34] H. J. Jeffrey, *Chaos game representation of gene structure*, *Nucleic acids research* **18** (1990) 2163–2170. 5, 9

[35] P. J. Deschavanne, A. Giron, J. Vilain, G. Fagot and B. Fertil, *Genomic signature: characterization and classification of species assessed by chaos game representation of sequences.*, *Molecular biology and evolution* **16** (1999) 1391–1399. 5, 12

[36] J. Avila Cartes, P. Bonizzoni, S. Ciccolella, G. Della Vedova and L. Denti, *Pangeblocks: customized construction of pangenome graphs via maximal blocks*, *BMC bioinformatics* **25** (2024) 344. 6

[37] J. Avila Cartes, S. Anand, S. Ciccolella, P. Bonizzoni and G. Della Vedova, *Accurate and fast clade assignment via deep learning and frequency chaos game representation*, *GigaScience* **12** (2023) giac119. 6

[38] H. F. Löchel and D. Heider, *Chaos game representation and its applications in bioinformatics*, *Computational and Structural Biotechnology Journal* **19** (2021) 6263–6271. 10

[39] S. Vinga, A. M. Carvalho, A. P. Francisco, L. M. Russo and J. S. Almeida, *Pattern matching through chaos game representation: bridging numerical and discrete data structures for biological sequence analysis*, *Algorithms for Molecular Biology* **7** (2012) 1–12. 12

[40] C. M. Bishop and N. M. Nasrabadi, *Pattern recognition and machine learning*, vol. 4. Springer, 2006. 14, 115

[41] B. Kulis et al., *Metric learning: A survey*, *Foundations and Trends® in Machine Learning* **5** (2013) 287–364. 15

[42] F. Schroff, D. Kalenichenko and J. Philbin, *Facenet: A unified embedding for face recognition and clustering*, in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 815–823, 2015. 15, 108, 113

[43] R. Hadsell, S. Chopra and Y. LeCun, *Dimensionality reduction by learning an invariant mapping*, in *2006 IEEE computer society conference on computer vision and pattern recognition (CVPR'06)*, vol. 2, pp. 1735–1742, IEEE, 2006. 15, 113

[44] B. W. Matthews, *Comparison of the predicted and observed secondary structure of t4 phage lysozyme*, *Biochimica et Biophysica Acta (BBA)-Protein Structure* **405** (1975) 442–451. 17

[45] J. Gorodkin, *Comparing two k-category assignments by a k-category correlation coefficient*, *Computational biology and chemistry* **28** (2004) 367–374. 17

[46] G. Jurman and C. Furlanello, *A unifying view for performance measures in multi-class prediction*, *arXiv preprint arXiv:1008.2908* (2010) . 17

[47] P. J. Rousseeuw, *Silhouettes: A graphical aid to the interpretation and validation of cluster analysis*, *Journal of Computational and Applied Mathematics* **20** (1987) 53–65. 18

[48] T. Caliński and J. Harabasz, *A dendrite method for cluster analysis*, *Communications in Statistics-theory and Methods* **3** (1974) 1–27. 18

[49] A. Schilling, A. Maier, R. Gerum, C. Metzner and P. Krauss, *Quantifying the separability of data classes in neural networks*, *Neural Networks* **139** (2021) 278–293. 19

[50] Y. LeCun, L. Bottou, Y. Bengio and P. Haffner, *Gradient-based learning applied to document recognition*, *Proceedings of the IEEE* **86** (1998) 2278–2324. 20, 112

[51] S. Ioffe, *Batch normalization: Accelerating deep network training by reducing internal covariate shift*, *arXiv preprint arXiv:1502.03167* (2015) . 20

[52] R. Kohavi, *A study of cross-validation and bootstrap for accuracy estimation and model selection*, *Morgan Kaufman Publishing* (1995) . 21

[53] L. Breiman, *Random forests*, *Machine learning* **45** (2001) 5–32. 21

[54] A. Fisher, C. Rudin and F. Dominici, *All models are wrong, but many are useful: Learning a variable's importance by studying an entire class of prediction models simultaneously*, *Journal of Machine Learning Research* **20** (2019) 1–81. 21

[55] K. Simonyan, A. Vedaldi and A. Zisserman, *Deep inside convolutional networks: Visualising image classification models and saliency maps*, 2013. 10.48550/ARXIV.1312.6034. 21, 82

[56] M. T. Ribeiro, S. Singh and C. Guestrin, *" why should i trust you?" explaining the predictions of any classifier*, in *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 1135–1144, 2016. 21, 96

[57] R. R. Selvaraju, A. Das, R. Vedantam, M. Cogswell, D. Parikh and D. Batra, *Grad-cam: Why did you say that? visual explanations from deep networks via gradient-based localization*, *CoRR* **abs/1610.02391** (2016) , [1610.02391]. 22, 96

[58] A. Shrikumar, P. Greenside and A. Kundaje, *Learning important features through propagating activation differences*, in *International conference on machine learning*, pp. 3145–3153, PMLR, 2017. 22, 96

[59] B. Letcher, M. Hunt and Z. Iqbal, *Gramtools enables multiscale variation analysis with genome graphs*, *Genome Biology* **22** (2021) 1–27. 24

[60] A. Guarracino, S. Heumos, S. Nahnsen, P. Prins and E. Garrison, *Odgi: understanding pangenome graphs*, *Bioinformatics* **38** (2022) 3319–3326. 24

[61] M. Equi, T. Norri, J. Alanko, B. Cazaux, A. I. Tomescu and V. Mäkinen, *Algorithms and complexity on indexing founder graphs*, *Algorithmica* **85** (2023) 1586–1623. 25

[62] T. Norri, B. Cazaux, D. Kosolobov and V. Mäkinen, *Linear time minimum segmentation enables scalable founder reconstruction*, *Algorithms for Molecular Biology* **14** (2019) 1–15. 25

[63] K. Hanauer, M. P. Seybold and J. Unterweger, *Covering Rectilinear Polygons with Area-Weighted Rectangles*, Dec., 2023. 10.48550/arXiv.2312.08540. 25

[64] M. Sinnl, *Exact and heuristic algorithms for the maximum weighted submatrix coverage problem*, *European Journal of Operational Research* **298** (May, 2022) 821–833. 25

[65] V. V. Vazirani, *Approximation Algorithms.* Springer, Berlin, Heidelberg, 2003, 10.1007/978-3-662-04565-7. 26

[66] J. Alanko, H. Bannai, B. Cazaux, P. Peterlongo and J. Stoye, *Finding all maximal perfect haplotype blocks in linear time*, *Algorithms for Molecular Biology* **15** (2020) 1–7. 26, 35

[67] P. Bonizzoni, G. Della Vedova, Y. Pirola, R. Rizzi and M. Sgrò, *Multiallelic maximal perfect haplotype blocks with wildcards via pbwt*, in *International Work-Conference on Bioinformatics and Biomedical Engineering*, pp. 62–76, Springer, 2023. 26

[68] D. Gusfield, *Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology.* Cambridge University Press. 27, 106

[69] E. Garrison and A. Guarracino, *Unbiased pangenome graphs*, *Bioinformatics* **39** (2023) btac743. 28

[70] R. Rizzi, M. Cairo, V. Mäkinen, A. I. Tomescu and D. Valenzuela, *Hardness of covering alignment: phase transition in post-sequence genomics*, *IEEE/ACM Transactions on Computational Biology and Bioinformatics* **16** (2018) 23–30. 56

[71] R. S. Hamilton, *Three-manifolds with positive ricci curvature*, *Journal of Differential geometry* **17** (1982) 255–306. 57

[72] G. Perelman, *The entropy formula for the ricci flow and its geometric applications*, *arXiv preprint math/0211159* (2002) . 57

[73] G. Perelman, *Ricci flow with surgery on three-manifolds*, *arXiv preprint math/0303109* (2003) . 57

[74] G. Perelman, *Finite extinction time for the solutions to the ricci flow on certain three-manifolds*, *arXiv preprint math/0307245* (2003) . 57

[75] Y. Ollivier, *Ricci curvature of metric spaces*, *Comptes Rendus Mathematique* **345** (2007) 643–646. 57

[76] Y. Ollivier, *Ricci curvature of markov chains on metric spaces*, *Journal of Functional Analysis* **256** (2009) 810–864. 57

[77] Y. Lin, L. Lu and S.-T. Yau, *Ricci curvature of graphs*, *Tohoku Mathematical Journal, Second Series* **63** (2011) 605–627. 57

[78] C.-C. Ni, Y.-Y. Lin, F. Luo and J. Gao, *Community detection on networks with ricci flow*, *Scientific reports* **9** (2019) 1–12. 57, 61

[79] J. Topping, F. Di Giovanni, B. P. Chamberlain, X. Dong and M. M. Bronstein, *Understanding over-squashing and bottlenecks on graphs via curvature*, *arXiv preprint arXiv:2111.14522* (2021) . 57

[80] L. Ambrosio, A. Bressan, D. Helbing, A. Klar, E. Zuazua, L. Ambrosio et al., *A user's guide to optimal transport*, *Modelling and Optimisation of Flows on Networks: Cetraro, Italy 2009, Editors: Benedetto Piccoli, Michel Rascle* (2013) 1–155. 58

[81] S. Bai, Y. Lin, L. Lu, Z. Wang and S.-T. Yau, *Ollivier ricci-flow on weighted graphs*, *American Journal of Mathematics* **146** (2024) 1723–1747. 74

[82] S. Khare, C. Gurry, L. Freitas, M. B. Schultz, G. Bach, A. Diallo et al., *Gisaid's role in pandemic response*, *China CDC Weekly* **3** (2021) 1049–1051. 79

[83] J. Hadfield, C. Megill, S. M. Bell, J. Huddleston, B. Potter, C. Callender et al., *Nextstrain: real-time tracking of pathogen evolution*, *Bioinformatics* **34** (Dec., 2018) 4121–4123. 79

[84] *Chapter 15 - Immunodeficiency*, in *Primer to the Immune Response (Second Edition)* (T. W. Mak, M. E. Saunders and B. D. Jett, eds.), pp. 377–421. Academic Cell, Boston, 2014. DOI. 79

[85] S. Ali, B. Bello, P. Chourasia, R. T. Punathil, Y. Zhou and M. Patterson, *Pwm2vec: An efficient embedding approach for viral host specification from coronavirus spike sequences*, *Biology* **11** (2022) . 79

[86] S. Ali and M. Patterson, *Spike2Vec: An Efficient and Scalable Embedding Approach for COVID-19 Spike Sequences*, in *2021 IEEE International Conference on Big Data (Big Data)*, pp. 1533–1540, Dec., 2021. DOI. 79

[87] S. Ali, B. Sahoo, N. Ullah, A. Zelikovskiy, M. Patterson and I. Khan, *A k-mer Based Approach for SARS-CoV-2 Variant Identification*, in *Bioinformatics Research and Applications* (Y. Wei, M. Li, P. Skums and Z. Cai, eds.), vol. 13064, pp. 153–164. Springer International Publishing, Cham, 2021. DOI. 79

[88] M. Cacciabue, P. Aguilera, M. I. Gismondi and O. Taboga, *Covidex: An ultrafast and accurate tool for sars-cov-2 subtyping*, *Infection, Genetics and Evolution* **99** (2022) 105261. 79, 84, 89, 91

[89] J. Singer, R. Gifford, M. Cotten and D. Robertson, *Cov-glue: A web application for tracking sars-cov-2 genomic variation*, *Preprints* (2020) . 79

[90] G. S. Randhawa, M. P. Soltysiak, H. El Roz, C. P. de Souza, K. A. Hill and L. Kari, *Machine learning using intrinsic genomic signatures for rapid classification of novel pathogens: Covid-19 case study*, *Plos one* **15** (2020) e0232391. 79

[91] D. C. Sengupta, M. D. Hill, K. R. Benton and H. N. Banerjee, *Similarity Studies of Corona Viruses through Chaos Game Representation*, *Comput Mol Biosci* **10** (Sep, 2020) 61–72. 80

[92] R. Touati, S. Haddad-Boubaker, I. Ferchichi, I. Messaoudi, A. E. Ouesleti, H. Triki et al., *Comparative genomic signature representations of the emerging covid-19 coronavirus and other coronaviruses: High identity and possible recombination between bat and pangolin coronaviruses*, *Genomics* **112** (2020) 4189–4202. 80

[93] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard et al., *Backpropagation applied to handwritten zip code recognition*, vol. 1, pp. 541–551, 1989. DOI. 80

[94] K. He, X. Zhang, S. Ren and J. Sun, *Deep residual learning for image recognition*, in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016. 81, 84

[95] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, 2017. 81

[96] S. M. Lundberg and S.-I. Lee, *A unified approach to interpreting model predictions*, in *Advances in Neural Information Processing Systems 30* (I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan et al., eds.), pp. 4765–4774. Curran Associates, Inc., 2017. 82

[97] J. Johnson, M. Douze and H. Jégou, *Billion-scale similarity search with GPUs*, *IEEE Transactions on Big Data* **7** (2019) 535–547. 98, 119

[98] C. Jain, L. M. Rodriguez-R, A. M. Phillippy, K. T. Konstantinidis and S. Aluru, *High throughput ani analysis of 90k prokaryotic genomes reveals clear species boundaries*, *Nature communications* **9** (2018) 5114. 100, 119

[99] G. Kim, D. Ahn, M. Kang, J. Park, D. Ryu, Y. Jo et al., *Rapid species identification of pathogenic bacteria from a minute quantity exploiting three-dimensional quantitative phase imaging and artificial neural network*, *Light: Science & Applications* **11** (2022) 190. 100

[100] R. Franco-Duarte, L. Černáková, S. Kadam, K. Kaushik, B. Salehi, C. Bevilacqua et al., *Advances in chemical and biological methods to identify microorganisms—from past to present. microorganisms 7: 130*, 2019. 100

[101] S.-M. Ha, C. K. Kim, J. Roh, J.-H. Byun, S.-J. Yang, S.-B. Choi et al., *Application of the whole genome-based bacterial identification system, truebac id, using clinical isolates that were not identified with three matrix-assisted laser desorption/ionization time-of-flight mass spectrometry (maldi-tof ms) systems*, *Annals of laboratory medicine* **39** (2019) 530–536. 100

[102] T. W. Battaglia, I. L. Mimpen, J. J. Traets, A. van Hoeck, L. J. Zeverijn, B. S. Geurts et al., *A pan-cancer analysis of the microbiome in metastatic cancer*, *Cell* **187** (2024) 2324–2335. 100

[103] C. R. Woese and G. E. Fox, *Phylogenetic structure of the prokaryotic domain: the primary kingdoms*, *Proceedings of the National Academy of Sciences* **74** (1977) 5088–5090. 100

[104] C. R. Woese, *Bacterial evolution*, *Microbiological reviews* **51** (1987) 221–271. 100

[105] R. Srinivasan, U. Karaoz, M. Volegova, J. MacKichan, M. Kato-Maeda, S. Miller et al., *Use of 16s rrna gene for identification of a broad range of clinically relevant bacterial pathogens*, *PloS one* **10** (2015) e0117617. 100

[106] K. Lagesen, P. Hallin, E. A. Rødland, H.-H. Stærfeldt, T. Rognes and D. W. Ussery, *Rnammer: consistent and rapid annotation of ribosomal rna genes*, *Nucleic acids research* **35** (2007) 3100–3108. 101

[107] R. L. Lindsey, L. M. Gladney, A. D. Huang, T. Griswold, L. S. Katz, B. A. Dinsmore et al., *Rapid identification of enteric bacteria from whole genome sequences using average nucleotide identity metrics*, *Frontiers in Microbiology* **14** (2023) . 101

[108] J. P. Meier-Kolthoff, J. S. Carbasse, R. L. Peinado-Olarte and M. Göker, *Tygs and lpsn: a database tandem for fast and reliable genome-based classification and nomenclature of prokaryotes*, *Nucleic acids research* **50** (2022) D801–D807. 101

[109] B. Ondov, T. Treangen, P. Melsted, A. Mallonee, N. Bergman and S. P. Koren, *(2016) mash: fast genome and metagenome distance estimation using minhash*, *Genome Biol* **17** 132. 101

[110] C. T. Parker, B. J. Tindall and G. M. Garrity, *International code of nomenclature of prokaryotes: prokaryotic code (2008 revision)*, *International journal of systematic and evolutionary microbiology* **69** (2019) S1–S111. 101

[111] D. A. Benson, M. Cavanaugh, K. Clark, I. Karsch-Mizrachi, J. Ostell, K. D. Pruitt et al., *Genbank*, *Nucleic acids research* **46** (2018) D41. 101

[112] K. Clark, I. Karsch-Mizrachi, D. J. Lipman, J. Ostell and E. W. Sayers, *Genbank*, *Nucleic acids research* **44** (2016) D67–D72. 101

[113] D. H. Parks, M. Imelfort, C. T. Skennerton, P. Hugenholtz and G. W. Tyson, *Checkm: assessing the quality of microbial genomes recovered from isolates, single cells, and metagenomes*, *Genome research* **25** (2015) 1043–1055. 101

[114] P. Yilmaz, L. W. Parfrey, P. Yarza, J. Gerken, E. Pruesse, C. Quast et al., *The silva and "all-species living tree project (ltp)" taxonomic frameworks*, *Nucleic acids research* **42** (2014) D643–D648. 101, 102

[115] J. L. B. Cineros and O. Lund, *Kmerfinderjs: a client-server method for fast species typing of bacteria over slow internet connections*, *BioRxiv* (2017) 145284. 102

[116] J. Zhao, J. P. Both, L. M. Rodriguez-R and K. T. Konstantinidis, *Gsearch: ultra-fast and scalable genome search by combining k-mer hashing with hierarchical navigable small world graphs*, *Nucleic Acids Research* **52** (2024) e74–e74. 102

[117] G. A. Blackwell, M. Hunt, K. M. Malone, L. Lima, G. Horesh, B. T. Alako et al., *Exploring bacterial diversity via a curated and searchable snapshot of archived dna sequences*, *PLoS biology* **19** (2021) e3001421. 102, 117

[118] D. E. Wood and S. L. Salzberg, *Kraken: ultrafast metagenomic sequence classification using exact alignments*, *Genome biology* **15** (2014) 1–12. 103

[119] J.-U. Ulrich and B. Y. Renard, *Fast and space-efficient taxonomic classification of long reads with hierarchical interleaved xor filters*, *Genome Research* (2024) gr–278623. 103

[120] P. T. West, A. J. Probst, I. V. Grigoriev, B. C. Thomas and J. F. Banfield, *Genome-reconstruction for eukaryotes from complex natural microbial communities*, *Genome research* **28** (2018) 569–580. 103

[121] M. Karlicki, S. Antonowicz and A. Karnkowska, *Tiara: deep learning-based classification system for eukaryotic sequences*, *Bioinformatics* **38** (2022) 344–350. 103

[122] L. Romeijn, A. Bernatavicius and D. Vu, *Mycoai: Fast and accurate taxonomic classification for fungal its sequences*, *Molecular Ecology Resources* (2024) e14006. 103

[123] C. Northcutt, L. Jiang and I. Chuang, *Confident learning: Estimating uncertainty in dataset labels*, *Journal of Artificial Intelligence Research* **70** (2021) 1373–1411. 104, 109, 111

[124] J. Kuan and J. Mueller, *Back to the basics: Revisiting out-of-distribution detection baselines*, *arXiv preprint arXiv:2207.03061* (2022) . 104, 105, 118

[125] D. Hendrycks and K. Gimpel, *A baseline for detecting misclassified and out-of-distribution examples in neural networks*, *arXiv preprint arXiv:1610.02136* (2016) . 105

[126] H. Li and R. Durbin, *Fast and accurate short read alignment with burrows–wheeler transform*, *bioinformatics* **25** (2009) 1754–1760. 106

[127] T. Bingmann, P. Bradley, F. Gauger and Z. Iqbal, *Cobs: a compact bit-sliced signature index*, in *String Processing and Information Retrieval: 26th International Symposium, SPIRE 2019, Segovia, Spain, October 7–9, 2019, Proceedings 26*, pp. 285–303, Springer, 2019. 106

[128] P. Indyk and R. Motwani, *Approximate nearest neighbors: towards removing the curse of dimensionality*, in *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pp. 604–613, 1998. 107

[129] Y. A. Malkov and D. A. Yashunin, *Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs*, *IEEE transactions on pattern analysis and machine intelligence* **42** (2018) 824–836. 107

[130] M. D. Zeiler and R. Fergus, *Stochastic pooling for regularization of deep convolutional neural networks*, *arXiv preprint arXiv:1301.3557* (2013) . 112

[131] X. Glorot, A. Bordes and Y. Bengio, *Deep sparse rectifier neural networks*, in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pp. 315–323, JMLR Workshop and Conference Proceedings, 2011. 114

[132] J. Lu, F. P. Breitwieser, P. Thielen and S. L. Salzberg, *Bracken: estimating species abundance in metagenomics data*, *PeerJ Computer Science* **3** (2017) e104. 117

[133] K. H. Schleifer, *Classification of bacteria and archaea: past, present and future*, *Systematic and applied microbiology* **32** (2009) 533–542.

[134] R. Rosselló-Mora and R. Amann, *The species concept for prokaryotes*, *FEMS microbiology reviews* **25** (2001) 39–67.

[135] M. Richter and R. Rosselló-Móra, *Shifting the genomic gold standard for the prokaryotic species definition*, *Proceedings of the National Academy of Sciences* **106** (2009) 19126–19131.

[136] L. Wayne, D. Brenner, R. Colwell, P. Grimont, O. Kandler, M. Krichevsky et al., *Report of the ad hoc committee on reconciliation of approaches to bacterial systematics*, *International Journal of Systematic and Evolutionary Microbiology* **37** (1987) 463–464.

[137] H. Li, *Minimap2: pairwise alignment for nucleotide sequences*, *Bioinformatics* **34** (2018) 3094–3100.

[138] J. P. Meier-Kolthoff, A. F. Auch, H.-P. Klenk and M. Göker, *Genome sequence-based species delimitation with confidence intervals and improved distance functions*, *BMC bioinformatics* **14** (2013) 1–14.

[139] J. Goris, K. T. Konstantinidis, J. A. Klappenbach, T. Coenye, P. Vandamme and J. M. Tiedje, *Dna–dna hybridization values and their relationship to whole-genome sequence similarities*, *International journal of systematic and evolutionary microbiology* **57** (2007) 81–91.

[140] G. Blackwell, M. Hunt, K. Malone, L. Lima, G. Horesh, B. T. F. Alako et al., *Additional material for article 'Exploring bacterial diversity via a curated and searchable snapshot of archived DNA sequences'*, .

[141] G. E. Hinton and R. R. Salakhutdinov, *Reducing the dimensionality of data with neural networks*, *science* **313** (2006) 504–507.

[142] M. Kokot, M. Długosz and S. Deorowicz, *Kmc 3: counting and manipulating k-mer statistics*, *Bioinformatics* **33** (2017) 2759–2761.

[143] J. Avila Cartes, "Complexcgr: encodings and image representations based on the chaos game representation of dna.." https://github.com/AlgoLab/ComplexCGR, 2024.

[144] J. Avila Cartes, "panspace: Embedding-based indexing for compact storage, rapid querying, and curation of bacterial pan-genomes.." https://github.com/pg-space/panspace, 2024.

[145] H. Iqbal, "Plotneuralnet: Latex code for making neural networks diagrams.." https://github.com/HarisIqbal88/PlotNeuralNet/tree/v1.0.0, 2024.

[146] Z. J. Wang, R. Turko, O. Shaikh, H. Park, N. Das, F. Hohman et al., *Cnn explainer: learning convolutional neural networks with interactive visualization*, *IEEE Transactions on Visualization and Computer Graphics* **27** (2020) 1396–1406.

[147] N. A. O'Leary, E. Cox, J. B. Holmes, W. R. Anderson, R. Falk, V. Hem et al., *Exploring and retrieving sequence and metadata for species across the tree of life with ncbi datasets*, *Scientific Data* **11** (2024) 732.

[148] F. Dabbaghie, S. K. Srikakulam, T. Marschall and O. V. Kalinina, *Panpa: generation and alignment of panproteome graphs*, *bioRxiv* (2023) 2023–01.

[149] Y. Ollivier et al., *A survey of ricci curvature for metric spaces and markov chains*, *Probabilistic approach to geometry* **57** (2010) 343–381.

[150] P. Wills and F. G. Meyer, *Metrics for graph comparison: a practitioner's guide*, *Plos one* **15** (2020) e0228728.

[151] L. Babai, *Graph isomorphism in quasipolynomial time*, in *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*, pp. 684–697, 2016.

[152] E. Saucan, R. Sreejith, R. Vivek-Ananth, J. Jost and A. Samal, *Discrete ricci curvatures for directed networks*, *Chaos, Solitons & Fractals* **118** (2019) 347–360.

[153] S. Bai, Y. Lin, L. Lu, Z. Wang and S.-T. Yau, *Ollivier ricci-flow on weighted graphs*, *arXiv preprint arXiv:2010.01802* (2020) .

[154] L. Cunha, Y. Diekmann, L. Kowada and J. Stoye, *Identifying maximal perfect haplotype blocks*, in *Brazilian Symposium on Bioinformatics*, pp. 26–37, Springer, 2018.

[155] R. Durbin, *Efficient haplotype matching and storage using the positional burrows–wheeler transform (pbwt)*, *Bioinformatics* **30** (2014) 1266–1272.

[156] V. Mäkinen and T. Norri, *Applying the positional burrows–wheeler transform to all-pairs hamming distance*, *Information Processing Letters* **146** (2019) 17–19.

[157] A. Naseri, D. Zhi and S. Zhang, *Multi-allelic positional burrows-wheeler transform*, *BMC bioinformatics* **20** (2019) 1–8.

[158] N. Rizzo and V. Mäkinen, *Indexable Elastic Founder Graphs of Minimum Height*, in *33rd Annual Symposium on Combinatorial Pattern Matching (CPM 2022)* (H. Bannai and J. Holub, eds.), vol. 223 of *Leibniz International Proceedings in Informatics (LIPIcs)*, (Dagstuhl, Germany), pp. 19:1–19:19, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. DOI.

[159] A. Sanaullah, D. Zhi and S. Zhang, *Haplotype threading using the positional burrows-wheeler transform*, in *22nd International Workshop on Algorithms in Bioinformatics (WABI 2022)*, Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2022.

[160] J. Sirén, J. Monlong, X. Chang, A. M. Novak, J. M. Eizenga, C. Markello et al., *Pangenomics enables genotyping of known structural variants in 5202 diverse genomes*, *Science* **374** (2021) abg8871.

[161] G. Hickey, J. Monlong, J. Ebler, A. Novak, J. M. Eizenga, Y. Gao et al., *Pangenome Graph Construction from Genome Alignment with Minigraph-Cactus*, Apr., 2023. 10.1101/2022.10.06.511217.

[162] F. J. Massey Jr, *The kolmogorov-smirnov test for goodness of fit*, *Journal of the American statistical Association* **46** (1951) 68–78.

[163] F. Sigaux, *Cancer genome or the development of molecular portraits of tumors*, *Bulletin de L'academie Nationale de Medecine* **184** (2000) 1441–7.

[164] B. H. Korte, J. Vygen, B. Korte and J. Vygen, *Combinatorial optimization*, vol. 1. Springer, 2011.

[165] D. Hermelin, D. Rawitz, R. Rizzi and S. Vialette, *The minimum substring cover problem*, *Information and Computation* **206** (2008) 1303–1312.

[166] I. Savnik, *Index data structure for fast subset and superset queries*, in *International Conference on Availability, Reliability, and Security*, pp. 134–148, Springer, 2013.

[167] Gurobi Optimization, LLC, *Gurobi Optimizer Reference Manual*, 2023.

[168] J. Avila Cartes, "pangeblocks." https://github.com/AlgoLab/pangeblocks, 2024.

[169] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard et al., *Backpropagation applied to handwritten zip code recognition*, *Neural computation* **1** (1989) 541–551.

[170] A. Krizhevsky, I. Sutskever and G. E. Hinton, *Imagenet classification with deep convolutional neural networks*, in *Advances in Neural Information Processing Systems* (F. Pereira, C. Burges, L. Bottou and K. Weinberger, eds.), vol. 25, Curran Associates, Inc., 2012.

[171] H. F. Löchel, D. Eger, T. Sperlea and D. Heider, *Deep learning on chaos game representation for proteins*, *Bioinformatics* **36** (06, 2019) 272–279, [https://academic.oup.com/bioinformatics/article-pdf/36/1/272/31813814/btz493.pdf].

[172] K. Dick and J. R. Green, *Chaos game representations amp; deep learning for proteome-wide protein prediction*, in *2020 IEEE 20th International Conference on Bioinformatics and Bioengineering (BIBE)*, pp. 115–121, 2020. DOI.

[173] Y. Ren, T. Chakraborty, S. Doijad, L. Falgenhauer, J. Falgenhauer, A. Goesmann et al., *Prediction of antimicrobial resistance based on whole-genome sequencing and machine learning*, *Bioinformatics* **38** (10, 2021) 325–334, [https://academic.oup.com/bioinformatics/article-pdf/38/2/325/42543458/btab681.pdf].

[174] P. Millán Arias, F. Alipour, K. A. Hill and L. Kari, *Delucs: Deep learning for unsupervised clustering of dna sequences*, *PLOS ONE* **17** (01, 2022) 1–25.

[175] D. L. Davies and D. W. Bouldin, *A cluster separation measure*, *IEEE transactions on pattern analysis and machine intelligence* (1979) 224–227.

[176] L. Liu, H. Jiang, P. He, W. Chen, X. Liu, J. Gao et al., *On the variance of the adaptive learning rate and beyond*, *arXiv preprint arXiv:1908.03265* (2019) .

[177] M. Zhang, J. Lucas, J. Ba and G. E. Hinton, *Lookahead optimizer: k steps forward, 1 step back*, *Advances in neural information processing systems* **32** (2019) .