Jorge Miguel
Ferreira da Silva

Aproximações algorítmicas de informação em análise de dados

Algorithmic information approximations in data analysis

Jorge Miguel Ferreira da Silva

Aproximações algorítmicas de informação em análise de dados

Algorithmic information approximations in data analysis

Tese apresentada à Universidade de Aveiro para preenchimento dos requisitos necessários à obtenção do grau de Doutor em Engenharia Informática, desenvolvida sob orientação científica do Doutor Sérgio Guilherme Aleixo de Matos, Professor Auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro e co-orientação científica do Doutor Diogo Rodrigo Marques Pratas, Investigador Auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro.

Thesis presented to the University of Aveiro for fulfillment of the necessary requirements to obtain the degree of Doctor in Informatic Engineering, developed under scientific supervision of the Doctor Sérgio Guilherme Aleixo de Matos, Assistant Professor at the Department of Electronics, Telecommunications and Informatics at the University of Aveiro and scientific co-supervision of Doctor Diogo Rodrigo Marques Pratas, Assistant Researcher at the Department of Electronics, Telecommunications and Informatics at the University of Aveiro.

Trabalho financiado pela Fundação para a Ciência e a Tecnologia (FCT) pela bolsa de doutoramento SFRH/BD/141851/2018.

Work funded by the Portuguese Foundation for Science and Technology (FCT) through the doctoral grant SFRH/BD/141851/2018.

o júri / the jury

presidente / president

Doutora **Anabela Botelho Veloso**,
Professora Catedrática, Universidade de Aveiro.

Doutor **Mário Alexandre Teles de Figueiredo**,
Professor Catedrático, Universidade de Lisboa;

Doutor **André Osório e Cruz de Azerêdo Falcão**,
Professor Associado, Universidade de Lisboa;

Doutor **André Nuno Carvalho Souto**,
Professor Auxiliar, Universidade de Lisboa;

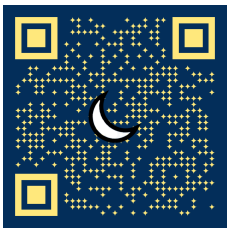
vogais / examiners committee

Doutor **Sérgio Guilherme Aleixo de Matos** (Orientador),
Professor Auxiliar, Universidade de Aveiro;

Doutor **Sónia Cristina Alexandre Gouveia**,
Investigadora Auxiliar, Universidade de Aveiro.

To my little Brother

"It's always darkest before the dawn."
– Thomas Fuller



"No pessimist ever discovered the secrets of the stars,
or sailed to an uncharted land,
or opened a new heaven to the human spirit."
– Helen Keller

"Life has a way of testing a person's will,
either by having nothing happen at all
or by having everything happen at once."
– Paulo Coelho

Acknowledgments

When I started this journey, I had a dream to make this world slightly better. Unfortunately, just when you think you have it all under control, life throws you a curve ball. The truth is that life can be challenging for everyone and make simple things hard. In my case, however, I was lucky enough to have good friends and kindhearted people in my life.
In that regard, a special thanks to Professor Sérgio Matos and researcher Diogo Pratas for allowing me to carry out my research project and for their guidance and support, which were essential to this work's success. I gratefully acknowledge the "Fundação para a Ciência e Tecnologia" (FCT) for making this thesis work possible through the grant **SFRH/BD/141851/2018**. I am also thankful to the Bioinformatics group at the Institute of Electronics and Informatics Engineering of Aveiro (IEETA) for providing me with equipment and guidance throughout this dissertation.
Finally, I express my deepest gratitude to all my mentors, family, and friends for their guidance, friendship, support, and patience. Particularly, I would like to thank:

- My loving parents and my partner Ana, my brother and Rufus who have supported me in every way possible. I can never repay what you have done for me;
- Professor Augusto Silva, who always mentored and guided me;
- Professor Carlos Costa, who introduced me to the bioinformatics group and always had my best interest at heart;
- All friends and colleagues from IEETA, which through the years, have had so much patience and taught me so much;
- My longtime friends, Pedro N., Saraiva, Miguel N., and Tiago S., thank you for the good times, conversations and outbursts;
- Rui Antunes, which gave me a helping hand every time I was having trouble with LaTeX;
- João Almeida, despite its rough attitude, was always there to help me when I needed it most;
- Tiago Almeida, which help me carry this heavy boulder to the finish line;
- God for giving me strength throughout this journey.

A final word to all of those who have helped me.
Saying thank you just does not seem enough. Without your help and emotional support, I wouldn't have finished this journey.
I hope to repay what you've done for me by helping others. I will use your selfless support as a model as I move forward.

Palavras-chave

Informação Algorítmico-Estatística, Complexidade Kolmogorov, Compressão de Dados, Genómica, Classificação de Dados.

Resumo

Apesar de serem altamente heterogéneos, todos os dados podem ser reduzidos a uma sequência de bits. A complexidade de Kolmogorov analisa os dados dessa maneira, medindo a complexidade de uma sequência de bits, ao determinar o comprimento de um dos programas mais pequenos que, quando executado, gera essa sequência de bits e pára.

Embora a complexidade de Kolmogorov não seja computável, ela pode ser aproximada e, como tal, é possível realizar análises de dados usando essas aproximações. Esta dissertação relaciona-se diretamente com este assunto, uma vez que estuda estas aproximações na descrição de diferentes tipos de dados, criando potenciais aplicações.

Primeiro, usamos medidas de aproximação de complexidade de Kolmogorov em dados genómicos. Demonstramos que o uso de compressores de dados específicos para quantificação da complexidade dos dados impacta profundamente a identificação taxonómica de genomas, a sua classificação e organização.

Em seguida, examinamos a aplicação destas medidas em objetos digitais bidimensionais, especificamente em pinturas artísticas. Usando estas medidas, desenvolvemos técnicas que podem ser valiosas para atribuição e validação de autoria de arte, categorização e organização de estilo de arte e explicação de conteúdo de arte.

Posteriormente, aplicamos essas medidas a dados gerados por Máquinas de Turing. Especificamente, usando estas medidas, investigamos a relação entre as complexidades algorítmicas e probabilísticas das fitas da Máquina de Turing. A complexidade é estudada globalmente por meio da investigação dos padrões gerados por Máquinas de Turing, criadas sequencialmente e localmente por meio de perfis de complexidade. Além disso, introduzimos um método para aumentar a complexidade probabilística mantendo a complexidade algorítmica.

Finalmente, usando o conhecimento do estudo efetuado à complexidade das fitas da Máquina de Turing, apresentamos uma metodologia que procura programas que geram aproximadamente a mesma sequência de dados que fornecemos ao programa.

Keywords Algorithmic-Statistical Information, Kolmogorov Complexity, Data Compression, Genomics, Data Classification.

Abstract Despite being highly heterogeneous, all data can be reduced to a string of bits. Kolmogorov complexity analyses data in this way, measuring a string's complexity by determining the length of a smallest program which, when executed, generates that string and halts.
Even though Kolmogorov complexity is noncomputable, it can be approximated, and as such, it is possible to perform data analysis using these approximations. This dissertation ties in directly with this subject since it studies Kolmogorov complexity approximations in data description and its potential applications.
First, we use Kolmogorov complexity approximation measures in genomic data. We demonstrate that using specific data compressors to quantify data complexity profoundly impacts genomic taxonomic identification, classification, and organization.
Afterwards, we examine the application of information-based measures in 2-dimensional digital objects, specifically artistic paintings. Using these measures, we developed techniques that can be valuable for art authorship attribution and validation, art style categorization and organization, and art content explanation.
Subsequently, we apply Kolmogorov approximations to data generated by Turing Machines. Specifically, using these measures, we investigate the relationship between the algorithmic and probabilistic complexities of the Turing Machine tapes. Complexity is studied globally by investigating probabilistic complexity patterns yielded by sequentially generated TMs, and locally using complexity profiles. Furthermore, we also introduce a method for increasing probabilistic complexity while retaining the same algorithmic complexity.
Finally, using the knowledge from studying Turing Machine tapes, we introduce a methodology that, given a string, searches for programs that generate approximately the same output.

Table of contents

List of figures

List of tables

List of abbreviations

A . . . . . adenine
A-T . . . . . adenine-thymine
AIT . . . . . Algorithmic Information Theory

BDM . . . . . Block Decomposition Method

C . . . . . cytosine
C-G . . . . . cytosine-guanine
CD . . . . . Conditional Compression Distance
CS . . . . . Computer Science
CTM . . . . . Coding Theorem Method

DNA . . . . . Deoxyribonucleic acid
dsDNA . . . . . double-stranded deoxyribonucleic acid
dsRNA . . . . . double-stranded ribonucleic acid

FCM . . . . . Finite-Context Model
FISH . . . . . Fluorescence In Situ Hybridization

G . . . . . guanine
GC . . . . . GC-Content
GNB . . . . . Gaussian Naive Bayes

ID . . . . . Information Distance
IRs . . . . . inverted repeats
ITRs . . . . . inverted terminal repeats

KNN . . . . . K-Nearest Neighbors

LDA . . . . . Discriminant Analysis

MDL . . . . . Minimum Description Length
MML . . . . . Minimum Message Length
mRNA . . . . . messenger ribonucleic acid

NC . . . . . Normalized Compression

NCD . . . . . Normalized Compression Distance
NID . . . . . Normalized Information Distance
NR . . . . . normalized redundancy
NRC . . . . . Normalized Relative Compression

ORFs . . . . . open reading frames

R . . . . . redundancy
RNA . . . . . Ribonucleic acid

SL . . . . . Sequence Length
ssDNA . . . . . single-stranded deoxyribonucleic acid
ssRNA . . . . . single-stranded ribonucleic acid
SVM . . . . . Support Vector Machine

T . . . . . thymine
TM . . . . . Turing Machine
TNTM . . . . . Total Number of Turing Machines
tRNA . . . . . transfer ribonucleic acid

U . . . . . uracil
ULS . . . . . Universal Levin Search

XGB . . . . . XGBoost

Chapter 1

Introduction

[a] Trinity- Jorge Miguel Silva, 2020.

"The man who moves a mountain
begins by carrying away small stones."
– Confucius, Confucius: The Analects

1.1 Overview

Humans are social creatures that rely heavily on information communication as a survival tool. After language development, the need to share information efficiently led humans to create written systems. These systems, unlike other modes of communication (such as a painting), use a finite and discrete set of symbols to express a concept [1, 2]. Any written language can be thought of in this way since messages are formed by combining and arranging symbols in specific patterns. Later, the continued improvement of communication led to the development of computing machines and information science.

The first well-documented mechanical computer was the difference engine by Charles Babbage in 1822, which gave rise to the analytical engine in 1837. This engine encapsulated most of the elements of modern computers [3, 4]. A century later, a universal calculating machine (Turing Machine) was introduced by Alan Turing [5]. During this time, Harry Nyquist, Ralph Hartley, and Claude Shannon helped lay the groundwork of information theory.

Interestingly, Computer Science can be defined as a discipline that studies complexity, information structures [6], and the algorithmic processes that describe and transform information [7]. These definitions reinforce the understanding that Computer Science (CS) has historical roots in information theory and can be seen as a generalization of information theory concerned not only with the transmission of information but also with its transformation and interpretation.

This dissertation ties in directly with this subject by studying Kolmogorov complexity approximations as data descriptors and describing novel applications for them. Furthermore, we will tackle one of the most fundamental questions in CS, *"What can be efficiently automated?"* [8], by creating prototypes of novel applications that use/are Kolmogorov complexity approximations.

1.2 Motivation

Gregory Chaitin described Algorithmic Information Theory (AIT) as the results of "putting Shannon's information theory and Turing's computability theory into a cocktail shaker and shaking vigorously. The basic idea is to measure the complexity of an object by the size in bits of the smallest program for computing it" [9]. An adequate description since AIT explores the relationship between computation and the information of its generated strings [10].

Informally, a string's information content is equal to the size of its most compressed, self-contained representation. A self-contained representation is a program that outputs the original string when executed.

Most lossless compression algorithms are limited to finding simple probabilistic regularities since they were designed for fast storage reduction. Although compressors' probabilistic nature has already been much studied, algorithmic modelling has yet to be fully

explored.

Given that lossless data compression is closely linked to the ideas of Minimum Message Length (MML) [11] and Kolmogorov complexity [12–14], it seems reasonable to incorporate algorithmic modelling into compressors. Nonetheless, noise and other characteristics of natural data reduce the efficiency of programs that measure information based on pure algorithmic schemes rather than those that use probabilistic methods.

This thesis explores this dynamic between algorithmic and probabilistic methods and how Kolmogorov complexity approximations can be used as descriptors of different data types. Specifically, we first analyse the application of these approximations in genomic sequences (1 dimension), in images of artistic paintings (2 dimensions), and finally on algorithmically generated data. The latter type of data is used to study the inversion problem and how we may be able to approximate a solution.

1.3 Methodology

Computer Science offers a wide variety of methodologies that can be applied when carrying out a research project [15, 8]. However, we opted to apply the experimental CS methodology approach in our work. This methodology is widely used in different fields such as machine learning and natural language processing [8], and since it is most effective with problems that require a complex software solution and to evaluate new solutions for problems [15, 8], it seems to be the most adequate methodology to fulfil the work we intend to develop.

This methodology can be divided into two phases. The first constitutes an exploratory phase when the researcher identifies the questions that should be asked in order to solve the problem and then identifies the concepts that facilitate creation of the solution to the problem. This is followed by the evaluation phase, when the researcher attempts to answer these questions using the concepts identified in the exploratory phase. This methodology usually culminates in the development of a prototype system that demonstrates the created solution is feasible [16].

In this work, the identified research question is: ***"Are current Kolmogorov complexity approximations efficient data descriptors and valuable in novel applications?"*** In turn, this question can be divided into a set of questions such as: *"Are Kolmorogov complexity approximations appropriate for different data types?"*; *"What examples of applications can be developed using Kolmogorov complexity approximations?"* and *"Is it computationally feasible to create an approximation solution to the inversion problem?"*

After some research, we hypothesized that it may be possible to answer these questions by applying Kolmogorov complexity approximation measures to data obtained from different sources. Using these methods, we will study the underlying structure of data and perform classification and identification tasks.

Information-based measures are approximations of the Normalized Information Dis-

tance, a *"universal similarity measure, and is an objective recursively invariant notion by the Church–Turing thesis"* [17]. Since these metrics do not use background knowledge from input data, they do not evaluate specific features. Consequently, they can potentially be applied to any type of digital data [18], and therefore, seem the most adequate to represent data and measure information similarity between strings.

Although some indications favour these hypotheses, we must conduct meticulous validation to demonstrate their validity. This will predominantly be done by comparing our method with other state-of-the-art solutions. Furthermore, we have created open-source code for the developed prototypes, which serve as a proof of concept of the developed work and allow the user to verify the validity of our experiments.

1.4 Objectives

This doctoral work investigates Kolmogorov complexity approximations in different settings. From a methodological standpoint, we will evaluate the applicability of Kolmogorov complexity approximations in different data types, showcasing their effectiveness as data descriptors.

As such, this work will include the following research tasks:

- Investigate the capability of Kolmogorov complexity approximations as data descriptors. For that purpose, we will analyse the behaviour of compression, and Block Decomposition Method measures in synthetic and natural data, with 1 and 2 dimensions, and from algorithmic and probabilistic sources.
- Investigate possible applications and use cases where Kolmogorov complexity approximation measures can be used to achieve state-of-the-art results in classification tasks.
- Investigate possible methods that create approximate solutions to the inverse problem.

1.5 Thesis Outline

This document is divided into seven chapters:

- Chapter 1 is the introduction.
- Chapter 2 gives a background to information theory, its approximations, and derived measures.
- Chapter 3 describes the complexity analysis of 1d structure strings.
- Chapter 4 describes the complexity analysis performed on 2d structure strings.

- Chapter 5 shows the developed work performed in probabilistic and algorithmic information.
- Chapter 6 showcases the developed work in determining a program approximating a given string.
- Finally, Chapter 7 summarizes the findings and future work of the thesis.

1.6 Contributions

International Journals

- **Jorge Miguel Silva**, Diogo Pratas, Tânia Caetano, Sérgio Matos. The complexity landscape of viral genomes. GigaScience, vol. 11, August 2022.
- **Jorge Miguel Silva**, Diogo Pratas, Rui Antunes, Sérgio Matos, Armando J. Pinho. Automatic analysis of artistic paintings using information-based measures. Pattern Recognition, p. 107864, February 2021.
- Diogo Pratas, **Jorge Miguel Silva**. Persistent minimal sequences of SARS-CoV-2. Bioinformatics, vol. btaa686, p. 4, August 2020.
- **Jorge Miguel Silva**, Eduardo Pinho, Sérgio Matos, Diogo Pratas. Statistical Complexity Analysis of Turing Machine Tapes with Fixed Algorithmic Complexity Using the Best-Order Markov Model. Entropy, vol. 22, no. 1, January 2020.
- Diogo Pratas, Morteza Hosseini, **Jorge Miguel Silva**, Armando J. Pinho. A reference-free lossless compression algorithm for DNA sequences using a competitive prediction of two classes of weighted models. Entropy, vol. 21, p. 1074, November 2019.

Conference Proceedings

- **Jorge Miguel Silva**, João Almeida. The value of compression for taxonomic identification, Proceedings of the 35th IEEE International Symposium on Computer-Based Medical Systems, CBMS, Shenzhen, China, p. 276-281, July 2022.
- **Jorge Miguel Silva**, Diogo Pratas, Tânia Caetano, Sérgio Matos. Feature-Based Classification of Archaeal Sequences Using Compression-Based Methods. Proceedings of the 10th Iberian Conference on Pattern Recognition and Image Analysis, IbPRIA, Aveiro, Portugal, p. 309-320, May 2022.
- **Jorge Miguel Silva**, Diogo Pratas, Tânia Caetano, Sérgio Matos. Archaea Taxonomic Classification. Proceedings of the 27th Portuguese Conference on Pattern Recognition, RecPad 2021, Évora, Portugal, November 2021.

- **Jorge Miguel Silva**, Diogo Pratas, Sérgio Matos. Comparison and Evaluation of Information-based Measures in Images. Proceedings of the 26th Portuguese Conference on Pattern Recognition, RecPad 2020, Évora, Portugal, October 2020.
- **Jorge Miguel Silva**, Diogo Pratas, Sérgio Matos. Evaluation of Statistical Complexity in Viral Genome Sequences. 25th Portuguese Conference on Pattern Recognition, RECPAD2019, Porto, Portugal, October 2019.

Open-source Software

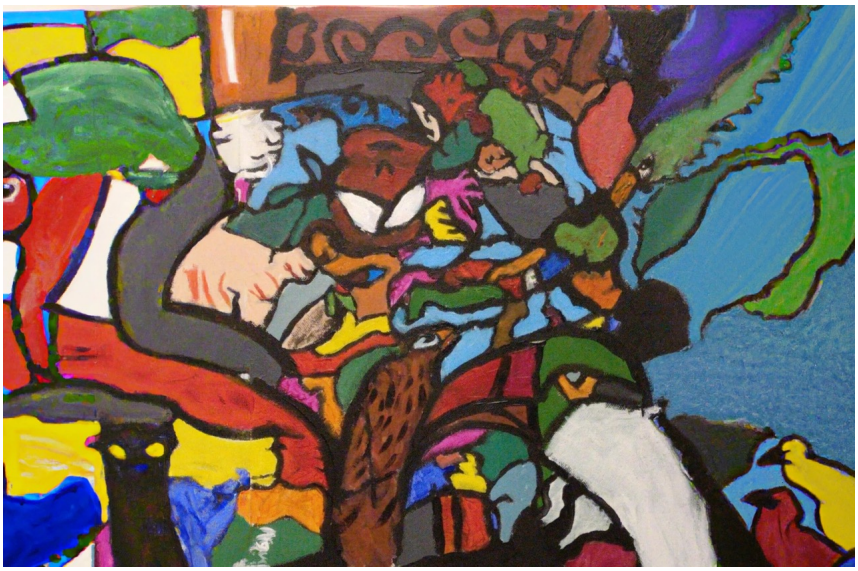
- TM Neural Finder Repository. Link: https://github.com/bioinformatics-ua/TM-Neural-Finder
- SPTTM Repository. Link: https://github.com/jorgeMFS/spttm;
- COMPressor tAxonomic ClassificaTion (C.O.M.P.A.C.T.) Repository. Link: https://github.com/bioinformatics-ua/COMPACT;
- Complexity ANalysis VirAl Sequences (C.A.N.V.A.S.) Repository. Link: https://github.com/jorgeMFS/canvas;
- Archaea Taxonomic Classification (ARCHAEA) Repository. Link: https://github.com/jorgeMFS/Archaea;
- Classification and identification of Archaea (ARCHAEA2) Repository. Link: https://github.com/jorgeMFS/Archaea2;
- Measuring probabilistic-algorithmic information of artistic paintings (PANTHER) Repository. Link: https://github.com/asilab/panther;
- Turing Machine Recreator (TMRecreator). Link: https://github.com/jorgeMFS/TMRecreator;
- TMCompression Repository. Link: https://github.com/asilab/TMCompression;
- Benchmark in ALgorithmic And Natural data ComprEssion (B.A.L.A.N.C.E.) Repository. Link: https://github.com/jorgeMFS/balance.git.

Websites

- PANTHER Website. Link: http://panther.web.ua.pt/;
- CANVAS Website. Link: https://asilab.github.io/canvas/.

Chapter 2

Background



[b]Animal Entropy- Jorge Miguel Silva, 2019.

"Information can only be acquired in two ways:
by choice or by chance."
– Joey Lawsin, Originemology

2.1 Contextualization

As mentioned in Chapter 1, this work aims to investigate Kolmogorov complexity approximations as data descriptors and exemplify potential novel applications.

Since data can take many forms and come from many sources, creating a suitable representation that accommodates these variations is challenging. Consequently, researchers tend to focus on specific niches with specific data to achieve better results. Contrarily, in this work, we use algorithmic Kolmogorov information approximations to describe and represent different types of data. To do this, we base ourselves on the assumption that making a lossless representation of the data's information simplifies the pattern inference process and the measurement of similarities between patterns. We will try to demonstrate this assumption throughout this dissertation using different data types and sources. We expect that aitll provide relevant information for pattern inference and enable data comparison. Based on this assumption, in this chapter, we will survey regarding information theory, AIT background, information-based measures, and its approximations.

2.2 Information theory and compression

Informally, information can be thought of as some message stored or transmitted using some medium. For instance, when painting, a message is being represented using a continuous set of patterns with a seemingly endless number of possible forms. However, the development of written systems created the need to divide our environment into a finite number of atomic units, which were expressed using symbols [1, 2]. Any written language can be thought of in this way since messages are formed by arranging symbols in specific patterns. This process is independent of the symbols used, the selecting process, or language, since information is just a choice from a pool of possible symbols (alphabet).

The need to create faster and more efficient ways of transmitting information across long distances created an engineering challenge: the increment of noise over long distances, making it unworkable to separate the signal from the noise [19]. As such, there was a need to create mechanisms capable of encoding the source message as a high-energy signal and decoding the transmitted message on the target. This issue was solved by implementing an encoder on the message's source and a decoder on the message's target (Figure 2.1). First, the encoder modified the source message information, adding redundancy. Afterwards, the decoder received the coded transmitted data and decoded the message, trying to replicate the source message. Information theory was created as a consequence of this challenge and the attempts to solve it.

2.2.1 Information theory

The first known attempts to quantify information were made by Nyquist (1924) [20] and Hartley (1928) [21]. Nyquist, interested in engineering aspects of telegraph signals, addressed the problem of quantifying "intelligence" and the "line speed" at which a commu-

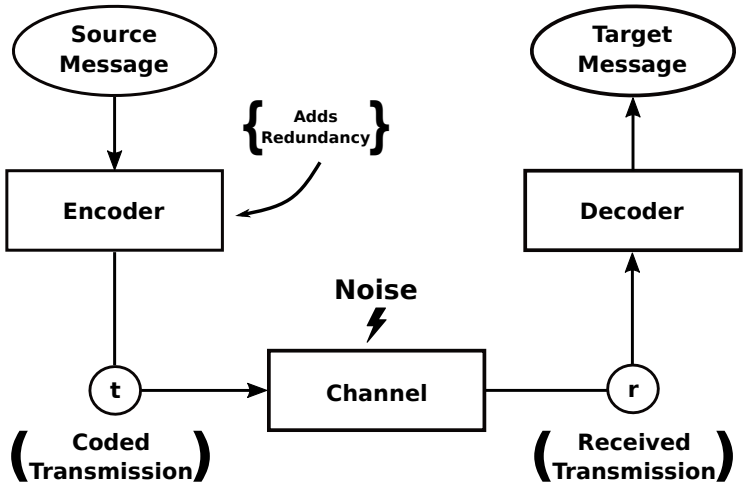


Figure 2.1: Transmission of information through a channel.

nication system can transmit it. Hartley introduced the notion of information, which could be quantitatively measured and depended only on the receiver's capacity to distinguish the sequence of symbols that had been intended by the sender, regardless of any associated meaning or semantic aspect the symbols might represent. He defined the entropy (H) for each symbol as

$$H = n \log(s), \tag{2.1}$$

where n is the number of symbols on the message, and s is the number of possible symbols (alphabet cardinality).

Later in 1948, Claude Shannon made contributions to the information theory field by creating a quantitative communication model as a probabilistic process underlying information theory. Shannon introduced the notion of information entropy and redundancy of a source, the concept of mutual information and side information, and the channel capacity of a noisy channel. He also defined the *bit* as the information entropy of a binary random variable that is 0 or 1 with equal probability, or the information that is obtained when the value of such a variable becomes known [22]. Shannon built upon the work of Nyquist and Hartley by defining the notion of average information, also called Shannon entropy. Shannon entropy is defined as

$$H = -\sum_{i=0}^{N} p_i \times \log_2 p_i, \tag{2.2}$$

where p_i is the probability of each character. The logarithm base two is used due to the binary scale [23]. Shannon formalization assumes that the objects encoded are outcomes of a known random source. It ignores the object itself and only considers the features of the random source, one of the possible results of which is the object [24].

Following the work of Shannon, in 1960, Solomonoff presented the basic ideas of AIT in work where he devised a method to overcome problems associated with the application

of Bayes's rule in statistics [25]. Later on, in a two-part paper published in 1964 [26, 27], he introduced the notion of complexity (now broadly known as *Kolmogorov complexity*) as an auxiliary concept to obtain a universal a priori probability and proved the invariance theorem governing AIT. This universal a priori probability $M(x)$ is defined as the probability that the output of a universal Turing Machine (TM) U starts with the string x when provided with fair coin flips on the input tape.

Algorithmic information theory was also developed independently by Kolmogorov and Chaitin in 1965 and 1966, respectively. Kolmogorov improved upon Shannon's probabilistic information description by adding two quantitative information definitions: combinatorial and algorithmic [28].

Solomonoff, Kolmogorov, and Chaitin showed that, among all the algorithms that decode strings from their codes, there is an optimal one. This algorithm, for all strings, allows codes as short as allowed by any other, up to an additive constant that depends on the algorithms but not on the strings themselves. Concretely, algorithmic information is a measure that quantifies the information by determining its complexity. This complexity (K) of the string s is given by

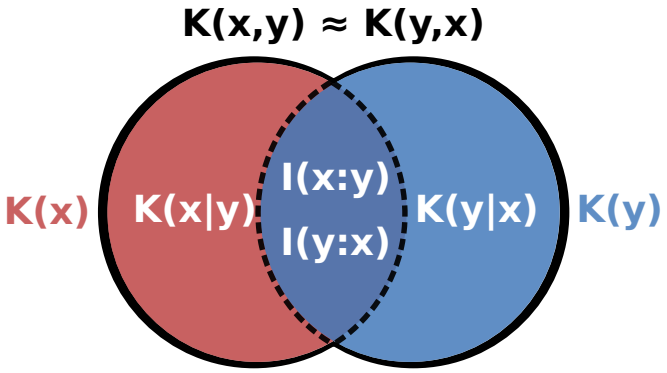
$$K(s) := \min_{p}\{l(p) : U(p) = s\}, \tag{2.3}$$

where l is the length of a shortest binary program p that computes s on a universal TM U and halts [28].

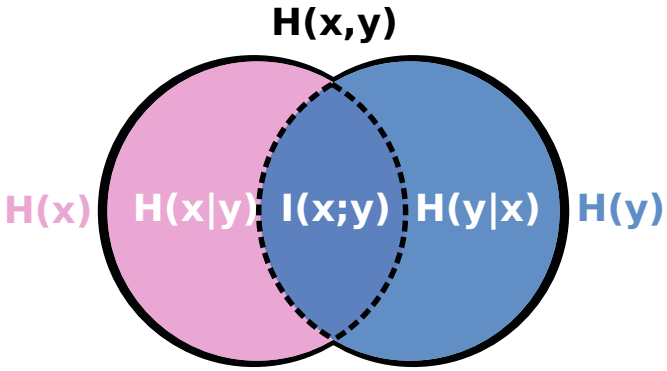
This notion of *algorithmic Kolmogorov complexity* became widely adopted and is currently the standard for performing information quantification. It differs from Shannon's entropy because it considers that the source creates structures that follow algorithmic schemes [29, 30], rather than perceiving it as a probabilistic function that generates symbols. While Solomonoff applied this idea to define a *universal probability* of a string on which inductive inference of the succeeding digits of the string can be created, Kolmogorov used it to define several functions of strings, such as complexity, randomness, and information. It is also worth mentioning that besides the algorithmic nature [31], Chaitin also carried out substantial work regarding the halting problem [32–34]. Furthermore, Chaitin introduced Ω [10, 35] a non-computable number [35], defined as the probability that an optimal computer halts, where the optimal computer is a universal decoding algorithm used to define the notion of program-size complexity.

Given two strings, x and y, the Conditional Kolmogorov complexity, $K(x|y)$ is defined as the length of a shortest binary program which, having y given as an auxiliary input, computes x and halts. And finally, the Conjoint Kolmogorov complexity of the strings x and y, $K(x, y)$, is defined as the length of a shortest binary program which, without any auxiliary information, computes both x and y (and how to separate them) and halts. In turn, these three definitions are related by the chain rule

$$\underbrace{K(x,y)}_{\text{Conjoint complexity}} = \underbrace{K(x)}_{\text{Complexity}} + \underbrace{K(y|x)}_{\text{Conditional complexity}}, \tag{2.4}$$



(a) Diagram showing the relation between algorithmic mutual information and different Kolmogorov complexities



(b) Diagram showing the relation between Shannon mutual information and different entropies

Figure 2.2: Diagrams depicting the relation between algorithmic mutual information and Kolmogorov complexity, and Shannon mutual information and entropy.

when ignoring constants that asymptotically become irrelevant. Using the chain rule of Equation (2.4) and the symmetric property of the conjoint Kolmogorov complexity, $K(x, y) = K(y, x)$, we can quantify the algorithmic mutual information, $I(x : y)$, which provides the mutual dependence between the two strings, as

$$\begin{aligned} I(y : x) &= K(x) - K(x|y), \\ I(x : y) &= K(y) - K(y|x), \\ I(x : y) &= I(y : x). \end{aligned} \tag{2.5}$$

Using Equation (2.4) of the chain rule and Equation (2.5) we can construct a diagram, shown in Figure 2.2a that ties together the algorithmic mutual information, $I(x : y)$, Kolmogorov complexity, $K(x)$ and $K(y)$, Kolmogorov conditional complexity, $K(x|y)$ and $K(y|x)$, and Kolmogorov conjoint complexity, $K(x, y)$ and $K(y, x)$, of the string x and y.

This relationship shows a high degree of similarity to the mutual information concept introduced by Shannon, $I(x; y)$, shown in Figure 2.2b, where mutual information is related to the joint entropy $H(x, y)$, individual entropy, $H(x)$ and $H(y)$, and the conditional entropy $H(x|y)$ and $H(y|x)$ [36].

There are several variants of *algorithmic complexity*, the most successful of which was the prefix complexity introduced by Levin [37], Gács [38], and Chaitin [10]. Furthermore, an axiomatic approach to Kolmogorov complexity based on Blum axioms [39] was introduced by Burgin [40].

From the roots of *algorithmic Kolmogorov complexity* [14], Wallace formulated [41] the idea of minimum message length (MML). MML is described as compressed two-part codes for the data corresponding to replacing negative-log probabilities in Bayes' rule by Shannon–Fano code lengths [42]. Independently from Wallace, Rissanenn described the Minimum Description Length (MDL) principle in a series of papers starting with his paper in 1978 [11]. The basic idea underlying the MDL principle is that statistical inference can be used as an attempt to find regularity in the data. MDL is based on two main pillars: regularity implies high compression, and compression can be considered learning. Any regularity in the data can be used to compress it; as such, regularity in data may be identified with the ability to compress it. On the other hand, compression can be viewed as a way of learning since the more we can reduce data without loss of information, the more we understand its underlying structure. MDL joins these two insights and tells us that, for a given set of hypotheses H and dataset D, we should try to find the hypothesis or combination of hypothesis in H that compresses D the most.

Another possible way to quantify information was given by Bennett, who introduced the notion of Logical Depth. Essentially, it adds to Kolmogorov complexity the notion of time and, as such, can be defined as the time required by a standard universal TM U to generate a given string from an algorithmically random input [43].

An area strongly connected with AIT is the *theory of algorithmic randomness*, which studies random individual elements in sample spaces, mostly the set of all infinite binary sequences, e.g., a sequence of coin tosses represented as a binary string. While Kolmogorov's formalization of classical probability theory assigns probabilities to sets of outcomes and determines how to calculate such probabilities, it does not distinguish between individual random and non-random elements. For instance, in a uniform distribution, a sequence of n zeros has the same probability as any other outcome of n coin tosses, namely 2^{-n}. However, there is an intrinsic notion that such sequences are not random, which is even more pronounced in infinite sequences. The current view of algorithmic randomness proposes three paradigms to determine what a random object is: *unpredictability* (in a random sequence, it is difficult to predict the next element of a sequence); *incompressibility* (a random sequence is not feasibly compressible); and measure of theoretical *typicalness* (a random sequence passes all feasible statistical tests). Martin-Löf in 1966 [44] defined a random sequence based on the measure of theoretical typicalness. Earlier researchers such as Von Mises [45] gave insights into describing a random sequence by formalizing a test for randomness. If a sequence passed all tests for randomness, it would be a random sequence. The problem was that, if we considered all possible tests, any sequence would not be random because it would be rejected by the test whose sole purpose was to reject that particular sequence. Martin-Löf's critical insight was to formally use the theory of

computation to define the notion of a test for randomness. Namely, he did not consider the set of all tests but the set of effective tests (computable tests). Martin-Löf also proved that this set of all effective tests for randomness can be subsumed by a universal test for randomness [44]. Unfortunately, deciding if a sequence passes the test is not computable. Although applying the universal test for randomness is unfeasible in practice, this notion can be approximated. An example can be found in data compressors since, given an arbitrary sequence and applying a series of models to compress the string, it is possible, although in a very limited way, to evaluate the randomness of the string. The idea behind using data compressors is that if a string is well compressed, there are repetitive patterns, meaning it is not random.

It is necessary to mention that other authors have made approaches to randomness via incompressibility (e.g., prefix-free Kolmogorov complexity [37, 38, 10]) or unpredictability (constructive martingales [46]). Despite the progress in information theory, quantifying information is still one of the most challenging open questions in computer science since no computable measure encapsulates all concepts surrounding information. Thus, one usually chooses between two options to quantify information: *Shannon entropy* or an approximation of *algorithmic complexity*.

Shannon entropy poses some problems since it is not invariant to the description of the object and its probability distribution [47]. Furthermore, it lacks an invariance theorem, forcing us to decide on a characteristic shared by the objects of interest [48]. On the other hand, *algorithmic complexity* is only approximately attainable since the Kolmogorov complexity is non-computable [14]. These approximations are computable variants of the Kolmogorov complexity and are bounded by time and resources.

Data compressors have been used to approximate the Kolmogorov complexity, as the bit stream produced by a lossless data compression algorithm allows the reconstruction of the original data with the appropriate decoder, and therefore, can be seen as an upper bound of the *algorithmic complexity* of the sequence [49]. However, the problem with using fast general-purpose data compressors is that most of their implementations are based on estimations of entropy [50] and thus are no more related to *algorithmic complexity* than to Shannon entropy. It is possible to embed specialized algorithms into data compressors to achieve state-of-the-art compression results [51]. These results show the importance of efficiently combining probabilistic and algorithmic models. Nonetheless, contrary to probabilistic data, which may be explored in advance, algorithmic modelling requires exhaustive search, human knowledge, or machine learning.

A substantial result in algorithmic modelling was the algorithm proposed by Levin to solve inversion problems [52, 53]. The Universal Levin Search (ULS) consists of an iterative search algorithm that interleaves the execution of all possible programs on a universal TM U, sharing computation time equally among them until one of the executed programs manages to solve the inversion problem provided. This search uses Levin complexity, a resource-bounded generalization of the *algorithmic Kolmogorov complexity* making it computable. In the context of a TM, these resources are the maximum number of cells

of the work tape used (space) and the number of execution steps (time). As such, a time-bounded version of Equation (2.3) can be obtained as

$$K_t(s) := \min_p \{l(p) + \log(\text{time}(p)) : U(p) = s\}, \tag{2.6}$$

where the time taken to generate the string s is considered, executing all possible programs in lexicographic order [14]. Despite this, the ULS is currently intractable due to hidden multiplicative constants in the running time and the need to make verification fast. To bridge this gap, Hutter proposed a more general and asymptotically fastest algorithm, which solves these problems at the cost of an even larger additive constant [54, 55]. More recent approaches such as the Coding Theorem Method (CTM) [56] and its derivation, the Block Decomposition Method (BDM) [48], approximate local estimations of *algorithmic complexity* providing a closer connection to the algorithmic nature. The main idea is to decompose the quantification of complexity for segmented regions using small TMs [56]. To model the probabilistic nature, such as noise, it commutes into a Shannon entropy quantification.

2.2.2 Information distances and approximations

Using the knowledge obtained from the definitions of Kolmogorov complexity, it is possible to create a distance based on the quantity of information. This was first proposed by Charles Bennett et al. in 1998, when he introduced the notion of Information Distance (ID) [57] as

$$ID(x, y) = \max\{K(x|y), K(y|x)\}, \tag{2.7}$$

as the maximum length of a shortest binary program for a reference universal prefix TM which, with input x, computes y, as well as the inverse, with input y, computes x.

In 2004 the Normalized Information Distance (NID) was introduced as a normalized version of the ID [17] as

$$NID(x, y) = \frac{\max\{K(x|y), K(y|x)\}}{\max\{K(x), K(y)\}}. \tag{2.8}$$

The problem with the NID metric is that, while appealing, its use is unpractical since the Kolmogorov complexity is non-computable [58, 59], although it may be approximated.

The Coding Theorem Method and the Block Decomposition Method

While the Kolmogorov complexity is non-computable, it can be approximated with programs with that purpose. As previously mentioned, a possible approximation is the CTM [56], and its improved version, BDM [48], which approximate local estimations of *algorithmic complexity* providing a closer relationship to the algorithmic nature. This approximation decomposes the quantification of complexity for segmented regions using small TMs [56].

The CTM uses the algorithmic probability between the frequency with which a string is produced using a random program and its *algorithmic complexity*. The more frequent a string is, the lower its Kolmogorov complexity, and the lower the frequency of strings, the higher their complexity. The BDM increases the capability of a CTM, approximating local estimations of algorithmic information based on Solomonoff-Levin's algorithmic probability theory. In practice, it approximates the algorithmic information, and when it loses accuracy, it approximates the Shannon entropy. This approach has shown encouraging results for many different purposes [60–62]. However, it has also shown underestimation issues related to side information [63].

It is possible to perform two types of normalization of the BDM. The first one is given by the number of elements (length) of the digital object as

$$NBDM_1(x) = \frac{BDM(x)}{|x| \log_2 |\theta|}, \tag{2.9}$$

where $|\theta|$ is the size of the alphabet and $|x|$ is the length of the string x.

The second is the normalization of the BDM performed using a minimum complexity object (BDM_{Min}) and a maximum complexity object (BDM_{Max}). A minimum complexity object is filled with only one symbol, like a binary string of only zeros. In contrast, a maximum complexity object is an object that, when decomposed (by a given decomposition algorithm), yields slices that cover the highest CTM values and are repeated only after all possible slices with a given shape have been used once. Using these two objects, the NBDM_2 for a given string can be computed as

$$NBDM_2(x) = \frac{BDM(x) - BDM_{Min}}{BDM_{Max} - BDM_{Min}}, \tag{2.10}$$

where $BDM(x)$ is the BDM value of x, BDM_{Min} is the minimum complexity object, and BDM_{Max} is the maximum complexity object.

Kolmogorov complexity is invariant only up to a constant factor, which depends on the choice of a description language $K = K' + L$, where K is the total complexity, K' is the description of the object and L is the description of the language. As such, by performing the normalization according to Equation (2.10), the normalization aims to remove the constant factor as

$$\frac{K - K_{Min}}{K_{Max} - K_{Min}} = \frac{K' + L - K'_{Min} - L}{K'_{Max} + L - K'_{Min} - L} = \frac{K' - K'_{Min}}{K'_{Max} - K'_{Min}}, \tag{2.11}$$

where K_{Max} and K_{Min} are the maximum and minimum Kolmogorov complexity objects and K'_{Max} and K'_{Min} are the maximum and minimum Kolmogorov complexity description of the objects.

Data compression measures

Another approximation of the Kolmogorov complexity is provided by data compressors. Approaches based on data compression are a natural solution to measure complexity, since with the appropriate decoder, the bit stream produced by a lossless compression algorithm allows reconstruction of the original data, and therefore, can be seen as an upper bound of the *algorithmic complexity* of the sequence (see [49] for a mathematical explanation of safe approximation).

Using the compression-based approach, a normalized version, known as the NC was created as

$$NC(x) = \frac{C(x)}{|x| \times \log_2 |\theta|}, \tag{2.12}$$

where x is the string, $C(x)$ represents the number of bits needed by the lossless data compression program to represent the string x, $|\theta|$ is the size of the alphabet and $|x|$ is the length of the string x.

The NC compares the information contained in strings independently from their sizes. However, in order to create metrics that compare two strings using lossless compression algorithms, the Conditional Compression Distance (CD) (Equation (2.13)) and its normalized counterpart, the Normalized Compression Distance (NCD) were developed as

$$CD(x, y) = \max\{C(x|y), C(y|x)\}, \tag{2.13}$$

and

$$\begin{aligned} NCD(x, y) = \frac{\max\{C(x, y) - C(x), C(y, x) - C(y)\}}{\max\{C(x), C(y)\}} \Leftrightarrow \\ NCD(x, y) = \frac{C(xy) - \min\{C(x), C(y)\}}{\max\{C(x), C(y)\}}. \end{aligned} \tag{2.14}$$

They assume that the chain rule of Kolmogorov complexity is mapped to the compression domain and that the conjoint compression, $C(x, y)$, corresponds to the total number of bits required to compress the conjoint information content of the strings x and y, as well as the information on how to split them. This process is usually approximated through concatenation [64]. Both these compression-based distances compute the distance between two strings using the number of bits needed to describe one of them when a description of the other is available, as well as the number of bits required to describe each of them.

The NCD is a normalized metric distance ($NCD\ \epsilon(0, 1]$). It is a direct computation of the NID and generates a non-negative value, where distances near 0 indicate similarity between two strings, and values near 1 indicate dissimilarity.

Usually, to compute the NCD metric, the chain rule is applied (Equation (2.14)) instead of performing a direct substitution of the Kolmogorov Conditional complexity by the Conditional Compression. This is due to the fact that to make a direct replacement, the compressors needed to be capable of performing conditional compression ($C(x|y)$ *and* $C(y|x)$),

which was initially not achievable by most compressors [65]. Consequently, the NCD usually used the conjoint compression ($C(x,y)$ *and* $C(y,x)$), which was computed as the compression of the concatenation of the x and y strings. However, performing the conjoint compression by assuming concatenation reduced the efficiency of the measurement [66].

A more efficient way to compute the NCD would be to apply a direct form of conditional compression as

$$NCD(x,y) = \frac{\max\{C(x|y), C(y|x)\}}{\max\{C(x), C(y)\}}. \tag{2.15}$$

This approach defines conditional compression, $C(x|y)$, as the number of bits needed by the lossless compression program to represent the digital object x given the digital object y as an auxiliary input, and $C(y|x)$ as the inverse [67, 68].

It is also worth mentioning that NCD measures the information between two strings and is robust to some degree of noise [69]. This behaviour is verified as long as the compressor is normal (possess the property of Distributivity, Symmetry, Monotonicity, Idempotency, and Density [36, 66]) and has an internal model capable of performing a correct representation of the strings' nature.

2.2.3 Normalized Relative Compression

The NCD measures a distance between two strings, in other words, it respects the properties of identity, symmetry and triangle inequality. However, there are some cases where the usage of semi-distances is more advantageous, namely when it is intended to measure the information of a digital object relative to another. This implies that two distance properties cannot be fulfilled, namely the property of symmetry and the triangle inequality [64].

An important method to approximate relative information between strings is by using relative compressors which can be based on Markov models [70] or dictionaries [71–73]. These relative compressors perform the compression of the digital object x relatively to y, $C(x||y)$, by first modelling and organizing the data of the digital object y, without knowing the content of the digital object x. Then, the model of the digital object y is frozen and used exclusively to measure the number of bits needed to describe x, with the information from the y object. Furthermore, $C(x||y)$, can be seen as the sum of the information content provided, symbol by symbol, after processing the complete x, according to

$$C(x||y) = \sum_{i=1}^{|x|} C(x_i||y). \tag{2.16}$$

To assess the overall quantities of relative information between two strings, the Normalized Relative Compression (NRC) was developed as

$$NRC(x,y) = \frac{C(x||y)}{|x| \times \log_2 |\theta|}. \tag{2.17}$$

The NRC can be seen as a string's fraction that cannot be represented by another string, regardless of the redundancy of this fraction. Although this measure cannot answer specific questions relative to distances (since it is a semi-distance), it is helpful since it can infer insights into dissimilarity or completeness. Unlike the NCD, the NRC does not require the computation of the self-similarity term, $C(x)$, and only depends on one compression. This decreases the computation time and provides a more predictable behaviour since it relies on only one approximation function. In contrast, the NCD uses a ratio between two approximation functions [74].

2.3 Summary

This chapter analyses the information theory, approximations of the Kolmogorov complexity, and its derived distances and measures.

Since the success of classification and identification tasks largely depends on feature selection and data representation, the success of our work depends on performing a correct data representation [75, 76]. Furthermore, considering different representations can entangle and hide different explanatory factors of variation behind the data, by creating an adequate representation of the data, we will identify the underlying explanatory factors hidden in the observed environment of low-level sensory data [75]. Finally, although specific domain knowledge can be used to design representations, we assume that learning with generic priors is more advantageus because it may lead to more robust and condensed data representations.

On the other hand, using compression measures can be quite valuable since they can provide insights into dissimilarity or completeness between data.

In the next chapter, we will start by analysing the applicability of *algorithmic Kolmogorov complexity* approximations in describing and detecting patterns of 1-dimensional biological data.

Chapter 3

Complexity analysis of natural sequences

[c]The metamorphosis of man - Jorge Miguel Silva, 2020.

"The butterfly's attractiveness derives not only from colours
and symmetry: deeper motives contribute to it.
We would not think them so beautiful if they did not fly,
or if they flew straight and briskly like bees, or if they stung,
or above all if they did not enact the perturbing mystery of metamorphosis:
the latter assumes in our eyes the value of a badly decoded message,
a symbol, a sign."
– Primo Levi

3.1 Contextualization

This chapter focuses on studying and applying the Kolmogorov complexity on strings encoded as 1d data structures. Since Kolmogorov complexity provides several ways to examine different natural processes, we use it to analyse genomic sequences, which are strings with finite length and quaternary alphabet. We perform this analysis by first testing two ways of approximating the Kolmogorov complexity, the block decomposition method and lossless data compressors. Efficient data compressors that consider the probabilistic and algorithmic characteristics of the data seem more capable of identifying small programs embedded in this type of biological data. As such, we use them to analyse and classify genomic sequences. As will be shown in this chapter, the use of specific data compressors to quantify data complexity (Kolmogorov complexity) profoundly impacts viral genomes' identification, classification, and organization. This chapter follows the structure:

- Research questions and contributions;
- Biological background;
- Data compression in genomic data;
- Methods;
- Complexity analysis of viral genomes;
- Taxonomic classification;
- Summary.

3.2 Research questions and contributions

This chapter aims to use approximations of the Kolmogorov complexity to analyse natural genomic sequences. By doing so, we aim to answer several questions: What is the best approach to detecting and quantifying small programs embedded in natural sequences? What information can be obtained by analysing the complexity landscape of viral genomes? Moreover, how much information is present in a viral genome sequence? Can this information be used for classification and viral taxonomic identification? If so, can this be true for other genomic sequences?

In the following sections, we will answer these questions, present the information obtained from performing an extensive complexity analysis of the viral genome, and demonstrate that this information can be used to obtain state-of-the-art results in the taxonomic identification of organisms.

3.3 Biological background

In this chapter, we work with Deoxyribonucleic acid (DNA) and Ribonucleic acid (RNA) genomic sequences. To introduce several concepts, we provide background regarding general aspects of organisms with the notions of genome and proteome, as well as a more in-depth description of viruses, archaea, and finally, a description of Inverted Repeats.

3.3.1 Life

Life is a quality that discerns matter that has biological processes from that which does not. Living matter has biological processes, such as signalling and self-sustaining mechanisms, and has, by definition, the capability for growth, response to stimuli, metabolism, energy transformation, and reproduction.

There are many different types of life, including bacteria, fungus, protists, archaea, and plants. Furthermore, in the case of viruses, they are considered to be matter near the edge of being alive. This classification occurs because they share some traits with living beings but are incapable of metabolizing and require a host cell to make new products.

The gene is the heredity unit, and the cell is the basic structural and functional unit in Earth's life forms. All cells possess cytoplasm enclosed within a membrane containing numerous biomolecules like proteins and nucleic acids. They reproduce through cell division, where the parent cell splits into two or more daughter cells and passes its survival/behaviour instructions (genes) onto a new generation, sometimes producing genetic variation.

Organisms are individual entities of life composed of cells which are open systems that maintain homeostasis and have a life cycle. Organisms can be prokaryotes or eukaryotes, and in the case of eukaryotes can either be unicellular or multi-cellular.

Prokaryotes are characterized by being the most diverse group of organisms and being unicellular. They are separated into the bacteria and archaea domains [77].

On the other hand, eukaryotes are portrayed by the presence of membrane-bound compartments in the cytoplasm. Specifically, a membrane-bound cell nucleus, and other membrane-bound compartments (called organelles), i.e. mitochondria, Golgi apparatus, and the chloroplasts [78]. Eukaryotes comprise the animal, plant and fungi domains. Despite having very different morphologies and physiologies, all living organisms are time-limited [79], the end of an organism's existence being marked by death, which is the irreversible cessation of all biological activities that maintain it.

To avoid the organism's complete eradication, they perform a replication process where all or some of their heredity unities (genes) are passed to their offspring. This process can occur using a single individual (asexual process) or combining information from two progenitor organisms (sexually). Due to the pressure of natural selection and genetic variation, organisms have diversified and evolved, giving rise to various domains of life and millions of species that can be traced back to a common ancestor, as shown in Figure 3.1.

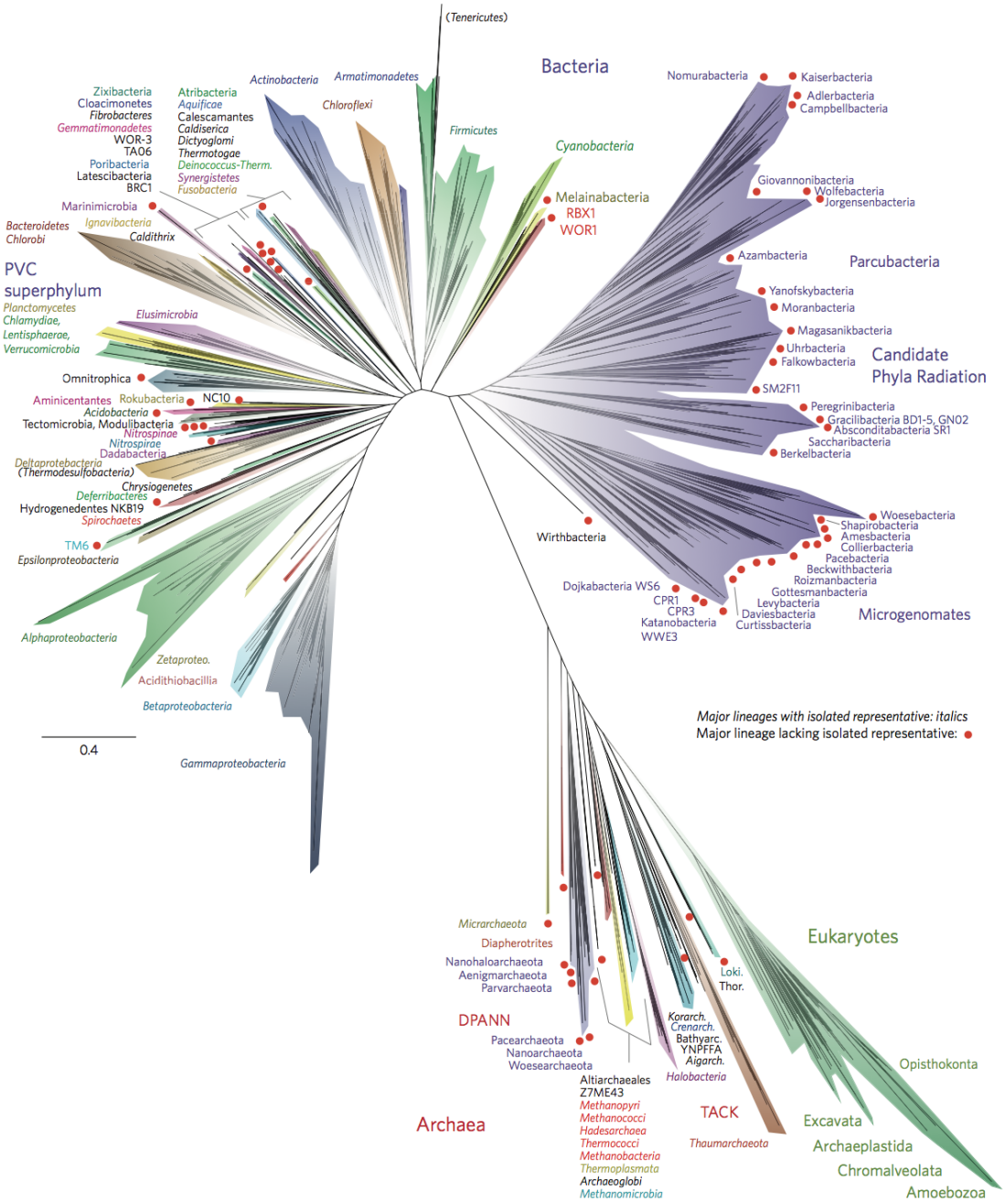


Figure 3.1: A metagenomic representation of the tree of life using ribosomal protein sequences of a tree of life according to Hug *et al.*[80].

Figure 3.1 shows the separation of bacteria, archaea, and eukaryotes. Despite their structural resemblance, bacteria and archaea are very different types of prokaryotes. Archaea have genes and metabolic pathways more closely related to eukaryotes, particularly transcription and translation enzymes. In addition, archaeal biochemistry is distinct, as proven by the isoprene-composed and ether-bond phospholipids of their cell membranes. Furthermore, archaea exist in practically every habitat despite being initially discovered

in extreme environments, such as hot springs and salt lakes.

On the other hand, as seen in the Figure 3.1, viruses are not included in the tree of life, not only because they do not share characteristics with cells but also because while cellular life has a single, common origin, viruses are polyphyletic, meaning they have many evolutionary origins. As such, a possible description of their place in the tree of life could be dead leaves that have fallen off a tree at different points in time.

3.3.2 Genome and proteome

Despite all differences, all living beings and viruses possess genetic information. Living beings use DNA, whereas some viruses can use RNA. DNA and RNA sequences have a quaternary alphabet, adenine (A), cytosine (C), guanine (G), and thymine (T) for DNA and uracil (U) in RNA (instead of thymine). Adenine and guanine are categorized as purines, while cytosine and thymine as pyrimidines. Watson and Crick described the double helix structure of DNA [81]. This discovery showcased Chargaff's parity rule [82], where the total percentage of complementary nucleotides, i.e. adenine-thymine (A-T) and cytosine-guanine (C-G), in a double-stranded deoxyribonucleic acid (dsDNA) molecule, should be equal.

The dsDNA molecule is composed of two anti-parallel chains containing genes that encode the instructions necessary for the development and survival of the organisms. These genes can have coding (exons) and non-coding (intron) regions. The coding regions of the genes (exons) are used to create proteins. In living beings, this occurs in a two-step process: The first step is transcription, where DNA is used to create a single-stranded RNA molecule known as messenger ribonucleic acid (mRNA), whose nucleotide sequence is complementary to the DNA used. The second step is translation, where a mature mRNA molecule is used as a template for synthesizing a new protein. This process can be different in the case of viruses.

In the translation of RNA to protein, one amino acid is added to the protein strand for every three bases in the RNA, called codons. This process occurs using a specialized RNA molecule called transfer ribonucleic acid (tRNA), which has three unpaired bases (anticodon) complementary to the codon it reads on the mRNA and is also covalently linked to the amino acid defined by the complementary codon.

When the tRNA attaches to its complementary codon in an mRNA strand, the ribosome binds its amino acid payload to the new polypeptide chain, which is synthesized from the amino terminus to the carboxyl terminus. The polypeptide chain has complex interactions, which create a three-dimensional structure that characterizes a protein and is fundamental for its functioning.

3.3.3 Viruses

Viruses are a strong driving force of life and evolution. They are the shortest and most abundant life realm, estimated at around 10^{31} particles [83]. Likewise, viruses occupy

almost every ecosystem [84–86] and infect all types of life forms [87, 88].

They are submicroscopic biological infectious agents that require living cells of an organism to be active for replication [89]. They can exist outside of their host in the form of independent particles named virions composed of the genetic material (DNA or RNA) enclosed by the capsid. This protein shell protects the viral genome, and at the same time, it is extracellular and promotes its entry into the host cells [90].

Most viruses possess capsids with helical (Figure 3.2 A) or icosahedral (Figure 3.2 B) arrangements [91, 92]. Different viruses, like bacteriophages, have developed other structures composed of elongated capsids attached to a cylindrical tailed sheet (Figure 3.2 C) [93]. Others have an outer lipid bilayer named viral envelope (Figure 3.2 D), which is constituted by a modified form of the host's cell membranes. Viroids have naked genomes without any protective layer. Like viruses, they use the host's machinery to replicate, but their genomes do not encode proteins [94]. Furthermore, some viruses are dependent on another virus species in the host cell to be transmitted to new cells. They were named 'satellites' and may represent evolutionary intermediates of viroids and viruses [95, 96]. Regarding mutability, the viral and viroid realm is the most mutable [97] of the realms.

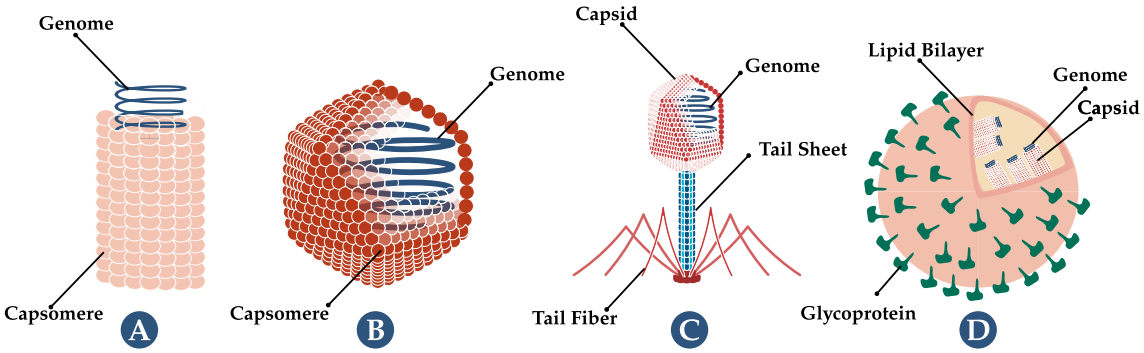


Figure 3.2: Illustrations of types of virus morphology. Virus (**A**) is a helical virus, where the capsoid has a helical shape that envelops the genomic material, virus (**B**) is icosahedral following cubic symmetry, (**C**) depicts a complex virus, namely a bacteriophage with a prolate capsid protecting the genomic material, and (**D**) is virus covered by a viral envelop.

Viral genomes can be of dsDNA, single-stranded deoxyribonucleic acid (ssDNA), double-stranded ribonucleic acid (dsRNA), single-stranded ribonucleic acid (ssRNA) or mixed nature, being linear or circular molecules [98]. The ssRNA viruses can be further classified as positive- or negative-ssRNA, depending on the sense of their RNA strand. These features determine the viral replication and mRNA synthesis pathways. For instance, (+)-ssRNA is directly translated into proteins by the host cell's ribosomes, acting as mRNA. On the other hand, (-)-ssRNA needs to be converted to a (+)-ssRNA by an RNA-dependent RNA polymerase (RdRp) before translation. RdRp also transcribes dsRNA to mRNA (using the negative strand as a template), and it is indispensable for the replication of RNA viral genomes. Finally, ssDNA and dsDNA usually use the host's DNA-dependent RNA polymerase to form mRNA. However, before this process, ssDNA

is converted to a dsDNA by a DNA polymerase upon cell invasion [99], which is also the enzyme involved in the replication of DNA viruses. The RdRps have a high error rate due to their low proofreading activity and, therefore, replication of RNA viruses is much more prone to mutation than that of DNA viruses [100].

Viruses have a vast variation in size, ranging from around 10 nm with small genomes to viruses with similar dimensions and genome sizes to bacteria and archaea [101, 102]. These viruses are called giant viruses and contain many unique genes currently not found in other life forms.

There can also be hybrid viruses [103], making it difficult to identify species [104]. There are several possible combinations for the creation of a hybrid virus. One possible way of occurring is the infection of a host's cell by two or more related viruses and consequential exchange of sequences between viruses. The result is the creation of a new variant derived from the parental genomes. Another possible way is the recombination of RNA viral genomes with the host's RNA. Finally, there is evidence that small DNA viruses could have been created by recombination events between RNA viruses and DNA plasmids [103].

Although the origin of viruses is still uncertain, they play an essential role in the evolution of living organisms since they are horizontal gene transfer vehicles. This biological phenomenon increases genetic diversity. Furthermore, it occasionally allows viral genetic material to integrate into the host genomes, transferred vertically to its offspring. This property is so preponderant in evolution that the origin of the eukaryotic nucleus might be related to this process [105–107].

Additionally, viral genomic integration allows us to infer the evolutionary distance between hosts by observing the shared virus integrated into their genomes. For instance, in humans, viruses frequently establish persisting infections [108] and imprint their genetic material in the tissues throughout life, displaying phylogeographic patterns. These can be used as markers to understand the human population history and migrations better and provide new insights into unidentified individuals' origins on both global and local scales [109]. In this respect, the JC polyomavirus is one of the most comprehensively studied viruses. Its genotype-specific global spread has been suggested to indicate the origins of modern [110] and ancient humans [111–113]. Furthermore, a worldwide study supported the co-dispersal of this virus with major human migratory routes and its co-divergence with human mitochondrial and nuclear markers [114].

3.3.4 Inverted Repeats

Inverted repeats (IRs) are nucleotide sequences with a downstream reverse complement copy, causing a self-complementary base-pairing region [115]. Consequently, IRs usually fold into different secondary structures (hairpin- and cruciform-like structures, pseudoknots) that participate or interfere in many cellular processes in all forms of life, including DNA replication [116, 100]. Due to these traits, IRs play an essential role in genome stabilization and destabilization [117], contributing to mutability. This mutability can

create diseases in the short term [118], but over long periods leads to cellular evolution and genetic diversity [119]. In many viruses, IRs in pseudoknots are involved in ribosomal frameshifting. This translational mechanism allows the production of different proteins encoded by overlapping open reading frames (ORFs) of the same mRNA [120, 121]. This feature allows them to encode a more significant amount of genetic information in small genomes and constitutes another level of gene regulation [122].

The genomes of some viruses, such as parvovirus, are flanked by inverted terminal repeats (ITRs) that form hairpin structures functioning as a duplex origin of replication sequence [116, 123]. Therefore, these ITRs contain most of the cis-acting information needed for viral replication and viral packaging [123]. In adeno-associated viruses, ITRs are essential for intermolecular recombination and circularization of genomes [124]. IRs can also function as termination transcription signals, especially in giant viruses [125, 126].

3.4 Data compression in genomic data

In genomics, sequences can be codified as messages using a four-symbol alphabet ($\theta = \{A, C, G, T\}$ for DNA sequences and $\theta = \{A, C, G, U\}$ for RNA sequences). These messages contain instructions for survival and replication of the organism, its morphology and historical marks from previous generations [127]. Initially, genomic sequences were compressed with general-purpose data compressors such as gzip [128], bzip2 [129], or LZMA [130]. However, this paradigm shifted towards using a specific compression algorithm after the introduction of BioCompress [131]. Genomic data compressors can outperform general-purpose data compressors since they are designed to consider specific genomic properties such as the presence of a high number of copies and substitutional mutations, and multiple rearrangements, such as inverted repeats [132, 133].

Given this advantage of using specific data compressors for the compression of genomic data, several algorithms have emerged to model these genomic data behaviours [134]. Specifically, several algorithms have been created to model repetitions and inverted repetitions in the genome regions through simple bit encoding, dictionary approaches and context modelling [135–145].

Currently, state-of-the-art data compressors have different objectives, such as optimizing for compression strength or prioritizing a balance between compression speed and compression capability. Examples of the latter are NAF (Nucleotide Archival Format) [146, 147] and MBGC (Multiple Bacteria Genome Compressor) [148], which are more suitable for large data collections and are frequently used by computational biologists. Data compressors focused on compressibility at the expense of more computational resources, on the other hand, generally apply probabilistic and algorithmic model mixtures combined with arithmetic encoding. Among the best compressors regarding compression ratio performance for various genomic sequences, the best results are provided by cmix [149], XM [150], Jarvis [151], and Geco3 [51]. For additional information regarding data compressors' compressibility capacity of genomic sequences, see [152]. Cmix [149] is a

general-purpose lossless data compression program that optimises compression ratio at the cost of high CPU/memory usage. It is based on PAQ compressors [153, 154] but dramatically increases the amount of processing per input bit and computational memory. Current updates include LSTM (Long Short-Term Memory) based models [155]. The XM compressor [150] uses three types of experts: repeat models, a low-order context model, and a short memory context model. On the other hand, Jarvis [151] uses a competitive prediction model that estimates for each symbol the best class of models to be used. There are two classes of models: weighted context models and weighted stochastic repeat models, where both classes of models use specific sub-programs to handle inverted repeats efficiently. Finally, GeCo3 [51], currently one of the best performing reference-free data compressors, uses neural networks to improve upon the results of specific genomic models of GeCo2 [156]. Specifically, the neural networks are used in mixing multiple contexts and substitution-tolerant context models of GeCo2. Furthermore, GeCo3 has embedded subprograms capable of detecting genome-specific patterns, such as inverted repeats.

3.5 Methods

This section describes the measures used in this chapter. Specifically, we first define and contextualize the information-based measures used: the Normalized Block Decomposition Method, the NC with different subprograms, the Normalized Compression Capacity (NCC), the difference between NCs, and the minimal bi-directional complexity profiles. Afterwards, we define another important measure, GC-Content. Finally, we described the classification pipeline, specifically, the features and classifiers used as well as the metrics used to evaluate the model's performance.

3.5.1 Information-based measures

In this subsection, we describe the NC, the minimal bi-directional complexity profiles, and the Normalized Block Decomposition Method (NBDM).

Normalized Block Decomposition Method (NBDM)

In this chapter, we use BDM to perform a comparison with the Normalized Compression. Consequentially we considered the normalization of the BDM (NBDM) by sequence length BDM as described in Equation (2.9) of Chapter 2. In this case, since we have a four-symbol alphabet in genomic sequences, $log_2(4) = 2$, and NBDM is computed as

$$NBDM(x) = \frac{BDM(x)}{2 \times |x|}. \tag{3.1}$$

Normalized Compression

We used NC as described in Chapter 2. Since we have a four-symbol alphabet in genomic sequences, the NC is computed as

$$NC(x) = \frac{C(x)}{2 \times |x|}. \tag{3.2}$$

Given the normalization, the NC can compare the proportions of information contained in the strings independently from their sizes [157]. If the compressor is efficient, then it can approximate the quantity of probabilistic-algorithmic information in data using affordable computational resources. In our work, to determine the NC, we made use of the state-of-the-art genome compressor GeCo3 [51], using level 16, which yielded the best average results (benchmark provided in the results section).

Besides computation of the NC using the standard configuration of this model, we also computed the NC using GeCo3 with three subprogram configurations. These subprogram configurations address the different compression aspects:

- $IR_0 \rightarrow$ uses the regular context model without IR detection;
- $IR_1 \rightarrow$ uses IR detection simultaneously with the regular context model;
- $IR_2 \rightarrow$ uses the IR detection sub-program without regular context models.

There was a need to determine the sequences with the highest normalized compression capacity (NCC) in some cases. When the compressor was only using the subprogram IR_2, NCC was computed as $NCC_{IR_2}(x) = 1 - NC_{IR_2}$. Only positive values were considered to filter computations where the compressor could not compress the sequence sufficiently. Another measure used to quantify inverted repeats was the difference between NC_{IR_0} and NC_{IR_1}.

Minimal bi-directional complexity profiles

A complexity profile is a numerical sequence describing for each symbol (x_i) of a sequence x the number of bits required for its compression assuming a causal order [158]. A minimal bi-directional complexity, $B(x)$, profile assumes the minimal representation of compressing the sequences using both directions independently, namely $\overrightarrow{C}(x_i)$ as from the beginning to the end of the sequence, and $\overleftarrow{C}(x_i)$ as from the end to the beginning [159]. Accordingly, these profiles are defined as

$$B(x_i) = \min\{\overrightarrow{C}(x_i), \overleftarrow{C}(x_i)\}. \tag{3.3}$$

The construction of these profiles follows a pipeline formed of many transformations, including reversing, segmenting, inverting, and the use of specific low-pass filters after data compression to achieve better visualization. To compute these profiles, we use the GTO toolkit [160].

The generation of these profiles is robust to localize specific features in the sequences, namely low and high complexity sequences, inverted repeat regions and duplications, among others.

3.5.2 Other measures

The other two measures used to perform viral analysis and classification are the GC-Content (GC) and the length of the viral genome $|x|$.

GC-Content (GC) represents the proportion of guanine (G) and cytosine (C) bases out the quaternary alphabet ($\theta = \{A, C, G, T/U\}$). The GC percentage is given by the number of cytosine (C) and guanine (G) bases in a viral genome x with length $|x|$ according to

$$\mathcal{GC}(x) = \frac{100}{|x|} \sum_{i=1}^{|x|} \mathcal{N}(x_i || x_i \in \xi), \tag{3.4}$$

where x_i is each symbol of x (assuming causal order), ξ is a subset of the genomic alphabet containing the symbols $\{G, C\}$ and $\mathcal{N}$ the program that counts the numbers of symbols in ξ.

GC-content is variable between different organisms and correlates with the organism's life-history traits, genome size [161], and GC-biased gene conversion [162]. Furthermore, in RNA viruses, excess C to U substitutions accounted for 11–14% of the sequence variability of viruses, indicating that a decrease in GC-content is a potent driver of RNA viruses' diversification and longer-term evolution [163]. As such, this measure helps perform viral classification.

On the other hand, it was shown that the number of base stackings (typical arrangement of nucleobases found in the three-dimensional structure of nucleic acids) is one of the most critical elements contributing to the thermal stability of double-stranded nucleic acids. Furthermore, due to the relative locations of exocyclic groups, GC pairings have higher stacking energy than AT or AU pairs [164]. This energy accumulation in the GC pair in an organism's genome makes the DNA more prone to mutation. Thus, over time, a species tends to decrease its GC-content to become more stable [165], giving us further information regarding viral characterization.

3.5.3 Classification

We tested several machine learning algorithms to perform the genomic and taxonomic classification task: Linear Discriminant Analysis (LDA) [166], Gaussian Naive Bayes (GNB) [167], K-Nearest Neighbors (KNN) [168], Support Vector Machine (SVM) [169], and XGBoost classifier (XGB)[170].

Linear Discriminant Analysis is a generalization of Fisher's linear discriminant, a method used in statistics and other fields, to find a linear combination of features that separates classes of objects. The resulting combination can be used as a linear classifier [166]. Gaussian Naive Bayes is defined as a supervised machine learning classification algorithm based on the Bayes theorem following Gaussian normal distribution [167]. K-Nearest Neighbors is another approach to data classification, taking distance functions into account and performing classification predictions based on the majority vote of its neighbours [168]. Support Vector machines are supervised learning models with associated

learning algorithms that, using data, construct a hyperplane in high-dimensional space to perform classification [169]. Finally, XGBoost [170] is an efficient open-source implementation of the gradient boosted trees algorithm. Gradient boosting is a supervised learning algorithm that predicts a target variable by combining the estimates of a set of simpler models. Specifically, new models are created that predict the residuals or errors of prior models and then added together to make the final prediction. This task uses a gradient descent algorithm to minimize the loss when adding new models. XGBoost can use this method in both regression and classification predictive modelling problems.

The accuracy and weighted F1-score were used to select and evaluate the classification performance of the measures. Accuracy is the proportion of correct classifications in the total number of cases examined, while the F1-score is computed using the precision and recall of the test. We utilized the weighted version of the F1-Score due to the presence of imbalanced classes.

For comparison of the results obtained, we assessed the outcomes obtained using a random classifier. For that purpose, for each task, we determined the probability of a random sequence being correctly classified (p_{hit}) as

$$p_{hit} = \sum_{i=0}^{n} [p(c_i) * p_{correct}(c_i)], \tag{3.5}$$

where $p(c_i)$ is the probability of each class, determined as

$$p(c_i) = \frac{|samples_{class}|}{|samples_{total}|}.$$

On the other hand, $p_{correct}(c_i)$ is the probability of that class being correctly classified. In the case of a random classifier,

$$p_{correct}(c_i) = \frac{1}{|classes|}.$$

3.6 Complexity analysis of viral genomes

Despite the significant impact of viruses on the evolution of living beings and the ecosystem, our understanding of viruses is still relatively limited compared to other realms of life. In particular, the complexity landscape of viruses is unknown. Complexity analysis of genome sequences is not new and is frequently performed by data compressors, which serve as an upper bound to Kolmogorov complexity. Many examples of these studies appeared after the creation of the first data compressor for DNA sequences [171]. Specifically, data compression has been used to detect repeated sequences in the Plasmodium falciparum DNA, and observed patterns were related to large-scale chromosomal organization, and gene expression control [171]. The XMAligner tool [172] was created for pairwise genome local alignment, which considers a pair of nucleotides from two sequences related if their mutual information in context is significant. They used a lossless compression method

to measure the information content of nucleotides in sequences. Graph compression was used to compare large biological networks [173]. This method involved compressing the original network structure and then measuring the similarity of the two networks using the compression ratio of the concatenated networks. The method was applied to several organisms, showing an efficient ability to measure the similarities between metabolic networks. Data compression was used to approximate the Kolmogorov complexity and applied to data derived from sequence alignment data [174]. This process identified a novel way of predicting three aspects of protein structure: secondary structures, inter-residue contacts and the dynamics of switching between different protein states. An analysis of the complexity of different DNA genomes was performed, demonstrating various evolution-related findings linked with complexity, notably that archaea have a higher relative complexity than bacteria and eukaryotes on a global scale. Furthermore, viruses have the most complex sequences according to their size [157]. Metagenomic composition analysis of a sedimentary ancient DNA sample was performed using relative compression of whole-genome sequences [175]. The results showed that several viruses and bacteria expressed high levels of similarity relative to the samples. Finally, an alignment-free tool was created to accurately find genomic rearrangements of DNA sequences following previous studies, which took alignment-based approaches or performed Fluorescence In Situ Hybridization (FISH) [176].

Given the applicability of compression methods in the analysis of genomic sequences and intending to understand viruses better, in this section, we conduct an extensive complexity analysis of the viral world through the automatic computational analysis of its genome complexity and associated characteristics. Specifically, we use a genomic compressor to analyse the complexity across viral taxonomies and quantify the algorithmic information embedded in viral genome sequences better represented by small programs.

Since studying the complexity of a DNA/RNA sequence requires efficient data compressors that take into account the probabilistic and algorithmic characteristics of the data, we compared several state-of-the-art genomic data compressors and another approximation of the Kolmogorov complexity besides data compression. This comparison was made to evaluate their ability to detect IRs with increasing levels of mutations. Consequently, the best method was used to analyse viruses' complexity and overall abundance of inverted repeats and to construct cladograms.

The following subsections describe the database and discuss the results obtained.

3.6.1 Data description

The dataset is composed of 12,163 complete reference genomes from 9,605 viral taxa retrieved from NCBI database on 22 January 2021 [1]. The download was performed in a custom manner to retrieve the taxonomic id, host and geolocation of each reference

[1] https://www.ncbi.nlm.nih.gov/labs/virus/vssi/#/virus?SeqType_s=Nucleotide&VirusLineage_ss=Viruses,%20taxid:10239&SourceDB_s=RefSeq&GenomeCompleteness_s=complete&CreateDate_dt=1998-01-01T00:00:00.00Z%20TO%202021-01-22T23:59:59.00Z

genome. The metadata header was removed from each sequence using the GTO toolkit [160], where any nucleotide outside the quaternary alphabet $\{A, C, G, T/U\}$, was replaced by a random nucleotide from the quaternary alphabet. Notice that the sequences with symbols outside the alphabet are scarce. Finally, the type of genome and the taxonomic description of each sequence were retrieved using Entrez-direct [177].

Then, the retrieved NCBI sequences were filtered to remove possibly contaminated or poorly sequenced sequences. Firstly, using the taxonomic metadata, sequences that did not hold complete taxonomic information down to the genus rank and any sequences that maintained a taxonomic description of unclassified were removed. Secondly, we applied a filter to remove outlier sequences. Specifically, after computing all sequences' length, GC-Content, and Normalized Complexities, sequences whose measure fell outside $\mu \pm 3 \times \delta$ (approximately 0.03% of all sequences) of any measure were removed. A total of 182 sequences were removed since they most likely have errors in the assembly process or contamination. After filtering, we kept 6,091 of the initial 12,163 sequences.

3.6.2 Determining the optimal way to quantify probabilistic-algorithmic information in viral sequences

Viral genomes have specific characteristics, for example, short length, high average complexity, and specific structures, that require the proper optimization of the data compressor to provide higher modelling adaptability and efficiency. As such, to find the best way to analyse the complexity of viral genomes, we performed three separate tests: one using synthetic data to simulate small program detection; another using a small sample of natural sequences to determine the best time-compressibility compressor; and a final step to determine the best configuration for the selected method.

Synthetic sequence benchmark

Genomes have small subprograms embedded in them. One of the most relevant for characterization is the subprogram that creates Inverted Repeats. These subsequences occurring in viral genomes are better described using simple algorithmic approaches. To benchmark the ability of different programs to quantify IRs accurately, we created a genomic sequence of 10,000 nucleotides in which the last 5,000 were inverted repeats of the first 5,000. This length was chosen since the median size of the viral genomes is 9,836 bases. This sequence was mutated incrementally from 0% to 10%, meaning that the number of IR bases decreases with the increase of nucleotide substitutions. For each sequence, the NC was computed with: i) GeCo3 [51], without and with the IR detection program (IR_0 and IR_2, respectively), ii) PAQ8 [178] and iii) Cmix [149] (Figure 3.3). The Normalized Block Decomposition Method (NBDM) [48] was also computed as a measure more predisposed to algorithmic nature quantification. Results show that GeCo3 with the IR_2 subprogram compresses the sequences better than the other programs, resulting in a lower NC at 0% mutational rate (Figure 3.3). No other compressor (cmix and PAQ8) could detect IRs and

compress the sequence. Furthermore, NBDM cannot detect the IRs, as shown by the same high NC across sequences with various mutation rates. It is also evident that GeCo3 with IR_2 can detect IRs even in the presence of substantial mutations (5% of mutation) and is sensitive to different levels of nucleotide substitutions, as seen by the increase in NC with the increase of the mutational rate (i.e. decrease of IRs). The difference between NC_{IR_0} and NC_{IR_1}, both computed with GeCO3, was also analysed as a supplementary measure to show the significance of IRs in genomic sequence compression. Its profile is inverse to the IR_2 and confirms that the accumulation of nucleotide substitutions decreases the number of IRs in the sequence.

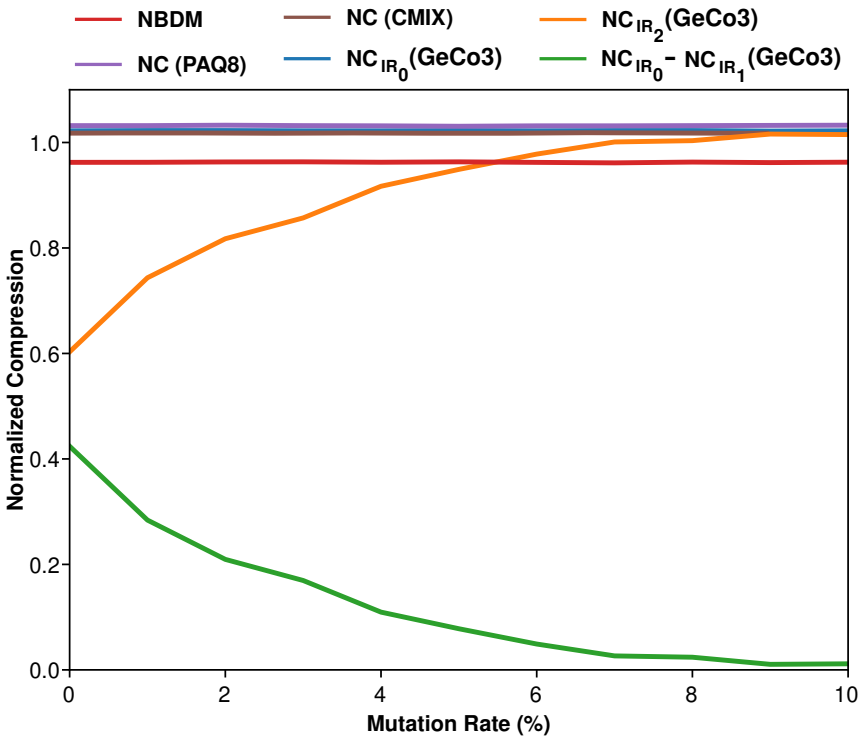


Figure 3.3: Plot describing the variation of NC and Normalized Block Decomposition Method (NBDM) with an increase of mutation rate of a sequence (0%-10%). The NC was computed using the state-of-the-art genomic compressor (GeCo3[51]) and a general-purpose compressor (PAQ8 [178]). The red line depicts the NBDM, the NC value using cmix with brown, and PAQ8 by a purple line. Furthermore, the GeCo3 compressor with (IR_2) and without the IR detection subprogram (IR_0) is shown in orange and blue lines, respectively. Finally, the green line shows $NC_{IR_0} - NC_{IR_1}$.

Comparison between state-of-the-art compressors

Cmix [149] and GeCo3 [51] are state-of-the-art genomic compressors. To assess the viability of each compressor, we tested their computational time and NC values on a small sample consisting of 8 medium size viral genomes. The results, presented in Figure 3.4, show that the compression ratio of GeCo3 is, on average, slightly better, with a much

more reasonable computational time (on average, three orders of magnitude faster than cmix).

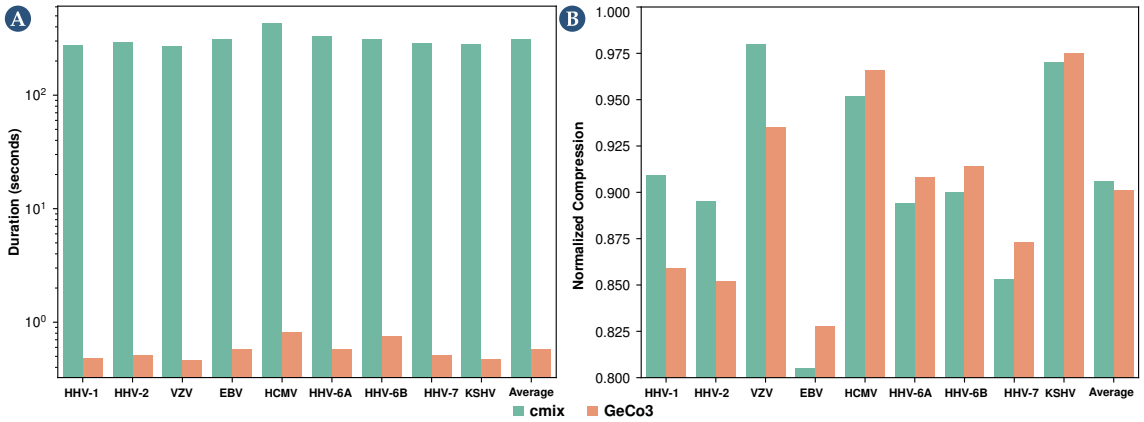


Figure 3.4: Comparison between cmix and GeCo3 when applied to various Human Herpesviruses regarding computational time and compression ratio obtained (NC).

Fine tuning GeCO3 parameters

Since GeCo3 provided better results in the sample and contains many types of compression levels [51], we next finetuned the GeCo3 parameters to find a better configuration that achieves good average compression results. To this end, we selected 19 levels of models in GeCo3, which correspond to the default 13 levels of the GeCo3 compressor and six others built for this task. The list of the levels used and the parameter descriptions can be found in Table A.1 and Table A.2 of Appendix A. The 13 default levels of the compressor have increasingly higher complexity and take longer to run since they use higher context models. Therefore, since the first and lightest level performed best, the other six custom-built levels were also built with lightweight models. These models were applied to each viral genome from the dataset, and their NC was computed.

We evaluated the frequency of each level yielding the lowest NC (provided the best compression for a given sequence; Figure 3.5 A) and determined the sum of the NC from the compression of all reference genomes for each model (Figure 3.5 B). Overall, we selected level 16 because it provided the lowest NC on average (28.38% as the best compression level) and the lowest NC sum from compressing all reference genomes. This level is formed of a mixture using a neural network with the following models:

- Model 1 $\rightarrow$ context-order of 1, an alpha parameter of 1 (without inverted repeats), and a gamma parameter of 0.7 (parameters explained in Table A.2 of Appendix A);
- Model 2 $\rightarrow$ context-order of 12, an alpha parameter of 1/50 (with inverted repeats), and a gamma parameter of 0.97.

The chosen level comprises two models with a small and average context model. This

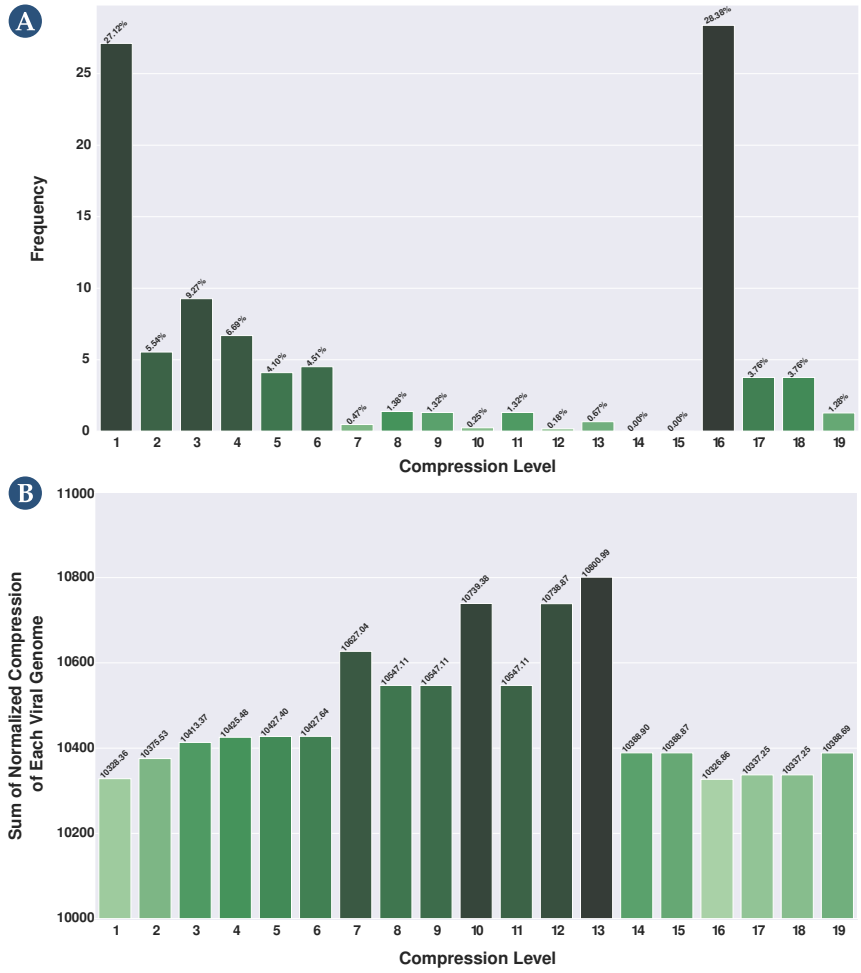


Figure 3.5: Selection of a level for GeCo3 from a pool of 19 levels. (**A**) depicts the frequency of each level providing the best NC results, and (**B**) shows for each level the sum of the NC from the compression of all reference genomes. For better visualization, please visit the website https://asilab.github.io/canvas/.

configuration performed better because most viral genomes are small and compact, where a small genomic space usually separates repetitions and IRs. Therefore, the depth of the models is more adapted to provide higher efficiency to the average of the viral genomes than, for example, a higher context model (higher than 13) that can perform marginally better in more extensive and repetitive sequences but which loses sensitivity in the average of the genomes.

Insights

During this analysis, GeCo3 was shown to have good computational time and compressibility, ability to detect IRs (unlike the other methods used), and was resistant to substitutional mutation up to 10%, showing that it can also deal with this extreme nature of genomic data, namely approximate IRs. For these reasons, GeCo3 was chosen as the method to perform this complexity analysis.

3.6.3 Viral genome analysis and its visualization

The core of the viral genomes was analysed in terms of complexity landscape, including the trends, singularities, and patterns for both the use or absence of IRs. The NC, using GeCo3, with IR_0, IR_1 and IR_2 subprograms was determined and the NCC_{IR_2} was calculated. The outcome was interpreted according to the genome type or the taxonomic group, together with the average of their genome sizes (Figure 3.6 and Table 3.1). Notice that the NC can compare proportions of the absence of redundancy independently from the sizes of the genomes. This value is complementary to the normalized redundancy. Specifically, the redundancy (R) of a sequence x is defined as

$$R(x) = |x| \log_2 |\theta| - C(x), \tag{3.6}$$

where $|x|$ is the length of the sequence, $|\theta|$ is the cardinality of the alphabet and $C(x)$ is the compressed size of x in bits, and the normalized redundancy (NR) as

$$NR(x) = 1 - \frac{C(x)}{|x| \log_2 |\theta|}. \tag{3.7}$$

Complexity landscape according to genome type

According to NCBI, the virus's genomes are of five types: dsDNA, ssDNA, dsRNA, ssRNA and mixed-DNA. On average, RNA viruses mutate faster than DNA viruses, double-strand viruses mutate slower than single-stranded viruses, and genome size correlates negatively with mutation rate [179]. Results show that ssDNA, followed by mixed-DNA and dsRNA viruses, have higher NC, whereas dsDNA genomes have the lowest (Figure 3.6; Table 3.1). In general, smaller genomes are less complex and are more likely to contain fewer repeats, and hence, have less redundancy, and ssDNA, mixed-DNA and dsRNA genomes have smaller average sequence lengths (3282 bp, 3258 bp, and 8377 bp; Table 3.1).

Table 3.1: Depiction of the genome type by the highest NC, normalized compression capacity (NCC) and difference. NCC is computed by $NCC = 1 - NC_{IR_2} > 0$, and the difference (diff) as difference $= NC_{IR_0} - NC_{IR_1}$. Furthermore, the shows the genomes' average sequence length (SL) and GC-Content (GC).

Normalized Compression				**Inverted Repeats**				**Difference**			
Genome	**NC**	**SL**	**GC**	**Genome**	**NCC**	**SL**	**GC**	**Genome**	**diff**	**SL**	**GC**
ssDNA	1.065	3282	0.447	**dsDNA**	0.029	84721	0.485	**ssDNA**	0.006	4672	0.435
mixedDNA	1.050	3258	0.491	**ssDNA**	0.026	5981	0.389	**dsDNA**	0.006	80636	0.470
dsRNA	1.047	8377	0.456	**ssRNA**	0.015	13425	0.393	**mixedDNA**	0.002	3311	0.434
ssRNA	1.013	9564	0.437	**dsRNA**	0.015	19911	0.396	**dsRNA**	0.001	6186	0.431
dsDNA	0.977	70353	0.481					**ssRNA**	0.001	10197	0.433

According to the NCC and the $NC_{IR_0} - NC_{IR_1}$ results, dsDNA and ssDNA have more

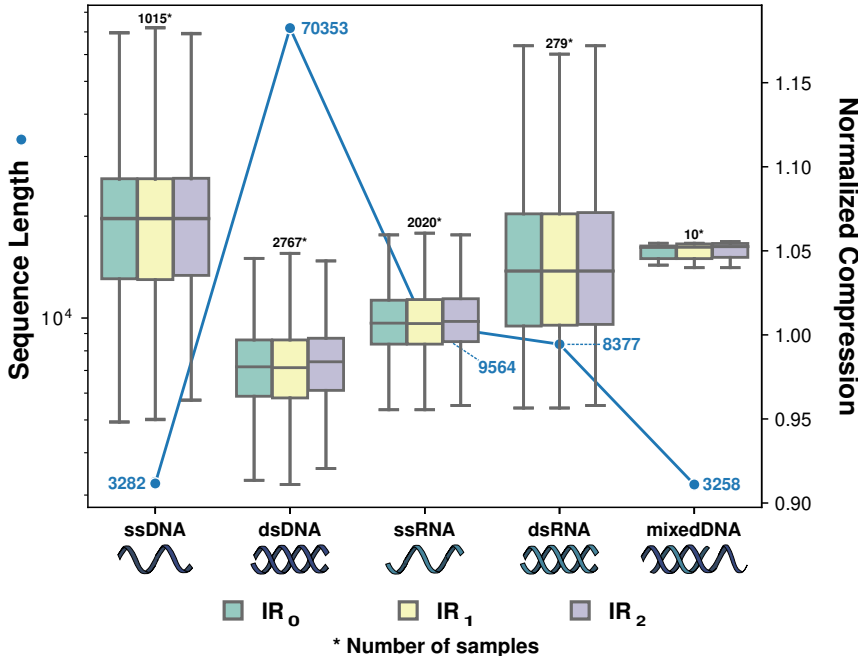


Figure 3.6: Average Normalized Compression (ANC) and average sequence length per viral group by genome type.

significant quantities of IRs than the other genome types. This can be due to ITRs present at the ends of some dsDNA viruses, such as Adenovirus and Ampullaviruses, and ssDNA virus as Parvoviruses, or other IRs structures that perform ribosomal frameshifting.

This analysis shows that the redundancy of dsDNA is higher than ssDNA, but for RNA viruses, the opposite occurs. The sequences that showed a higher normalized redundancy of the ssRNA relative to dsRNA have approximately the same length. However, the dataset of dsRNA has less than one order of magnitude in the number of sequences. This difference is natural since ssRNA is much more abundant than dsRNA. Nevertheless, this discrepancy could justify the higher normalized redundancy of ssRNA in the first instance. Although the lower average NC values of ssRNA are similar to dsRNA, the dsRNA has higher NC extremes. Therefore, we argue that this difference in the number of sequences in the dsRNA is not significant in changing the lower average of the ssRNA. Also, ssRNA is more prone to mutation than dsRNA [180]. On the other hand, extensive C to U mutations have been reported in many mammalian RNA viruses [163]. This behaviour was detected during a much faster evolution of the SARS-CoV- 2, an ssRNA virus [181]. Therefore, the faster average decrease of GC-content in ssRNA viruses explains a decrease in the ssRNA entropy, and hence, average NC. A higher GC-content (approximately 2%) of the dsRNA over ssRNA strengthens these outcomes (Table 3.1).

Complexity landscape according to taxonomic level

In complexity analysis of viral genomic sequences, when considering the realm taxonomic level (Figure 3.7), the lowest NC values were obtained for Adnaviria, Varidnaviria

and Duplodnaviria (Tables A.3 and A.4 in Appendix A). These results are consistent with the genomic grouping since they are composed exclusively of dsDNA viruses and have the highest sequence lengths. Hence, an inverse correlation between genome size and NC was generally observed as with the genome type analysis (Figures 3.6 and 3.7) and occurs across all taxonomic levels (Table A.4 of Appendix A). However, within these three realms, Adnaviria has the lowest sequence length and presents higher compressibility than Varidnaviria and Duplodnaviria, suggesting that the last are highly complex.

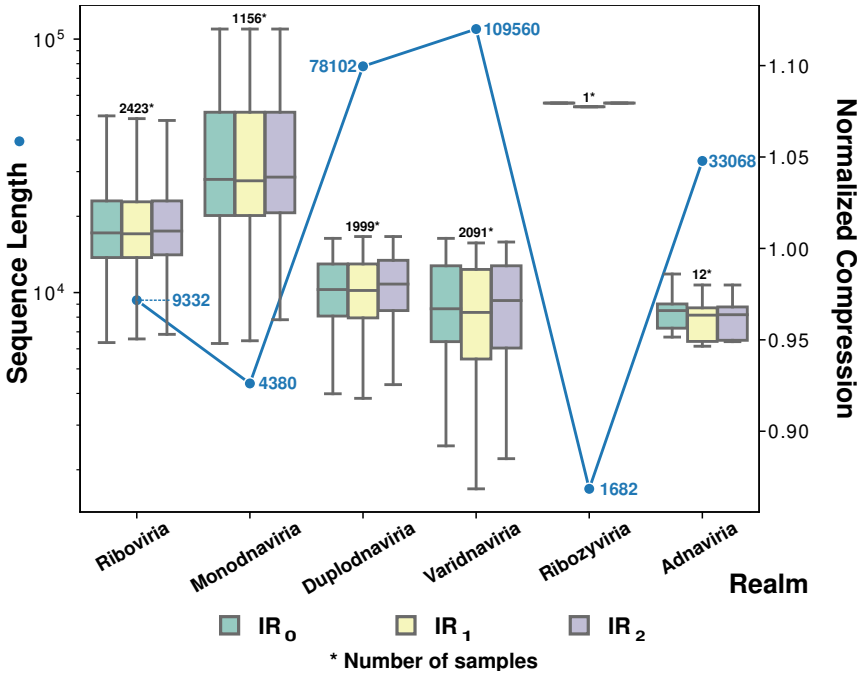


Figure 3.7: Average Normalized Compression (ANC) and average sequence length per viral group by realm. To view all boxplots by groups of realm, kingdom, phylum, class, order, family, and genus, please visit the website https://asilab.github.io/canvas/.

Regarding IRs, Adnaviria was the realm where the highest compression was obtained using the IR_2 subprogram (highest rate of IRs; Table A.5 in Appendix A). Consequently, its only recognized kingdom, Zilligvirae, also has one of the highest NCC values (Table A.5 in Appendix A). Adnaviria is a realm constituted of mostly A-form dsDNA viruses, and the ends of their genomes contain ITRs [182]. A-form is proposed to be an adaptation allowing DNA survival under extreme conditions since their hosts are hyperthermophiles and acidophiles microorganisms from the archaea domain [182, 183]. The fact that Adnaviria presented the lowest NC might indicate that their genomes require redundancy to survive such extreme environments. The kingdom of Trapavirae, belonging to the realm of Monodnaviria, is also composed of dsDNA viruses that infect halophilic archaea. Together with the kingdom of Zilligvirae, Trapavirae presented the highest difference between IRs and standard compression (Table A.6 in Appendix A). These results also support that IRs can stabilize the DNA of viruses in extreme environments. It has already been demonstrated that archaeal viruses with linear genomes use diverse solutions to protect and

replicate the genome ends, including covalently closed hairpins and terminal IRs [184].

Botourmiaviridae presented the highest complexity at the family level, followed by Alphasatellitidae and Tolecusatellitidae families (Table A.4). Botourmiaviridae is composed of ssRNA viruses that infect plants and filamentous fungi [185]. Curiously, plants and fungi have higher redundancy despite the lower redundancy of their pathogens. Alphasatellitidae and Tolecusatellitidae are families of satellite viruses that depend on the presence of another virus (helper viruses) to replicate their genomes. These satellite viruses have minimal genomes, making sense that they possess very low redundancy. Regarding IRs, Malacoherpesviridae, Herpesviridae, and Rudiviridae contained the highest $NC_{IR_0} - NC_{IR_1}$ difference (Table A.6 in Appendix A). Malacoherpesviridae and Herpesviridae are evolutionarily close dsDNA viruses, since they belong to the order of Herpesvirales [186]. Malacoherpesviridae encompasses the genera Aurivirus and Ostreavirus, which infect molluscs. Herpesviridae are also known as herpesviruses and have reptiles, birds and mammals as hosts. This family will be discussed in more detail in the following subsection. Rudiviridae is a family of viruses with linear dsDNA genomes that infect archaea. The viruses of these families are highly thermostable and can act as a template for site-selective and spatially controlled chemical modification. Furthermore, the two strands of the DNA are covalently linked at both ends of the genomes, which have long ITRs [187]. Again, these IRs could be an adaptation to stabilize the genome.

Complexity landscape of the Herpesviridae family

Here we analysed the complexity landscape of the genera of the Herpesviridae family in more detail, and the results show a significant variation between them (Figure 3.8 A). Mardivirus had the highest $NC_{IR_0} - NC_{IR_1}$ difference among all viruses, and only three other genera (out of thirteen) of herpesviruses were within the ten highest differences list (Table A.6 in Appendix A). Indeed, the genus Mardivirus had the highest compression, whereas the genus Lymphocryptovirus possessed very low compression with the IR_2 subprogram. We performed the minimal bi-directional complexity profiles of one sequence of each virus to visualize their distribution of complexity locally (Figure 3.8 C). As we can see, Human herpesvirus 4 (also known as Epstein-Barr virus) has more internal repeats (Figure 3.8 C, IR_0 profile) detected and fewer IRs (Figure 3.8 B; IR_2 profile). The opposite occurs with the Falconid herpesvirus-1 strain S-18, where IRs are more prominent than internal repetitions. Furthermore, these regions determined with compression profiles coincide with actual regions detected in the genome with other methods (Figure 3.8 C; first profile).

Next, we analyse a particular group of the Herpesviridae family, the human herpesviruses (HHVs), regarding their genome complexity and IRs abundance. Some of these viruses are involved in globally prevalent infections and cancers and are characterized by lifelong persistence with reactivations that can potentially manifest life-threatening conditions [188]. Globally, the HHVs present a higher redundancy than other viruses (Fig-

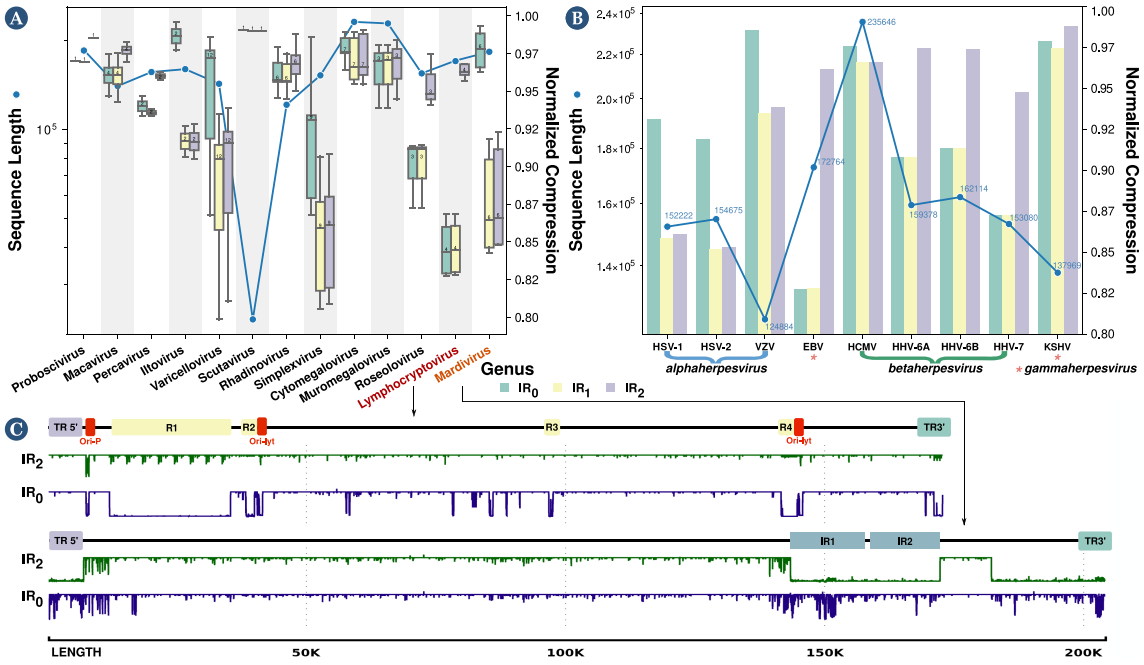


Figure 3.8: Average Normalized Compression (ANC) and average sequence length per the genera of the Herpesviridae family (**A**) and for various Human Herpesviruses (**B**).(**C**) minimal bi-directional complexity profiles of Lymphocryptovirus and Mardivirus. In panel A, two genera were selected, one with a low level of IRs (Lymphocryptovirus) and one with a high level of IRs (Mardivirus). Then, a representative reference sequence was selected (Lymphocryptovirus - Human herpesvirus 4 or EBV, NCBI Reference Sequence: $NC\_024450.1$; Mardivirus - Falconid herpesvirus 1 strain S-18, NCBI Reference Sequence: $NC\_009334.1$) and used to create minimal bi-directional complexity profiles.

ure 3.8 B). These viruses are divided into i) the alpha-subfamily members, namely herpes simplex virus type 1 and 2 (HSV-1 and HSV-2) and varicella-zoster virus (VZV), ii) the beta-subfamily of human cytomegalovirus (HCMV) and human herpesviruses 6A, 6B, and 7 (HHV-6A, HHV-6B, and HHV-7) and iii) the gamma-subfamily of Epstein-Barr virus (EBV) and Kaposi's sarcoma-associated herpesvirus (KSHV). Specifically, the EBV, one of the most potent cell transformation and growth-inducing viruses known, capable of *immortalizing* human B lymphocytes, contains a higher redundancy than the other HHVs (Figure 3.8 B). The other gamma-herpesvirus, KSHV, has the genome with the highest NC_{IR_1} (Figure 3.8 B). Unlike the beta- and gamma-subfamilies, the alpha-subfamily is characterized by a substantial quantity of IRs, as suggested by the NCs with IR_1 and IR_2 configurations (Figure 3.8 B). The VZV has the shortest genome and the highest NC within this group. These differences might be justified by the different rates of evolution within these genomes [189]. Considering the beta-subfamily members, HCMV contains a small proportion of IRs while having a substantial-high NC relative to other HHVs being analysed. Since the HCMV has the largest genome, this was surprising because the NC typically has an inverse correlation with the genome size and the quantity of IRs. The

other beta-subfamily members are the Human Herpesvirus 6A, 6B, and 7, which produced lower NCs (with IR_1 and IR_2 configurations) compared to the other HHVs, with a low quantity of IRs, an effect that their integrating function might favour. For instance, HHV-6A and 6B can integrate their genomes into the telomeres of latently infected cells [190, 191]. Thus, their genomes contain subsequences similar to the human telomere regions that can be formed by internal nucleotide repetitions [192]. As such, these are sequences with very low complexity, and hence, highly compressible.

Alternative visualization methods of the viral complexity landscape

After analysing the results, we wanted to display the results qualitatively. As such, we generated Cladograms depicting the redundancy (NC; Figure 3.9 A) and the prevalence of inverted repeats (NCC; Figure 3.9 B) on each taxonomic branch. In addition, we performed the same analysis to portray the relation between inverted and internal repetitions (Figure 3.10). These cladograms show the broad picture of the regions with more complex and less redundant sequences, regions rich in inverted repeats, and regions with a higher prevalence of inverted repeats relative to standard repetitions in the genomes.

Another way to analyse the results is by producing 3D-scatter plots of randomly sampled values obtained from computing the features' sequence length (SL), NC and GC-content (GC; Figure 3.11 A) or 2D-scatter plots of their projections (Figure 3.11 B and C), both concerning a particular taxonomic level (herein realm). Analysing the sequence length projections (Figure 3.11 B), it is evident that there is a logarithmic downtrend of the NC with the increase in sequence length. Thus, although longer sequences have, on average, greater complexity (absolute quantities), they have higher redundancy, which the data compressor takes advantage of to perform a better compression. On the other hand, the NC vs the GC-content displays a normal distribution around the 0.5 GC-mark, with higher complexities associated with a similar frequency of occurrence of the four bases A, C, G, T/U (Figure 3.11 C). This result also makes sense since, in principle, a well-distributed frequency of bases has more complex sequences which are harder to compress. The image also shows that the NC, GC and SL seem to discriminate between different taxonomic groups (Figure 3.11). As such, in the following section, we analyse the classification capability of these features.

Insights

During this analysis, we identify that, on average, dsDNA viruses are the most redundant (least complex) according to their size, and ssDNA viruses are the least redundant. Contrarily, dsRNA viruses show a lower redundancy relative to ssRNA viruses.

In the analysis at the taxonomic level, we have found indications that some viruses that infect extremophiles are more redundant and possess more IRs, indicating the presence of an adaptation to stabilize the genome in these environments.

In-depth analysis of the human herpesviruses indicated that higher compressibility and

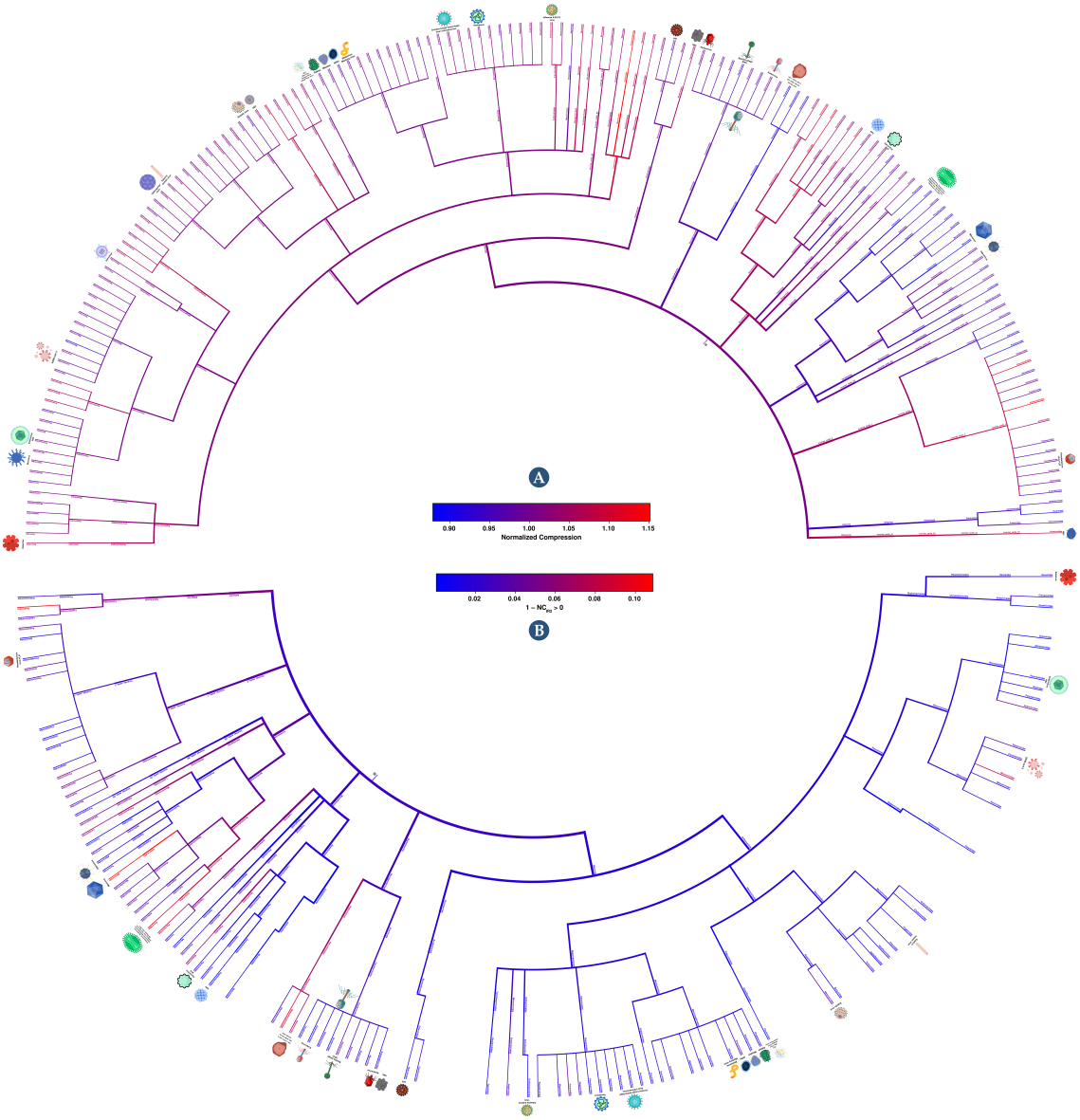


Figure 3.9: Cladograms showing average NC of each viral group (**A**), and the normalized compression capacity (NCC) (**B**). NCC results were obtained by $NCC = 1 - NC_{IR_2} > 0$. The colour red depicts the highest complexity, and blue the lowest. The first cladogram describes the NC of each taxonomic branch. Red shows genomes with less redundancy, and blue more redundancy. On the other hand, the second cladogram depicts the prevalence of inverted repeats on each taxonomic branch. Red indicates branches with genomes with a high percentage of inverted repeats, whereas blue shows branches with a low percentage. For better visualization, please visit the website https://asilab.github.io/canvas/.

abundance of inversions in herpesvirus might be associated with viral genome integration and showed that it is possible to provide the structural description of the viral genome using minimal bi-directional complexity profiles.

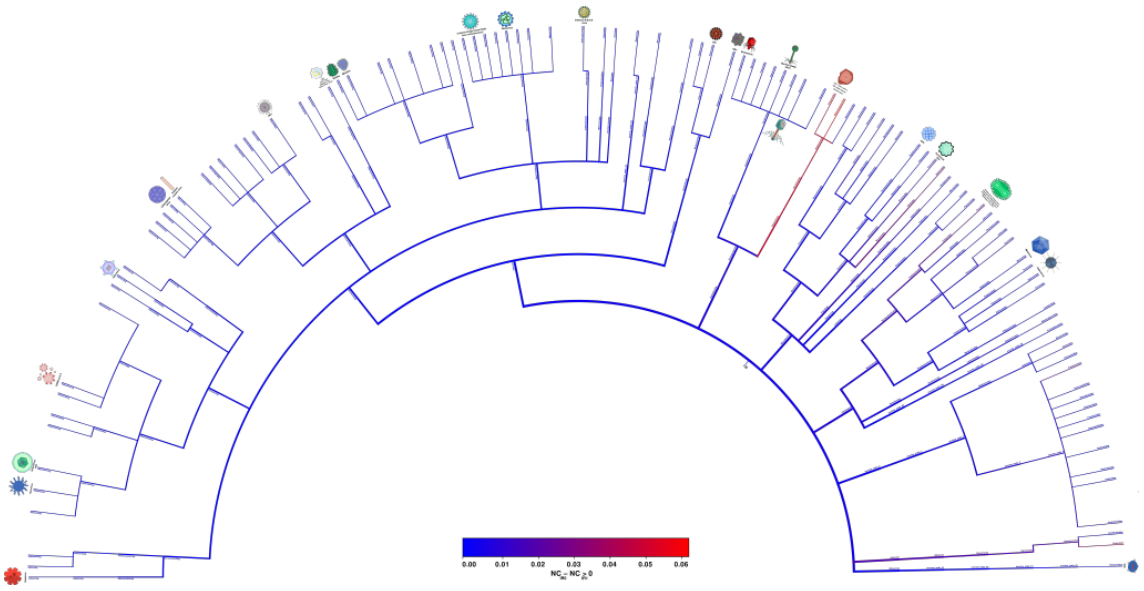


Figure 3.10: Cladogram showing average difference ($NC_{IR_0} - NC_{IR_1} > 0$). Red depicts the branches where on average, the genome possesses more inverted repetitions than internal repetitions (higher difference), whereas blue represents the branches with fewer inverted repetitions than internal repetitions (smaller difference).

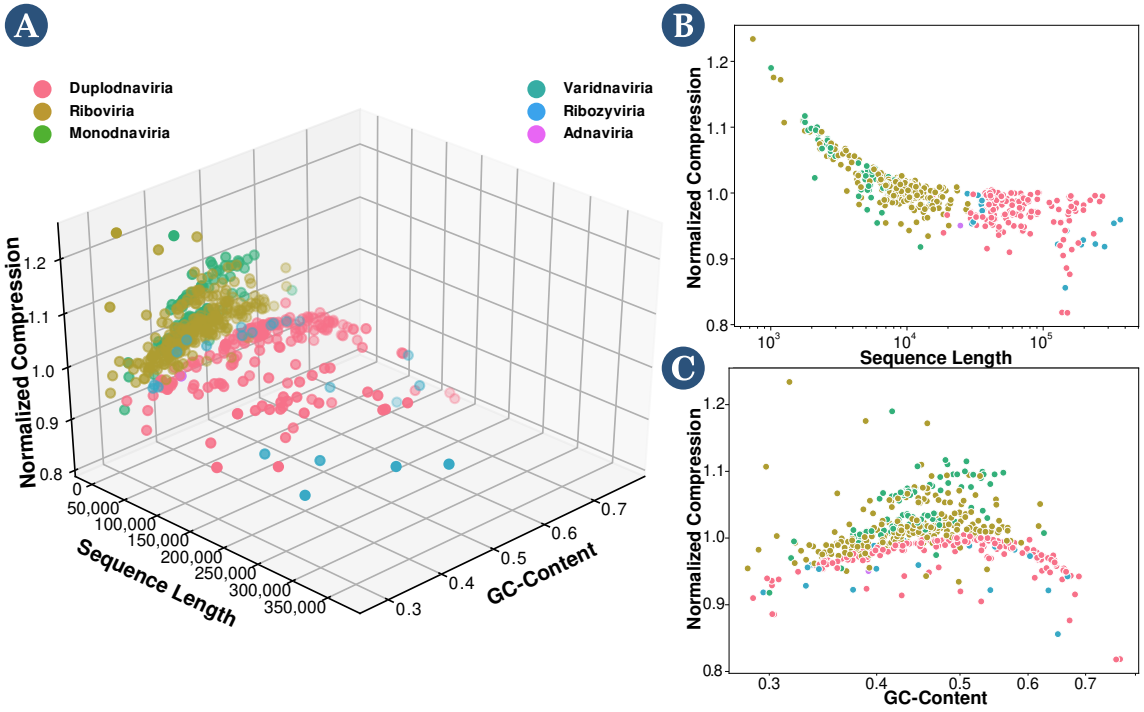


Figure 3.11: Scatter-plots of Normalized Compression vs. sequence length and GC-Content (**A**), Scatter-plots of Normalized Compression vs. sequence length (**B**) and Normalized Compression vs. GC-Content (**C**).

3.7 Taxonomic classification

Metagenomics is a vast field that enables the discovery of taxonomic and physiological information of species collected from their natural environment. Acquiring this information is especially important given the role that microorganisms play in terms of the structural and functional balance of the ecosystem, medicine and patient health status, as well as industrial and economic activity [193]. Due to its importance, this field has become instrumental in medicine, forensics, and exobiology [104].

The rapid growth of metagenomics has led to the generation of large amounts of genomics data. As a result, several institutions host distinct genomic repositories. These data can be used to help researchers conduct genomic analysis. However, to do so, a correct characterization and identification of the genomic samples present in these repositories are vital since repositories correctly catalogued facilitate filtering and selection for research. This characterization can be done by identifying the organisms in the metagenomics samples. However, the identification of the organism is not always conclusive [194]. The leading cause is that most classification pipelines rely on reference-based comparison approaches to perform organism identification [195], where the reconstructed sequence is compared to a collection of references stored in a database. Problems occur when performing reference-based identification since this method is ineffective when there is a variation between the sequences of known organisms in the database, when irregularities are introduced during the reconstruction process of the organism being identified, or when a new organism is being sequenced [196, 197].

Other approaches have recently been used to identify organisms without reference-based methods. Karlicki *et al.* [198] developed a deep learning-based categorization system. The system first classifies nuclear and organellar eukaryotic fractions. Afterwards, it separates organellar sequences into plastidial and mitochondrial categories. Zhang *et al.* [199] used k-mers as genomic features for viral genome identification.

In the previous section, we observed that the NC, GC and SL features seem capable of discriminating between different viral taxonomic groups. Consequently, in this section, we analyse and propose a feature-based methodology for classification that uses NC and other simple features. We will start by performing viral classification using a viral database and then archaea classification in a database of archaea genomic sequences.

3.7.1 Viral classification

SOTA of viral taxonomic classification

Although sequence alignment is essential for genomic analysis, the fact that pairwise and multiple alignment methods are often slow led to the popularization of fast alignment-free methods for sequence comparison. Most alignment-free methods are based on word frequencies for words of a fixed length or word-matching statistics. Others use the length of maximal word matches, and others rely on spaced-word matches (SpaM). These inex-

act word matches allow mismatches at certain pre-defined positions and can accurately estimate phylogenetic distances between DNA or protein sequences using a stochastic model of molecular evolution [200]. This approach has also been updated as the Multiple Spaced-Word Matches (Multi-SpaM) method, which is based on multiple sequence comparison and maximum likelihood [201]. Regarding viral sequences, many studies were performed on alignment-free sequence comparison and classification. For instance, Garcia et al. [202] developed a dynamic programming algorithm to create a classification tree using metagenome viruses. To create the classification tree, k-mer profiles of each virus' metagenome were created, and proportional similarity scores were generated and clustered. Using the JGI metagenomic and NCBI databases, the authors identified the correct virus (including its parent in the classification tree) 82% of the time. Zhang et al. [199] created an alignment-free method that employed k-mers as genomic features for a large-scale comparison of complete viral genomes. After determining the optimal k for all 3,905 complete viral genomes, a dendrogram was created, which shows consistency with the viral taxonomy of the ICTV and the Baltimore classification of viruses. He et al. [203] proposed an alignment-free sequence comparison method for viral genomes based on the location correlation coefficient. When applied to evolutionary analysis of common human viruses, including SARS-CoV-2, Dengue virus, Hepatitis B virus, and human rhinovirus, it achieves the same or even better results than alignment-based methods. Finally, Huang et al. [204] proposed a classification method based on discriminant analysis, employing the first and second moments of positions of each nucleotide of the genome sequences as features, and performed classification of genomes regarding their Baltimore classification and family (12 families), obtaining a maximum value of accuracy of 88.65% and 85.91%, respectively.

Feature-based archaea taxonomic classification

Considering these studies, here we aim to create an alignment-free feature-based classification method. We performed eight classification tasks using the viral dataset. Specifically, the sequences were classified regarding their genome type, realm, kingdom, phylum, class, order, family, and genus. To that end, we conducted a random 80-20 train-test split on the dataset to perform viral classification. Due to classes being imbalanced in the dataset, we performed several actions. First, we did not consider classes with fewer than four samples. Depending on the classification task, the number of samples decreased from 6,091 to the values shown in Table 3.2 (number of classes column). Secondly, we performed a stratified train-test to ensure each label's representability in the train and test sets. Finally, instead of performing k-fold cross-validation, we performed the random train-test split fifty times and retrieved the average of the evaluation metrics. Then, we computed the Accuracy and the Weighted F1-score to select the best-performing method.

We applied 5 types of classifiers: Discriminant Analysis (LDA) [166], Gaussian Naive Bayes (GNB) [167], K-Nearest Neighbors (KNN) [168], Support Vector Machine (SVM)

[169] and XGBoost (XGB) classifier [170].

We performed classification using six different features: Sequence Length (SL), GC-Content (GC), the NC values for the best performing model, and the NC of the same model with IR configuration to 0, 1 and 2. These six features were fed to all the classifiers, and the accuracy and weighted F1-score were measured to determine which classifier was best suited for this task.

Tables 3.2 and 3.3 depict the accuracy and weighted F1-score values obtained for each classifier. For all classification tasks, the best performing classifier was the XGBoost classifier.

Table 3.2: Accuracy results obtained for viral taxonomic classification tasks regarding genome type, realm, kingdom, phylum, class, order, family, and genus. The classifiers used were Linear Discriminant Analysis (LDA), Gaussian Naive Bayes (GNB), K-Nearest Neighbors (KNN), Support Vector Machine (SVM), and XGBoost classifier (XGB).

Classification	**Classes**	**Samples**	**LDA**	**GNB**	**SVM**	**KNN**	**XGB**
Genome	5	6089	67.32	74.14	72.41	84.40	**87.25**
Realm	5	5799	75.95	80.95	81.38	88.71	**92.57**
Kingdom	10	5788	73.49	78.76	78.41	85.49	**90.96**
Phylum	17	5778	61.59	56.75	55.88	71.28	**83.41**
Class	34	5845	51.15	52.95	47.56	63.47	**80.23**
Order	48	5838	48.89	55.65	48.89	60.62	**79.62**
Family	102	5990	36.64	43.24	27.05	42.99	**74.46**
Genus	360	4673	44.60	36.79	18.82	17.65	**68.71**

Table 3.3: F1-score (F1) results obtained for viral taxonomic classification tasks regarding genome type, realm, kingdom, phylum, class, order, family, and genus. The classifiers used were Linear Discriminant Analysis (LDA), Gaussian Naive Bayes (GNB), K-Nearest Neighbors (KNN), Support Vector Machine (SVM), and XGBoost classifier (XGB).

Classification	**Classes**	**Samples**	$\mathbf{F1_{LDA}}$	$\mathbf{F1_{GNB}}$	$\mathbf{F1_{SVM}}$	$\mathbf{F1_{KNN}}$	$\mathbf{F1_{XGB}}$
Genome	5	6089	0.6549	0.736	0.6989	0.836	**0.8662**
Realm	5	5799	0.7496	0.8001	0.7949	0.8817	**0.9234**
Kingdom	10	5788	0.7238	0.7640	0.7512	0.8410	**0.9039**
Phylum	17	5778	0.5824	0.5226	0.4435	0.6891	**0.8299**
Class	34	5845	0.4780	0.4562	0.3803	0.5896	**0.7963**
Order	48	5838	0.4435	0.4798	0.3832	0.5462	**0.7884**
Family	102	5990	0.3042	0.3517	0.1681	0.3429	**0.7323**
Genus	360	4673	0.3600	0.2956	0.0682	0.0621	**0.6561**

Following this, we analysed if all features were necessary. For that purpose, the XGBoost classifier was used with only the NC feature, the NC with SL and GC, and finally,

using all features. The obtained accuracies are shown in Table 3.4, and the weighted F1-score results are shown in Table 3.5.

Table 3.4: Results obtained for viral taxonomic classification task regarding the genome type, realm, kingdom, phylum, class, order, family, and genus using XGBoost classifier. The features used were the genome sequence length (SL), the GC-content (GC) and the NC values for the best model, the same model with IR configuration to 0, to 1 and 2. The results correspond to the accuracy, and the probability of a random sequence being correctly classified (p_{hit}) using a random classifier.

Classification	**Classes**	**Samples**	$\mathbf{p_{hit}}$	**Accuracy**				
				NC	**NC + GC**	**NC + SL + GC**	**All without SQ**	**All Features**
Genome	5	6089	20.00	75.57	80.60	87.11	81.24	**87.25**
Realm	5	5799	20.00	77.90	84.56	92.25	86.16	**92.57**
Kingdom	10	5788	10.00	76.44	82.51	90.82	84.06	**90.96**
Phylum	17	5778	5.88	63.97	70.69	82.36	73.21	**83.41**
Class	34	5845	2.94	59.83	65.90	79.05	68.66	**80.23**
Order	48	5838	2.08	58.44	65.08	78.20	67.88	**79.62**
Family	102	5990	0.98	43.35	54.06	72.46	58.34	**74.46**
Genus	360	4673	0.28	35.59	50.02	67.32	54.23	**68.71**

Table 3.5: F1-score obtained for the viral taxonomic classification task regarding genome type, realm, kingdom, phylum, class, order, family, and genus. The features used were the genome sequence length (SL), the GC-content (GC) and the NC values for the best model, the same model with IR configuration to 0, 1 and 2.

Classification	**Classes**	**Samples**	**F1-score**				
			NC	**NC + GC**	**NC + SL + GC**	**All without SL**	**All Features**
Genome	5	6089	0.7490	0.7988	0.8649	0.8051	**0.8662**
Realm	5	5799	0.7726	0.8401	0.9200	0.8569	**0.9234**
Kingdom	10	5788	0.7518	0.8131	0.9026	0.8295	**0.9039**
Phylum	17	5778	0.6234	0.6926	0.8194	0.7188	**0.8299**
Class	34	5845	0.5742	0.6404	0.7844	0.6705	**0.7963**
Order	48	5838	0.5568	0.6292	0.7736	0.6598	**0.7884**
Family	102	5990	0.4112	0.5187	0.7118	0.5636	**0.7323**
Genus	360	4673	0.3248	0.4661	0.6417	0.5089	**0.6561**

The best results are obtained when using all features. This improvement increased when the number of classes was higher, demonstrating that the different compression sub-

programs (IR_0, IR_1, and IR_2) are more helpful in classifying more specific taxonomic groups. The results show a decrease in accuracy and F1-score when there is an increase in the number of classes. Specifically, we obtained the best performance in the realm classification of the virus (accuracy - 92.57%, F1-score - 0.9234) and our lowest performance in genus classification (accuracy - 68.71%, F1-score - 0.6561). This decrease is mainly because the average number of samples per class decreases as the number of classes increases. As such, many classes may still have an insufficient number of samples to be accurately classified. Figure 3.12 represents the number of samples (genome sequences) per viral genus.

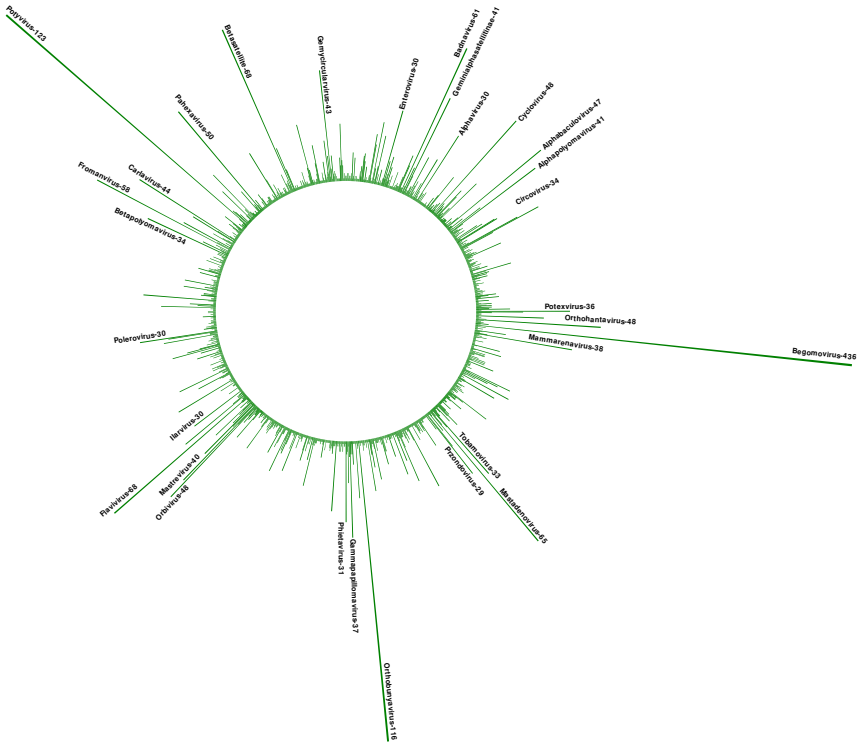


Figure 3.12: Frequency of genome sequences per viral genus using radial plot.

Furthermore, part of the classification inaccuracies can be explained by possible errors in the assembly process of the original sequence or eventual sub-sequence contamination of parts of the genomes. Moreover, other inaccuracies could be due to several genomes being reconstructed using older methods that have been improved since then [205].

Despite being pertinent, the alignment-free studies are not directly comparable due to sample size, absence of classification metrics and source code. Furthermore, the method proposed in this work is not only alignment-free but also feature-based, providing a higher level of flexibility since it does not resort directly to the reference genomes but rather features that the biological sequences share. Therefore, we compared our results with the outcome obtained using a random classifier as a measure of comparison. Specifically, for each task, we determined the probability of a random sequence being correctly classified (p_{hit}). Overall, there is a vast improvement over the random classifier, showing the

importance of the features used in the classification process.

These results show that we could automatically and accurately distinguish between viral genomes at different taxonomic levels using the XGBoost classifier with all features (NC with different configurations, GC-content and SL). However, a decrease in accuracy when approaching the lowest taxonomic levels was observed, which can be increased with future entries to the database. Furthermore, when analysing viral sequences from environmental samples, the original viral genome length is often unknown. Therefore, we computed the accuracy of a model that does not include this feature. Although we obtain a lower accuracy and F1-score, the results indicate that the method is still reliable for fast and efficient viral taxonomic identification in these scenarios.

3.7.2 Archea classification

Since the results obtained for in-depth taxonomic identification of viruses seem promising, we tried to analyse this methodology when applied to other genome sequences. In this case, we experimented with the archaea domain.

Data description

The dataset comprises all archaea genome samples (FASTA format) in the NCBI database on 10 January 2022. In total, 2,437 samples were retrieved. After retrieving all the data from the NCBI database, the dataset was processed according to Section 3.6.1 and filtered by discarding sequences without any taxonomic information. As a result of the filtering process, a total of 667 (of the initial 2,437) samples remained for classification. All these samples contain genomic sequences and their taxonomic descriptions.

Feature-based viral taxonomic classification

Using the archaea dataset, we performed taxonomic classification of the archaea regarding their phylum, class, order, family, and genus. The same setup as in viral analysis was performed, using the same classifiers and stratified train-test split. We also did not consider classes with fewer than four samples. Depending on the classification task, the number of samples decreased from 667 to the values shown in Table 3.6. The table also shows the results of feeding all genomic features to the classifiers.

For all classification tasks, the best performing classifier was the XGBoost classifier, with the second-best results being obtained using LDA for the majority of the classification tasks.

On the other hand, different features were fed to the XGBoost classifier to determine the best features to be used. Table 3.7 shows the most relevant entries (only NC feature and when combining all features).

Overall, the best results were obtained for all feature groups in the phylum classification task. The worst classifications were present in the order/family classification task despite

Table 3.6: Accuracy (Acc) and F1-score (F1) results for archaea taxonomic group classification using all features. The classifiers used were Linear Discriminant Analysis (LDA), Gaussian Naive Bayes (GNB), K-Nearest Neighbors (KNN), Support Vector Machine (SVM), and XGBoost classifier (XGB).

Group	Classes	Samples	SVM		GNB		KNN		LDA		XGB	
			Acc	F1-score	Acc	F1-score	Acc	F1-score	Acc	F1-score	Acc	F1-score
Phylum	5	660	44.36	0.4436	48.35	0.4835	68.45	0.6845	60.11	0.5600	**96.09**	**0.9606**
Class	10	615	29.10	0.2910	42.33	0.4233	52.41	0.5241	59.76	0.5674	**93.35**	**0.9314**
Order	20	634	23.64	0.2364	31.01	0.3101	36.53	0.3653	52.33	0.4804	**89.67**	**0.8930**
Family	29	623	22.84	0.2284	31.77	0.3177	33.92	0.3392	51.82	0.4703	**89.50**	**0.8889**
Genus	55	543	12.47	0.1247	27.69	0.2769	17.47	0.1747	60.67	0.5753	**91.50**	**0.9048**

Table 3.7: Accuracy (Acc) and F1-score (F1) results from archaea taxonomic group classification using XGBoost classifier. The features used were the genome's sequence length (SL), GC-content (GC), and NC.

Group	Classes	Samples	$\mathbf{P_{hit}}$	NC		All	
				Acc	F1-score	Acc	F1-score
Phylum	5	660	20.00	65.76	0.6423	**96.09**	**0.9606**
Class	10	615	10.00	50.65	0.4933	**93.35**	**0.9314**
Order	20	634	5.00	41.75	0.3963	**89.67**	**0.8930**
Family	29	623	3.45	41.57	0.3872	**89.50**	**0.8889**
Genus	55	543	1.82	37.47	0.3263	**91.50**	**0.9048**

not having the highest number of classes. In contrast, remarkable results were obtained for the genus classification task, which has the highest number of classes.

For comparison purposes, we assessed the outcomes obtained using a random classifier. Specifically, for each task, we determined the probability of a random sequence being correctly classified (p_{hit}). Overall, there is a substantial improvement relative to the random classifier, showing the features' importance in the classification process. For instance, the probability of a random classifier correctly identifying the genus of a given sequence was 1.82%, whereas, in our best classification, we obtained 91.50% accuracy. These results are particularly encouraging, given the small sample size and the diversity of labels in the dataset.

In the genomic features, despite the NC being the most relevant, the ensemble of features improved the accuracy and F1-score in all tasks. These results are significant in archaea, given that this classification is challenging since a set of these archaea have not been isolated in a laboratory and were only detected by their gene sequences in environmental samples.

Furthermore, these results are congruent with the results obtained for viral classification and reinforce that this method can be efficiently utilized in identifying organisms.

Insights

The results show that the efficient approximation of the Kolmogorov complexity of sequences can accurately perform taxonomic identification and classification. Regarding the features selected, surprisingly, rich genomic features suffice to correctly identify viruses and archaea's taxon, even with a large number of labels. These results are significant since they show that this methodology facilitates the classification of viruses and archaea, usually identified from metagenomic samples.

3.8 Summary

In this chapter, we have analysed the usage of approximations of the Kolmogorov complexity to analyse natural genomic sequences. The results show that efficient approximation of the Kolmogorov complexities of viral sequences as measures that quantify the absence of redundancy profoundly improves genomes' identification, classification, and organization. The same was verified for archaea taxonomic identification.

After optimization, we benchmark a specific data compressor (GeCo3) against other approaches for computing an upper bound of the sequence complexity. Specifically, GeCo3 was compared with high compression ratio general-purpose data compressors (PAQ and cmix) and a measure that combines small algorithmic programs and Shannon entropy (BDM). Unlike the other approaches, we show that GeCo3 can efficiently address and quantify regions properly described by simple algorithmic sources, namely inverted repeats (exact and approximate), among other characteristics.

Using an optimized compression level of GeCo3 in an extensive viral dataset, we provide a comprehensive landscape of the viral genome's complexity, comparing the viral genomes at several taxonomic levels while identifying the genome regarding the lowest and highest proportion of complexity. Specifically, on average, dsDNA viruses are the most redundant (least complex) according to their size, and ssDNA viruses are the least redundant. Contrarily, dsRNA viruses show a lower redundancy relative to ssRNA viruses.

By performing analysis at various taxonomic levels, we have also found evidence to support that viruses infecting archaeal extremophiles possess a more redundant genome and abundance of IRs. This suggests an adaptation of the genomes to resist the environment they inhabit.

We perform an in-depth analysis of the human herpesvirus regarding its genome complexity and abundance of IRs. We suggest that higher compressibility and abundance of inversions in herpesvirus may be associated with viral genome integration.

We describe and use minimal bi-directional complexity profiles of one sequence of each virus to visualize the distribution of complexity of these sequences locally. These profiles

can describe structural regions detected in the genome with other methods, proving the description capability of data compression at a structural level.

We reveal the importance of efficient data compression in genome classification tasks, explicitly showing that the complexity, when combined with simple measures (GC-content and size), is efficient in accurately distinguishing between viral and archaea genomes at different taxonomic levels without using direct comparisons between sequences.

The methods and results presented in this work provide new frontiers for studying viral genomes' complexity, magnifying the importance of developing efficient data compression methods for automatic and accurate viral analysis and its usage for organism taxonomic identification.

Chapter 4

Complexity analysis of artistic paintings

[d] Dream world - Jorge Miguel Silva, 2021.

"The purpose of art is to lay bare the questions
that have been concealed by the answers."
– James Baldwin

4.1 Contextualization

After using Kolmogorov complexity approximations to analyse 1-dimensional natural strings, specifically genomes, in this chapter, we will focus on studying and applying the Kolmogorov complexity approximations in 2-dimensional data structures. In this case, we will analyse images of artistic paintings.

We use the block decomposition method and lossless data compressors to examine the potential of these information-based measures as descriptors of images. We notice that both approaches have different merits and initially consider them both in analysing artistic paintings. After the first analysis of both author and style complexity, we use compression measures combined with the roughness exponent (α) of the two-point height difference correlation (HDC) function and observe that these measures can distinguish different styles. Next, we create different regional complexity descriptions of paintings and use them to create author complexity fingerprints. From these fingerprints, we create cladograms depicting relations and influences between authors. Finally, we use these measures to obtain state-of-the-art author and style classification results. This chapter follows the structure:

- Research questions and contributions;
- Introduction;
- Methods;
- Kolmogorov approximations in images;
- Artist painting analysis;
- Artist painting classification;
- Summary.

4.2 Research questions and contributions

In this chapter, we use approximations of the Kolmogorov complexity to analyse images. Concretely, we introduce novel solutions for automatic computational analysis of artistic paintings and the problem of artist authentication. When addressing artist authentication and classification, several questions arise: What defines a painter's style? How does the author expose information? How does the author differ from and relate to other artists? Furthermore, taking inspiration from information theory: How do we best quantify information in a painting? How is the information utilized across the canvas? Moreover, what can information quantification tell us about the author's style, way of painting, and relationships with other authors? These complex questions are at the core of this chapter, where we describe and compare solutions for unsupervised measures of

probabilistic and algorithmic information in images (2D) of artistic paintings.

Our contributions are as follows:

- We perform a direct comparison between state-of-the-art unsupervised probabilistic and algorithmic information measures to specify each measure's strengths and weaknesses.
- We show that hidden patterns and relationships present in artistic paintings can be identified by analysing their complexity.
- We show an efficient stylistic descriptor by combining the Normalized Compression and a measure of the paintings' roughness.
- We propose a new descriptor of the artists' style, artistic influences, and shared techniques.
- We show that average local complexity describes how each artist typically composes and distributes the elements across the canvas and, therefore, how their work is perceived.
- We demonstrate that these measures can serve as useful auxiliary features capable of improving current methodologies in the classification of artistic paintings.

4.3 Introduction

Artistic paintings are concrete visual expressions of human evolution and creativity to share emotions, values, visions, beliefs, and trends of history and culture. The creation, interpretation, and analysis of artistic paintings are social, contextual, subjective, passive, and, beyond superficial characteristics, complex to compute and automatize [206]. In particular, it is theorized that art is an output of social agents, particularly a human experience, that can only be imitated by machines [207].

The process of measuring the information contained in paintings is non-trivial. For example, artistic paintings contain information related to schools, periods, and artists [208]. The artistic community widely uses automatic computational analysis of artistic paintings for authentication of artistic paintings [209, 210]. Currently, this process does not substitute human experts completely, but it is an essential additional control for fraud and misleading detections [211]. Furthermore, applying new techniques and pre-existing ones that are new to the field can be helpful for authorship attribution, fraud detection, art style categorization and organization, and even for art content explanation.

The idea of automatic computational analysis of artistic paintings is mature [209, 210], and the artistic community has widely relied on it to authenticate artistic paintings. Specifically, the characteristics of artistic paintings have been analysed through several probabilistic techniques and properties, namely fractal [212], wavelet-based [209], hidden

Markov models [213, 214], Fisher kernel based [215], sparse coding model [216, 217], colour and brightness [210], illumination [218], stroke [219, 220], print index [221], and entropy-based analysis [222, 223]. Recently, the work of Machado and Lopes [223], using fractional calculus, showed the potentiality of measures based on entropy to describe the hierarchical clustering of paintings and their correlation with artistic movements.

Regarding style and author classification, several recent studies have proposed using Convolutional Neural Networks (CNNs). A straightforward approach is to combine features extracted from multiple CNN layers, as proposed by Peng *et al.* [224]. Another more effective approach is based on representing images by the principal components of a Gram matrix, which captures correlations across the different feature maps obtained from a convolutional layer of a pre-trained deep CNN, such as VGG16 or VGG19. Mao *et al.* [225] combine this representation with the features from all the five convolutional blocks of the VGG16, learning a joint representation that can simultaneously capture the content and style of visual arts. On the other hand, Chu *et al.* [226] apply a support vector machine (SVM) to the Gram representation to perform author and style classification. Then, they improve the results by automatically learning correlations between feature maps.

As previously mentioned, in this chapter, we bring the notion of Kolmogorov complexity to the field of artistic painting analysis and classification. Measuring the information in paintings requires fast, efficient, and automatic computation due to the diversity and large number of existing artistic paintings [227]. To measure the information (or complexity) contained in paintings, we first need to define the quantity of information in an image. We define the quantity of information of an image as the smallest number of bits required by a model to represent an image losslessly. To perform this task, the model searches for unknown patterns of similarity between sub-regions of the image [228–230] and uses this information to create this compressed representation of the image, relying exclusively upon the patterns of the two-dimensional pixels without using exogenous information.

In the following sections, we will explain how data compression and other Kolmogorov complexity approximations behave as descriptions of images and artistic paintings.

4.4 Methods

In this section, we describe the measures used, their normalizations, the methodology, and the compression benchmark performed.

4.4.1 Information-based measures

In this subsection, we describe the Normalized Compression and two BDM normalizations when applied to this problem. Then, we establish the local application of the Normalized Compression to create a complexity matrix for each author and the methods used to create a distance matrix and the cladogram.

Normalized Compression (NC)

We used NC as described in Chapter 2. Since we consider a binary matrix of each image, $|\theta| = 2, \log_2 2 = 1$. Given the normalization, the NC allows comparing the information contained in the strings independently from their sizes [157].

Normalized Block Decomposition Method (NBDM)

In this chapter we use both normalizations of BDM described in Equations (2.9) and (2.10) of Chapter 2. We perform a direct comparison between the NC and the NBDM_1, and we compare the two types of BDM normalization and their impact on the results.

Local complexity analysis using the Normalized Compression

The NC was used to approximate the local (or regional) complexity of images of artistic paintings. To that end, all the dataset images were divided into 16x16 blocks (256 equal regions) and the NC was computed for each block, generating a complexity matrix. Other patch sizes were also tested, specifically patch sizes of 8x8 and 32x32 blocks. Following this operation, the average complexity matrix was generated for each author, using the complexity matrices of their paintings. The average complexity matrices were then used to obtain a similarity matrix, in which the distance between matrices was determined as

$$\mathrm{d(A, B)} = \sum_{i=0}^{n} \sum_{j=0}^{n} \mid a_{ij} - b_{ij} \mid, \tag{4.1}$$

where d is the distance between the complexity matrix A and B, and a_{ij} and b_{ij} are the complexity values at the index i and j of matrices A and B, respectively. Subsequently, using the similarity matrix, a cladogram was computed resorting to two methods, namely UPGMA (unweighted pair group method with arithmetic mean) [231] and the Kruskal minimum spanning tree algorithm [232], in order to portray complexity relationships among different authors.

4.4.2 Two-point height difference correlation function

The two-point height difference correlation (HDC) function was computed to quantify brightness contrast as

$$\mathrm{HDC}(r) = \overline{[h(\vec{x} + \vec{r}) - h(\vec{x})]^2} = \frac{1}{N_r} \sum_{\vec{x}, |\vec{r}| = r} [h(\vec{x} + \vec{r}) - h(\vec{x})]^2, \tag{4.2}$$

where r is the distance between two-pixel points, the overbar represents the spatial average at a fixed distance r for all possible points, N_r is the number of possible pairs at a distance r, $h(x)$ is pixel intensity at position x. Using the HDC function, its roughness exponent as

$$\alpha = \frac{\log_{10}(HDC(r_{final})) - \log_{10}(HDC(r_{initial}))}{\log_{10}(r_{final}) - \log_{10}(r_{initial})}, \tag{4.3}$$

where the roughness exponent (α) is the slope of the HDC curve in a double logarithmic plot of the surface growth model. The slope was calculated from $r_{initial} = 10$ to r_{final}, which matches the point where the HDC function saturates, approximately 30% of the image's width.

4.4.3 Dataset

The dataset used for the complexity analysis of artistic paintings contains 4,266 images of paintings by 91 artists with approximate geometric sizes [208]. The 91 artists are well-known, such as Claude Monet, Frida Kahlo, Henri Matisse, Jackson Pollock, Picasso, Rembrandt, and Salvador Dali.

4.4.4 Assessment pipeline

In order to evaluate the information-based measures fairly, we designed a pipeline for processing images. It respects the following steps: obtaining the dataset images; converting the images to PGM format; quantization of the images to 8 bits (256 levels) using the Lloyd-Max algorithm; binarization of the images (conversion to 01 format in ASCII) and finally, applying the information-based measurements (NC, NBDM_1 and NBDM_2).

Quantization was performed to reduce the precision of the pixels (alphabet) in images, enabling the filtering of minor variations that might occur during the digitalization process. Lloyd-Max algorithm [233, 234] was used to make the errors small in the regions where the signal is most likely. In addition, the images were binarized since the BDM currently only supports a small alphabet.

4.5 Kolmogorov approximations in images

In this work, we develop, use, and compare unsupervised pattern recognition techniques to quantify information in images of artistic paintings. We rely on two approaches, namely data compression using the NC, and the Block Decomposition Method (BDM), to estimate information of both probabilistic and algorithmic sources.

4.5.1 Finding an effective data compressor

To compute the NC, we have to find an effective data compressor, meaning a compressor that best represents each image, while using practical resources. Since our aim is later to apply this measure to a dataset of artistic paintings, we compared seven compression tools, namely GZIP [128], BZIP2 [129], XZ [235], LZMA [130], AC [236], PPMD [237], and PAQ8 [238].

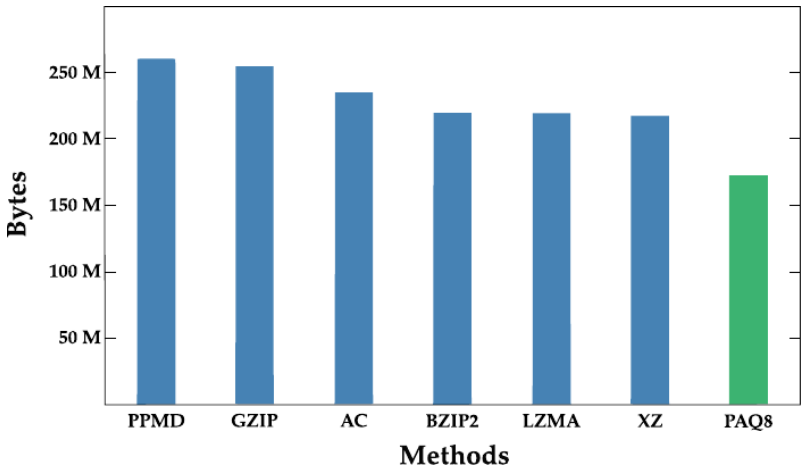


Figure 4.1: Benchmark of lossless data compression tools specifically for the processed dataset of artistic paintings. The y-axis depicts the sum of bytes to compress the dataset, where each image was compressed individually using each tool.

As depicted in Figure 4.1, the PAQ8 tool shows the best compression ratio for this dataset. It shows an improvement of $\approx 26\%$ over the second-best tool (XZ). The disadvantage is the use of higher RAM and substantially more computational time. Nevertheless, since our purpose is to find the number of bits of the shortest program to reproduce the image, it is affordable to spend these computational resources. Therefore, we used the PAQ8 tool to compress each of the quantized images. The code was compiled using the package provided from the PAQ website[154]. The version selected was PAQ8kx v7, and it ran with the mode with the highest level to achieve the best compression rates at the expense of speed.

The PAQ8 compressor uses a context mixing algorithm between many models independently predicting each quantized pixel's next bit [239]. The predictions are combined using a neural network and arithmetic coding [240, 241]. For more information on PAQ, see the work of Knoll and Freitas [153]. The computations ran on a single-core Ubuntu Linux computer at 2.13 GHz with 16 GB of RAM. Using this machine, the compression of the whole dataset with PAQ8 required approximately 270 hours of real-time without parallelization. The results indicate that PAQ8 is the most effective for this dataset.

4.5.2 Comparison of NC and BDM

In order to compare NC with BDM regarding capacity to represent probabilistic and algorithmic information in images, we performed three types of tests. Namely, we compared the robustness of both measures according to increasing rates of random pixel changes in paintings, tested their application on different types of images, and assessed the minimal information bounds.

In the first test, we assessed the impact of an increasing rate of pixel editions using a pseudo-random uniform distribution and compared both information-based measures.

This approach is not identical to image noise but rather a pure edition of pixels. For this purpose, for three authors (Theodore Gericault, Marc Chagall, and Rene Magritte), we selected a painting and made 50 adulterated copies of each with an increasing edition rate (from 1 to 50%). Finally, we measured the NC, the $NBDM_1$ (Equation (2.9)), and $NBDM_2$ (Equation (2.10)) in all the paintings. $NBDM_1$ considers the normalization by the length of the input object, $NBDM_2$ performs a normalization that aims to mimic the removal of the constant factor related to Kolmogorov complexity (see Equation (2.11)).

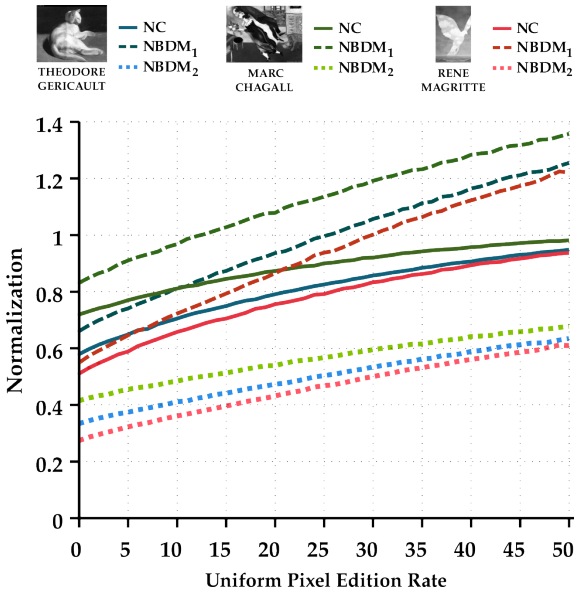


Figure 4.2: Information-based measures evaluation. Impact of increasing pseudo-random substitution on information-based measures: NC (approximated using the PAQ8 algorithm) and two BDM normalizations ($NBDM_1$ and $NBDM_2$).

Figure 4.2 depicts the values obtained for the NC and BDM. The results show that, when using the same type of normalization, NC is more robust to the increment of pixel edition than NBDM ($NBDM_1$). Since the $NBDM_2$ normalization does not consider the constant of the description language, it shows more robust behaviour than $NBDM_1$, which increases rapidly with the increase of pixel edition. Since NC and $NBDM_1$ have the same type of normalization, we will focus on comparing these normalizations from now on.

In the second test, we applied both measures to six datasets with distinct natures (9 images each) to understand how $NBDM_1$ and NC behave with different types of images. The six datasets were: artistic images from 2 different datasets [208, 242]; cellular automata images; diabetic retinopathy images [243]; chest computed radiography (CR) images [244], and photographic images [245].

The results are depicted in Figure 4.3. Overall, the NC and $NBDM_1$ show similar behaviour across the majority of the datasets. The exception to this the cellular automata dataset, which exhibit more algorithmic behaviour.

The cellular automata dataset is constituted by images created with small programs with simple rules. Whereas the compressor has difficulty compressing this type of image,

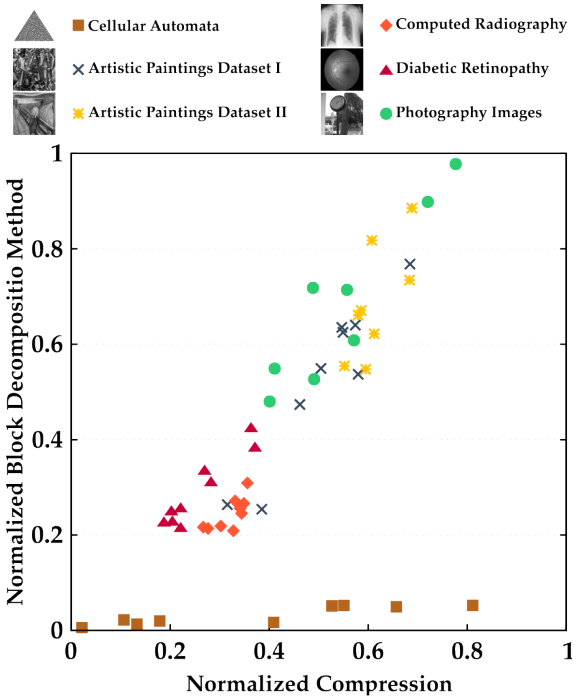


Figure 4.3: Information-based measures evaluation. Values of the NC and $NBDM_1$ for different types of images.

the BDM can point to their algorithmic nature and thus attribute them a very small value. This outcome shows the importance of the BDM in detecting simple algorithmic outputs embedded in data.

For the last test, we selected one of the most complex images identified by the NBDM in the previous test. Then, we used it to evaluate if the BDM could accommodate specific data alterations. This test is depicted in Figure 4.4.

After the binarization process, we performed a super-sample image transformation where each char was amplified to a 4x4 representation. This value was selected since the BDM has the default block size value of 4x4 in 2D structures. After this operation, the BDM was computed for the original and the super-sampled image. While the original image was measured with 370981 bits, the super-sampled image had only 79 bits. This abrupt decrease in complexity indicates that the BDM underestimates the amount of information contained in the object. This underestimation occurs because the BDM analyses object information in blocks instead of looking at the whole object. Specifically, blocks analysed by the BDM have the same size as the super-sample image transformation. As a result, the complexity attributed to each block is approximately zero since each block is composed of all zeros or ones. Hence, the overall value attributed to the complexity of the object will drop dramatically.

This analysis shows that BDM is not prepared to deal with the information associated with the choice of the model, unlike the NC. Instead, the NC relies on a lossless data compressor bounded by a maximum information channel capacity.

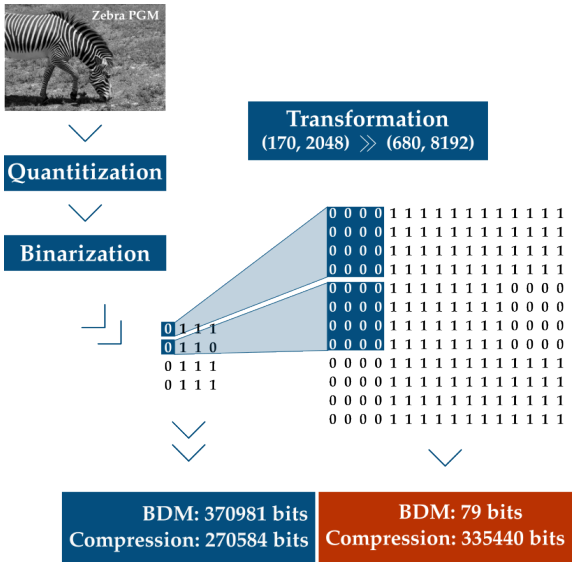


Figure 4.4: Information-based measures evaluation. Image transformation pipeline leading to BDM underestimation of the amount of information contained in the transformed object.

In these three tests, we can notice some advantages and limitations of both measures. However, ranking these measures is unfair because they have different characteristics and natures. Therefore, we will use the NC and NBDM in a combined mode to recover insights and characteristics from the images of the artistic paintings.

4.5.3 Insights

By benchmarking several compressors, we found that the most effective compressor for the artistic dataset under study is PAQ8.

When evaluating the NC and two normalizations of the BDM regarding their robustness when images undergo uniform pixel editing, we found that the NC is more robust than BDM with the same kind of normalization.

When we compared NC and NBDM using different images, we found that the results of the NC and NBDM are similar, except for the cellular automata dataset, which exhibited a more algorithmic behaviour. The cellular automata data was created with small programs with simple rules. While the compressor had difficulty compressing this data, BDM could approximate their algorithmic nature and thus assign them a value close to a minimal complexity value. The ability to identify an algorithmic nature incorporated in the data demonstrates the relevance of BDM as a measure.

Compression makes use of the digital object in its entirety to create the shortest possible representation without loss of information. This is reflected in the NC, a compression-based measure. In contrast, BDM divides the digital object into blocks and, based on the complexity of the blocks, estimates the image complexity in its entirety. Consequently, BDM cannot determine the information shared between the blocks, which causes it to

increase, compared to the NC, with the increase in uniform pixel editing.

On the other hand, the fact that BDM cannot determine the information shared between the blocks causes it to underestimate the amount of information contained in the object when performing a super-sample image transformation. Since the ampliation size was the same as the blocks analysed by BDM, the complexity attributed to each block was approximately zero. Consequently, the overall value attributed to the image complexity decreased dramatically.

4.6 Artist painting analysis

Here, we investigate using information measures to analyse the Painting-91 dataset[208]. In the following subsections, we present the results of applying the complexity approximation measures, combining the NC with the HDC function, measuring local complexity for different authors, and constructing a cladogram.

We also measure the impact of normalizing these images by applying the abovementioned measures to the dataset after normalization. Afterwards, we compare the average variation and percentage difference between the results obtained for each author. These results are shown in Section B.2.1.

4.6.1 Global measures analysis

In this subsection, we measure an approximation to the Kolmogorov complexity for the dataset of artistic paintings. The same pipeline, described in Section 4.4.4, was used, with the difference that the Lloyd-Max algorithm quantization was set to 16, 64, and 256 levels (4, 6, and 8 bits, respectively). It is important to note that the Lloyd-Max algorithm forced normalization of the images for the 16 and 64 levels, while 256 was the original level of the images, and as such, these images were not normalized. This process was performed to evaluate the impact of quantization on the measures used to approximate the Kolmogorov complexity in artistic painting images. From the results obtained, we show unknown characteristics and insights into temporal traits.

In general, the complexity of each painting follows the example of Figure 4.5. Paintings with low complexity are classified as abstract and minimalist, following simple patterns. As the complexity increases, we start to recognize paintings with different local complexities, meaning there are regions with high complexity and detail (generally at the centre/bottom of the paintings) surrounded by low complexity regions (same colour background), known as *chiaroscuro*. This pattern begins fading as the complexity increases, since the highest complexity paintings are also the most irregular, detailed, and convoluted.

Each artist's average complexity is described in Figure 4.6 as the average NBDM_1 and NC, respectively. Each artist has an associated colour, and lines of the same colour illustrate its relative positional deviation in different quantizations. Noticeably, quantization impacts the NBDM_1 more than the NC, since the relative positioning between authors

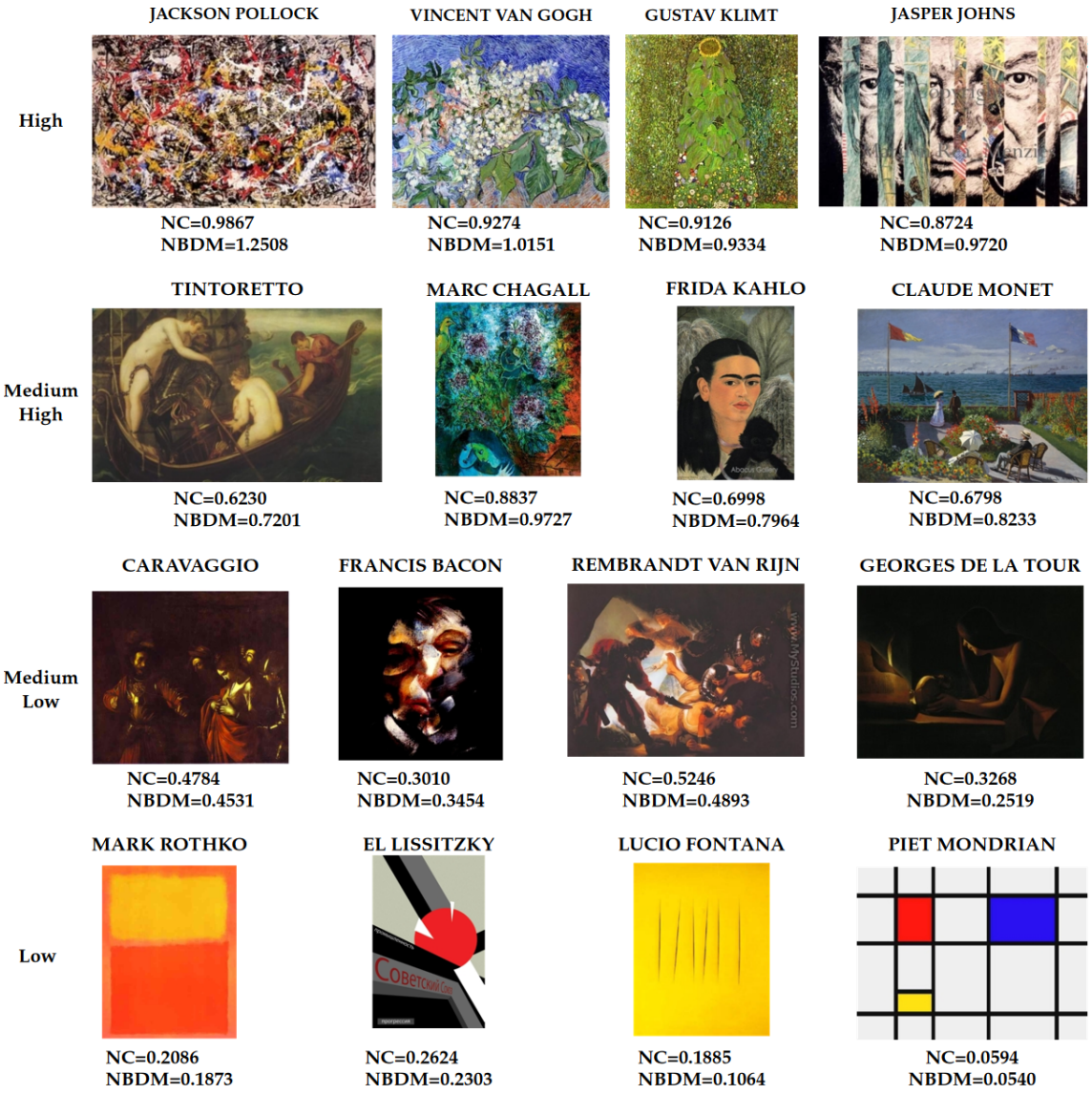


Figure 4.5: Examples of artistic paintings with different levels of complexity (8 bits quantization). The NC and NBDM_1 values are displayed bellow each painting.

varies more in the former. On average, the variation is 13.4 ± 11.37 relative positions of each author in NBDM_1, while in NC, the variation is 4.9 ± 4.3 positions.

Despite the higher variation in the NBDM_1, both measures can detect styles with low and high complexity. Artists such as Mark Rothko, Lucio Fontana, Piet Mondrian, and El Lissitzky can be easily identified on the low side of the complexity spectrum. Minimalism, Abstract Expressionism, and Constructivism movements are associated with these styles. On the other hand, artists also from Abstract Expressionism, such as Willem de Kooning, Jackson Pollock, and Jasper Johns, characterize the highest complexity side of the spectrum, as well as other artists with a more detailed and convoluted style, like Gustav Klimt and Vincent van Gogh.

Abstract Expressionism is characterized by aggressive features combined with random

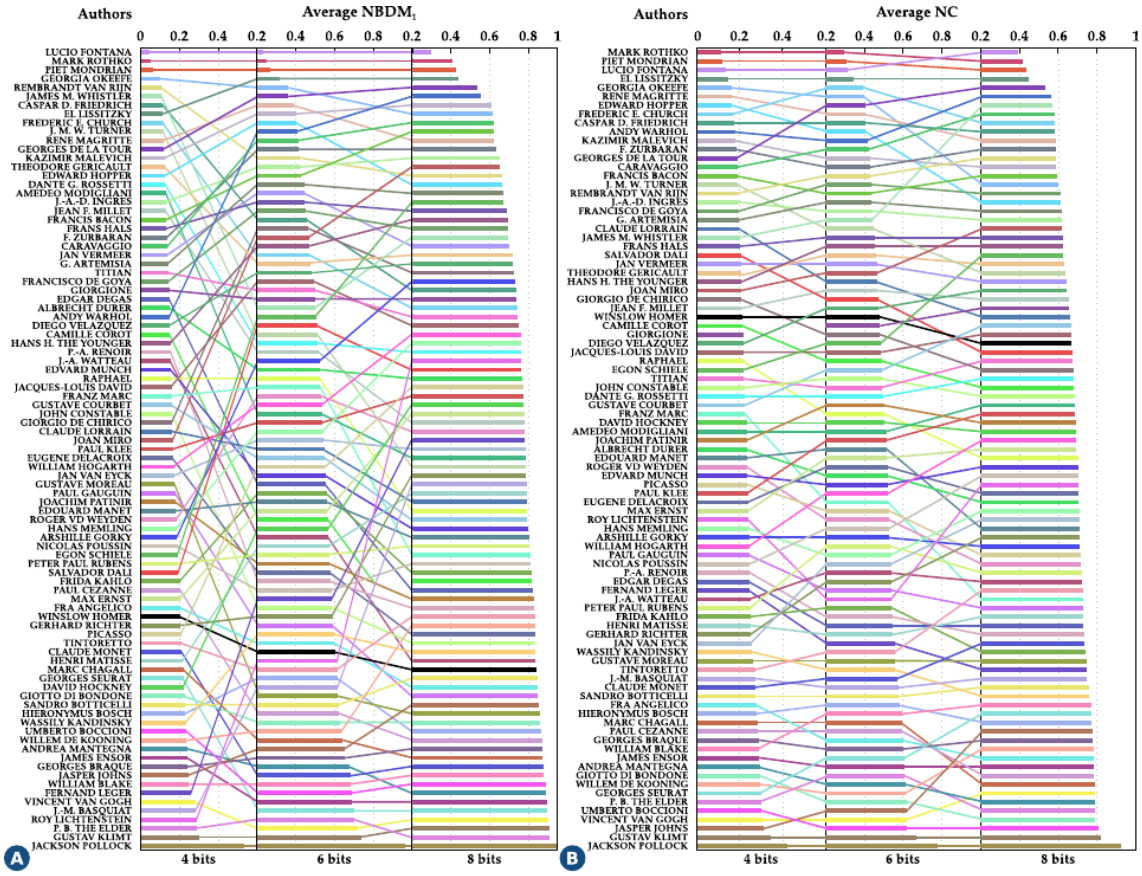


Figure 4.6: Average Normalized Block Decomposition Method using $NBDM_1$ (**A**), and Average Normalized Compression (**B**) for each author where images of paintings where quantized for 4, 6, and 8 bits. The authors are sorted by the value of $NBDM_1$ and NC, respectively. To see this result in more detail, please visit the website described in section B.3.

and geometric features and spontaneity [246]. Abstract Expressionism artists are present at both extremes of the complexity spectrum because this style is divided into two opposites, Action Painting and Colour Field. In Action Painting, the paint was thrown directly on the canvas, through instinctive gestures, where chance and randomness determined the evolution of painting [247]. This style is characteristic of artists like Jackson Pollock (known for the technique of "dripping") and Willem de Kooning. On the other hand, Colour Field is more mystical and meditative. This style of painting has few elements in the frames, indefinite limits, and explores the sensory effects of colour and the subtlety of chromatic relations [248]. A specific example of an artist that followed this trend was Mark Rothko. In all cases, Jackson Pollock had complexity values completely different from other artists, the average complexity of his paintings being approximate to random (normalized value close to 1). Although he denied his paintings were random, similar results were also found in previous work, which defined Jackson Pollock's dripping paintings as not typical artworks [210].

The same results for $NBDM_2$ are presented in Figure 4.7.

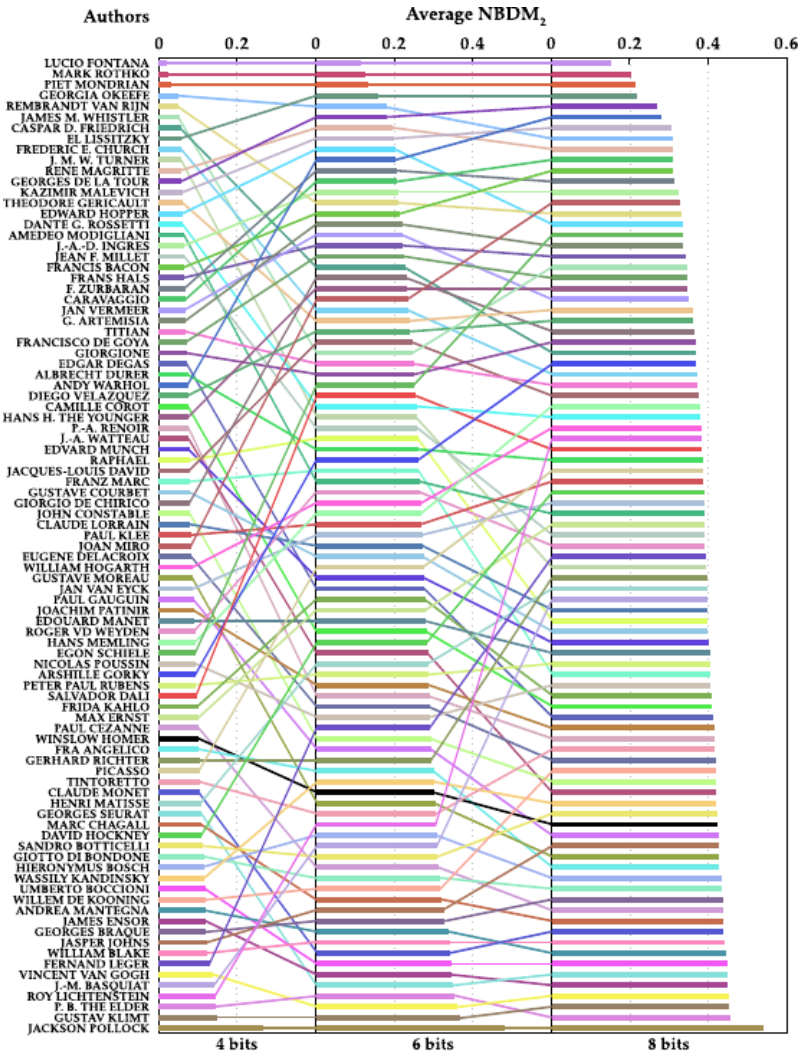


Figure 4.7: Author's average Normalized Block Decomposition Method (ANBDM) using $NBDM_2$ for 4, 6, and 8-bit quantization. The authors are sorted by the value of $NBDM_2$. To see this result in more detail, please visit the website described in section B.3.

This measure, on average, has a relative positional variation of 13.2 ± 13.2, a value slightly lower than in $NBDM_1$, although with a higher average standard deviation. This aspect, combined with the fact that the author positions vary slightly from their position in $NBDM_1$, demonstrates that, overall, normalization has minimal impact on the measure, and thus, does not influence the results obtained with BDM.

4.6.2 Combining the NC with the roughness exponent of HDC function

We used the average NC and the roughness exponent (α) of the two-point height difference correlation (HDC) function, which measures the roughness exponents of brightness surfaces, to assess the ability of these measures to distinguish different styles. Accordingly, we made use of style-labelled paintings available in the dataset. From these labelled im-

ages, we computed their author's average NC and the value of α. The roughness exponent was used as an additional measure since it has proven to be capable of some differentiation between styles [210]. We discarded the usage of BDM due to quantization impacting it more than the NC. Using the average NC and α of each labelled painter, we created a scatter plot (Figure 4.8) and represented each artistic movement as an ellipse, with the centre in the points centre of mass and with a width corresponding to the standard deviation.

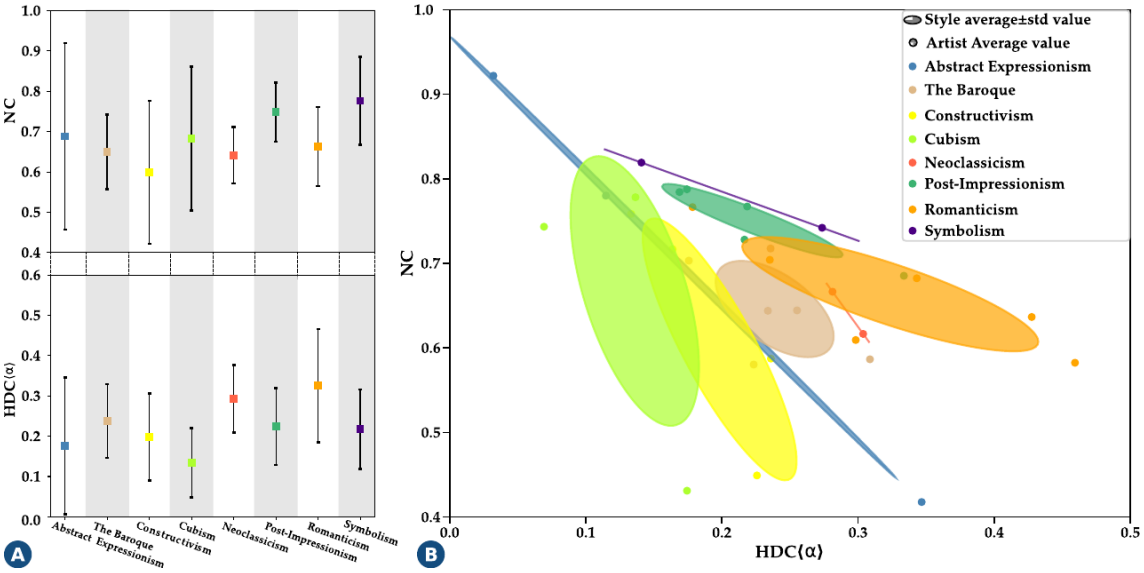


Figure 4.8: Combining the HDC with NC. (**A**) Average and standard deviation for each style in NC and α, respectively. (**B**) Results grouped by styles using average NC and average α of HDC for each artist labeled on the dataset.

As shown in Figure 4.8 (**A**), each measure alone is not capable of efficiently separating styles. However, when combined, the styles are well confined into different regions (except for Abstract Expressionism), showing that together these measures are representative of artistic movements. The roughness exponent α captures the brightness and relative spatial position level and is correlated to variations in painting techniques and genres [210]. The NC adds to the level of brightness and relative spatial position provided by the HDC, the notion of average information present in each artist's painting. This amount of information differs depending on the artistic movement and historical circumstances.

Interestingly, similar to NC, the roughness exponent of the HDC varies greatly in Abstract Expressionism, and in this artistic movement, there is an inverse correlation between the NC and α. Artists like Jackson Pollock and Willem de Kooning (Action Painting) presented a high average NC and a low α, whereas, Mark Rothko (Colour Field) had polar results. This atypical behaviour corroborates the big difference between the two currents of Abstract Expressionism. The Action Painting usage of instinctive gestures and randomness creates high NC values and spatial correlation approaching a random image. In contrast, in Colour Field, we get more minimalist images with high spatial contrast between regions but low complexity.

The capacity of these measures to differentiate styles will be further explored in Section 4.7, where we use these as features for style and author classification.

4.6.3 Local complexity of paintings

After exploring the global impact of NBDM and NC on paintings and styles, we focused on local complexity of the paintings.

Picking the best block size

When creating the fingerprints, we tried to select a patch size that is the minimum for the differences in the compression rate to be significant and capable of being used as a measure between paintings. To this end, we computed the regional complexity of each image for the 8x8, 16x16, and 32x32 blocks of the image. It is worth mentioning that for the regional complexity of 8x8 and 16x16, all images were divided into the same number of tiles, resulting in tiles with slightly different sizes. On the other hand, for the 32x32 regional complexities, the images were divided into blocks of the same size, except for the last tiles, which were the remainder of each image since the image size was not an exact multiple of the desired tile size. There are two reasons for this difference in the method of dividing blocks. Firstly, when we divide the image into a few blocks, each block has a reasonable size and therefore, the variation with the addition of a row or column does not affect the results of the compression. With small tiles, adding a column or row affects the compression results of each tile more significantly. Secondly, cropping tiles with the same size in larger blocks would cause a large remainder, which would be more inaccurate than distributing the remainder among the other blocks. The results of each author's fingerprints (computed for the 8x8, 16x16, and 32x32 blocks) are shown in the website (Section B.3). Furthermore, some illustrative results are exemplified in Figure 4.9.

Qualitatively, as can be seen in Figure 4.9, the images of the authors show similar patterns for each author. However, with the increase in patch size, from 16x16 blocks to 32x32 blocks, the patterns become less noticeable, as such a good balance between detail and differentiation would be the 16x16 blocks. Quantitatively, we computed the distance matrix for each author using images of different patch sizes. On those distances, we performed the Mantel test and measured the average difference between them. The results are shown in Table 4.1.

Since the matrices correspond to distances computed for the same measure but with a different number of blocks, the distance between authors is expected to be similar and yield a high correlation between distance matrices. The results show a high correlation between the distances computed from 8x8 and 32x32 fingerprints towards the 16x16 distance. However, there is a higher correlation (0.951) and lower average variation between 8x8 and 16x16 fingerprints distance than between 16x16 and 32x32 fingerprints distance (0.918). These results reveal that the increase in the number of blocks from 16x16 to 32x32 decreases consistency between the same author's fingerprints as the correlation between

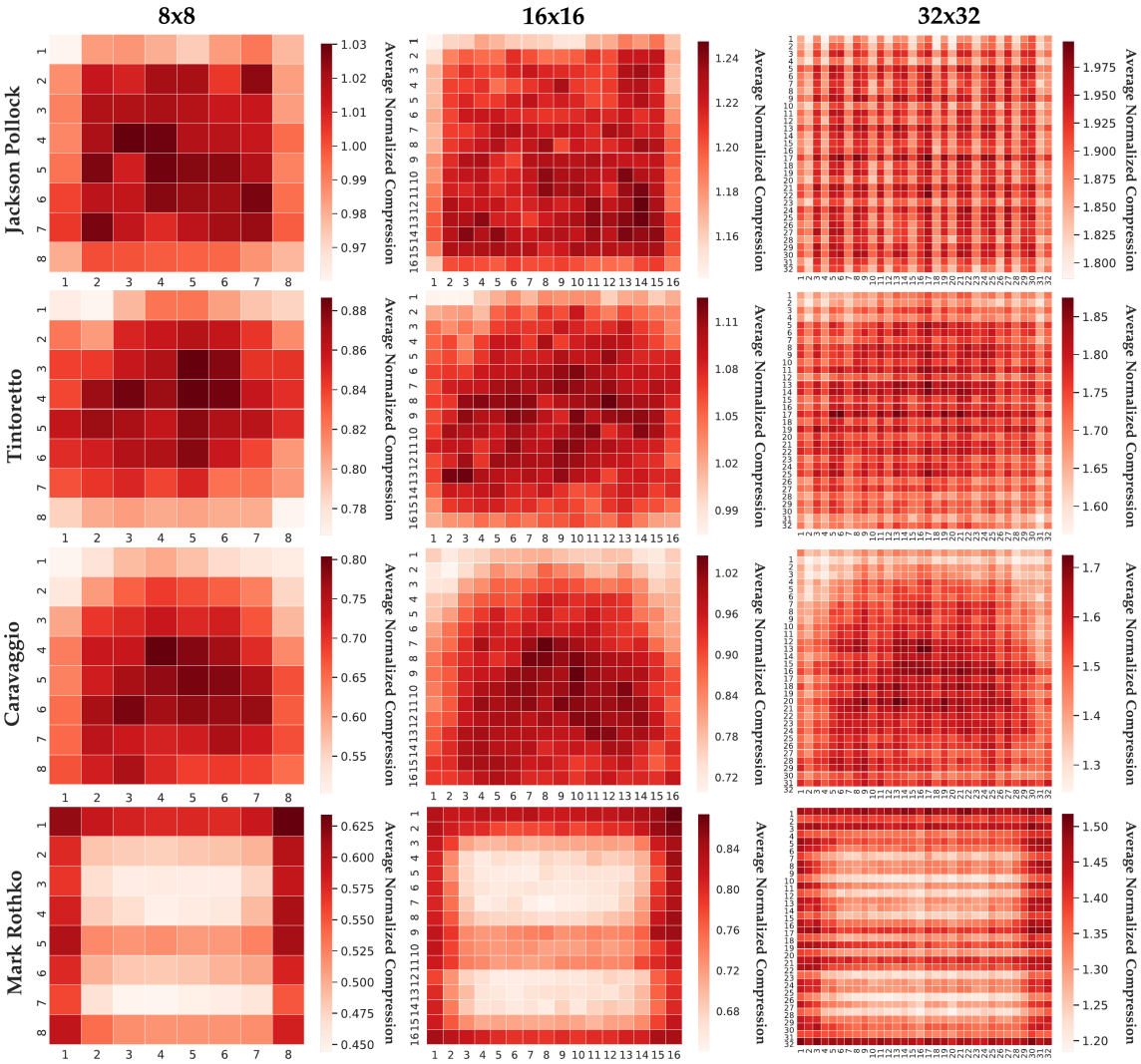


Figure 4.9: Heat maps of the local complexity matrix (fingerprint) of some authors for the different number of blocks the images were divided. This fingerprint shows the author's range of complexity and where they paint with more detail. To see all fingerprints, please visit the website described in section B.3.

distances decreases significantly. Thus, we can conclude that the 16x16 fingerprints can give a more optimized balance between detail and differentiation since it retains correlation to the 8x8 fingerprints and provides a more detailed map of the author's complexity range.

Analysing local complexity of paintings

To analyse the paintings' local complexity, we divided the images into identical quadrilateral sizes and measured the algorithmic information for each (16x16 blocks). Then, we computed the average of each quadrilateral for all the paintings for each painter. The results are shown in Figure 4.10, illustrating the same authors as those in Figure 4.5.

Table 4.1: Mantel Test between distance matrices and average difference between them. For the Mantel test, all results had a p-value of 0.001.

Comparison methods	**Mantel Test**	**Average ± Standard Deviation**
16x16 blocks*	0.955	3.262 ± 2.820
8x8 *vs* 16x16 blocks	0.951	14.959 ± 12.372
32x32 *vs* 16x16 blocks	0.918	74.092 ± 50.952

* Normalized *vs* non-normalized images.

However, the matrices of Figure 4.10 were computed using all the authors' paintings in the dataset. The complete results are available on the website described in section B.3.

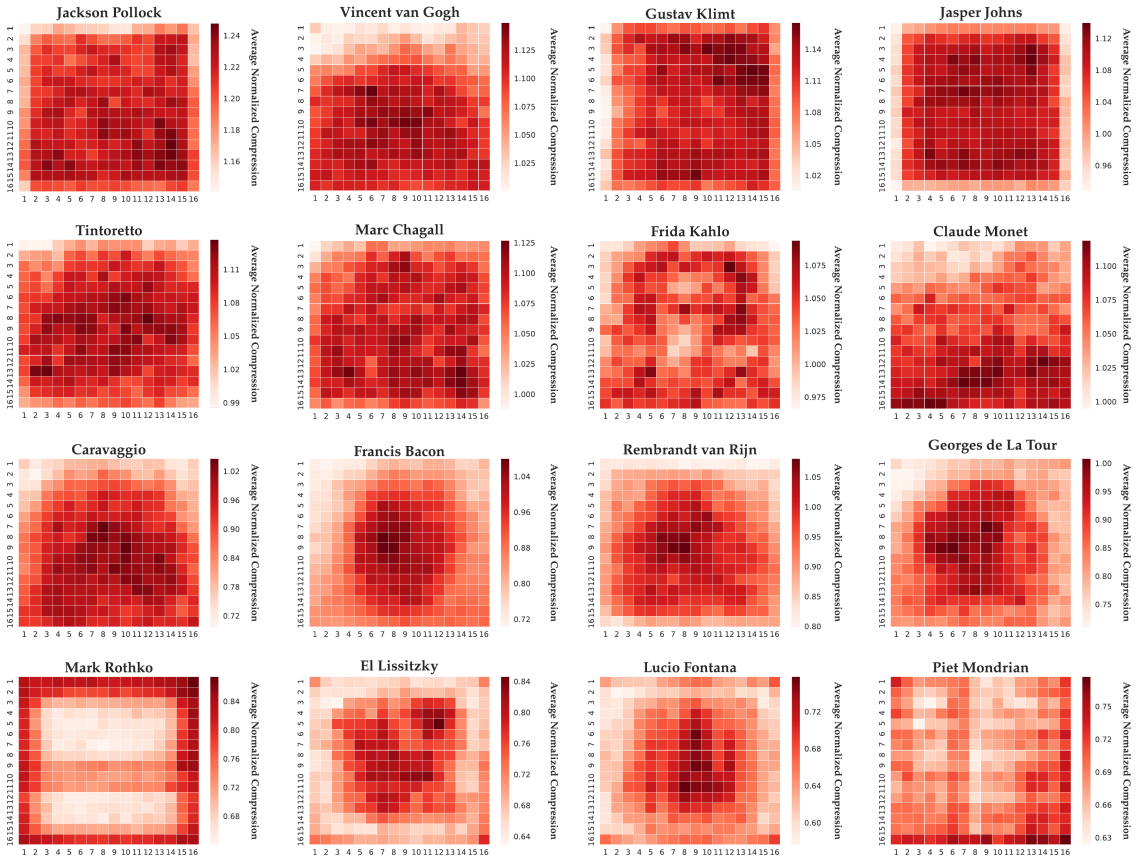


Figure 4.10: Heat maps of the local complexity matrix (fingerprint) of some authors, computed with the NC average. This fingerprint shows the author's range of complexity and the locations in the canvas painted with more detail (or complexity). To see all matrices, please visit the website described in section B.3.

All artists have a unique complexity matrix (fingerprint). This fingerprint shows, on average, where artists paint with more detail and emphasis, as well as their average range of complexity. For instance, Jackson Pollock and Jasper Johns show high complexity values dispersed over the canvas. At the same time, artists like Francis Bacon and George

de la Tour focus more on the centre of the canvas, and Mark Rothko and Piet Mondrian have their highest complexities around the paintings' borders.

Since the 16x16 fingerprints conveyed the best results regarding detail and differentiation, the cladograms were constructed utilizing the distance computed from the fingerprints with this block size. Two cladograms were constructed to portray the relations between different artists. One cladogram was constructed using the UPGMA algorithm, which is illustrated in Figure 4.11, and another was built using the Kruskal minimum spanning tree algorithm [232], which is depicted in the Figure B.1 in Section B.2.2.

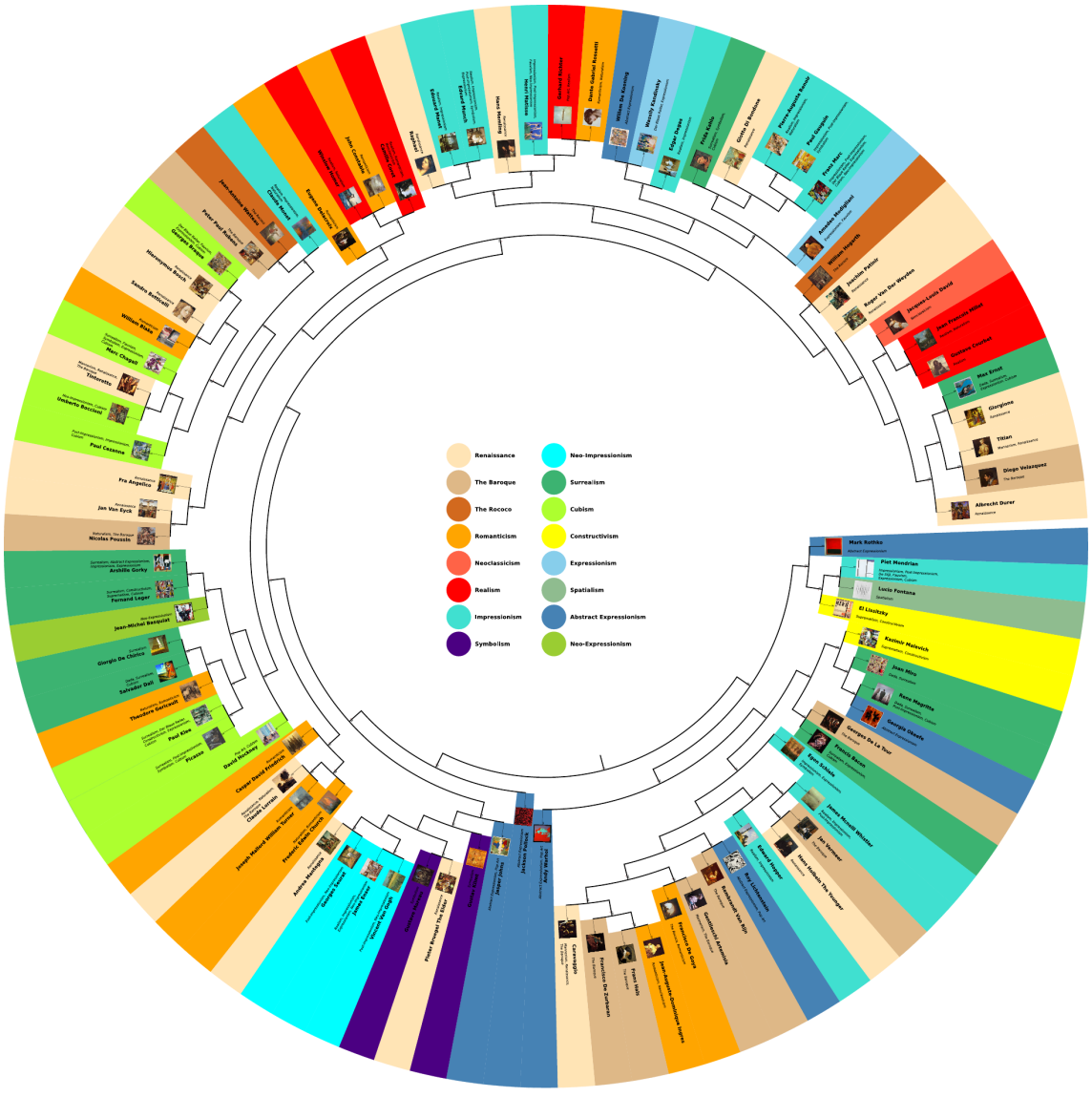


Figure 4.11: Artists' cladogram computed resorting to the UPGMA algorithm. Each artist has a sample painting and a colour associated with one of his styles (the colour was chosen based on nearest leaves) assigned to him, as well as a description of some styles usually associated with the author. To obtain an improved view of the cladogram, please visit the website described in section B.3.

The cladogram shows the fingerprint's ability to group artists from the same artistic movements. Broad groupings of artists from styles are present in the cladogram: Renaissance, Baroque, Romanticists, Impressionists, Surrealism, Cubism, and Abstract Expressionism. Also, the cladogram shows smaller groupings of sister leaf-nodes with the same style. The cladogram also depicts relationships of influence between authors of different artistic movements. For example, this relation is seen in the case of Titian, who influenced Diego Velazquez; Caravaggio, who influenced Francisco de Zurbarán; Frida Kahlo, who influenced Amedeo Modigliani; Sandro Botticelli who influenced William Blake; Claude Lorrain who influenced Joseph Mallord William Turner; and Peter Paul Rubens who influenced Jean-Antoine Watteau.

On the other hand, some authors seem unrelated in style and influence, for instance, Francis Bacon and Georges de la Tour, George Braque and Hieronymus Bosch, Peter Paul Rubens and Frida Kahlo, Max Ernst and Giorgione, and Rembrandt van Rijn and Roy Lichtenstein. There may be many reasons for this, for instance, the number of regions the images were divided in can be sub-optimal for some images of artistic paintings, decreasing the sensitivity of the measure and jeopardizing the cladogram's construction. On the other hand, the algorithm used to measure the similarity between matrices or the algorithm used to construct the cladogram (UPGMA) may not be the most appropriate for all cases. However, we have tested the Kruskal minimum spanning tree algorithm, which generated similar results (Section B.2.2). Additionally, these seemingly unrelated connections could reveal undiscovered elements and relationships. For instance, one of Roy Lichtenstein's early artistic idols was Rembrandt van Rijn. Moreover, if artists are not related regarding the artistic movement or influence, the vicinity between them could be representing another property. This aspect is not necessarily related to the period or movement the artists were inserted in, but rather, the way they projected their compositions, ideas, and impressions onto the canvas. Complexity can be approximated by the total number of properties transmitted by an object and detected by an observer. By dividing images into blocks of equal size and evaluating their local complexity, we are quantifying the local information being transmitted. On the other hand, by averaging the canvas results per artist, we obtain a matrix that describes how they present information to the observer. This information intertwines various notions critical to how the work is perceived, such as composition, which describes where the artist places the subject and how the background elements support it, as well as the unity, balance, movement, rhythm, focus, contrast, pattern, and proportion of the painting. For instance, the proximity between Hans Holbein and Vermeer could be due to both of them having used optics to achieve precise positioning in their compositions, namely by performing a combination of curved mirrors, *camera obscura*, and *camera lucida* [249].

The fingerprints can also convey the notion of space by depicting where the positive (subject itself, usually more detailed) and the negative (the area of painting around it) areas are on the canvas. Artists can play with a balance between these two spaces to further influence how viewers interpret their work. Therefore, the similarity between different

artists concerning the regional (local) complexity can reflect the similarity in thought regarding their approaches to painting. For instance, the proximity between Francis Bacon and Georges de la Tour could be due to the former being heavily influenced by the Baroque style and making dramatic use of contrasts of light and shadow. These methods are characteristic of the *chiaroscuro* principle and its radicalization in the Tenebrista school (signature style of Georges de la Tour) [250, 251]. The intense contrasts of light and shadow highlight the characters, and although exaggerated, lighting increases the feeling of realism, making the muscles and facial expressions more evident. Simultaneously, the presence of large blackened areas highlights the chromatic reach and the illuminated space, which acquire their value as elements of the composition.

4.6.4 Insights

We found that paintings with low complexity are abstract, minimalist, and follow simple patterns. Paintings with a slightly higher average complexity possess different regional complexities, specifically, a region with high complexity and detail surrounded by a background of low complexity. This noticeable pattern begins to fade with more complexity, and the most complex paintings are globally irregular, detailed, and convoluted.

Regarding the average complexity values for each artist, we found that NC and NBDM behave similarly, but quantization has a bigger impact on NBDM. We also establish that the low side of the complexity spectrum was characterized by Abstract Expressionism, Minimalism, Constructivism movements, with authors such as Mark Rothko, Lucio Fontana, Piet Mondrian, and El Lissitzky. Also, artists from Abstract Expressionism characterized the high complexity side of the spectrum, such as Willem de Kooning, Jackson Pollock, and Jasper Johns, as well as other artists with a more detailed and convoluted style, like Gustav Klimt and Vincent van Gogh. Due to two different currents (Colour Field with authors with low average complexity and Action Painting with authors with high complexity), Abstract Expressionism was present at the polar ends of the spectrum. In all cases, Jackson Pollock had average complexity values that were completely different from other artists, the average complexity of his paintings being close to random. Although he denied being a creator of random paintings, this result and others [210] seem to indicate that Jackson Pollock's dripping paintings are not typical artworks, possibly related to the artist including many symbolic layers and dispersion intentions over the canvas.

When evaluating the artists' average NC and the roughness exponent (α) of the HDC function in the labelled images of the dataset, we found that styles are well confined into different regions, showing that the combination of these measures gives a robust representation of artistic movements. Specifically, the NC adds to the brightness and relative spatial position shown by the roughness exponent, the notion of average information in each artist's painting, consistent within the same style and historical circumstances. We also detected that in Abstract Expressionism, the NC is inversely correlated to α. Artists related to Colour Field painting presented a high α and low NC, whereas artists related

to Action Painting presented the exact polar results (low α and high NC).

We divided the image into equal quadrilateral parts and estimated the local complexity of each painting on the dataset, using it to ascertain each artist's average regional matrix (fingerprint). Data complexity measures the total number of properties transmitted by a digital object and detected by an observer (plus the language used). By dividing images into blocks of equal size and evaluating its local complexity, we quantified the local information being transmitted. Furthermore, by averaging the canvas results per artist, we obtain a unique fingerprint that describes how the author exposes information to the observer. Among other things, these fingerprints give specific insights into each artist's way of painting, showing where, on average, they paint with more detail and give more emphasis, while also providing insights into each artist's range of complexity. Using these matrices, we computed a distance matrix and utilized it to construct a cladogram. We discovered that these cladograms aggregated authors of the same style close to each other and artists' influencing relationships, like Francis Bacon and Georges de la Tour, and George Braque and Hieronymus Bosch. Furthermore, we observed proximity between artists due to shared methods and techniques which are not correlated with the time or artistic movement. For example, the proximity between Hans Holbein and Vermeer, who do not share styles but who both used optics to achieve precise positioning in their compositions. This evidence shows that artists' fingerprints contain critical information about how the work is perceived, such as composition, unity, balance, movement, rhythm, focus, contrast, pattern, and proportion of the painting and space.

4.7 Artist painting classification

To evaluate the use of these measures quantitatively for classification purposes, we assessed their impact when used as additional features to improve state-of-the-art classification methods.

4.7.1 Evaluation of measures for classification purposes

To perform quantitative evaluation, we recreated a recently published state-of-the-art (SOTA) method as a baseline and improved the results by combining our proposed measures. Based on current methods [225, 226], we extracted a Gram representation using the first convolutional layer from the fifth convolutional block of the VGG16 network, which was pre-trained with the ImageNet dataset (no significant result difference was found between the use of VGG16 or VGG19). Then, principal component analysis (PCA) was applied to the Gram matrix to reduce the dimensionality, and finally, this vector was provided to an SVM to perform classification.

Afterwards, the features obtained from computing the HDC and the regional complexity were used for author and style classification using the XGBoost classifier [170] and combined with the SOTA classifier via a Voting Classifier ensemble. The results of the

SOTA and ensemble classifiers, applied to the Paintings-91 dataset in the author and style classification task using the labels provided in the dataset, are shown in Table 4.2.

Table 4.2: Accuracy results obtained for the test set in style and author classification task using state-of-the-art (SOTA), SOTA with regional complexity (RC) feature and ensemble with our measures (RC and HDC).

Task	Classes	Images	SOTA	SOTA+RC	SOTA+RC+HDC+RC
Style	13	2338	0.622	0.644	0.650
Author	91	4266	0.480	0.490	0.500

The results show that including the regional complexity increased the accuracy of the results by 2.2 p.p. and 1.0 p.p in the style and author classification tasks, respectively. Moreover, the overall inclusion of the proposed measures (HDC + RC) increased the accuracy in both classification tasks by 2.8 p.p. and 2.0 p.p. in the style and author classification tasks, respectively. These results indicate that these predictors are useful auxiliary features capable of improving current methodologies in classifying artistic paintings. These results are congruent with those obtained by Nanny *et al.* [252], since handcrafted features and non-handcrafted features seem to extract different information from the input images, and as a result, the fusion of the two types of features improves the results obtained when using non-handcrafted features only. Furthermore, regional complexity (RC) has a higher impact on improved accuracy than the HDC features, demonstrating the importance and distinction of regional complexity as a feature.

4.7.2 Insights

We show that regional complexity and HDC extract information that differs from non-handcrafted features and improve current methodologies in classifying artistic paintings.

Regional complexity provided the most significant increase in accuracy in the classification tasks, showing its relevance as a descriptor of images of artistic paintings.

4.8 Summary

In this chapter, we introduce novel solutions to the field of computer analysis of artistic paintings and the problem of artist classification and authentication. Specifically, we assessed the viability of unsupervised measures that approximate the quantity of probabilistic and algorithmic information to perform these tasks.

Our direct comparison between NC and BDM allowed us to understand the strengths and weaknesses of both measures. Although BDM has difficulty dealing with uniform pixel edition and information quantification given the block representability, it serves as a useful tool to measure and identify data content similar to simple algorithms. On the

other hand, the NC is more robust to data alterations (pixel edition and quantization) and can measure the quantity of information without underestimation.

Regarding the application of information-based measures in artistic paintings, we studied and developed techniques that can be valuable for art authorship attribution and validation, art style categorization and organization, and art content explanation. Namely, the NC proved to be a robust measure that gives us some insight into the complexity of different styles, showing hidden patterns and relationships in artistic paintings that share the same range in complexity. Furthermore, it could be a stylistic descriptor when coupled with the roughness exponent α. On the other hand, fingerprints depict how each author typically spreads content on canvas. Thus, they can provide a suitable means of art content explanation and be valuable for art authorship attribution and validation. Moreover, since complexity approximations provide insights into artists' way of painting, they can be used as a means of relating authors, therefore being useful in depicting artists' stylistic influences, and shared techniques. Additionally, using the distance between the artists' regional complexity, we find some interesting links between artists regarding the use of space, technique, composition, rhythm, and proportion.

Finally, we demonstrated that the regional complexity and the HDC function of the paintings could serve as useful auxiliary features capable of improving current methodologies in author and style classification of images of artistic paintings.

Chapter 5

Complexity analysis of Turing Machines

[e] Oak Tree - Jorge Miguel Silva.

"Come up to me on the mountain and stay here,
and I will give you the tablets of stone
with the law and commandments
I have written for their instruction."
– Moses, Bible: Exodus 24:12

5.1 Contextualization

In the previous chapters, we analysed 1d and 2d data using Kolmogorov complexity approximations, specifically data compressors and the BDM. We found that these approximations help describe, understand, and classify these data types. In this chapter, we take a closer look at data generated from algorithmic sources, specifically Turing Machines (TMs). Sources that generate symbolic sequences with the same *algorithmic complexity* may differ in probabilistic complexity. For example, in the case of Turing Machines, machines with the same *algorithmic complexity* can create tapes with different statistical complexity.

Specifically, in this chapter, we will use a compression-based approach to measure the global and local probabilistic complexity of specific TM tapes. Both measures are estimated using the best-order Markov model. For the global measure, we use the NC, while for the local measures, we define and use normal and dynamic complexity profiles to quantify and localize lower and higher regions of probabilistic complexity.

We assess the validity of our methodology on synthetic and natural genomic data, showing that it is tolerant to increasing rates of editions and block permutations. Regarding analysis of the tapes, we localize patterns of higher probabilistic complexity in two regions for a different number of machine states. Furthermore, we show that these patterns are generated by a decrease in the tape's length, caused by small rule cycles. Additionally, we use BDM to analyse the TM tapes.

Finally, we provide a simple algorithm to increase the probabilistic complexity of a TM tape while retaining the same *algorithmic complexity*. The structure of this chapter is as follows:

- Research questions and contributions;
- Turing Machines;
- Methods;
- Viability assessment of the NC;
- Global analysis of TM tapes;
- Analysis of probabilistically complex TM tapes;
- NC and BDM comparison;
- Increasing the probabilistic complexity of TM tapes;
- Summary.

5.2 Research questions and contributions

In this chapter, we study data generated from algorithmic sources and assess their probabilistic patterns. The sources have identical conditions and are evaluated using global and local measures. When performing this analysis, several questions arise: Is there a correlation between probabilistic and *algorithmic complexity*? Is there a way to quantify and localize higher and lower regions of probabilistic complexity? Can we systematically increase probabilistic complexity while retaining the same *algorithmic complexity*? This chapter revolves around these challenging questions. To try and answer them, we fix the source exclusively to an algorithmic nature, combining different rule configurations while keeping the number of states and cardinality fixed. This configuration is used to measure and analyse the probabilistic complexity of the TM tapes using compression-based approaches approximated by the best-order Markov model. First, we assess this compression-based approach according to different levels of symbol substitutions and permutations of contiguous blocks of symbols. After definition and assessment, we identify some patterns in these TMs. Then, we define and introduce the normal and dynamic complexity profiles as local measures to quantify and localize higher and lower regions of probabilistic complexity. Next, we compare the BDM with our compression-based approach. Finally, we describe a simple algorithm to increase the probabilistic complexity of the tape while maintaining the same amount of *algorithmic complexity*.

Our contributions are as follows:

- We showed that in two regions Turing Machines have on average higher NC (and lower tape length).
- We localized these regions and identified the cause as short cycles in the rules that output tapes with smaller length through the complexity profiles.
- We analyse statistically complex TM tapes.
- We compare BDM with compression-based measures.
- We create a simple algorithm to increase the probabilistic complexity of the tape while maintaining the same amount of *algorithmic complexity*.

5.3 Turing Machines

Turing's proposal on automatic machines (Turing Machines) simplified the concept of decision machines by defining an abstract machine that handles symbols on a strip of tape according to a table of rules. Despite the simplicity of the mathematical model, a TM (TM) is capable of simulating the logic behind any computer algorithm provided to it [253]. Specifically, a TM is composed of a rule table, a head, and a state, and operates on a memory tape divided into discrete cells. The machine starts with an initial state and

its head positioned at a given point of the tape. In this position, it scans a single symbol of the tape and based on the scanned symbol, initial state, and instruction table, the TM writes a symbol on the tape and changes its position. This process repeats continuously until the machine enters its final state, causing a halt in the computation [254].

Formally, a TM can be defined as a 7-tuple $T = \langle Q, \Gamma, b, \theta, \delta, q_0, F \rangle$, where: Q is a finite, non-empty set of *states*; Γ is a finite, non-empty set of *tape alphabet symbols*; $b \in \Gamma$ is the *blank symbol*; $\theta \subseteq \Gamma \setminus \{b\}$ is the set of *input symbols*; $q_0 \in Q$ is the *initial state*; and $F \subseteq Q$ is the set of *final states* or *accepting states*. Finally, the *transition function,* δ is characterized by $\delta : (Q \setminus F) \times \Gamma \not\to Q \times \Gamma \times \{L, R, N\}$, where L is left shift, R is right shift, and N is no shift. The initial tape contents are said to be *accepted* by T if it eventually enters a *final state* of F and halts. Despite this definition, there are many variants of these machines. For instance, there are models which allow symbol erasure or no writing, models that do not allow motion [255], and others which discard the stop criteria or final state, and consequently, do not halt.

With this model, a mathematical description of a simple device capable of arbitrary computations was developed, capable of proving properties of computation in general, and in particular of proving the uncomputability of the halting problem [5].

5.4 Methods

5.4.1 Turing Machines configuration

In this chapter, we analyse the output of simple TMs. All TMs have a binary or ternary set of symbols $\theta_2 = \{0, 1\}$ *or* $\theta_3 = \{0, 1, 2\}$, with a matching set of tape symbols $\Gamma = \{0, 1\}$ *or* $\Gamma = \{0, 1, 2\}$. These TMs have the following specific conditions:

- The machines start with a *blank* tape (all set to zero, $b =$"0");
- The machines do not have an internal condition to halt ($F = \emptyset$);
- The halting is performed by an external condition representing a certain number of iterations that is set the same in every TM;
- The TMs are restricted to read only one tape character at a time and perform three types of movement on the tape, namely move one cell to the left, move one cell to the right, or stay in the same position.

Table 5.1 shows a matrix rule M example for a TM with $\#Q = 2$ and $\#\theta = 2$. Given a scanned symbol and the internal state of the TM, there is a triple that provides the TM information regarding the next symbol to write on the tape from the set θ_2, the next movement of the head relative to the tape (left = 0, stay = 1, right = 2) and the next internal state from the set $Q = \{0, 1\}$. This matrix table is used to fill the tape across a given number of iterations. Each TM starts with the initial state of "0", and the tape with the initial symbol "0". With these initial conditions, the TM reads the tape's symbol and,

given the value read and the rule matrix, changes the tape symbol at the current position, its internal state, and the position of the head on the tape.

Table 5.1: Rule matrix for a TM with $\#Q = 2$ and $\#\theta = 2$.

Symbol		**0**	**1**	
State	**0**	(0, 1, 1)	(0, 2, 0)	(write, move, state)
	1	(1, 0, 1)	(1, 2, 0)	

5.4.2 Search approaches

Since we want to study the probabilistic complexity of TM tapes for different configurations, we created many TMs, as specified in Section 5.4.1 with a small cardinality of states and alphabet $(\#Q, \#\theta)$. These TMs were then analysed as a whole, followed by a more detailed analysis targeting TMs with specific configurations. For $\#Q \times \#\theta \leq 6$, we performed a sequential search through all possible TMs. Since the total number of Turing Machines (TNTM) increases in a super-exponential way $((3 \times \#\theta \times \#Q)^{\#\theta \times \#Q}$, as shown in Figure 5.1), it becomes computationally intractable to run all the TMs for larger values of $(\#Q, \#\theta)$. Therefore, to approximate the results of a complete traversal of the TMs domain, we applied a Monte Carlo algorithm [256] to select TMs from their total pool for $\#Q \in \{4, \ldots, 10\}$ and $\#\theta = 2$, since this algorithm is widely used to obtain qualitative information regarding the behaviour of large systems [257].

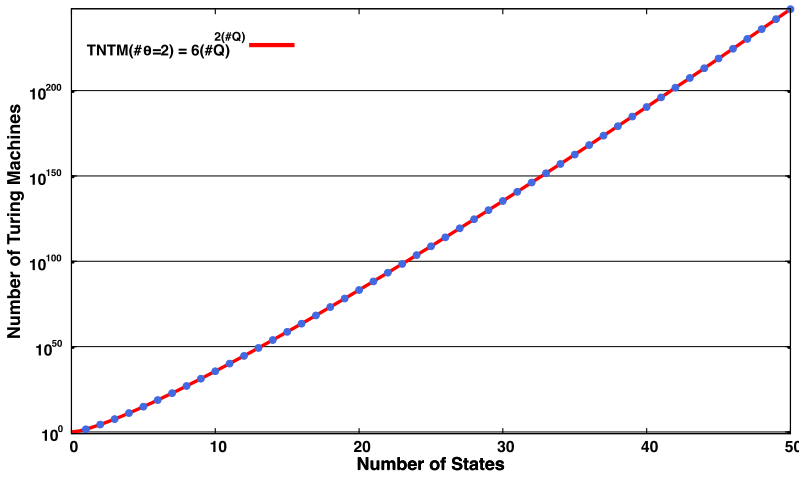


Figure 5.1: Super-exponential growth in TNTM, with the increase in number of states (#Q), and a fixed alphabet cardinality ($\#\theta = 2$).

To perform a sequential search through all possible TMs of a pair $(\#Q, \#\theta)$, we consider that each TM is represented by its rule matrix M and define a *total order relation* between them. Furthermore, we consider this order to start with an initial TM (tm_0), which has its rule matrix filled with the same tuple $(0, 0, 0)$. The definition of next$(M) \rightarrow M$ was based on a succession of increments that would carry to the next significant attribute on

overflow. When the last possible M is reached, M returns to its configuration of tm_0. As an example, for $\#Q = \#\theta = 2$, all four elements of the first TM rule matrix would start at $(0, 0, 0)$, and the last TM would have all elements defined as $(1, 2, 1)$.

To provide a unique numerical identifier (id) for each TM within the set of TNTM for a pair $(\#Q, \#\theta)$, we mapped $M \rightarrow id$ (Algorithm 1). The inverse function $id \rightarrow M$ is well defined and can be inferred from the former algorithm.

Algorithm 1: Mapping a TM rule matrix M to a unique identifier id, in order to traverse through all TMs.

$\#R = \#Q \times \#\theta \times 3$;
$id = 0$;
for each c in M **do**
  $id = id \times \#R + c.write + (c.move + c.state \times 3) \times \#\theta$;
end
return id;

In contrast, in the Monte Carlo approach, rather than starting in one rule matrix and ending in another, TMs are randomly configured by sampling each cell's components (write, move, state) from uniform distributions.

In both search cases, we took advantage of multiple processing cores of the same machine by subdividing the entire group of TMs into different jobs. For the sequential search, the domain was evenly split into as many partitions as the number of jobs in parallel. However, this separation was not required in the Monte Carlo algorithm. At the end of each job, the extracted measures were collected into a single measure set.

5.4.3 Probabilistic complexity

We used data compression to measure the probabilistic complexity of the output tape created by the TM after n iterations. The NC was used as the compression measure (Equation (2.12)). $C(x)$ is computed as

$$C(x) = -\sum_{i=1}^{|x|} \log_2 P(x_i|k), \tag{5.1}$$

where P is the probability of each string symbol x_i occurring given a context k.

Probability P is computed using a Markov model, which is a finite-context model that predicts the following outcomes given a past context k [144]. Specifically, a Markov model loads the input using a given context k and updates its internal model. This internal model is used to compute the probability of any character being read at a given point.

The Markov model operates in two distinct stages: estimation and update. During the estimation stage, the model leverages the given context k to predict the probability of each character in the input sequence. The update stage consists of refining the internal model based on the observed input, thus improving the prediction accuracy.

In the estimation stage, the Markov model computes the probability of a character x_i occurring, given the context k, as follows:

$$P(x_i|k) = \frac{N(x_i, k) + \alpha}{N(k) + \alpha|\theta|}, \tag{5.2}$$

where $N(x_i, k)$ is the number of times the character x_i appears after the context k, $N(k)$ is the total number of occurrences of the context k in the input sequence, α is Laplace smoothing and $|\theta|$ is the size of the alphabet.

During the update stage, the model refines the internal representation by incrementing the corresponding counters for $N(x_i, k)$ and $N(k)$ each time a new character x_i is observed after the context k. This process allows the model to learn and adapt to the input data, enhancing the prediction accuracy.

In the case of Algorithm 2, the tape produced by the TM is provided to the Markov model with context k. Then, this model is used to determine the NC by computing the normalized summation of the probability of each character occurring on the tape.

Algorithm 2: Determine the NC of a generated TM tape.

```
Input  : tape
Output: nc
for ( i = 0; i < tape.size(); ++i ) {
    element_value = markov_table.get_element_occurrence_for_context(tape[i]);
    sum_context_occurrences = summation(markov_table.get_context_occurrences(tape[i]));
    markov_table.update_model(tape[i])+=1;
    value += log_2(element_value/sum_context_occurrences);
}
tape_length = tape.get_length();
normalizer = tape_length × log_2(alphabet_cardinality);
return nc = value / normalizer;
```

5.4.4 Normal and dynamic complexity profiles

Some probabilistically complex tapes were analysed individually by studying how the NC behaved with the increase in the number of characters currently written on the tape (length of the tape). This study was carried out in two forms: a normal complexity profile and a dynamic complexity profile of the tapes. The normal complexity profile is computed after the TM is halted by an external condition representing a certain number of iterations, while the dynamic complexity profile is computed during the TM execution. It is also worth mentioning that the normal complexity profile was applied to the sequence of rules used by the TMs.

A normal complexity profile can be seen as a numerical sequence $\overrightarrow{N}(x_i)$ containing values that express the predictability of each element from x given a compression function $C(x)$. Assuming x_a^b is a subsequence of x from position a to b, we define a complexity profile as

$$\overrightarrow{N}(x_i) = C(x_i|x_1^{i-1}). \tag{5.3}$$

Notice that $C(x)$ has a causal effect, which means it is assumed that, for $\overrightarrow{N}(x_i)$, we

have to access the elements previously $\overrightarrow{N}(x_1), \overrightarrow{N}(x_{...}), \overrightarrow{N}(x_{i-1})$ by order. The profile was normalized to compare tapes with different alphabets according to

$$\overrightarrow{C}(x_i) = \frac{\overrightarrow{N}(x_i)}{\log_2 |\theta|}, \tag{5.4}$$

where $\log_2 |\theta|$ is the normalization factor by the cardinality of the alphabet.

The number of bits needed to describe x can be computed as the sum of the number of bits of each x_i, namely,

$$C(x) = \sum_{i=1}^{|x|} \overrightarrow{N}(x_i), \tag{5.5}$$

where, as i increases, the compressor is asymptotically able to accurately predict the following outcomes, because it creates an internal model of the data. In other words, C is memorizing, and in some cases, learning.

The dynamic complexity profile can be defined as

$$\overrightarrow{D}(x_t) = NC(x_1^i, t), \tag{5.6}$$

where x_1^i is the x sequence generated at the time iteration t considering that x is described from position 1 to i. As such, the dynamic profile was computed by providing the tape to the Markov model during the TM's execution time (while the tape is being edited), computing the NC in small intervals as described in Equation (2.12).

5.4.5 Increasing the probabilistic complexity of TM's tape

We also investigated the formulation of a methodology capable of consistently increasing the TM tapes' probabilistic complexity while maintaining the machines' *algorithmic complexity*. To this end, we created two methods:

- Method I aims to increase probabilistic complexity by optimizing the impact of the rules on the TM's probabilistic complexity (aggregation of fundamental rules).
- Method II aims to globally increase probabilistic complexity by iteratively changing the TM's rules.

The first method provided exciting conclusions regarding the interaction between critical rules, whereas the second proposed method consistently increases the probabilistic complexity of the tapes. It is worth mentioning that in every computation of the NC, the best Markov model was selected, and the set of TMs has specific conditions (specified in Section 5.4.1).

Method I

The main principle of the method described in Algorithm 3 is to maximize the relevance of a given rule on the TM by computing an impact measurement of the rule using the NC.

This metric is used as a selection criterion of rules when merging two randomly generated TMs.

The first premise is that a rule can be essential in a certain sub-network of rules at a particular time. The second premise is that we can successfully join essential rules relevant to the source matrix by maximizing impact metrics. Even if specific rules lost relevance during the TM merging process, since the value of their impact decreased, these rules would be replaced by others that would fit the TM rule matrix better.

Algorithm 3: Method I: pseudo-code algorithm.

```
   Input  : merge_number, number_of_states, alphabet_size
   Output: tm
 1 rule_matrix_1 = rule_matrix.random(number_of_states, alphabet_size);
 2 for ( it = 0; it < merge_number; ++it ) {
 3 |   nc_matrix_1 = Compute_nc_matrix(rule_matrix);
 4 |   rule_matrix_2 = rule_matrix.random(number_of_states, alphabet_size);
 5 |   nc_matrix_2 = Compute_nc_matrix(rule_matrix);
 6 |   rule_matrix_1 = tm_merge(rule_matrix_1, nc_matrix_1, rule_matrix_2, nc_matrix_2);
 7 }
 8 return tm(rule_matrix);
 9 Function tm_merge(rule_matrix_1, nc_matrix_1, rule_matrix_2, nc_matrix_2):
10 |   for ( index = 0; index < rule_matrix_1.size(); ++index ) {
11 |   |   nc_rule_1 = nc_matrix_1[index];
12 |   |   nc_rule_2 = nc_matrix_2[index];
13 |   |   if (nc_rule_2 > nc_rule_1) then
14 |   |   |   rule_matrix_1[index] = rule_matrix_2[index];
15 |   |   end
16 |   }
17 |   return rule_matrix;
18 Function Compute_nc_matrix(rule_matrix):
19 |   tm = tm(rule_matrix);
20 |   nc_original = nc(tm, number_iterations);
21 |   summation_nc=0; for ( i = 0  i < rule_matrix.size(); ++i ) {
22 |   |   original_rule = rule_matrix[i];
23 |   |   for ( rule = (0, 0, 0) ... (w_max, 2, s_max) ) {
24 |   |   |   if (rule ≠ original_rule) then
25 |   |   |   |   rule_matrix[i] = rule;
26 |   |   |   |   summation_nc += nc(tm, number_iterations);
27 |   |   |   end
28 |   |   }
29 |   |   average_nc = summation_nc/(number_possible_rules - 1);
30 |   |   rule_relevance = nc_original - average_nc;
31 |   |   nc_matrix[i] = rule_relevance;
32 |   |   rule_matrix[i] = original_rule;
33 |   }
34 |   return nc_matrix;
```

The algorithm can be decomposed into steps: (1) Generate a random rule matrix for a TM. (2) Run TM through n iterations. (3) Compute NC of resulting TM using its tape. (4) (i) For each element in the rule matrix, change its content sequentially to every

possible outcome $((0,0,0)\ldots(w_{max},2,s_{max}))$ other than the original rule, and maintain the remaining matrix unedited; (ii) for each outcome, run the TM n iterations and compute the resulting tape's NC; (iii) compute the average of all resulting NC values; (iv) compute the impact of the original rule in the element by calculating the difference between the original NC from Step (3) and the average NC from Step (iii); and (v) store the resulting value ($impact \in [-1,1]$) in an impact matrix, with the same size of the rule matrix, at the same index of the element in the rule matrix. (5) Repeat Steps (1)–(4) to generate another random rule matrix for a TM and obtain its impact matrix. (6) Merge the first and second TMs by selecting the rules with the highest impact value, creating a new TM. (7) Calculate the impact matrix of the new TM. (8) Keep generating and merging other new randomly generated TMs for n iterations.

This impact metric is vital as it allows us to assess the relevance of a given rule since:

1. An impact of zero means that the rule is not used by TM, as the mean probabilistic complexity from changing it does not alter from the original;

2. An impact value closer to 1 means that the rule is more relevant, as changing the rule results in a decrease in the probabilistic complexity of the tape compared to the original rule;

3. An impact closer to -1 signifies that the rule decreases the probabilistic complexity of the tape.

Method II

In contrast to the previous method, this more straightforward method iteratively attempts to increase the probabilistic complexity of the tape by changing TM rules. This method, described in Algorithm 4, looks at the global impact a rule has on the probabilistic complexity of the TM's tape and tries to increase the global NC.

The algorithm goes as follows: (1) While the length of the TM is smaller than a limit (set as 100 here), generate a random rule matrix, run TM through n iterations and determine the length of the generated tape. (2) Compute the NC of the resulting TM using its tape. (3) For the number of iterations defined: (i) randomly select a matrix rule index; (ii) randomly change the value of the rule; (iii) run TM for n iterations; (iv) determine the length and NC of the generated tape; and (v) if the resulting NC is higher than the previous maximum and the length is higher than 100 characters, keep the new rule. Finally, (4) Retrieve the new TM.

In the next section, we assess our compression-based method regarding noise (substitutions) and permutations in synthetic and actual genomic data.

Algorithm 4: Method II: pseudo-code algorithm.

```
Input  : number_of_states,alphabet_size,number_of_iterations
Output: tm
while length<100 do
    rule_matrix = rule_matrix.random(number_of_states,alphabet_size);
    length=tm(rule_matrix).get_length();
end
nc = nc(rule_matrix);
for ( it = 0; it ≠ number_of_iterations; ++it ) {
    random_index=random(rule_matrix.size());
    rule=rule_matrix[random_index];
    new_rule=rule.random();
    rule_matrix[random_index]=new_rule;
    nc_rule=nc(rule_matrix);
    new_length=tm(rule_matrix).get_length();
    if (nc<nc_rule AND new_length>100) then
        nc=nc_rule;
    else
        rule_matrix[random_index]=rule;
    end
}
return tm(rule_matrix);
```

5.5 Viability assessment of the NC

In this section, we validate using the NC computed with the best Markov model. To make this assessment, we made use of synthetic and natural inputs. The synthetic input was a string of 500 zeros followed by 500 ones, whereas the natural inputs were the complete genome sequences of the *Microplitis demolitor bracovirus segment O* and the *Human parvovirus B19 isolate BX1* (both retrieved from NCBI [1]).

5.5.1 Assessment

For each type of data, the substitution probability of the string increased from 0% to 100%, and the string was randomly permuted in blocks of increasingly smaller sizes. At each point, the NC was computed using the best Markov model ($k \in \{2, \ldots, 9\}$) (the model that provides the highest compression for that given string) and the obtained results were plotted as a heat map (Figure 5.2). It is worth mentioning that, since the generated synthetic data have low probabilistic complexity and genomic data usually have high probabilistic complexity, in synthetic data, the substitution was randomly set to generate any symbol of the alphabet $\theta = \{0, 1\}$ in order to increase the complexity of the string. On the other hand, in genomic data, the substitution was fixed to generate always a specific nucleotide (symbol of the alphabet) to decrease the complexity continually.

As shown in Figure 5.2, the NC behaves as expected, increasing as the substitution rate of the strings increases in synthetic data and decreasing in biological data.

[1] https://www.ncbi.nlm.nih.gov/nuccore/

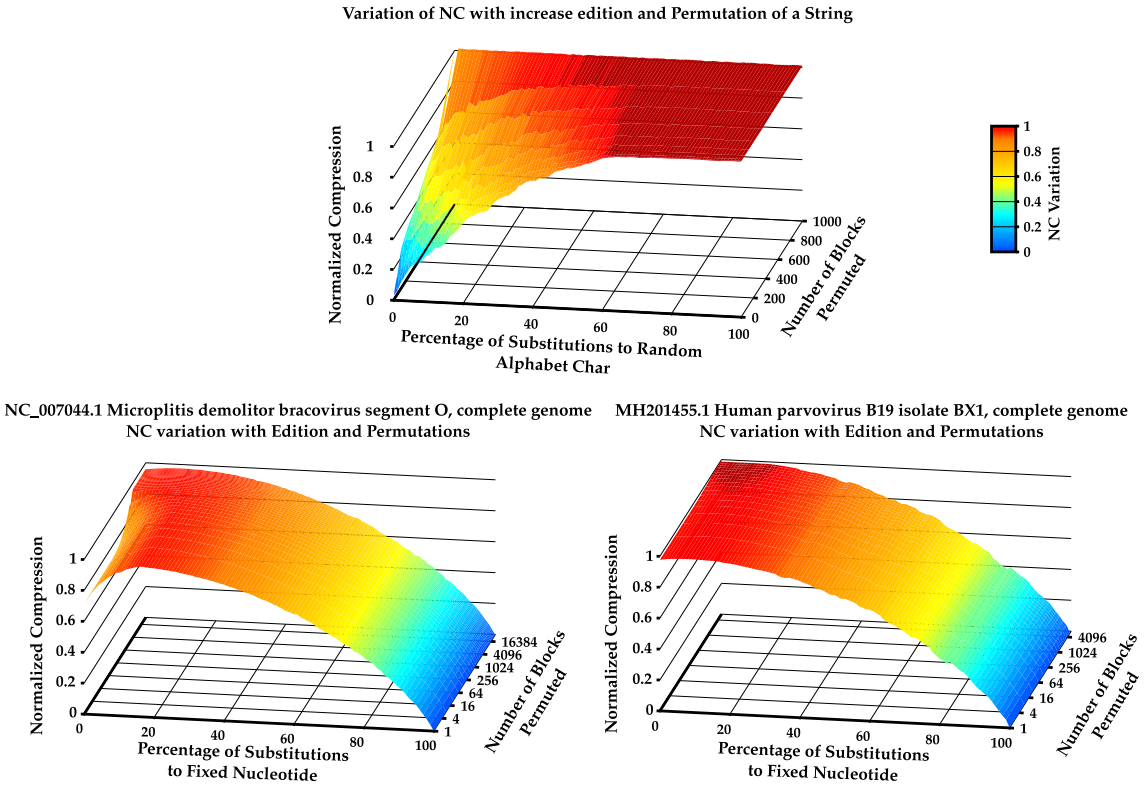


Figure 5.2: Heat map of Normalized Compression with an increase in permutation and edition rate. Generated string starting with 500 zeros followed by 500 ones (**top**); NC_-007044.1 *Microplitis demolitor bracovirus segment O*, complete genome (**bottom-left**); and MH201455.1 *Human parvovirus B19 isolate BX1*, complete genome (**bottom-right**).

The same occurs regarding a random permutation of the synthetic and natural strings. There was a significant increase in NC with the increase in the number of blocks permuted for the synthetic input. The same occurred in genomic data. Although less noticeable due to being more probabilistically complex than the synthetic sequence, an increase in the NC with an increase in the number of blocks permuted can be seen in both genomic sequences. This occurrence is more pronounced in the *Microplitis demolitor bracovirus* genome sequence than in the *Human parvovirus B19 isolate BX1* genome sequence since the latter has higher probabilistic complexity.

5.5.2 Insights

The results of the assessment suggest that the NC is a tolerant way of dealing with substitutions and permutations when computed in this manner. As such, this methodology could be successfully applied to measure the output of TM tapes since, besides being an ultra-fast method of obtaining information about the tape, it can cope with the presence of substitutions and permutations in a string and is thus a good estimator of probabilistic complexity on a tape.

5.6 Global analysis of Turing Machine tapes

This section analyses the probabilistic complexity of TMs with a different alphabet and state cardinality. We look at regions with spikes in NC and investigate the reason for their occurrence using global metrics and average rule complexity profiles.

5.6.1 Probabilistic complexity patterns of Turing Machines

All TMs ran for 50,000 iterations. This value was selected since a considerable number of iterations can still be performed in a reasonable computational time. The tapes produced by each TM were analysed with Markov models of context $k \in \{2, \dots, 9\}$, and the smallest NC was selected. Table 5.2 shows the average length of the tape and NC as well as its standard deviation obtained for each group of TMs with a different pair of $(\#Q, \#\theta)$. Due to the limited availability of computational resources, the number of sampled TMs in the Monte Carlo search approach was the same order of magnitude as the TNTM for $\#Q = 3$ and $\#\theta = 2$.

Table 5.2: The average and maximum standard deviation of the length of the tape and NC obtained in each TM group.

$\#\theta$	#Q	TM No.	Search Approach	Mean Amp ± std	Mean NC ± std
2	2	20,736	Sequential	32,055 ± 20,836	0.22724 ± 0.420,95
2	3	34,012,224	Sequential	29,745 ± 20,609	0.22808 ± 0.42330
3	2	34,012,224	Sequential	32,603 ± 19,923	0.17887 ± 0.38230
2	4	34,000,000	Monte Carlo	28,146 ± 20,348	0.22753 ± 0.42403
2	5	34,000,000	Monte Carlo	26,932 ± 20,092	0.22643 ± 0.42403
2	6	50,000,000	Monte Carlo	25,963 ± 19,856	0.22512 ± 0.42363
2	7	50,000,000	Monte Carlo	25,164 ± 19,636	0.22356 ± 0.42285
2	8	30,000,000	Monte Carlo	24,477 ± 19,433	0.22219 ± 0.42215
2	9	30,000,000	Monte Carlo	23,882 ± 19,245	0.22079 ± 0.42134
2	10	30,000,000	Monte Carlo	23,357 ± 19,068	0.21933 ± 0.42039

A moving average low-pass filter was applied to the obtained values of data compression, the tape's length was normalized by the maximum size obtained for its pair $(\#Q, \#\theta)$, and the results were expressed as plots for $\#Q \in \{2, \dots, 6\}$, as presented in Figure 5.3. We did not create the plots for higher cardinalities due to the under-sample of computed TM relative to the TNTM. Nevertheless, the plots show an interesting relationship between the tape's length and the NC. In particular, the NC tends to be higher for tapes with a shorter length. There are two significant reasons for this: firstly, smaller strings have few elements and thus are harder to compress. This occurs mainly in machines that only produce 1 to 10 elements in the tape. Secondly, there is a correlation between smaller tapes created by TMs with intertwined rules and a high NC value. This increase in probabilistic

complexity in short tapes is caused by TMs systematically re-writing portions of the tape several times and consequently increasing the probabilistic complexity.

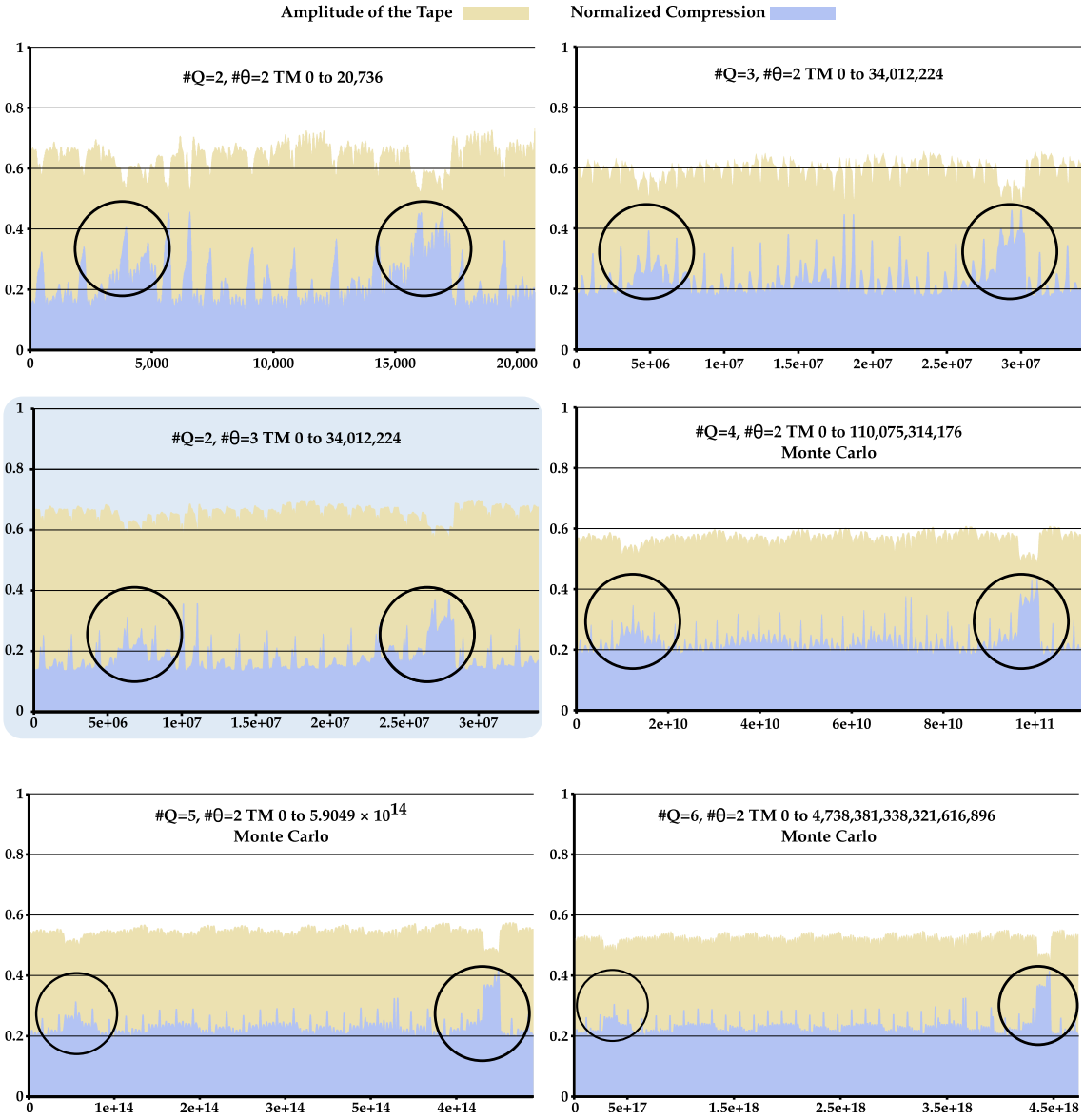


Figure 5.3: Plot of all TMs in Table 5.2. NC value is in blue, and the tape's normalized length is in yellow. The x-axes of the plots represent the index of the TM computed according to Algorithm 1. The **blue background** is the plot that corresponds to the group of TMs with $\#\theta = 3$ and $\#Q = 2$; all other plots have $\#\theta = 2$.

Overall, the TM plots show two large regions with a significant NC spike and a smaller tape length. Moreover, these regions persist for TMs with a different $\#Q$ and $\#\theta$, at least for the cardinality under examination (see circles in Figure 5.3).

To examine this phenomenon, we computed the average bits required to represent the generated tapes inside these regions, the NC, and the tape's length. Furthermore, we sampled from this pool 5000 TMs from inside and outside this region (except for $\#Q$ and

$\#\theta = 2$, with only 2000 selected). These sampled TMs ran for 1000 iterations. Figure 5.4 depicts these results:

- The length of TM's tape (top-left);
- The required bits to perform compression of the tape (top-middle);
- The corresponding NC value (top-right);
- The average required bits to perform compression of index rules used by each TM (bottom-left);
- The corresponding NC value (bottom-right).

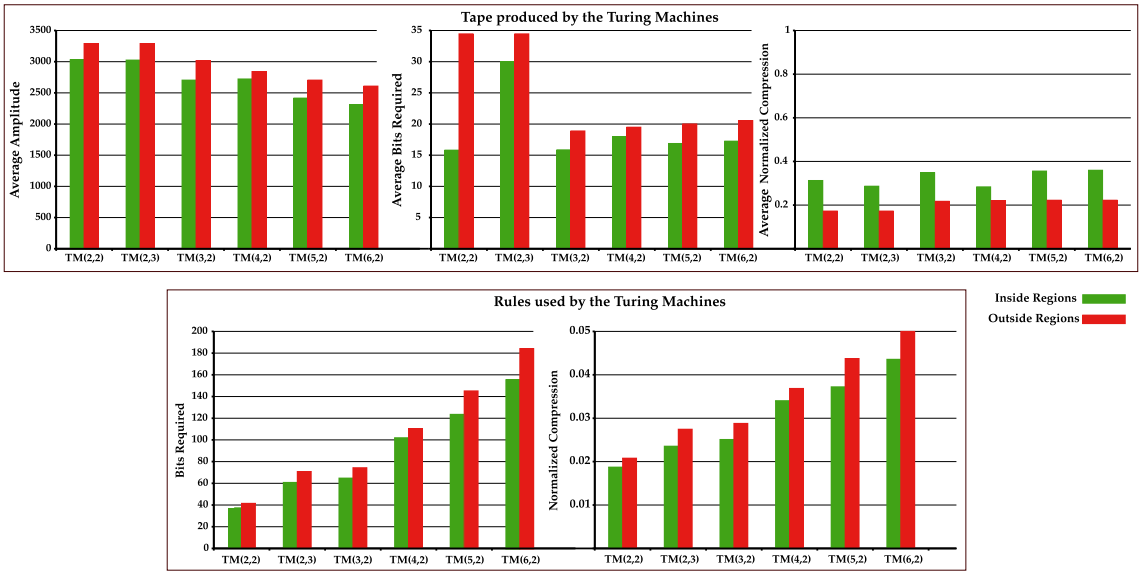


Figure 5.4: The average value for the length of TM's tape (**top-left**); average required bits to perform compression of the tape (**top-middle**); and average NC value (**top-right**), inside and outside the regions marked with circles in Figure 5.3. The average bits required (**bottom-left**); and the average NC value obtained for the rules used by the TM (**bottom-right**), inside and outside the regions marked with circles in Figure 5.3

By analysing Figure 5.4, we observe that, for all TM$(\#Q, \#\theta)$, the length of the tape and bits required to compress it are, on average, higher outside the circle lines than inside this region. In contrast, the average NC is higher inside the region. This information demonstrates that, despite the seemingly higher probabilistic complexity, tapes are, in fact, not more probabilistically complex but rather simply smaller in size. On average, the tapes inside this region require fewer bits to represent than those outside this region. Consequently, the results are only caused by the logarithmic normalization factor of the NC, which creates this discrepancy in values, not the complexity of the sequence itself.

Regarding the NC and the average bits required to represent the index of the rules

used, we notice that in both cases, the indexes of the rules are, on average, more easily compressible inside the regions under analysis (Figure 5.4, bottom). This data indicates the presence of smaller rule cycles in these regions, making it easier for the Markov models to compress them. To see this, we performed the normal complexity profile of these rules and computed their average. Figure 5.5 shows the important regions of these rule complexity profiles (all complete profiles are shown in Section C.2.1, Figure C.1).

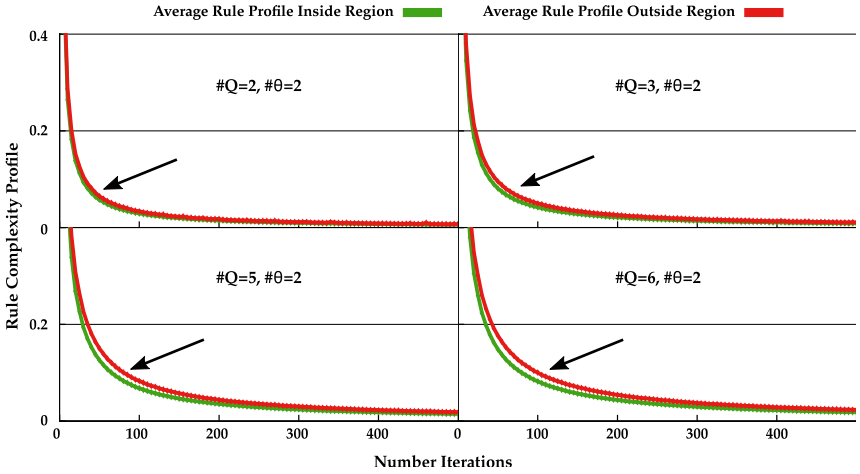


Figure 5.5: Regional capture of average rule complexity profiles obtained from pseudo-randomly selected TMs with $\#Q \in \{2, 3, 5, 6\}$ and $\#\theta = 2$ up to 1000 iterations.

The rule complexity profiles have a higher complexity outside the region than inside. Furthermore, the decrease in complexity occurs faster inside the region than outside. This difference is incremented with the increase in the cardinality of states. These results indicate that the rule cycles are smaller inside this region. These small rule cycles (that create low probabilistic complexity) are going back and forth in tiny regions of the tape, decreasing its overall size. This decrease in size coupled with the low complexity of the rules produces, on the one hand, models that require fewer bits to represent the tape and, on the other hand, a higher NC value due to its dividing factor $|x|$.

We conclude that given the same number of iterations, TM tapes in these regions possess smaller lengths and higher NC due to similar short cycles. The regions in Figure 5.3 were created because these short cycles occurred in TMs, which were grouped due to the sequential generation order of the TM rules (changes in the rules).

5.6.2 Insights

Using the Normalized Compression, we showed that some TMs have higher Normalized Compression (and smaller tape length) in two regions (shown up to six states). These regions correspond to probabilistic low complexity regions, and have a high NC due to tape size.

We localized these regions and, using average rule complexity profiles, identified this grouping as being caused by short cycles in the rules that output tapes with smaller

lengths. Since the probabilistic complexity of the TM tapes was measured assuming a sequential generation order of the rules (changes in the sequential rules), these similar short cycles with similar configurations were grouped together.

5.7 Analysis of probabilistically complex Turing Machine tapes

In this section we look at probabilistically complex TM tapes by applying the normal and the dynamic complexity profiles.

5.7.1 Analysis using normal and dynamic complexity profiles

To create the normal and dynamic complexity profiles, we filtered from each pair $(\#Q, \#\theta)$ in Table 5.2 the top 15 TMs with a tape length larger than 100 characters and the highest NC. Figure 5.6 depicts the normal and dynamic complexity profiles obtained from the TMs with the highest NC for their state cardinality when run until each has a tape length of 10,000.

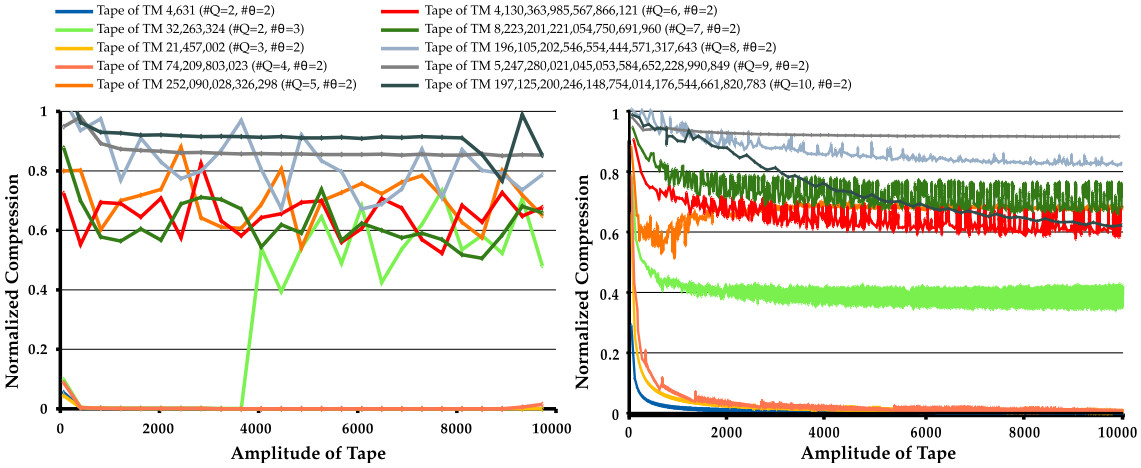


Figure 5.6: Normal complexity profiles (**left**); and dynamic complexity profiles (**right**) obtained for some of the filtered TMs. Each TM has a different cardinality of states or alphabet.

We can draw some conclusions from the normal complexity profile from the results. Firstly, all machines have tape regions that are harder to compress given the prior knowledge of the Markov table, thus the spikes. These spikes are regions where the tape pattern has been altered due to changes in the machine's state or part of the tape being rewritten.

Furthermore, similarly to Figure 5.2 (**top**), where an increase in the number of string editions leads to an increase in the probabilistic complexity of the string, the profiles of Figure 5.6 (**left**) show the tape regions which have suffered more edition. This edition is caused by complex rule interchange, which is harder to compress (more probabilistically complex).

Regarding the dynamic complexity profile, TMs that create a tape with simple repetitive patterns have a dynamic complexity profile that decreases their NC across time until it reaches approximately zero. This behaviour is also observed in TMs with the highest NC ($\#Q \in \{2,3,4\}, \#\theta = 2$). However, with the increase in the number of rules, the compression capacity of these tapes starts to decrease due to more complex patterns being generated. These patterns change during the creation of the tape due to interlocked changes in TM states and changes in received inputs.

As expected, in both the normal and dynamic complexity profiles, with the increase in $\#Q$ and $\#\theta$, there is an increase in the number and size of spiky regions (normal complexity profile) and the value of NC (dynamic complexity profiles). These results imply that the number of rules influences the amount of information on the tape. Notably, TMs with more rules have the potential to generate tapes that are generally harder to compress.

The results obtained show that these profile methods can localize higher probabilistic complexity for different purposes. On the one hand, the normal complexity profile can localize regions of higher and lower probabilistic complexity (assuming the machine has reached the external halting condition) and localize regions of short cycles of rules. On the other hand, the dynamic complexity profile is capable of localizing temporal dynamics of probabilistic complexity, showing where there is an abrupt change in probabilistic complexity.

5.7.2 Insights

In this section, we studied probabilistically complex TM tapes by applying the normal and dynamic profiles described in Section 5.4.4.

The normal and dynamic complexity profiles serve as approximate measures that can detect complexity change in the events through spatial and temporal quantities. The normal complexity profiles localize higher and lower probabilistic complexity regions assuming the machine reaches the external halting condition. In contrast, the dynamic complexity profiles localize temporal dynamics of probabilistic complexity through the dynamics of the tape (while running). The latter identifies when a change in complexity occurs at a certain depth. It is worth mentioning that a similar concept, Algorithmic Information Dynamics (AID), was introduced by Zenil, Kiani, and Tegnnér [258]. AID explores the effects of perturbations to systems and data (such as strings or networks) on their underlying computable models. It uses algorithmic probability between variables to determine the probability for the dependency (and direction between variables) and the set of candidate models explaining such a connection. On the other hand, the dynamic complexity profiles localize temporal dynamics of probabilistic complexity through the modification cycles of the tape. Namely, it shows where there is an abrupt change in the probabilistic complexity during the TM's execution. Moreover, it detects the rules that directly influence the tape's probabilistic complexity during the TM's run time. Although these notions are similar, they differ because one relies on NC to compute the probabilistic complexity of the tape

in a dynamic temporal way, whereas the other uses algorithmic probability to determine the smallest program to represent that pattern after the perturbations have taken place.

5.8 NC and BDM comparison

After considering the relationship between the *algorithmic complexity* of TMs and the probabilistic complexity produced by their tapes and applying complexity profiles as local measures to quantify and localize higher and lower regions of probabilistic complexity, in this section, we compared the NC and the BDM regarding their ability to detect the probabilistic complexity of tapes.

5.8.1 Comparison analysis between NC and BDM

To perform the comparison between NC and BDM, we pseudo-randomly selected 10,000 TMs with 6, 8, or 10 states and a binary alphabet, and ran each for 50,000 iterations. The NC and the one-dimensional normalized BDM (according to [48]) were applied to the tapes obtained. The average results can be observed in Table 5.3, and the overall behaviour for TMs with $\#Q = 10$ and $\#\theta = 2$ can be seen in Figure 5.7 (images for all analysed states are shown in Figure C.2 in Appendix C). Notice that, in Figure 5.7, the results are presented after applying a moving average low-pass filter. Furthermore, tape length was normalized by the maximum size obtained for its pair $(\#Q, \#\theta)$ and, to observe BDM compared to the NC, the BDM results were scaled in the left image by a 10^2 factor. An example with non-scaled BDM is also presented in the right image.

Table 5.3: The average and maximum standard deviation of the NC and NBDM obtained in each TM group.

$\#\theta$	#Q	TM No.	Mean Amp ± std	Mean BDM ± std	Mean NC ± std
2	6	10,000	26,359.6 ± 19,821	0.00042801 ± 0.011328	0.21631 ± 0.41753
2	8	10,000	24,471.8 ± 19,353	0.00045401 ± 0.010250	0.21975 ± 0.42054
2	10	10,000	23,123 ± 19,157	0.00060360 ± 0.012539	0.22696 ± 0.42404

Similar to Table 5.2, there is a decrease in the tape's length when the number of states increases. Additionally, in this sample, BDM and NC increase with the number of states. The results indicate that the algorithmic nature of BDM allows for a tiny representation of these data. Internally, BDM uses values derived from the computation of small TM. Consequently the BDM can represent TM tapes with very small values. Nevertheless, for a higher number of states, the BDM is progressively approximated by our methodology.

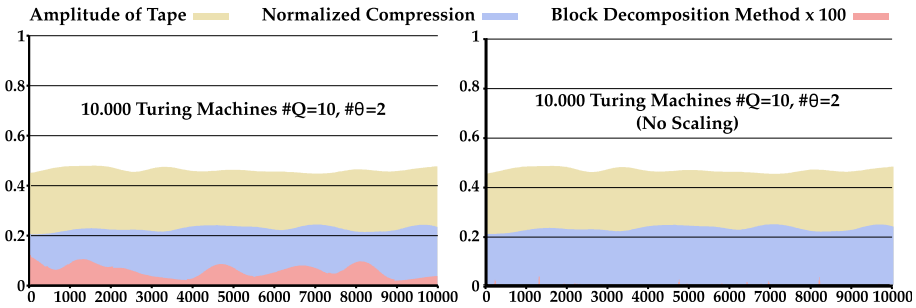


Figure 5.7: Comparison between the NC and BDM for 10,000 TM with $\#Q = 10$ and $\#\theta = 2$ that ran over 50,000 iterations: (**Left**) BDM scaled by a factor of 10^2; and (**Right**) same example but with non-scaled BDM.

5.8.2 Insights

Herein we compared the NC and the BDM in analysing the TM tapes. BDM efficiently describes the TM tapes since it uses TM values for their complexity estimation. Nevertheless, for a higher number of states, the BDM progressively approximates the statistical values since the algorithmic approach of the BDM is constituted by a set of small TMs, currently up to a maximum number of five states in a binary alphabet. Since BDM has been computed for this number of states, it lacks the information to represent some Turing Machine tapes after this state cardinality. This effect grows with an increment in the number of states.

On the other hand, the probabilistic approach used in the BDM is more simplistic than ours, making the proposed method more relevant for this specific application. Furthermore, since this analysis aims to quantify the probabilistic complexity of the TM tapes with the same *algorithmic complexity*, the BDM is unsuitable for use in this use case because the BDM has algorithmic models in its internal composition. Nevertheless, for an application of quantification of both algorithmic and probabilistic complexity, a combination of both methods would approximate the complexity of strings efficiently.

5.9 Increasing the probabilistic complexity of Turing Machine tapes

In this section, we analyse the impact of methods I and II on increasing the probabilistic complexity of TM tapes.

5.9.1 Applying methods I and II

Methods I and II were implemented and tested regarding their ability to increase probabilistic complexity. Method I was instantiated 200 times, where in each instance, 100 TMs were merged according to Algorithm 3. Method II was instantiated 2000 times (10 times more since it is a faster algorithm) for 1000 iterations. In both methods, all TMs

had a $\#Q = 10$ and $\#\theta = 2$, and were executed for 50,000 tape iterations. The compared results can be observed in Section 5.9.1.

Regarding method I (Section 5.9.1, left), both top and bottom plots show that this algorithm decreases the tapes' length by half while retaining approximately the same bits required to represent the tape and slightly increasing the NC. This method shows inconclusive results as the bits required to compress the tape are, on average, approximately the same. Therefore, the tape length shrinkage is responsible for the increase in NC. However, this could mean that information on the tapes is being condensed.

Moving on to the results obtained for method II, both top and bottom plots show that this algorithm, on average, decreases tape length while significantly increasing the bits required to represent the tape and the NC value. Besides being much faster than the previous method, it can systematically increase the probabilistic complexity of TMs. These results are a consequence of the way this method behaves. The method tries to change the rules by looking at the outcome of the TM's tape NC rather than augmenting

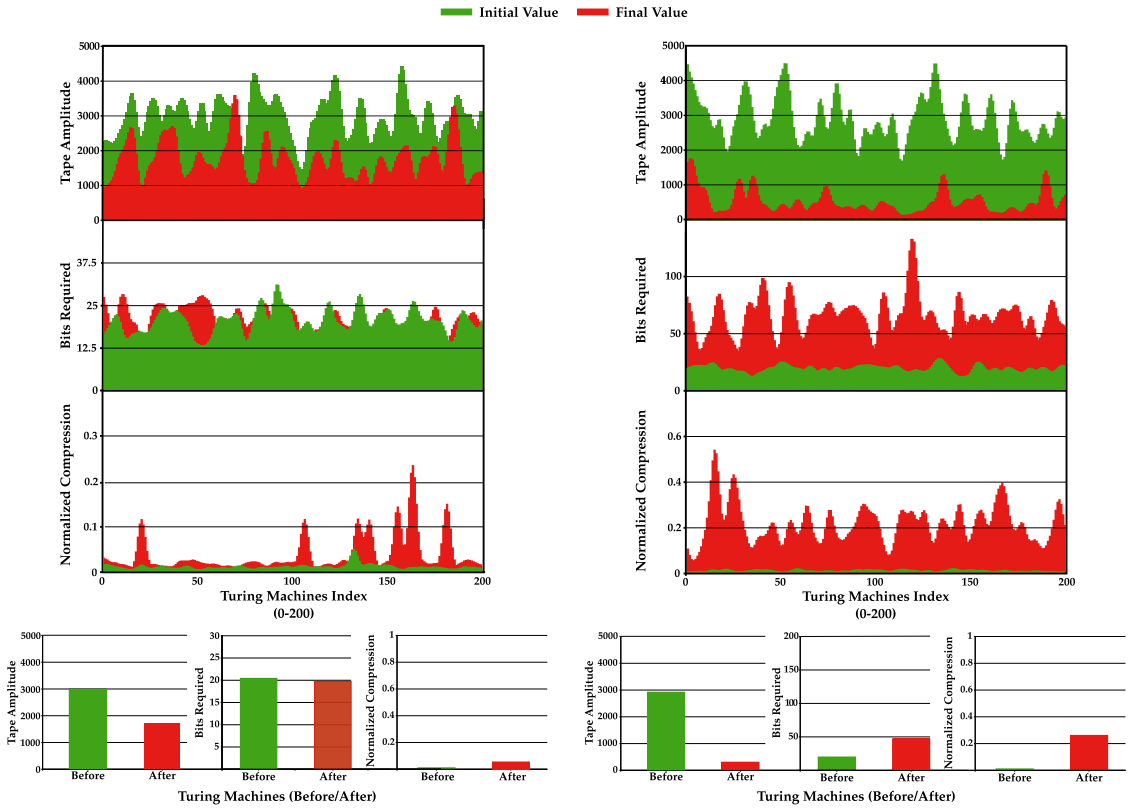


Figure 5.8: Comparison between method I (**left**) and method II (**right**). (**Top**) Plots show the length of the tapes, bits required to represent the sequence, and the NC obtained for 200 TMs after a low-pass filter was applied. (**Bottom**) Plots show the average tape length (**bottom-left**); average bits required (**bottom-middle**); and NC (**bottom-right**). Green and red represent TMs before and after the method was applied, respectively. For method I, the average corresponds to 200 instances, and for method II to 2000.

the rule's impact on the tape. Figure 5.9 presents two tape evolution examples using method II.

Tape 1 Before:01010101010101010101010101010101010101010101010101010101010101
Tape 1 After: 11111111111001111011110011101111001100111011111101111001100111
Tape 2 Before:00000000000000000000000000000000000000000000000000000000000000
Tape 2 After: 11110101101111111011010011101001000000100111100010010011110 1

Figure 5.9: The first 59 characters of TMs' tapes before and after method II was applied.

To evaluate method II better, we altered the number of rule iterations it uses to improve ($it \in \{50, 100, 200, \dots, 32,000\}$), the number of iterations the TM can write on the tape ($n \in \{500, 1000, 2000, \dots, 8000\}$), and computed for 2000 instances:

- The final average length of the tape;
- The variation of the bits required (BR) to represent the tape ($\overline{\text{BR}_{\text{var}}} = \frac{1}{n} \times \sum_{i=1}^{n}(BR_{final} - BR_{initial})$);
- The variation of NC ($\overline{\text{NC}_{\text{var}}} = \frac{1}{n} \times \sum_{i=1}^{n}(NC_{final} - NC_{initial})$).

The results of this process are shown in Figure 5.10.

It is possible to observe in Figure 5.10 that both rule iterations and iterations on the tape play a significant role in the probabilistic complexity of the TM. The method demonstrates that as the number of rule iterations increases (iterations that try to increase the NC value of the tape), both the variation of the NC (maximum average increase of 40%) and bits required (maximum average increase of 55 bits) increase. At the same time, the length of the tape decreases (minimum of 111 characters of tape length).

On the other hand, the variation of the NC and bits required act interestingly. Firstly, for a reduced number of rule iterations, the NC is higher for a lower number of tape iterations, showing an overall improvement of the NC due to the small tape size. However, as the number of rule iterations increases, so does the variation of the NC and bits required, showing an actual increase in the probabilistic complexity of the TMs' tapes. This result indicates that the sequence is getting increasingly complex with the number of iterations, as the only way to sustain an increase in complexity is by creating rules with non-repeatable patterns. As such, the decrease in tape length observed when using method II implies that the rules obligate the rewriting of certain parts of the tape, making it more compact and, thus, harder to describe by a probabilistic compressor. Accordingly, a simple adaptation of method II can be made to decrease probabilistic complexity while maintaining the same *algorithmic complexity* (the opposite). This adaptation allows the variation of the probabilistic complexity of a TM tape according to the desired value while maintaining the same *algorithmic complexity*.

5.9.2 Insights

In this section, we evaluated methods to increase the probabilistic complexity of TM tapes while maintaining the same amount of *algorithmic complexity*. Although method I

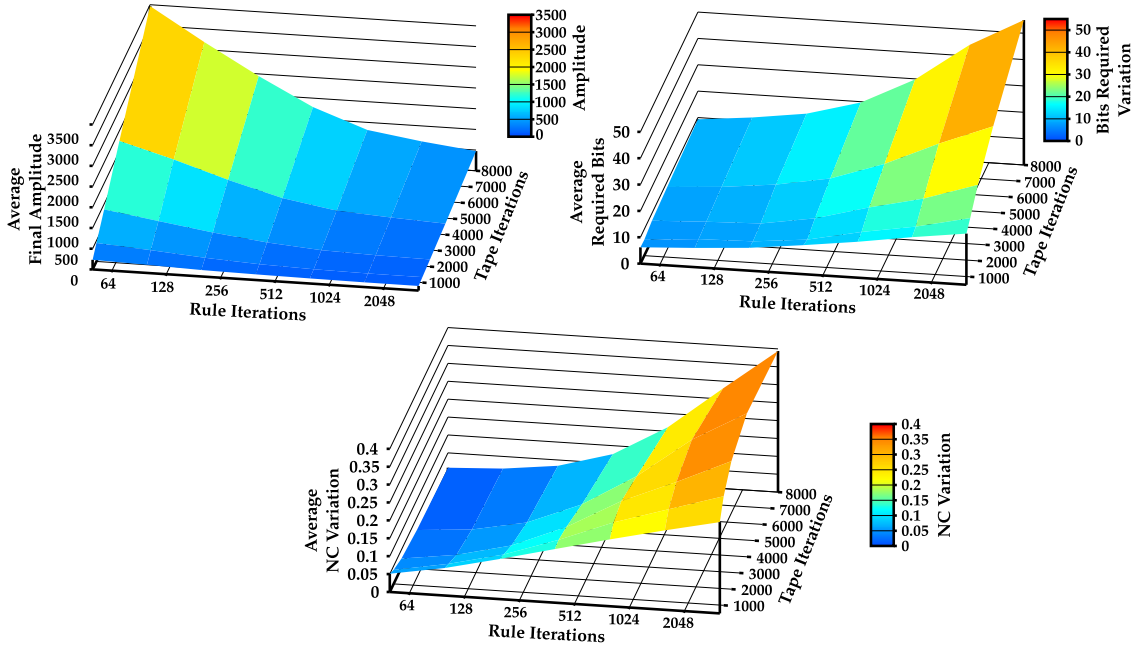


Figure 5.10: Average final length of the tape (**top-left**); variation of the bits required to represent the string (**top-right**); and variation of the NC (**bottom**), with the increase in number of rule iterations and tape iterations.

provided inconclusive results, method II could effectively increase probabilistic complexity while retaining *algorithmic complexity*. The value of the probabilistic complexity can be set to the desired number instead of the maximum using a simple variation of method II. This algorithm can be applied for ultra-fast generation from a short program, outputting tapes with a desired probabilistic complexity while the *algorithmic complexity* remains low. This algorithm may have many applications in genomics, metagenomics, and virology. For example, the output of these TMs can be used as an input to test and quantify the accuracy of genome assembly algorithms for different complexities from non-stationary sources. The significant advantage is that the output has high-speed generation without the need to store any sequence besides the program's configurations. Another application is the simulation of progressive mutations in metagenomic communities. It can also complement absent regions of genome viruses given sequencing incapabilities or DNA/RNA degradation (using similar probabilistic complexity). A broader application is the benchmark of compression algorithms used to localize complexity in the data given possible algorithmic sources.

5.10 Summary

Turing Machine tapes with the same *algorithmic complexity* may have very different probabilistic complexities. In this study, we analysed the behaviour of TMs with specific conditions regarding the probabilistic complexity of their generated tapes. To carry out this study, we created low *algorithmic complexity* TMs (a small number of states and

alphabet). We used a compression-based measure approximated by the best-order Markov model to measure the quantity of probabilistic complexity in the output tapes.

After assessing the compression-based approach regarding symbol editions and block transformations, we identified that TMs with the same number of states and alphabet produce distinct tapes with higher NC patterns in two regions. Furthermore, we showed that these two regions are related to repetitive short cycles in the configuration matrix, given sequential modifications in the matrix that create shorter tapes.

We introduced normal and dynamic complexity profiles as approximate measures to quantify local complexity. The normal complexity profiles localize regions of higher and lower probabilistic complexity for machines that reached the external halting condition. The dynamic complexity profiles localize probabilistic complexity's temporal dynamics through the tape's modification cycles. We showed examples using both profiles and applied them to achieve some findings in this chapter regarding global quantities.

Additionally, we compared a measure that uses both algorithmic and probabilistic approaches (BDM) to analyse the tapes. The fact that the tapes were created by TMs with the mentioned specific conditions makes BDM an efficient describer of the tapes. However, for a higher number of states, BDM is progressively approximated by our methodology.

Finally, we developed an algorithm to increase a tape's probabilistic complexity progressively. This algorithm creates a tape that evolves through a stochastic optimization rule matrix that is modified according to the quantification of the NC. As such, this algorithm can be modified to variate the value of the NC. We pointed out some of its potential applications in the bioinformatics field.

Chapter 6

On solving the inverse problem through approximation

[f] Abraham meets Melchizedek - Jorge Miguel Silva.

"When you find your path, you must not be afraid.
You need to have sufficient courage to make mistakes.
Disappointment, defeat, and despair
are the tools God uses to show us the way."
– Paulo Coelho

6.1 Contextualization

Despite being highly heterogenous, all data can be reduced to a string of bits. Kolmogorov Complexity analyses data in this way. It estimates the complexity of data as the length of a smallest program that, when run, generates the string and halts. Despite Kolmogorov complexity being noncomputable, it can be approximated.

In the previous chapter, we evaluated the usage of NC to select the best Markov model that minimizes the complexity quantity of TM tape. Although the *algorithmic complexity* of the TMs that generate the tapes is the same for a set of $(\#Q, \#\theta)$ since the number of rules is the same, the output tapes can vary widely in size and form (a good example being the Busy Beaver [259]). Hence, the approximation of the probabilistic complexity provided by the NC of the tapes generated differs substantially from tape to tape. While it is relatively easy to compress repeated sequences, the compression of some sequences with an algorithmic scheme has proven difficult to do by simple probabilistic compressors. However, the string is simple to describe using the rule matrix. This problem is an inverse problem since the rule matrix given the produced TM tape.

Suppose it is possible to develop a close algorithmic representation of a digital object, even if it is not the smallest representation possible. In that case, the applications are tremendous since we can (besides losslessly representing data in a compressed manner) get insights into the underlying structure of the data itself. Detecting algorithms in data can allow for better data compression, comprehension, and behavioural prediction.

As such, in this chapter we aim to create a program that can represent approximately any given string. However, this problem is very complex due to the high dimensionality and heterogeneity of the data. Consequently, this chapter presents only an initial proposal, and there is still much work to be done in order to function correctly for any type of string. The remainder of this chapter is divided as follows:

- Research Questions and Contributions;
- Considerations regarding the inverse problem;
- Methods;
- Results;
- Summary.

6.2 Research questions and contributions

As previously mentioned, *algorithmic Kolmogorov complexity*, although non-computable, can be approximated. One meaningful approximation is performed by data compressors, which are an upper bound of the *algorithmic complexity* of the sequence. However, most lossless data compression algorithms are limited to finding simple proba-

bilistic regularities because these algorithms have been designed for fast storage reduction. Still, some compressors like GeCo3 [51], and PAQ [239, 153] are designed for compression ratio improvement at the expense of more computational resources. These lossless data compression algorithms are hybrids between probabilistic and algorithmic schemes. This algorithmic approach differs from most lossless data compression algorithms since it considers that the source creates structures that follow algorithmic schemes and, as such, does not only perceive data as being generated from a probabilistic function. Consequently, they detect some algorithmic schemes in data and use them in data compression. But, more importantly, they show the importance of efficiently combining models that address probabilistic and algorithmic nature. Raw data is not only algorithmic in nature, and its characteristics, quantity of noise, and precision, on average, affect the efficiency of programs that measure information using pure algorithmic schemes more than those of a more probabilistic nature. Although nature can be based on algorithmic schemes, it is not limited to them. This other part is based on non-computable sources that make logical descriptions inconsistent [260, 261].

This chapter will explore a possible approximation solution for the inverse problem. Specifically, we propose a method that, given a target string y, tries to find a program that approximately describes that string. So how do we look for a program that generates the algorithmic schemes present in data? Furthermore, how do we do this in a way that is tolerant to noise and from a single example? These are the questions at the core of this chapter, which we will try to answer. When dealing with this problem, it is crucial to create a solution that leverages the algorithmic and probabilistic methods for the solution to be tolerant to noise and flexible in finding various algorithmic schemes in data. Another aspect that needs to be considered is the computational time required by this method, as it needs to be executed in feasible computational time.

As a result, we have made the following contributions in this chapter:

- A method of approximating a solution for the inverse problem;
- Three different search solutions.

6.3 Considerations regarding the inverse problem

6.3.1 Global aspects

Mathematical models are a key tool in the scientific pursuit of understanding our world. When a mathematical model is known, it is employed to predict the system's behaviour, given some parameters that describe a system. This approach is known as mathematical modelling, and the parameters are called model parameters.

In the inverse problem, causal factors are estimated using a set of observations. Since they provide information about parameters that are not directly observable, they are among the most paramount mathematical issues in science and mathematics, always being

present: e.g., estimating a planet's density from measurements of its gravitational field; determining a function that generates a sequence number; determining from where a fire weapon was shot, given the position of the bullet hole; decrypting a cryptographic message; finding a program that generated a given string.

Formally in an inverse problem, given the generated observation data (d_{obs}), we seek to identify the model parameters. As such, we look for the model parameters p such that (at least approximately)

$$F(p) = d_{obs}, \tag{6.1}$$

where F is the forward map, $F(p)$ is the data predicted by model p. Using this formulation, we can also think of the discrepancy between d_{obs} and $F(p)$ ($D(d_{obs}, F(p))$) as the residuals associated with the model. They are essential to assess the model's viability since their analysis indicates if the assumed model is realistic or not. The biggest challenge of the inverse problem is that it is an unstable process: noise and errors can be tremendously amplified, making a direct solution challenging. This is especially true for the case of raw data, which is common in scientific areas and is why probabilistic approaches have become prevalent when trying to solve these types of problems.

6.3.2 Inverse problem in case of study

Despite the progress of probabilistic methods, there is value in studying the algorithmic modelling of problems. For example, in data compression, although the probabilistic approaches have been highly studied, algorithmic modelling is yet to be fully explored.

In our case, finding the program that defines a given string allows us to losslessly represent compressed data and get insights into the underlying structure of the data itself. This is because data compression can be viewed as a way of learning since the more we can reduce data without losing their description, the more we understand their underlying structure [11]. Consequently, we could understand its structure better by finding an algorithmic program that describes a string in a compressed manner.

As seen in the previous chapter, sequences that follow algorithmic schemes are easier to compress with the algorithmic knowledge of how to compress them. The complex part is the inverse problem of how to determine from a target string what is the algorithm that generated it. We know of two approaches to solve this problem: human inspection and automatic computational search. Human inspection is primarily subjective, unpredictable, and requires programming time, while automatic computation is incremental and constant. Furthermore, given the size of the problem, it is only rational to use a computational solution.

Unfortunately, it is intractable to perform a sequential search due to the super-exponential growth of TMs associated with higher cardinalities of the alphabet or states. One might try to tackle the problem with a resource-bounded approach similarly to Levin [52, 53] or Hutter search [54, 55], although, in search of long strings, the problem also be-

comes unfeasible. More recent approaches are the CTM [56] and its derivation, the Block Decomposition Method [48]. These approaches decompose the quantification of complexity for segmented regions using small TMs. Furthermore, they change into a Shannon entropy quantification when modelling the probabilistic nature, such as noise. This numerical method that implements the ULS in a resource-bounded way has shown promising results in different applications. However, as seen in Chapters 3 and 4, it also has shown problems such as the incapacity to detect small programs such as IRs and underestimating the amount of information contained in the object.

Inspired by a data compressor that combines models addressing the probabilistic and algorithmic nature of data, in the following section, we introduce a novel method that aims to create an approximate solution to the inversion problem by discovering Turing Machines that define or approximately define a given string.

6.4 Methods

6.4.1 General configuration

Usually, data compressors use a probabilistic approach to look for patterns in strings. The program follows a fixed set of rules to analyse string patterns, and using these patterns creates a probabilistic model that it uses to compress the data. Instead of following this reasoning, our proposal balances probabilistic and algorithmic methods. This equilibrium is performed using algorithmic methods to generate output and probabilistic methods to assess similarities between the generated and target strings.

Our proposal has two modules to approximate an algorithmic source that produces a target string y. Module 1 uses Turing Machines as string generators (their tapes), and module 2 uses Finite-Context Model (FCM) to determine the loss between the generated string x and target string y. Turing Machines have the same configuration as those referenced in the previous chapter. Each TM starts at state zero, interacts with a blank tape (tape full of zeros), and stops after n iterations, producing the tape x. The generated string x is then compared to the target string y.

To compare the generated string with the target tape, we used FCMs. Each Markov model loads the generated tape using a context k and updates its internal model. This internal model is used to compute the probability of any character being read at a given point. After reading the generated strings, the FCMs are used to compute the loss function between strings. This process of generation and evaluation of similarity is performed until a solution is found or a N number of executions are performed. At this point, the program outputs its best results obtained.

6.4.2 Loss function

To determine the similarity between a target string and the generated TM tape, we computed a loss (L) using the FCMs. Specifically, FCMs were loaded with the target

string ($FCM(y)$) and saved in the program's memory. When a generated string x is created, other FCMs with the same contexts as $FCM(y)$ are created and loaded with x ($FCM(x)$). The models are paired for each context k and then used to compute the loss function.

The loss is determined using three main components, two based on the Kullback–Leibler divergence (D_{KL}) and one based on a length difference penalization.

Particularly, the loss is determined according to

$$L = D_{KL}(y \parallel x)_{cond} + D_{KL}(y \parallel x)_{seq} + \lambda(L(|x|, |y|)), \tag{6.2}$$

where $D_{KL}(y \parallel x)_{cond}$ is the Kullback–Leibler divergence determined between the conditional distributions of the FCM from the predicted string (x) given the target string (y); $D_{KL}(y \parallel x)_{seq}$ is the Kullback–Leibler divergence determined between distributions of occurrence of a substring s of size $(k+1)$, where k is the context of the FCM; λ is a constant multiplication factor and $L(|x|, |y|)$ is the loss penalization from the size difference between x and y.

The conditional Kullback–Leibler divergence ($D_{KL}(y \parallel x)_{cond}$) is determined as

$$D_{KL}(y \parallel x)_{cond} = \frac{1}{|c|} \sum_{i=0}^{|c|} \sum_{j=0}^{|\theta|} P^y(\theta_j|c_i) \times \log_2 \left(\frac{P^y(\theta_j|c_i)}{P^x(\theta_j|c_i)} \right), \tag{6.3}$$

where $|\theta|$ is the cardinality of the alphabet θ, $|c|$ is the cardinality of all prefixes for an alphabet θ and a context k, determined by $|c| = \theta^k$. $P^y(\theta_j|c_i)$ is the conditional probability of a letter θ_i, given the prefix c_j.

The Kullback–Leibler divergence for a given prefix c_j followed by the alphabet letter θ_i is determined as

$$D_{KL}(y \parallel x)_{seq} = \frac{1}{|c|} \sum_{i=0}^{|c|} \sum_{j=0}^{|\theta|} P^y(c_i\theta_j) \times \log_2 \left(\frac{P^y(c_i\theta_j)}{P^x(c_i\theta_j)} \right), \tag{6.4}$$

where $P^y(c_i\theta_j)$ is the probability of a given sequence of prefix c_j followed by the alphabet letter θ_i occurring.

And finally, the loss penalization from the size difference between x and y is computed as

$$L(|x|, |y|) = \begin{cases} -log_2\left(\frac{|y|}{|x|}\right), |x| \leq |y| \\ -log_2\left(\frac{|x|}{|y|}\right), |x| > |y| \end{cases}. \tag{6.5}$$

6.4.3 Search approaches

In order to perform the search for strings, we defined three search approaches:

- Sequential search;

- Monte Carlo search;
- Guided search.

Sequential and Monte Carlo searches served as baseline methods to compare the guided search. It is worth mentioning that for all search approaches, multiple processing cores were used.

To perform a sequential search, we used the same approach explained in Chapter 5, where we navigate through all possible TMs of a pair $(\#Q, \#\theta)$ by considering that each TM is represented by its rule matrix M and define a *total order relation* between them. The unique numerical identifier (id) was computed using the same notation explained in Section 5.4.2 and Algorithm 1.

Monte Carlo algorithm [256] was used again since this algorithm is widely used to obtain qualitative information regarding the behaviour of large systems [257]. As explained in Section 5.4.2, rather than starting in one rule matrix and ending in another, the method was used to configure TMs randomly by sampling each cell's components (write, move, state) from uniform distributions.

The guided search is a derivation of A*, an informed search algorithm drafted in terms of weighted graphs. In the A* algorithm, the search starts from a specific starting node of a graph and seeks to find a path to a goal node having the smallest cost. It holds a tree of paths originating at the start node and extends those paths one edge at a time until its termination criterion is met. In our derivation, length penalization was considered as the heuristic, and the cost was the average of the Kullback–Leibler divergences. Furthermore, one node is composed of a TM rule matrix and the loss obtained from the tape generated after n_{it} iterations.

Initially, the program receives as input the alphabet cardinality, state cardinality, the number of tape iterations (n_{it}) each Turing Machine runs, and the number of attempts to approximate the solution ($n_{total\_attempts}$). Then, with this knowledge, our algorithm selects n starting nodes (one node each running thread) with a random state matrix and associated cost. In each thread, the algorithm will run $n_{total\_attempts}/n_{treads}$ attempts to find a TM that represents a given TM tape.

After defining each thread's initial node, each starting node will determine its successors. In this case, each successor has a rule matrix that differs in one rule from its ancestor node. In total, the number of possible successors (n_{p_suc}) for a node is given by

$$n_{p\_suc} = \Big(\#\theta \times \#Q \times 3 - 1\Big)\Big(\#\theta \times \#Q\Big). \tag{6.6}$$

Using each successor's rule matrix, a TM is created and run for n_{it} iterations to obtain its generated tape x. Then, utilizing the generated tape, a loss is computed towards the target string. This pair is inserted into a node and added to the list of nodes to open. This list is ordered by the nodes with fewer costs (in this case, the lower loss). The ancestor node is then moved to a synchronized, unordered set of visited nodes shared

among threads. Afterwards, a successor node not included in the set of visited nodes is opened. The process is repeated until the number of attempts is reached or a solution is found. During this process, the nodes which provided the smallest costs are retained in a top K list.

6.4.4 Guided search optimizations

Due to the sheer size of the search space, some other optimizations were performed on this search. Namely, the creation of a patience mechanism, the selection of random successors, selective loss increment for successor nodes, and providing a good initialization of the starting nodes.

The patience mechanism was performed to prevent nodes from getting stuck on a specific local minimum. Each thread has a maximum patience number ($MaxPatience$) which, once reached, stops the search process in that thread, discards the nodes to be opened and restarts the search with a new node using a random rule matrix. The maximum patience number is a function of the current loss value as

$$MaxPatience = \frac{1}{2 \times l_c} + c, \tag{6.7}$$

where l_c is the loss of the current node being opened, and c is the baseline maximum patience value. This heuristic value and the function that relates loss and maximum patience were refined by trial and error. The mechanism works by incrementing the patience value (which starts at zero) every time the current node has the same loss as its ancestor. On the other hand, when loss decreases, the patience value returns to zero. When the level of patience reaches the maximum value for patience, the search in that thread stops and is restarted with a new never-visited node with a rule matrix filled with pseudo-randomly generated rules. After this restart, the patience value returns to zero, and the process starts again.

Another improvement to the algorithm was the selection of random successors from the total pool of possible successors. Due to the large number of possible successors each node has and the space of search characteristic of this problem, instead of opening all possible nodes, only a sample of random successors is selected to compute its loss. This number of successors also depends upon the loss of the current node being opened since the aim is to have fewer successor nodes of a node with high loss rather than nodes with smaller losses. As such, at a given point, the number of successor nodes is determined by

$$n_{suc} = \frac{2}{l_c} + n_{min}, \tag{6.8}$$

where n_{min} is the minimal number of successors for a given node, determined as

$$n_{min} = log_2(n_{p\_suc} + 1) \times 100. \tag{6.9}$$

Once again, these values are heuristics determined by trial and error.

Another mechanism introduced to navigate the search space faster, and avoid local minima, is the selective loss penalization of successor nodes. Concretely, instead of running all iterations of the TM, each successor runs for a small number of iterations first. Following this short run, the nodes that, at that point, have the same loss as the ancestor node (at the same number of iterations) are attributed a final loss equal to the ancestor node plus a penalization (1% loss of the ancestor node).

Finally, the initialization is performed, and instead of generating a random node for each thread, each thread generates 10 initial nodes. Since the node to open is a priority queue, the first node to be used is the one with the smallest loss.

6.4.5 Data representation

The bit representation of the target string y, is defined as $B(y)$. In the case of finding a program that represents y, the generated string x is the same as y and the number of bits ($B(y)$) is expressed by

$$B(y) = B_{algo}(y), \tag{6.10}$$

where $B_{algo}(y)$ is the number of bits that a Turing Machine when executed for n iterations outputs the target string y. Likewise, $B_{algo}(y)$ can be defined as

$$B_{algo}(y) = rm_{bits} + \lceil log_2(N_{tape_{it}}) \rceil, \tag{6.11}$$

where $\lceil log_2(N_{tape_{it}}) \rceil$ is the bits required to represent the number of tape iterations used by the TM; and rm_{bits} is the bits required to represent the rule matrix, which is computed as

$$rm_{bits} = \Big(\lceil log_2(\#Q) \rceil + \lceil log_2(\#\theta) \rceil + \lceil log_2(M) \rceil \Big) \Big(\#Q \times \#\theta \Big), \tag{6.12}$$

where M is the number of movements the Turing Machine can take, which is always 3 (left, fixed, right); $\lceil log_2(\#Q) \rceil$ is the number of bits required to represent the number of states used; and $\lceil log_2(\#\theta) \rceil$ is the bits required to represent the alphabet used.

If the program does not find a program that represents the string ($x \neq y$), and obtains only an approximation of the target string y, the bits required to describe y, will be computed as

$$B(y) = B_{algo}(y) + \lceil C(y||x) \rceil, \tag{6.13}$$

where $B_{algo}(x)$ is the number of bits required to represent the best program found that represents the target string y (according to Equation (6.11)), and $C(y||x)$ is the relative compression. $C(y||x)$ can be seen as the number of bits used to represent y, having exclusively the information from the string x. It measures the residual bits associated with the tape generated using the algorithmic model and is computed as

$$C(y||x) = -\sum_{i=0}^{|y|} log_2(P(y_i|x)). \tag{6.14}$$

The bits required to simply represent the string y is determined as

$$B(y) = \lceil |y| \times log_2(\#Q) \rceil. \tag{6.15}$$

6.5 Performance evaluation using synthetic data

To evaluate the performance of each method, we devised an experiment that, using method I described in Chapter 5, created TMs with increasingly more probabilistic complexity. Specifically, for each state-alphabet cardinality pair, 10 tapes were generated after 1000 iterations of a TM, where each tape was generated to fit into an NC bucket. The first tape $NC \in]\ 0, 0.2]$, the second $NC \in]\ 0.2, 0.4]$, and so on. These tapes were generated for $\#\theta \in \{2, 4\}$ and $\#Q \in \{2, 4, 8, 16, 32, 64\}$.

After their creation, these tapes were used to assess the performance of the search methods. All three search methods were executed for each of the tapes. The search TMs were initialized with the same state and alphabet as the tapes generated by the Turing Machines. If a tape was generated with a two state and a two symbol alphabet ($\#\theta$, $\#Q) = 2$, the search was only executed in this space. Furthermore, the iterations used to generate the tapes were also provided to the search program. All searches had a maximum of 10,080,000 attempts to find the target tape, and run in 18 threads (each thread searched for 560,000 TM). Each attempt corresponded to a TM being tested as the generator of the solution. After the program finds a program that generates the string or if the maximum number of executions is reached, the program halts and outputs the best-obtained results to a file. The number 10,080,000 was selected because it roughly corresponds to 10 minutes of computation using 18 cores. It is worth mentioning that our program has a seed input parameter, and as such, all these searches can be replicated.

6.6 Results

Table 6.1 shows the results obtained from the sequential, Monte Carlo and guided search for the generated TM tapes with $\#\theta \in \{2, 4\}$ and $\#Q \in \{2, 4, 8, 16, 32, 64\}$. For each file, the search runs for, at the maximum, 10,080,000 attempts, corresponding to approximately 10 minutes.

Table 6.1: Results obtained for sequential, Monte Carlo and guided search. The table shows the minimum loss obtained per file for each search performed as well as the bits required to represent the target object as a string $B(y)$, and with our program $(B_{algo}(x) + \lceil C(y||x) \rceil)$. In bold are the searches that found a TM that generates the target tape y and the TM that could be more compressed with our method rather than simply using the tape as representation.

#Q,#θ	NC	Min loss			B(y)	$B_{algo}(x) + \lceil C(y\|\|x) \rceil$		
		Sequential	Monte Carlo	Guided		Sequential	Monte Carlo	Guided
2, 2	]0.0,0.2]	**0,00**	**0,00**	**0,00**	252	**36**	**36**	**36**
		0,00	**0,00**	**0,00**	168	**27**	**27**	**27**
2, 4	]0.0,0.2]	**0,00**	**0,00**	**0,00**	334	**51**	**51**	**51**
		0,00	**0,00**	**0,00**	402	**60**	**60**	**60**
	]0.2,0.4]	3,37E-03	1,75E-03	**0,00**	128	**59**	**59**	**59**
		9,90E-02	5,32E-02	3,53E-03	120	**57**	**57**	**51**
	]0.4,0.6]	**0,00**	**0,00**	**0,00**	572	**59**	**59**	**59**
		4,80E-03	**0,00**	**0,00**	501	**70**	**59**	**59**
	]0.6,0.8]	2,13E-02	4,71E-03	8,44E-05	185	**58**	**57**	**57**
2, 8	]0.0,0.2]	1,29E-02	**0,00**	**0,00**	501	**129**	**117**	**117**
		0,00	**0,00**	**0,00**	401	**107**	**107**	**107**
	]0.2,0.4]	8,43E-02	**0,00**	**0,00**	113	116	115	115
		1,56E-01	2,75E-03	1,57E-04	104	118	113	112
	]0.4,0.6]	5,91E-03	**0,00**	**0,00**	668	**115**	**115**	**115**
		5,20E-02	**0,00**	**0,00**	384	**108**	**107**	**107**
	]0.6,0.8]	2,08E-02	**0,00**	**0,00**	239	**107**	**107**	**107**
		9,53E-02	3,23E-03	**0,00**	194	**127**	**113**	**113**
2, 16	]0.0,0.2]	1,57E-03	**0,00**	**0,00**	144	236	235	235
		0,00	**0,00**	**0,00**	334	**235**	**235**	**235**
	]0.2,0.4]	1,02E-01	2,32E-04	**0,00**	126	245	242	242
		1,06E-01	2,74E-04	**0,00**	162	257	242	242
	]0.4,0.6]	7,43E-03	**0,00**	**0,00**	668	**254**	**243**	**243**
		5,85E-03	**0,00**	**0,00**	668	**235**	**235**	**235**
	]0.6,0.8]	6,76E-01	**0,00**	**0,00**	377	603	**242**	**242**
		1,40E-01	**0,00**	**0,00**	236	245	242	242
	]0.8,1,0]	6,20E-01	3,02E-05	**0,00**	1001	1227	**235**	**235**
		6,37E-01	9,57E-03	**0,00**	371	597	**235**	**235**
2, 32	]0.0,0.2]	1,14E-02	**0,00**	**0,00**	144	542	531	531
		1,53E-03	**0,00**	**0,00**	600	**523**	**523**	**523**
	]0.2,0.4]	4,88E-02	**0,00**	**0,00**	119	530	529	529
		1,49E-01	**0,00**	**0,00**	126	525	523	523
	]0.4,0.6]	1,61E-02	**0,00**	**0,00**	329	523	523	523
		6,52E-01	**0,00**	**0,00**	230	744	529	529

#Q,#θ	NC	Min loss			B(y)	$B_{algo}(x) + \lceil C(y\|\|x) \rceil$		
		Sequential	Monte Carlo	Guided		Sequential	Monte Carlo	Guided
2, 64	]0.6,0.8]	7,39E-01	**0,00**	**0,00**	601	1115	531	531
		5,63E-01	**0,00**	**0,00**	358	756	523	523
	]0.8,1,0]	9,01E-01	5,52E-02	6,79E-03	134	648	549	553
		7,24E-01	**0,00**	**0,00**	261	775	523	523
	]0.0,0.2]	4,40E-02	**0,00**	**0,00**	498	1164	1163	1163
		1,17E-01	**0,00**	**0,00**	108	1173	1170	1170
	]0.2,0.4]	1,51E-01	**0,00**	**0,00**	148	1174	1169	1169
		8,78E-02	**0,00**	**0,00**	112	1174	1171	1171
	]0.4,0.6]	5,84E-01	1,27E-04	**0,00**	252	1352	1169	1169
		6,36E-01	**0,00**	**0,00**	221	1375	1163	1163
	]0.6,0.8]	5,53E-01	**0,00**	**0,00**	554	1512	1163	1163
		3,59E-01	**0,00**	**0,00**	305	1184	1170	1170
	]0.8,1,0]	9,41E-01	6,22E-04	3,32E-04	459	1613	1177	1177
		1,01E+00	**0,00**	1,54E-05	282	1436	1163	1163
4, 2	]0.0,0.2]	2,97E-02	**0,00**	**0,00**	256	**61**	**59**	**59**
		0,00	5,09E-04	**0,00**	466	**60**	**60**	**60**
	]0.2,0.4]	7,59E-02	1,06E-01	4,92E-02	210	**75**	**77**	**59**
		4,65E-02	4,32E-02	2,56E-02	218	**53**	**53**	**51**
	]0.4,0.6]	1,01E-01	9,32E-02	3,03E-02	256	**59**	**56**	**52**
4, 4	]0.0,0.2]	1,94E-04	**0,00**	**0,00**	996	**108**	**107**	**107**
		4,35E-05	**0,00**	**0,00**	1992	**118**	**118**	**118**
	]0.2,0.4]	4,48E-02	2,62E-02	**0,00**	368	**131**	**161**	**113**
		2,88E-02	1,42E-03	1,07E-04	276	**111**	**107**	**107**
	]0.4,0.6]	3,50E-02	1,64E-03	**0,00**	300	**109**	**108**	**107**
		2,70E-01	1,11E-01	3,52E-02	364	421	**264**	**125**
4, 8	]0.0,0.2]	3,05E-02	**0,00**	**0,00**	576	**254**	**243**	**243**
		3,77E-04	**0,00**	**0,00**	288	**236**	**235**	**235**
	]0.2,0.4]	5,16E-03	**0,00**	**0,00**	1142	**254**	**243**	**243**
		3,40E-02	**0,00**	**0,00**	1000	**237**	**235**	**235**
	]0.4,0.6]	2,01E-01	3,82E-02	3,34E-02	216	347	246	253
		3,07E-01	1,04E-03	**0,00**	1092	1141	**253**	**242**
	]0.6,0.8]	4,46E-01	3,01E-01	2,07E-01	250	437	409	367
		3,12E-01	2,13E-01	1,05E-01	212	405	394	268
4, 16	]0.0,0.2]	1,08E-01	**0,00**	2,18E-05	1334	**525**	**523**	**523**
		7,24E-05	**0,00**	**0,00**	1992	**534**	**534**	**534**
	]0.2,0.4]	2,11E-01	2,89E-03	5,56E-05	386	846	547	533
		1,23E-01	3,39E-02	1,39E-02	212	550	532	530
	]0.4,0.6]	1,92E-01	6,83E-02	3,63E-02	478	709	546	535
		1,65E-01	3,16E-03	**0,00**	356	817	531	529
	]0.6,0.8]	3,53E-01	2,62E-01	1,36E-01	264	739	637	612

#Q,#θ	NC	Min loss			B(y)	$B_{algo}(x) + \lceil C(y\|\|x) \rceil$		
		Sequential	Monte Carlo	Guided		Sequential	Monte Carlo	Guided
		4,95E-01	3,67E-01	3,17E-01	220	723	726	617
4, 32	]0.0,0.2]	4,55E-02	**0,00**	**0,00**	574	1165	1163	1163
		3,49E-02	**0,00**	**0,00**	402	1174	1172	1172
	]0.2,0.4]	1,95E-01	1,26E-01	9,75E-02	238	1193	1193	1179
		2,88E-01	9,84E-02	8,43E-02	268	1201	1193	1192
	]0.4,0.6]	4,77E-01	8,43E-02	6,88E-02	342	1429	1187	1174
		2,65E-01	1,01E-02	1,30E-04	242	1359	1163	1163
	]0.6,0.8]	4,40E-01	2,42E-01	1,69E-01	254	1400	1313	1270
		4,93E-01	3,60E-01	2,69E-01	210	1356	1317	1250
	]0.8,1,0]	4,87E-01	4,56E-01	4,49E-01	230	1376	1369	1313
		4,99E-01	3,92E-01	3,88E-01	216	1351	1321	1355
4, 64	]0.0,0.2]	7,05E-04	**0,00**	**0,00**	668	2591	2580	2580
		1,26E-01	3,26E-02	2,60E-02	522	2589	2580	2582
	]0.2,0.4]	1,02E-01	1,50E-02	1,48E-02	362	2605	2586	2592
		5,34E-01	1,60E-01	1,70E-01	582	3136	2608	2605
	]0.4,0.6]	5,07E-01	3,11E-01	2,52E-01	348	2902	2793	2731
		5,33E-01	2,73E-01	2,53E-01	582	3107	2860	3119
	]0.6,0.8]	4,48E-01	2,32E-01	2,94E-01	320	2874	2710	2755
		5,10E-01	3,14E-01	3,28E-01	244	2798	2773	2699
	]0.8,1,0]	5,91E-01	4,77E-01	4,77E-01	242	2796	2789	2789
		6,10E-01	4,75E-01	4,75E-01	246	2800	2789	2789

The results show that the methodology functions accurately for synthetic data. Independently of the searching mechanism used, the model finds candidate programs that approximate the tape after the 10,080,000 attempts. Two variables affect the approximation: the NC of the tape generated and the TNTM. On the one hand, the higher the NC of a tape, the more complex it is, and the fewer examples of programs representing that string. On the other hand, with the increase of state and alphabet cardinality, the TNTM increases, and the larger the searching space. As a result, the search becomes increasingly more difficult. In terms of search, the sequential search obtained the worst results, followed by the Monte Carlo search, whereas the best results were systematically achieved by the guided search.

Moreover, guided search found the most solutions, indicating this methodology's superiority. This is better observed in Table 6.2, where the minimal average loss of each method is illustrated, as well as the percentage number of found solutions. As can be seen, the best minimal average loss for the experiments was achieved by the guided search (5,12E-02). Furthermore, it obtained the most significant number of found solutions (60,64%).

However, this outperformance did not translate to the compression of the tape, where Monte Carlo and guided search obtained similar results, despite guided search using, on

Table 6.2: Global measures obtained by developed methodology.

Global Measures	**Sequential**	**Monte Carlo**	**Guided**
Average Loss	2,33E-01	6,24E-02	5,12E-02
Number of solutions	8	47	57
Found solutions (%)	8,51	50,00	60,64
Average Bits ($B_{algo}(x) + \lceil C(y\|\|x) \rceil$)	815,30	725,21	719,50
Compressed	35	41	41
Compressed (%)	37,23	43,62	43,62

average, fewer bits. This is the result of fundamentally 2 aspects. First, the number of attempts that were provided for each search. With such a long search space, finding optimal solutions with only 10 minutes of computation is complex. Secondly, the length of the generated tapes was small (they were generated with only 1000 iterations), which made the tapes (especially the more complex ones) have small sizes. Even though all generated tapes had a minimum length of at least 100 symbols, this size is small, and as a result, the bits required to represent the program is larger than the bits required to represent the tape itself.

6.7 Insights

Using our method in synthetic data revealed that navigating such an ample space and finding programs that represent the strings is possible using our methodology.

The results also indicate that guided search outperformed other methods. The results also show that less statistically complex tapes are easier to find because they occur more commonly within the search space. Nonetheless, data compression results showed similar results between guided and Monte Carlo searches. The two main factors for this were a low number of attempts to find the optimal solution by the guided search and the size of the target string.

We defined a time-constrained evaluation method to create a mechanism that would validate our methodology. We were sure the target string existed within that number of iterations and alphabet and state cardinality. Unfortunately, to do so, we could only search for small tapes (the minimum size of each generated tape was 100 symbols). Consequently, the compression suffered since in many cases more bits were required to represent the program than those to represent the target string itself.

These results seem promising, however, there are some aspects to consider. First, the program knows the number of states, alphabet, and iterations the tape generated. Without this knowledge, the search space increases. This point may be overcome if we first assume that the alphabet of the program is the same as the one present in the string. After fixing the alphabet cardinality, we can perform a random search between state cardinality with a

fixed number of iterations to finetune the location of the search. After choosing the space to search, we can look for the best number of iterations by using the parameter (w), which provides a range from the initial number of iterations where loss computations would be determined. The iterations that provided the best results would be set as the new number of iterations, as the cycle would repeat. Furthermore, the w interval would increase with smaller losses.

The second aspect is that the search was performed in synthetically generated data, and raw data is much noisier and more complex. Only with more testing can we determine the best approach to take.

6.8 Conclusions

This chapter proposes a methodology to approximate the inverse problem and find programs representing a given string. Our initial results show that finding solutions in vast spaces is possible in a short amount of time. This validates that our methodology based on a loss using the Kullback-Leibler divergence works and that an approximation method is possible using a derivation of the A* search. Nonetheless, more work needs to be done to fine-tune the solution by testing it in a more significant number of instances and applying it to raw data.

Chapter 7

Conclusions and Future Work

[g] Übermensch - Jorge Miguel Silva, 2020.

"The light shines in the darkness,
and the darkness did not overcome it."
– John, Bible: John 1:5

7.1 Contextualization

In this dissertation, we investigated Kolmogorov complexity approximations as data descriptors and described novel applications for different data types. Specifically, we first examined 1-dimensional data, using genomic sequences for this analysis. In this analysis, we portrayed the viral complexity landscape of viruses. Furthermore, we detected small programs (IRs) in genomic sequences using Kolmogorov complexity approximations and performed an in-depth taxonomic classification using compression-based measures.

Next, we investigated the usage of Kolmogorov complexity approximations of 2-dimensional data by using a dataset of artistic paintings. We showed that approximations could improve state-of-the-art classification results and serve as an explanatory descriptor of art and how artists paint. Subsequently, we analysed pure algorithmic data by investigating TM outputs and the relation between probabilistic and *algorithmic complexity*. Among other things, we described an algorithm to progressively increase a tape's probabilistic complexity while retaining the same *algorithmic complexity*. Finally, in the last chapter, we reflect on the inversion problem and possible workaround to solve it by approximation. To that end, we use compression-based measures and searching strategies to detect programs representing the output. This last chapter completes this doctoral thesis by presenting some closing remarks. Specifically, we highlight the most relevant findings from this thesis, indicate some of the limitations of our work, and point out future research directions obtained from the research developed during this Ph.D.

7.2 Relevant findings

The main results of this thesis arise from the comprehensive experiments carried out at different levels and with different data types. Notwithstanding, all studies involved the usage of Kolmogorov complexity approximations. Our main findings are summarized below:

- BDM can be used to measure and identify data content generated by simple algorithms. On the other hand, it cannot detect programs embedded into data, such as inverted repeats, and has difficulty dealing with information quantification due to block representability. Furthermore, BDM efficiently describes the TM tapes since it uses TM values for their complexity estimation. Nevertheless, for a higher number of states, the BDM is progressively approximated by the NC computation since the algorithmic approach of the BDM is constituted by small TMs.

- The NC computation using data compressors with embedded subprograms can accurately detect some algorithmic patterns, such as inverted repeats. It is robust to data alterations (pixel edition, substitutions, permutations and quantization), and can measure the quantity of information without underestimation. It is also a good data descriptor, providing good classification results in diverse tasks, such as

taxonomic identification and author and style attribution.

- We have demonstrated the importance of Kolmogorov complexity approximations in data description and classification. Moreover, we have shown that this information can be combined with other simple features to achieve state-of-the-art results.

- We describe and use minimal bi-directional complexity profiles of one sequence of each virus to visualize the distribution of complexity of these sequences locally. These profiles can describe actual regions detected in the genome with other methods, proving the description capability of data compression at a structural level.

- We proposed and described an average regional complexity matrix of an artist (fingerprint) by dividing the image into equal quadrilateral parts and averaging the estimated local complexity of each painting. Complexity measures the total number of properties (plus the language used) transmitted by an object and detected by an observer. By dividing images into blocks of equal size and evaluating its local complexity, we quantified the local information being transmitted. This fingerprint is a good data descriptor and helpful for classification and content explanation.

- We described the normal and dynamic profiles for analysing TM tapes. These complexity profiles serve as approximate measures that can detect complexity change in the events through spatial and temporal quantities. The normal complexity profiles localize higher and lower probabilistic complexity regions assuming the machine reaches the external halting condition. In contrast, the dynamic complexity profiles localize temporal dynamics of probabilistic complexity through the dynamics of the tape (while running). The latter identifies when a change in complexity occurs at a certain depth. Moreover, it detects the rules that directly influence the tape's probabilistic complexity during the TM's run time.

- We developed an algorithm to increase a tape's probabilistic complexity progressively. This algorithm creates a tape that evolves through a stochastic optimization rule matrix that is modified according to the quantification of the NC. We pointed out some of the applications of this algorithm in the bioinformatics field.

At a biological level, we identified that:

- On average, dsDNA viruses are the most redundant (least complex) according to their size, and ssDNA viruses are the least redundant. Contrarily, dsRNA viruses show a lower redundancy relative to ssRNA viruses.

- Some viruses that infect extremophiles are more redundant and possess more IRs, indicating the presence of an adaptation to stabilize the genome in these environments.

- In human herpesviruses, higher compressibility and abundance of inversions could

be linked with viral genome integration.

- It is possible to provide the structural description of the viral genome using minimal bi-directional complexity profiles.
- Rich genomic features suffice to correctly identify viruses and archaea's taxon.

In the artistic field, we found that:

- On average, paintings with low complexity are abstract and minimalist with simple patterns. Paintings with a slightly higher average complexity possess different regional complexities, specifically, a region with high complexity and detail surrounded by a background of low complexity. This noticeable pattern begins to fade with more complexity, and the most complex paintings are globally irregular, detailed, and convoluted.
- The low side of the complexity spectrum was characterized by Abstract Expressionism, Minimalism, and Constructivism movements, with authors such as Mark Rothko, Lucio Fontana, Piet Mondrian, and El Lissitzky. Also, artists from Abstract Expressionism characterized the high complexity side of the spectrum, such as Willem de Kooning, Jackson Pollock, and Jasper Johns, as well as other artists with a more detailed and convoluted style, like Gustav Klimt and Vincent van Gogh. Due to two different currents (Colour Field with authors with low average complexity and Action Painting with authors with high complexity), Abstract Expressionism was present at the polar ends of the spectrum. In all cases, Jackson Pollock had average complexity values that were completely different from other artists, the average complexity of his paintings being close to random. Although he denied being a creator of random paintings, this result and others indicate that Jackson Pollock's dripping paintings are not typical artworks, possibly related to the artist's inclusion of many symbolic layers and dispersion intentions over the canvas.
- When evaluating the artists' average NC and the roughness exponent (α) of the HDC function in the label images of the dataset, we found that styles are well confined into different regions, showing that the combination of these measures gives a robust representation of artistic movements. Specifically, the NC adds to the brightness and relative spatial position shown by the roughness exponent, the notion of average information in each artist's painting, consistent within the same style and historical circumstances. We also detected that the NC is inversely correlated to α in Abstract Expressionism. Artists related to Colour Field painting presented a high α and low NC, whereas artists related to Action painting presented the exact polar results (low α and high NC).
- The proposed complexity fingerprints give specific insights regarding each artist's way of painting, showing where, on average, artists paint with more detail and give

more emphasis while also providing insights into each artist's range of complexity. Furthermore, it provides critical information concerning how the work is perceived, such as composition, unity, balance, movement, rhythm, focus, contrast, pattern, and proportion of the painting and space.

- Cladograms built using the fingerprints aggregated authors of the same style close to each other and artists' influencing relationships, like Francis Bacon and Georges de la Tour, and George Braque and Hieronymus Bosch. Furthermore, proximity between artists is also caused by shared methods and techniques, which are not correlated with the period or artistic movement. For example, Hans Holbein and Vermeer are close but do not share styles. However, both used optics to achieve precise positioning in their compositions.

As regards the statistical and *algorithmic complexity* analysis of TM tapes, we have found that:

- Using the Normalized Compression, we showed that some TMsv have higher Normalized Compression (and lower tape length) in two regions (shown up to six states). Due to tape size, these regions correspond to probabilistically low complexity regions and have a high NC. Using average rule complexity profiles, we localized these regions and identified the cause as short cycles in the rules that output tapes with lower amplitude. Since the probabilistic complexity of the Turing tapes was measured assuming a sequential generation order of the rules (changes in the rules sequential), it groups these similar short cycles given their similar configurations.

- We analysed the behaviour of complex TM tapes using normal and dynamic complexity profiles. In both the normal and the dynamic complexity profiles, with the increase in $\#Q$ and $\#\theta$, there is an increase in the number and size of spiky regions (normal complexity profile) and the value of NC (dynamic complexity profiles). These results imply that the number of rules influences the amount of information on the tape. Notably, TMs with more rules have the potential to generate tapes that are generally harder to compress.

Lastly, regarding the inversion problem chapter, we have found that:

- It is possible to perform an efficient search within a vast space to find programs that represent a target string.

- The TMs found could be used to describe strings efficiently and thus be used to compress them.

7.3 Future research directions and work limitations

During the Ph.D., many of the studies developed were produced as a proof of concept or initial study to validate that analysis of complexity could be applied successfully to various areas and types of data and have many possible applications. Consequently, there were many aspects left to explore due to time constraints. Below, we offer some possible research directions that can be investigated in the future.

Future research related to the complexity of genomic sequence analysis can take a wide variety of directions, including:

- Increase the dataset size of the study to accommodate not only viruses and archaea but also other realms of life, such as bacteria, fungi, plants, and animals.
- Perform genomic sequence translation to aminoacid sequences and study the impact its NC has on classification, as well as evaluate its complexity across different taxonomic levels.
- Augment the pool of features being used, such as persistent minimal sequences across the genome species or automatic features that can be extracted using neural networks.
- Evaluate the impact of using different compressors in in-depth taxonomic classification.
- Develop an easy-to-use and deployed pipeline for in-depth taxonomic classification of metagenomic samples.
- Lastly, with the tools developed, a closer look could be taken at each genome species regarding its complexity and create tools that can better explain the genomic morphology.

Regarding future continuations in the analysis of artistic paintings, many possible lines of work can be considered. Specifically:

- We analysed the images of paintings by converting them to monochrome. However, it would be insightful to separate the colour channels and analyse them separately, therefore studying the influence of colour in the paintings in terms of complexity and how they would improve classification results.
- It could be helpful to explore how to separate different fingerprint characteristics and detect unknown repeated patterns that appear multiple times in a painting by creating and analysing their complexity surfaces [230] as well as the classification results when using these metrics.
- Another interesting study would be to replicate the work in Chapter 4 using a competitive compressor that would select the best compressor model for each painting

or region.

- Finally, the work regarding artistic classification is only a proof of concept. Therefore, it would be interesting to create ties with specialists in the area from museums and forensics. These connections would allow us to expand the dataset under analysis, have fake art copies, and provide other helpful technical features that could be fed to the model and improve the classification and authentication of artworks.

Regarding the analysis of algorithmic and probabilistic complexity analysis of TM tapes, future research includes:

- The implementation of an algorithm that, using a short program, outputs tapes with a desired probabilistic complexity while retaining the same *algorithmic complexity*. Implementing and providing this algorithm could be valuable in many fields, such as genomics, metagenomics, and virology. For instance, it could be used as an input to test and quantify the accuracy of genome assembly algorithms for different complexities with non-stationary sources assuming the probabilistic line (high-speed generation without the need to store any sequence besides the configurations). It could also simulate progressive mutations in metagenomic communities or complement absent regions of genome viruses given sequencing incapabilities or DNA/RNA degradation. Furthermore, a broader application is the benchmark of compression algorithms used to localize complexity in the data given possible algorithmic sources.

- Use dynamic complexity profiles to localize temporal dynamics of probabilistic complexity through the modification cycles of the tape. This information could potentially improve the speed of approximation in the inversion problem (Chapter 6).

Lastly, regarding the inversion problem, much work is still ahead of us, namely:

- The search algorithms only function after providing the state and alphabet cardinality and the number of iterations used. This limits the program's utility as it has to be the user who defines these parameters. Since we can infer the cardinality of the alphabet that generated a given string, a possible solution could be to initially perform a Monte Carlo search between different states and verify which space seems more promising. This initial search should consider the target string's size and the bits required to represent the TM rule matrix. After deciding on the search space, a standard guided search would follow. On the other hand, to solve the fixed iteration problem, a solution could be to start the search with a predefined number of iterations and modify within a range related to the loss value. The initial number of iterations should be correlated with the size of the target string and its complexity. After defining the initial number of iterations, a variable would be defined that delimited the range of search. The range would be defined regarding the loss of a current node, increasing when the loss decreases.

- Due to the vast search space, creating a fast and efficient search is essential. Many optimizations may increase the search program's speed, but these have yet to be tested. For instance, improvement mechanisms can be made to penalize TMs that create small cycles depending on the distribution dispersion. In the case of very complex tapes, TMs that have been shown to have smaller cycles within a small number of iterations could be penalized. Another improvement regarding speed is to perform computations of the TMs using GPUs.

- Regarding the accuracy of the search, other mechanisms should be explored. For example, the usage of reinforcement could improve node choice and facilitate navigation within the space. Another option is to model the problem so that it can use deep learning. However, this requires transposing this discrete problem into the continuum space to apply the chain rule.

- Finally, we have yet to systematically test the searching mechanism in raw data, which is highly heterogeneous, diverse, and noisy. Although our methodology can deal with noise, testing is required. On the other hand, since strings of this nature are usually large, a possible way to overcome performing so many iterations would be to provide a highly complex tape to the Turing Machines before performing the search.

As a final note regarding the global aspects of research, we have only scratched the surface of possible applications and use cases where Kolmogorov complexity approximations could be used. We hope that in the future, research will be carried out to make progress in this field and its applications.

"When you want something, all the universe conspires to help you achieve it."
– Paulo Coelho, The Alchemist

たとえ私の体が壊れても、誰かが私の場所を取って、この世界をより良い場所にします。

References

[1] David Sacks. *Letter perfect: the marvelous history of our alphabet from A to Z*. Broadway Books, 2004, p. 395. ISBN: 9780767911733. (Cit. on pp. 2, 8).

[2] Johanna Drucker. *The alphabetic labyrinth: the letters in history and imagination*. 1995, p. 320. ISBN: 0500016089. (Cit. on pp. 2, 8).

[3] B Jack Copeland. "The modern history of computing." In: (2000). (Cit. on p. 2).

[4] Maxwell Herman Alexander Newman. "General principles of the design of all-purpose computing machines." In: *Proceedings of the Royal Society of London. Series A. Mathematical and Physical Sciences* 195.1042 (1948), pp. 271–274. (Cit. on p. 2).

[5] A M Turing. "On Computable Numbers, with an Application to the Entscheidungsproblem." In: *Proceedings of the London Mathematical Society* s2-42.1 (1936), pp. 230–265. DOI: 10.1112/plms/s2-42.1.230. (Cit. on pp. 2, 80).

[6] Peter Wegner. "Research Paradigms in Computer Science." In: *Proceedings of the 2Nd International Conference on Software Engineering*. ICSE '76. San Francisco, California, USA: IEEE Computer Society Press, 1976, pp. 322–330. (Cit. on p. 2).

[7] Computing Curricula. "Computer Science, Final Report, The Joint Task Force on Computing Curricula." In: *IEEE Computer Society and Association for Computing Machinery, IEEE Computer Society* (2001). (Cit. on p. 2).

[8] Gordana Dodig-Crnkovic. "Scientific methods in computer science." In: *Proceedings of the Conference for the Promotion of Research in IT at New Universities and at University Colleges in Sweden, Skövde, Suecia*. 2002, pp. 126–130. (Cit. on pp. 2, 3).

[9] Cristian S Calude. *Information and randomness: an algorithmic perspective*. Springer Science & Business Media, 2002. (Cit. on p. 2).

[10] Gregory J Chaitin. "A theory of program size formally identical to information theory." In: *J. Assoc. Comput. Mach.* 22 (1975), pp. 329–340. ISSN: 0004-5411. DOI: 10.1145/321892.321894.
URL: https://doi.org/10.1145/321892.321894 (cit. on pp. 2, 10, 12, 13).

[11] Jorma Rissanen. "Modeling by shortest data description." In: *Automatica* 14.5 (1978), pp. 465–471. (Cit. on pp. 3, 12, 104).

[12] Ming Li, Xin Chen, Xin Li, Bin Ma, and Paul Vitányi. "The similarity metric." In: *IEEE transactions on Information Theory* 50.12 (2004), pp. 3250–3264. (Cit. on p. 3).

[13] Rudi Cilibrasi and Paul Vitányi. "Clustering by compression." In: *IEEE Transactions on Information theory* 51.4 (2005), pp. 1523–1545. (Cit. on p. 3).

[14] Ming Li, Paul Vitányi, et al. *An introduction to Kolmogorov complexity and its applications*. Vol. 3. Springer, 2008. (Cit. on pp. 3, 12–14).

[15] J Amaral, Michael Buro, Renee Elio, Jim Hoover, Ioanis Nikolaidis, Mohammad Salavatipour, Lorna Stewart, and Ken Wong. "About Computing Science Research Methodology, 2011." In: *URL http://citeseerx. ist. psu. edu/viewdoc/summary* (). (Cit. on p. 3).

[16] National Research Council (U.S.). Committee on Academic Careers for Experimental Computer Scientists. *Academic careers for experimental computer scientists and engineers*. National Academy Press, 1994, p. 139. ISBN: 0309049318. (Cit. on p. 3).

[17] Ming Li, Xin Chen, Xin Li, Bin Ma, and Paul Vitanyi. "The similarity metric." In: *IEEE Transactions on Information Theory* 50.12 (Dec. 2004), pp. 3250–3264. ISSN: 0018-9448. DOI: 10.1109/TIT.2004.838101. (Cit. on pp. 4, 14).

[18] Rómulo Antão, Alexandre Mota, and J A Tenreiro Machado. "Kolmogorov complexity as a data similarity metric: application in mitochondrial DNA." In: *Nonlinear Dynamics* 93.3 (Aug. 2018), pp. 1059–1071. ISSN: 1573-269X. DOI: 10.1007/s11071-018-4245-7.
URL: https://doi.org/10.1007/s11071-018-4245-7 (cit. on p. 4).

[19] Harold A Innis and A John (Alexander John) Watson. *Empire and communications*. Dundurn Press, 2007, p. 287. ISBN: 1550026623. (Cit. on p. 8).

[20] Nyquist H. "Certain Factors Affecting Telegraph Speed." In: *Bell System Technical Journal* 3.2 (1924), pp. 324–346. DOI: 10.1002/j.1538-7305.1924.tb01361.x. (Cit. on p. 8).

[21] Hartley R. "Transmission of Information." In: *Bell System Technical Journal* 7.3 (1928), pp. 535–563. DOI: 10.1002/j.1538-7305.1928.tb01236.x. (Cit. on p. 8).

[22] John B Anderson and Rolf Johnnesson. *Understanding information transmission*. John Wiley & Sons, 2006. (Cit. on p. 9).

[23] Shannon C E. "A Mathematical Theory of Communication." In: *Bell System Technical Journal* 27.3 (1948), pp. 379–423. DOI: 10.1002/j.1538-7305.1948.tb01338.x. (Cit. on p. 9).

[24] Peter D Grünwald and Paul Vitányi. "Kolmogorov complexity and information theory. With an interpretation in terms of questions and answers." In: *Journal of Logic, Language and Information* 12.4 (2003), pp. 497–529. (Cit. on p. 9).

[25] Ray J Solomonoff. "A preliminary report on a general theory of inductive inference." In: Citeseer. 1960. (Cit. on p. 10).

[26] Ray J Solomonoff. "A formal theory of inductive inference. Part I." In: *Information and control* 7.1 (1964), pp. 1–22. (Cit. on p. 10).

[27] R J Solomonoff. "A formal theory of inductive inference. I." In: *Information and Control* 7 (1964), pp. 1–22. ISSN: 0019-9958. (Cit. on p. 10).

[28] Andrei N Kolmogorov. "Three approaches to the quantitative definition ofinformation'." In: *Problems of information transmission* 1.1 (1965), pp. 1–7. (Cit. on p. 10).

[29] Daniel Hammer, Andrei Romashchenko, Alexander Shen, and Nikolai Vereshchagin. "Inequalities for Shannon Entropy and Kolmogorov Complexity." In: *Journal of Computer and System Sciences* 60.2 (2000), pp. 442–464. ISSN: 0022-0000. DOI: https://doi.org/10.1006/jcss.1999.1677. (Cit. on p. 10).

[30] Teresa Henriques, Hernâni Gonçalves, Luís Antunes, Mara Matias, João Bernardes, and Cristina Costa-Santos. "Entropy and compression: two measures of complexity." In: *Journal of evaluation in clinical practice* 19.6 (2013), pp. 1101–1106. (Cit. on p. 10).

[31] Gregory J Chaitin. "On the length of programs for computing finite binary sequences." In: *J. Assoc. Comput. Mach.* 13 (1966), pp. 547–569. ISSN: 0004-5411. DOI: 10.1145/321356.321363.
URL: https://doi.org/10.1145/321356.321363 (cit. on p. 10).

[32] Gregory Chaitin. "On the difficulty of computations." In: *IEEE Transactions on Information Theory* 16.1 (1970), pp. 5–9. (Cit. on p. 10).

[33] Gregory J Chaitin. "Incompleteness theorems for random reals." In: *Advances in Applied Mathematics* 8.2 (1987), pp. 119–146. (Cit. on p. 10).

[34] Gregory J Chaitin, Asat Arslanov, and Cristian Calude. *Program-size complexity computes the halting problem*. Tech. rep. Department of Computer Science, The University of Auckland, New Zealand, 1995. (Cit. on p. 10).

[35] Charles H Bennett. "On random and hard-to-describe numbers." In: *Randomness and complexity*. World Sci. Publ., Hackensack, NJ, 2007, pp. 3–12. DOI: 10.1142/9789812770837_0001.
URL: https://doi.org/10.1142/9789812770837_0001 (cit. on p. 10).

[36] Ming Li and Paul Vitányi. *An Introduction to Kolmogorov Complexity and Its Applications*. Springer, 2008, p. 790. ISBN: 978-0-387-33998-6. DOI: 10.1007/978-0-387-49820-1. arXiv: arXiv:1011.1669v3. (Cit. on pp. 11, 17).

[37] L A Levin. "Laws on the conservation (zero increase) of information, and questions on the foundations of probability theory." In: *Problemy Peredači Informacii* 10.3 (1974), pp. 30–35. ISSN: 0555-2923. (Cit. on pp. 12, 13).

[38] P Gač. "The symmetry of algorithmic information." In: *Dokl. Akad. Nauk SSSR* 218 (1974), pp. 1265–1267. ISSN: 0002-3264. (Cit. on pp. 12, 13).

[39] Manuel Blum. "A machine-independent theory of the complexity of recursive functions." In: *J. Assoc. Comput. Mach.* 14 (1967), pp. 322–336. ISSN: 0004-5411. DOI: 10.1145/321386.321395.
URL: https://doi.org/10.1145/321386.321395 (cit. on p. 12).

[40] M S Burgin. "Generalized Kolmogorov complexity and duality in computational theory." In: *Dokl. Akad. Nauk SSSR* 264.1 (1982), pp. 19–23. ISSN: 0002-3264. (Cit. on p. 12).

[41] Chris S Wallace and David M Boulton. "An information measure for classification." In: *The Computer Journal* 11.2 (1968), pp. 185–194. (Cit. on p. 12).

[42] Paul Vitányi and Ming Li. "Minimum description length induction, Bayesianism, and Kolmogorov complexity." In: *IEEE Transactions on information theory* 46.2 (2000), pp. 446–464. (Cit. on p. 12).

[43] Charles H Bennett. *Logical depth and physical complexity*. Citeseer, 1988. (Cit. on p. 12).

[44] Per Martin-Löf. "The definition of random sequences." In: *Information and control* 9.6 (1966), pp. 602–619. (Cit. on pp. 12, 13).

[45] R v Mises. "Grundlagen der Wahrscheinlichkeitsrechnung." In: *Mathematische Zeitschrift* 5.1-2 (1919), pp. 52–99. (Cit. on p. 12).

[46] Claus-Peter Schnorr. "A unified approach to the definition of random sequences." In: *Mathematical systems theory* 5.3 (1971), pp. 246–258. (Cit. on p. 13).

[47] Hector Zenil, Narsis A Kiani, and Jesper Tegnér. "Low-algorithmic-complexity entropy-deceiving graphs." In: *Phys. Rev. E* 96.1 (2017), pp. 012308, 12. ISSN: 2470-0045. DOI: 10.1103/physreve.96.012308.
URL: https://doi.org/10.1103/physreve.96.012308 (cit. on p. 13).

[48] Hector Zenil, Santiago Hernández-Orozco, Narsis A Kiani, Fernando Soler-Toscano, Antonio Rueda-Toicen, and Jesper Tegnér. "A decomposition method for global evaluation of Shannon entropy and local estimations of algorithmic complexity." In: *Entropy* 20.8 (2018), p. 605. (Cit. on pp. 13, 14, 32, 95, 105).

[49] Peter Bloem, Francisco Mota, Steven de Rooij, Luís Antunes, and Pieter Adriaans. "A Safe Approximation for Kolmogorov Complexity BT - Algorithmic Learning Theory." In: *International Conference on Algorithmic Learning Theory*. Ed. by Peter Auer, Alexander Clark, Thomas Zeugmann, and Sandra Zilles. Cham: Springer International Publishing, 2014, pp. 336–350. ISBN: 978-3-319-11662-4. (Cit. on pp. 13, 16).

[50] Hector Zenil, Liliana Badillo, Santiago Hernández-Orozco, and Francisco Hernández-Quiroz. "Coding-theorem like behaviour and emergence of the universal distribution from resource-bounded algorithmic probability." In: *International Journal of Parallel, Emergent and Distributed Systems* 34.2 (2019), pp. 161–180. (Cit. on p. 13).

[51] Milton Silva, Diogo Pratas, and Armando J Pinho. "Efficient DNA sequence compression with neural networks." In: *GigaScience* 9.11 (Nov. 2020). giaa119. ISSN: 2047-217X. DOI: 10.1093/gigascience/giaa119. eprint: https://academic.oup.com/gigascience/article-pdf/9/11/giaa119/34251844/giaa119.pdf. (Cit. on pp. 13, 26–28, 32–34, 103).

[52] Leonid A Levin. "Universal enumeration problems." In: *Problemy Peredači Informacii* 9.3 (1973), pp. 115–116. ISSN: 0555-2923. (Cit. on pp. 13, 104).

[53] Leonid A Levin. "Randomness conservation inequalities: information and independence in mathematical theories." In: *Inform. and Control* 61.1 (1984), pp. 15–37. ISSN: 0019-9958. DOI: 10.1016/S0019-9958(84)80060-1.
URL: https://doi.org/10.1016/S0019-9958(84)80060-1 (cit. on pp. 13, 104).

[54] Marcus Hutter. "The fastest and shortest algorithm for all well-defined problems." In: *Internat. J. Found. Comput. Sci.* 13.3 (2002), pp. 431–443. ISSN: 0129-0541. DOI: 10.1142/S0129054102001199.
URL: https://doi.org/10.1142/S0129054102001199 (cit. on pp. 14, 104).

[55] Marcus Hutter. *Universal artificial intelligence: Sequential decisions based on algorithmic probability*. Springer Science & Business Media, 2004. (Cit. on pp. 14, 104).

[56] Fernando Soler-Toscano, Hector Zenil, Jean-Paul Delahaye, and Nicolas Gauvrit. "Calculating Kolmogorov complexity from the output frequency distributions of small Turing machines." In: *PloS one* 9.5 (2014). (Cit. on pp. 14, 105).

[57] Charles H Bennett, Péter Gâcs, Ming Li, Paul Vitânyi, and Wojciech H Zurek. "Information distance." In: *IEEE Transactions on Information Theory* 44.4 (1998), pp. 1407–1423. ISSN: 00189448. DOI: 10.1109/18.681318. arXiv: 1006.3520. (Cit. on p. 14).

[58] Sebastiaan A Terwijn, Leen Torenvliet, and Paul Vitányi. "Nonapproximability of the normalized information distance." In: *Journal of Computer and System Sciences* 77.4 (2011), pp. 738–742. ISSN: 0022-0000. DOI: https://doi.org/10.1016/j.jcss.2010.06.018. (Cit. on p. 14).

[59] Gregory J Chaitin. "On the Length of Programs for Computing Finite Binary Sequences." In: *J. ACM* 13.4 (Oct. 1966), pp. 547–569. ISSN: 0004-5411. DOI: 10.1145/321356.321363. (Cit. on p. 14).

[60] Hector Zenil, Fernando Soler-Toscano, Kamaludin Dingle, and Ard A Louis. "Correlation of automorphism group size and topological properties with program-size complexity evaluations of graphs and complex networks." In: *Physica A: Statistical Mechanics and its Applications* 404 (2014), pp. 341–358. (Cit. on p. 15).

[61] Vera Kempe, Nicolas Gauvrit, and Douglas Forsyth. "Structure emerges faster during cultural transmission in children than in adults." In: *Cognition* 136 (2015), pp. 247–254. (Cit. on p. 15).

[62] Hector Zenil, Fernando Soler-Toscano, Jean-Paul Delahaye, and Nicolas Gauvrit. "Two-dimensional Kolmogorov complexity and an empirical validation of the Coding theorem method by compressibility." In: *PeerJ Computer Science* 1 (2015), e23. (Cit. on p. 15).

[63] Jorge Miguel Silva, Diogo Pratas, Rui Antunes, Sérgio Matos, and Armando J Pinho. "Automatic analysis of artistic paintings using information-based measures." In: *Pattern Recognition* 114 (2021), p. 107864. (Cit. on p. 15).

[64] Diogo Pratas. "Compression and analysis of genomic data." In: (2016). (Cit. on pp. 16, 17).

[65] R Cilibrasi and Paul Vitanyi. "Clustering by compression." In: *IEEE Transactions on Information Theory* 51.4 (Apr. 2005), pp. 1523–1545. ISSN: 0018-9448. DOI: 10.1109/TIT.2005.844059. (Cit. on p. 17).

[66] Manuel Cebrián, Manuel Alfonseca, and Alfonso Ortega. "Common Pitfalls Using the Normalized Compression Distance: What to Watch Out for in a Compressor." In: *Commun. Inf. Syst.* 5.4 (2005), pp. 367–384. (Cit. on p. 17).

[67] N Nikvand and Z Wang. "Generic image similarity based on Kolmogorov complexity." In: *2010 IEEE International Conference on Image Processing*. Sept. 2010, pp. 309–312. DOI: 10.1109/ICIP.2010.5653405. (Cit. on p. 17).

[68] Diogo Pratas and Armando J Pinho. "A Conditional Compression Distance that Unveils Insights of the Genomic Evolution." In: *2014 Data Compression Conference*. IEEE, Mar. 2014, pp. 421–421. ISBN: 978-1-4799-3882-7. DOI: 10.1109/DCC.2014.58. (Cit. on p. 17).

[69] Manuel Cebrián, Manuel Alfonseca, and Alfonso Ortega. "The normalized compression distance is resistant to noise." In: *IEEE Transactions on Information Theory* 53.5 (2007), pp. 1895–1900. (Cit. on p. 17).

[70] Diogo Pratas, Armando J Pinho, and Paulo Jorge S G Ferreira. "Pratas2016EfficientCO of Genomic Sequences." In: *2016 Data Compression Conference (DCC)* (2016), pp. 231–240. (Cit. on p. 17).

[71] J Ziv and N Merhav. "A measure of relative entropy between individual sequences with application to universal classification." In: *IEEE Transactions on Information Theory* 39.4 (July 1993), pp. 1270–1279. ISSN: 0018-9448. DOI: 10.1109/18.243444. (Cit. on p. 17).

[72] David Pereira Coutinho and Mário A T Figueiredo. "Text Classification Using Compression-Based Dissimilarity Measures." In: *International Journal of Pattern Recognition and Artificial Intelligence* 29.05 (2015), p. 1553004. DOI: 10.1142/S0218001415530043. (Cit. on p. 17).

[73] David Pereira Coutinho and Mário A T Figueiredo. "An Information Theoretic Approach to Text Sentiment Analysis." In: *ICPRAM*. 2013. (Cit. on p. 17).

[74] Sebastiaan A Terwijn, Leen Torenvliet, and Paul Vitányi. "Nonapproximability of the normalized information distance." In: *Journal of Computer and System Sciences* 77.4 (2011). JCSS IEEE AINA 2009, pp. 738–742. ISSN: 0022-0000. DOI: https://doi.org/10.1016/j.jcss.2010.06.018. (Cit. on p. 18).

[75] Yoshua Bengio, Aaron Courville, and Pascal Vincent. "Representation Learning: A Review and New Perspectives." In: (). (Cit. on p. 18).

[76] Michael I Jordan and Tom M Mitchell. "Machine learning: Trends, perspectives, and prospects." In: *Science* 349.6245 (2015), pp. 255–260. (Cit. on p. 18).

[77] Carl R Woese, Otto Kandler, and Mark L Wheelis. "Towards a natural system of organisms: proposal for the domains Archaea, Bacteria, and Eucarya." In: *Proceedings of the National Academy of Sciences* 87.12 (1990), pp. 4576–4579. (Cit. on p. 21).

[78] Sam Griffiths-Jones, Russell J Grocock, Stijn Van Dongen, Alex Bateman, and Anton J Enright. "miRBase: microRNA sequences, targets and gene nomenclature." In: *Nucleic acids research* 34.suppl_1 (2006), pp. D140–D144. (Cit. on p. 21).

[79] James L Bernat, Charles M Culver, and Bernard Gert. "On the definition and criterion of death." In: *Annals of Internal Medicine* 94.3 (1981), pp. 389–394. (Cit. on p. 21).

[80] Laura A Hug, Brett J Baker, Karthik Anantharaman, Christopher T Brown, Alexander J Probst, Cindy J Castelle, Cristina N Butterfield, Alex W Hernsdorf, Yuki Amano, Kotaro Ise, et al. "A new view of the tree of life." In: *Nature microbiology* 1.5 (2016), pp. 1–6. (Cit. on p. 22).

[81] James D Watson and Francis HC Crick. "Molecular structure of nucleic acids: a structure for deoxyribose nucleic acid." In: *Nature* 171.4356 (1953), pp. 737–738. (Cit. on p. 23).

[82] Erwin Chargaff. "Chemical specificity of nucleic acids and mechanism of their enzymatic degradation." In: *Experientia* 6.6 (1950), pp. 201–209. (Cit. on p. 23).

[83] Roger W Hendrix, Graham F Hatfull, Michael E Ford, Margaret CM Smith, and R Neil Burns. "Evolutionary relationships among diverse bacteriophages and prophages: all the world's a phage." In: *Horizontal gene transfer*. Elsevier, 2002, pp. 133–VI. (Cit. on p. 23).

[84] Nuala A O'Leary, Mathew W Wright, J Rodney Brister, Stacy Ciufo, Diana Haddad, Rich McVeigh, Bhanu Rajput, Barbara Robbertse, Brian Smith-White, Danso Ako-Adjei, et al. "Reference sequence (RefSeq) database at NCBI: current status, taxonomic expansion, and functional annotation." In: *Nucleic acids research* 44.D1 (2016), pp. D733–D745. (Cit. on p. 24).

[85] Robert A Edwards and Forest Rohwer. "Viral metagenomics." In: *Nature Reviews Microbiology* 3.6 (2005), pp. 504–510. (Cit. on p. 24).

[86] C Martin Lawrence, Smita Menon, Brian J Eilers, Brian Bothner, Reza Khayat, Trevor Douglas, and Mark J Young. "Structural and functional studies of archaeal viruses." In: *Journal of Biological Chemistry* 284.19 (2009), pp. 12599–12603. (Cit. on p. 24).

[87] Eugene V Koonin, Tatiana G Senkevich, and Valerian V Dolja. "The ancient Virus World and evolution of cells." In: *Biology direct* 1.1 (2006), p. 29. (Cit. on p. 24).

[88] Stephen Nayfach, Simon Roux, Rekha Seshadri, Daniel Udwary, Neha Varghese, Frederik Schulz, Dongying Wu, David Paez-Espino, I-Min Chen, Marcel Huntemann, et al. "A genomic catalog of Earth's microbiomes." In: *Nature biotechnology* 39.4 (2021), pp. 499–509. (Cit. on p. 24).

[89] Editorial. "Microbiology by numbers." In: *Nature Reviews Microbiology* 9 (9 2011), p. 628. ISSN: 1740-1534. DOI: 10.1038/nrmicro2644. (Cit. on p. 24).

[90] James H Strauss and Ellen G Strauss. "CHAPTER 1 - Overview of Viruses and Virus Infection." In: *Viruses and Human Disease (Second Edition)*. Ed. by James H Strauss and Ellen G Strauss. Second Edition. London: Academic Press, 2008, pp. 1–33. ISBN: 978-0-12-373741-0. DOI: https://doi.org/10.1016/B978-0-12-373741-0.50004-0. (Cit. on p. 24).

[91] Jack Lidmar, Leonid Mirny, and David R Nelson. "Virus shapes and buckling transitions in spherical shells." In: *Physical Review E* 68.5 (2003), p. 051910. (Cit. on p. 24).

[92] Graziano Vernizzi and Monica Olvera de la Cruz. "Faceting ionic shells into icosahedra via electrostatics." In: *Proceedings of the National Academy of Sciences* 104.47 (2007), pp. 18382–18386. (Cit. on p. 24).

[93] Antoni Luque and David Reguera. "The structure of elongated viral capsids." In: *Biophysical journal* 98.12 (2010), pp. 2993–3003. (Cit. on p. 24).

[94] Efthimia Mina Tsagris, Ángel Emilio Martínez de Alba, Mariyana Gozmanova, and Kriton Kalantidis. "Viroids." In: *Cellular microbiology* 10.11 (2008), pp. 2168–2179. (Cit. on p. 24).

[95] Mart Krupovic and Virginija Cvirkaite-Krupovic. "Virophages or satellite viruses?" In: *Nature Reviews Microbiology* 9.11 (2011), pp. 762–763. (Cit. on p. 24).

[96] Nigel J Dimmock, Andrew J Easton, and Keith N Leppard. *Introduction to modern virology*. John Wiley & Sons, 2016. ISBN: 9781405136457. (Cit. on p. 24).

[97] Selma Gago, Santiago F Elena, Ricardo Flores, and Rafael Sanjuán. "Extremely high mutation rate of a hammerhead viroid." In: *Science* 323.5919 (2009), pp. 1308–1308. (Cit. on p. 24).

[98] Diego Simón, Juan Cristina, and Héctor Musto. "Nucleotide composition and codon usage across viruses and their respective hosts." In: *Frontiers in Microbiology* 12 (2021). (Cit. on p. 24).

[99] David Baltimore. "Expression of animal virus genomes." In: *Bacteriological reviews* 35.3 (1971), pp. 235–241. (Cit. on p. 25).

[100] Kayla M Peck and Adam S Lauring. "Complexities of viral mutation rates." In: *Journal of virology* 92.14 (2018), e01031–17. (Cit. on p. 25).

[101] Jean-Michel Claverie, Hiroyuki Ogata, Stéphane Audic, Chantal Abergel, Karsten Suhre, and Pierre-Edouard Fournier. "Mimivirus and the emerging concept of "giant" virus." In: *Virus research* 117.1 (2006), pp. 133–144. (Cit. on p. 25).

[102] J-M Claverie, C Abergel, and H Ogata. "Mimivirus." In: *Lesser Known Large dsDNA Viruses*. Springer, 2009, pp. 89–121. (Cit. on p. 25).

[103] Jerome E Foster and Gustavo Fermin. "Chapter 4 - Origins and Evolution of Viruses." In: *Viruses*. Ed. by Paula Tennant, Gustavo Fermin, and Jerome E Foster. Academic Press, 2018, pp. 83–100. ISBN: 978-0-12-811257-1. DOI: https://doi.org/10.1016/B978-0-12-811257-1.00004-8. (Cit. on p. 25).

[104] Antonio Amorim, Filipe Pereira, Cíntia Alves, and Oscar García. "Species assignment in forensics and the challenge of hybrids." In: *Forensic Science International: Genetics* 48 (2020), p. 102333. (Cit. on pp. 25, 44).

[105] William Martin and Eugene V Koonin. "Introns and the origin of nucleus–cytosol compartmentalization." In: *Nature* 440.7080 (2006), pp. 41–45. (Cit. on p. 25).

[106] Thomas Cavalier-Smith. "Origin of the cell nucleus, mitosis and sex: roles of intracellular coevolution." In: *Biology direct* 5.1 (2010), p. 7. (Cit. on p. 25).

[107] Masaharu Takemura. "Medusavirus Ancestor in a Proto-Eukaryotic Cell: Updating the Hypothesis for the Viral Origin of the Nucleus." In: *Frontiers in Microbiology* 11 (2020), p. 2169. ISSN: 1664-302X. DOI: 10.3389/fmicb.2020.571831. (Cit. on p. 25).

[108] Mari Toppinen, Antti Sajantila, Diogo Pratas, Klaus Hedman, and Maria F Perdomo. "The Human Bone Marrow Is Host to the DNAs of Several Viruses." In: *Frontiers in Cellular and Infection Microbiology* 11 (2021). ISSN: 2235-2988. DOI: 10.3389/fcimb.2021.657245. (Cit. on p. 25).

[109] Mari Toppinen, Diogo Pratas, Elina Väisänen, Maria Söderlund-Venermo, Klaus Hedman, Maria F Perdomo, and Antti Sajantila. "The landscape of persistent human DNA viruses in femoral bone." In: *Forensic Science International: Genetics* 48 (2020), p. 102353. (Cit. on p. 25).

[110] Hiroshi Ikegaya and Hirotaro Iwase. "Trial for the geographical identification using JC viral genotyping in Japan." In: *Forensic science international* 139.2-3 (2004), pp. 169–172. (Cit. on p. 25).

[111] Hansjürgen T Agostini, Richard Yanagihara, Victor Davis, Caroline F Ryschkewitsch, and Gerald L Stoner. "Asian genotypes of JC virus in Native Americans and in a Pacific Island population: markers of viral evolution and human migration." In: *Proceedings of the National Academy of Sciences* 94.26 (1997), pp. 14542–14546. (Cit. on p. 25).

[112] Chie Sugimoto, Tadaichi Kitamura, Jing Guo, Mohammed N Al-Ahdal, Sergei N Shchelkunov, Berta Otova, Paul Ondrejka, Jean-Yves Chollet, Sayda El-Safi, Mohamed Ettayebi, et al. "Typing of urinary JC virus DNA offers a novel means of tracing human migrations." In: *Proceedings of the National Academy of Sciences* 94.17 (1997), pp. 9191–9196. (Cit. on p. 25).

[113] Chie Sugimoto, Masami Hasegawa, Huai-Ying Zheng, Vladimir Demenev, Yoshiharu Sekino, Kazuo Kojima, Takeo Honjo, Hiroshi Kida, Tapani Hovi, Timo Vesikari, et al. "JC virus strains indigenous to northeastern Siberians and Canadian Inuits are unique but evolutionally related to those distributed throughout Europe and Mediterranean areas." In: *Journal of Molecular Evolution* 55.3 (2002), pp. 322–335. (Cit. on p. 25).

[114] Diego Forni, Rachele Cagliani, Mario Clerici, Uberto Pozzoli, and Manuela Sironi. "You will never walk alone: codispersal of JC polyomavirus with human populations." In: *Molecular biology and evolution* 37.2 (2020), pp. 442–454. (Cit. on p. 25).

[115] Morteza Hosseini, Diogo Pratas, and Armando J Pinho. "On the role of inverted repeats in DNA sequence similarity." In: *International Conference on Practical Applications of Computational Biology & Bioinformatics*. Springer. 2017, pp. 228–236. (Cit. on p. 25).

[116] Mari Toppinen. "Parvoviral genomes in human soft tissues and bones over decades." PhD thesis. Helsingin yliopisto, 2021. (Cit. on pp. 25, 26).

[117] Irina Voineagu, Vidhya Narayanan, Kirill S Lobachev, and Sergei M Mirkin. "Replication stalling at unstable inverted repeats: interplay between DNA hairpins and fork stabilizing proteins." In: *Proceedings of the National Academy of Sciences* 105.29 (2008), pp. 9936–9941. (Cit. on p. 25).

[118] J John Bissler. "DNA inverted repeats and human disease." In: *Front Biosci* 3.4 (1998), pp. d408–d418. (Cit. on p. 26).

[119] Ching-Tai Lin, Wei-Hsin Lin, Yi Lisa Lyu, and Jacqueline Whang-Peng. "Inverted repeats as genetic elements for promoting DNA inverted duplication: implications in gene amplification." In: *Nucleic Acids Research* 29.17 (2001), pp. 3529–3538. (Cit. on p. 26).

[120] John F Atkins, Gary Loughran, Pramod R Bhatt, Andrew E Firth, and Pavel V Baranov. "Ribosomal frameshifting and transcriptional slippage: From genetic steganography and cryptography to adventitious use." In: *Nucleic acids research* 44.15 (2016), pp. 7007–7078. (Cit. on p. 26).

[121] Olivier Namy, Stephen J Moran, David I Stuart, Robert JC Gilbert, and Ian Brierley. "A mechanical explanation of RNA pseudoknot function in programmed ribosomal frameshifting." In: *Nature* 441.7090 (2006), pp. 244–247. (Cit. on p. 26).

[122] Martin Mikl, Yitzhak Pilpel, and Eran Segal. "High-throughput interrogation of programmed ribosomal frameshifting in human cells." In: *Nature communications* 11.1 (2020), pp. 1–18. (Cit. on p. 26).

[123] Susan F Cotmore and Peter Tattersall. "Parvoviruses: small does not mean simple." In: *Annual review of virology* 1 (2014), pp. 517–537. (Cit. on p. 26).

[124] Ziying Yan, Roman Zak, Yulong Zhang, and John F Engelhardt. "Inverted terminal repeat sequences are important for intermolecular recombination and circularization of adeno-associated virus genomes." In: *Journal of virology* 79.1 (2005), pp. 364–379. (Cit. on p. 26).

[125] Deborah Byrne, Renata Grzela, Audrey Lartigue, Stéphane Audic, Sabine Chenivesse, Stéphanie Encinas, Jean-Michel Claverie, and Chantal Abergel. "The polyadenylation site of Mimivirus transcripts obeys a stringent 'hairpin rule'." In: *Genome research* 19.7 (2009), pp. 1233–1242. (Cit. on p. 26).

[126] Jean-Michel Claverie and Chantal Abergel. "Mimivirus and its virophage." In: *Annual review of genetics* 43 (2009), pp. 49–66. (Cit. on p. 26).

[127] Edward R Dougherty and Ilya Shmulevich. *Genomic signal processing and statistics*. Vol. 2. Hindawi Publishing Corporation, 2005. (Cit. on p. 26).

[128] Jean Gailly and Mark Adler. *The gzip home page*. http://www.gzip.org/. accessed May 16, 2020. (Cit. on pp. 26, 58).

[129] *bzip2*. http://www.bzip.org/. accessed May 16, 2020. (Cit. on pp. 26, 58).

[130] Igor Pavlov. *7-Zip*. https://www.7-zip.org/. accessed May 16, 2020. (Cit. on pp. 26, 58).

[131] Stéphane Grumbach and Fariza Tahi. "Compression of DNA sequences." In: *[Proceedings] DCC93: Data Compression Conference*. IEEE. 1993, pp. 340–350. (Cit. on p. 26).

[132] Loren H Rieseberg. "Chromosomal rearrangements and speciation." In: *Trends in ecology & evolution* 16.7 (2001), pp. 351–358. (Cit. on p. 26).

[133] G Shirleen Roeder and Gerald R Fink. "DNA rearrangements associated with a transposable element in yeast." In: *Cell* 21.1 (1980), pp. 239–249. (Cit. on p. 26).

[134] Mikel Hernaez, Dmitri Pavlichin, Tsachy Weissman, and Idoia Ochoa. "Genomic data compression." In: *Annual Review of Biomedical Data Science* 2 (2019), pp. 19–37. (Cit. on p. 26).

[135] Stéphane Grumbach and Fariza Tahi. "A new challenge for compression algorithms: genetic sequences." In: *Information Processing & Management* 30.6 (1994), pp. 875–886. (Cit. on p. 26).

[136] Giovanni Manzini and Marcella Rastero. "A simple and fast DNA compressor." In: *Software: Practice and Experience* 34.14 (2004), pp. 1397–1411. (Cit. on p. 26).

[137] Neva Cherniavsky and Richard Ladner. "Grammar-based compression of DNA sequences." In: *DIMACS Working Group on The Burrows-Wheeler Transform* 21 (2004). (Cit. on p. 26).

[138] Gergely Korodi and Ioan Tabus. "An efficient normalized maximum likelihood algorithm for DNA sequence compression." In: *ACM Transactions on Information Systems (TOIS)* 23.1 (2005), pp. 3–34. (Cit. on p. 26).

[139] Gregory Vey. "Differential direct coding: a compression algorithm for nucleotide sequence data." In: *Database* 2009 (2009). (Cit. on p. 26).

[140] Kamta Nath Mishra, Anupam Aaggarwal, Edries Abdelhadi, and DPC Srivastava. "An efficient horizontal and vertical method for online DNA sequence compression." In: *International Journal of Computer Applications* 3.1 (2010), pp. 39–46. (Cit. on p. 26).

[141] P Raja Rajeswari and Allam Apparao. "GENBIT Compress-Algorithm for repetitive and non repetitive DNA sequences." In: *International Journal of Computer Science and Information Technology* 2 (2010), pp. 25–29. (Cit. on p. 26).

[142] Ashutosh Gupta and Suneeta Agarwal. "A novel approach for compressing DNA sequences using semi-statistical compressor." In: *International Journal of Computers and Applications* 33.3 (2011), pp. 245–251. (Cit. on p. 26).

[143] Zexuan Zhu, Jiarui Zhou, Zhen Ji, and Yu-Hui Shi. "DNA sequence compression using adaptive particle swarm optimization-based memetic algorithm." In: *IEEE Transactions on Evolutionary Computation* 15.5 (2011), pp. 643–658. (Cit. on p. 26).

[144] Armando J Pinho, Paulo Jorge S G Ferreira, António JR Neves, and Carlos AC Bastos. "On the representability of complete genomes by multiple competing finite-context (Markov) models." In: *PloS one* 6.6 (2011), e21588. (Cit. on pp. 26, 82).

[145] Diogo Pratas, Armando J Pinho, and Paulo Jorge S G Ferreira. "Efficient compression of genomic sequences." In: *2016 Data Compression Conference (DCC)*. IEEE. 2016, pp. 231–240. (Cit. on p. 26).

[146] Kirill Kryukov, Mahoko Takahashi Ueda, So Nakagawa, and Tadashi Imanishi. "Nucleotide Archival Format (NAF) enables efficient lossless reference-free compression of DNA sequences." In: *Bioinformatics* 35.19 (2019), pp. 3826–3828. (Cit. on p. 26).

[147] Kirill Kryukov. *Kirillkryukov/NAF: Nucleotide archival format - compressed file format for DNA/RNA/protein sequences*. accessed May 5, 2022.
URL: https://github.com/KirillKryukov/naf (cit. on p. 26).

[148] Szymon Grabowski and Tomasz M Kowalski. "MBGC: Multiple Bacteria Genome Compressor." In: *GigaScience* 11 (2022). (Cit. on p. 26).

[149] Byron Knoll. *Byronknoll/cmix: Cmix is a lossless data compression program aimed at optimizing compression ratio at the cost of high CPU/memory usage.Byron Knoll*. accessed May 5, 2022.
URL: https://github.com/byronknoll/cmix (cit. on pp. 26, 32, 33).

[150] Minh Duc Cao, Trevor I Dix, Lloyd Allison, and Chris Mears. "A simple statistical algorithm for biological sequence compression." In: *2007 Data Compression Conference (DCC'07)*. IEEE. 2007, pp. 43–52. (Cit. on pp. 26, 27).

[151] Diogo Pratas, Morteza Hosseini, Jorge M Silva, and Armando J Pinho. "A reference-free lossless compression algorithm for DNA sequences using a competitive prediction of two classes of weighted models." In: *Entropy* 21.11 (2019), p. 1074. (Cit. on pp. 26, 27).

[152] Kirill Kryukov, Mahoko Takahashi Ueda, So Nakagawa, and Tadashi Imanishi. "Sequence Compression Benchmark (SCB) database—A comprehensive evaluation of reference-free compressors for FASTA-formatted sequences." In: *GigaScience* 9.7 (2020), giaa072. (Cit. on p. 26).

[153] Byron Knoll and Nando de Freitas. "A machine learning perspective on predictive coding with PAQ8." In: *2012 Data Compression Conference*. IEEE. 2012, pp. 377–386. (Cit. on pp. 27, 59, 103).

[154] Avatar Johannes Buchner. *PAQ*. https://github.com/JohannesBuchner/paq/. accessed May 16, 2020. (Cit. on pp. 27, 59).

[155] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory." In: *Neural computation* 9.8 (1997), pp. 1735–1780. (Cit. on p. 27).

[156] Diogo Pratas, Morteza Hosseini, and Armando J Pinho. "GeCo2: An optimized tool for lossless compression and analysis of DNA sequences." In: *International Conference on Practical Applications of Computational Biology & Bioinformatics*. Springer. 2019, pp. 137–145. (Cit. on p. 27).

[157] Diogo Pratas and Armando J Pinho. "On the approximation of the Kolmogorov complexity for DNA sequences." In: *Iberian Conference on Pattern Recognition and Image Analysis*. Springer. 2017, pp. 259–266. (Cit. on pp. 28, 31, 57).

[158] Armando J Pinho, Sara P Garcia, Diogo Pratas, and Paulo Jorge S G Ferreira. "DNA sequences at a glance." In: *PloS one* 8.11 (2013), e79922. (Cit. on p. 28).

[159] Armando J Pinho, Diogo Pratas, Paulo Jorge S G Ferreira, and Sara P Garcia. "Symbolic to numerical conversion of DNA sequences using finite-context models." In: *2011 19th European Signal Processing Conference*. IEEE. 2011, pp. 2024–2028. (Cit. on p. 28).

[160] João R Almeida, Armando J Pinho, José L Oliveira, Olga Fajarda, and Diogo Pratas. "GTO: a toolkit to unify pipelines in genomic and proteomic research." In: *SoftwareX* 12 (2020), p. 100535. (Cit. on pp. 28, 32).

[161] Jonathan Romiguier, Vincent Ranwez, Emmanuel JP Douzery, and Nicolas Galtier. "Contrasting GC-content dynamics across 33 mammalian genomes: relationship with life-history traits and chromosome sizes." In: *Genome research* 20.8 (2010), pp. 1001–1009. (Cit. on p. 29).

[162] Laurent Duret and Nicolas Galtier. "Biased gene conversion and the evolution of mammalian genomic landscapes." In: *Annual review of genomics and human genetics* 10 (2009), pp. 285–311. (Cit. on p. 29).

[163] Peter Simmonds and M Azim Ansari. "Extensive C-> U transition biases in the genomes of a wide range of mammalian RNA viruses; potential associations with transcriptional mutations, damage-or host-mediated editing of viral RNA." In: *PLoS pathogens* 17.6 (2021), e1009596. (Cit. on pp. 29, 37).

[164] Peter Yakovchuk, Ekaterina Protozanova, and Maxim D Frank-Kamenetskii. "Base-stacking and base-pairing contributions into thermal stability of the DNA double helix." In: *Nucleic acids research* 34.2 (2006), pp. 564–574. (Cit. on p. 29).

[165] Han Chen and Chris-Kriton Skylaris. "Analysis of DNA interactions and GC content with energy decomposition in large-scale quantum mechanical calculations." In: *Physical Chemistry Chemical Physics* 23.14 (2021), pp. 8891–8899. (Cit. on p. 29).

[166] Geoffrey J McLachlan. *Discriminant analysis and statistical pattern recognition*. Vol. 544. John Wiley & Sons, 2004. (Cit. on pp. 29, 45).

[167] Irina Rish et al. "An empirical study of the naive Bayes classifier." In: *IJCAI 2001 workshop on empirical methods in artificial intelligence*. Vol. 3. 2001, pp. 41–46. (Cit. on pp. 29, 45).

[168] Gongde Guo, Hui Wang, David Bell, Yaxin Bi, and Kieran Greer. "KNN Model-Based Approach in Classification." In: *Springer Berlin Heidelberg* (2003). Ed. by Robert Meersman, Zahir Tari, and Douglas C Schmidt, pp. 986–996. DOI: 10.1007/978-3-540-39964-3_62. (Cit. on pp. 29, 45).

[169] Nello Cristianini, John Shawe-Taylor, et al. *An introduction to support vector machines and other kernel-based learning methods*. Cambridge university press, 2000. (Cit. on pp. 29, 30, 46).

[170] Tianqi Chen and Carlos Guestrin. "XGBoost: A Scalable Tree Boosting System." In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '16. San Francisco, California, USA: ACM, 2016, pp. 785–794. ISBN: 978-1-4503-4232-2. DOI: 10.1145/2939672.2939785. (Cit. on pp. 29, 30, 46, 74).

[171] Linda Stern, Lloyd Allison, Ross L Coppel, and Trevor I Dix. "Discovering patterns in Plasmodium falciparum genomic DNA." In: *Molecular and Biochemical Parasitology* 118.2 (2001), pp. 175–186. (Cit. on p. 30).

[172] Minh Duc Cao, Trevor I Dix, and Lloyd Allison. "A genome alignment algorithm based on compression." In: *BMC bioinformatics* 11.1 (2010), pp. 1–16. (Cit. on p. 30).

[173] Morihiro Hayashida and Tatsuya Akutsu. "Comparing biological networks via graph compression." In: *BMC systems biology*. Vol. 4. BioMed Central. 2010, pp. 1–11. (Cit. on p. 31).

[174] Robert Paul Bywater. "Prediction of protein structural features from sequence data based on Shannon entropy and Kolmogorov complexity." In: *PloS one* 10.4 (2015), e0119306. (Cit. on p. 31).

[175] Diogo Pratas and Armando J Pinho. "Metagenomic composition analysis of sedimentary ancient DNA from the Isle of Wight." In: *2018 26th European Signal Processing Conference (EUSIPCO)*. IEEE. 2018, pp. 1177–1181. (Cit. on p. 31).

[176] Morteza Hosseini, Diogo Pratas, Burkhard Morgenstern, and Armando J Pinho. "Smash++: an alignment-free and memory-efficient tool to find genomic rearrangements." In: *GigaScience* 9.5 (2020), giaa048. (Cit. on p. 31).

[177] Jonathan Kans. *Entrez direct: E-utilities on the UNIX command line*. National Center for Biotechnology Information (US), 2020. (Cit. on p. 32).

[178] Matt Mahoney. *Data compression programs*. Ed. by Florida Tech Department of Computer Sciences. 2009. (Cit. on pp. 32, 33).

[179] Rafael Sanjuán and Pilar Domingo-Calap. "Mechanisms of viral mutation." In: *Cellular and molecular life sciences* 73.23 (2016), pp. 4433–4448. (Cit. on p. 36).

[180] Brian WJ Mahy. "The evolution and emergence of RNA viruses." In: *Emerging infectious diseases* 16.5 (2010), p. 899. (Cit. on p. 37).

[181] Peter Simmonds. "Rampant C→ U hypermutation in the genomes of SARS-CoV-2 and other coronaviruses: causes and consequences for their short-and long-term evolutionary trajectories." In: *Msphere* 5.3 (2020), e00408–20. (Cit. on p. 37).

[182] David Prangishvili, Elena Rensen, Tomohiro Mochizuki, Mart Krupovic, et al. "ICTV virus taxonomy profile: Tristromaviridae." In: *Journal of General Virology* 100.2 (2019), pp. 135–136. (Cit. on p. 38).

[183] Mart Krupovic, Jens H Kuhn, Fengbin Wang, Diana P Baquero, Valerian V Dolja, Edward H Egelman, David Prangishvili, and Eugene V Koonin. "Adnaviria: a new realm for archaeal filamentous viruses with linear A-form double-stranded DNA genomes." In: *Journal of Virology* (2021), JVI–00673. (Cit. on p. 38).

[184] Mart Krupovic, Virginija Cvirkaite-Krupovic, Jaime Iranzo, David Prangishvili, and Eugene V Koonin. "Viruses of archaea: structural, functional, environmental and evolutionary genomics." In: *Virus research* 244 (2018), pp. 181–193. (Cit. on p. 39).

[185] María A Ayllón, Massimo Turina, Jiatao Xie, Luca Nerva, Shin-Yi Lee Marzano, Livia Donaire, Daohong Jiang, and ICTV Report Consortium. "ICTV virus taxonomy profile: Botourmiaviridae." In: *The Journal of general virology* 101.5 (2020), p. 454. (Cit. on p. 39).

[186] Keith W Savin, Benjamin G Cocks, Frank Wong, Tim Sawbridge, Noel Cogan, David Savage, and Simone Warner. "A neurotropic herpesvirus infecting the gastropod, abalone, shares ancestry with oyster herpesvirus and a herpesvirus associated with the amphioxus genome." In: *Virology journal* 7.1 (2010), pp. 1–9. (Cit. on p. 39).

[187] Andrew MQ King, Elliot Lefkowitz, Michael J Adams, and Eric B Carstens. *Virus taxonomy: ninth report of the International Committee on Taxonomy of Viruses*. Vol. 9. Elsevier, 2011. (Cit. on p. 39).

[188] Lari Pyöriä, Maija Jokinen, Mari Toppinen, Henri Salminen, Tytti Vuorinen, Veijo Hukkanen, Constanze Schmotz, Endrit Elbasani, Päivi M Ojala, Klaus Hedman, et al. "HERQ-9 is a new multiplex PCR for differentiation and quantification of all nine human herpesviruses." In: *Msphere* 5.3 (2020), e00265–20. (Cit. on p. 39).

[189] Joel D Baines and Philip E Pellett. "Genetic comparison of human alphaherpesvirus genomes." In: *Human herpesviruses: biology, therapy, and immunoprophylaxis* (2007). (Cit. on p. 40).

[190] Xiaoxi Liu, Shunichi Kosugi, Rie Koide, Yoshiki Kawamura, Jumpei Ito, Hiroki Miura, Nana Matoba, Motomichi Matsuzaki, Masashi Fujita, Anselmo Jiro Kamada, et al. "Endogenization and excision of human herpesvirus 6 in human genomes." In: *PLoS Genetics* 16.8 (2020), e1008915. (Cit. on p. 41).

[191] Ramesh Rajaby, Yi Zhou, Yifan Meng, Xi Zeng, Guoliang Li, Peng Wu, and Wing-Kin Sung. "SurVirus: a repeat-aware virus integration caller." In: *Nucleic acids research* 49.6 (2021), e33–e33. (Cit. on p. 41).

[192] Giulia Aimola, Georg Beythien, Amr Aswad, and Benedikt B Kaufer. "Current understanding of human herpesvirus 6 (HHV-6) chromosomal integration." In: *Antiviral research* 176 (2020), p. 104720. (Cit. on p. 41).

[193] JK Choudhari, J Choubey, MK Verma, T Chatterjee, and BP Sahariah. "Metagenomics: the boon for microbial world knowledge and current challenges." In: *Bioinformatics*. Elsevier, 2022, pp. 159–175. (Cit. on p. 44).

[194] Michael Vilsker, Yumma Moosa, Sam Nooij, Vagner Fonseca, Yoika Ghysens, Korneel Dumon, Raf Pauwels, Luiz Carlos Alcantara, Ewout Vanden Eynden, Anne-Mieke Vandamme, Koen Deforche, and Tulio de Oliveira. "Genome Detective: an automated system for virus identification from high-throughput sequencing data." In: *Bioinformatics* 35.5 (2019), pp. 871–873. DOI: 10.1093/bioinformatics/bty695. (Cit. on p. 44).

[195] Shifu Chen, Changshou He, Yingqiang Li, Zhicheng Li, and E Melancon III Charles. "A Computational Toolset for Rapid Identification of SARS-CoV-2, other Viruses, and Microorganisms from Sequencing Data." In: *Briefings in Bioinformatics* 22.2 (2021), pp. 924–935. DOI: 10.1101/2020.05.12.092163. (Cit. on p. 44).

[196] Irina Abnizova, Steven Leonard, Tom Skelly, Andy Brown, David Jackson, Marina Gourtovaia, Guoying Qi, Rene Te Boekhorst, Nadeem Faruque, Kevin Lewis, and Tony Cox. "Analysis of context-dependent errors for illumina sequencing." In: *J Bioinform Comput Biol* 10.2 (2012). DOI: 10.1142/S0219720012410053. (Cit. on p. 44).

[197] Rene Te Boekhorst, F M Naumenko, N G Orlova, Elvira Rasimovna Galieva, A M Spitsina, Irina Chadaeva, Yuriy Orlov, and Irina Abnizova. "Computational problems of analysis of short next generation sequencing reads." In: *Vavilov Journal of Genetics and Breeding* 20.6 (2016), pp. 746–755. DOI: 10.18699/VJ16.191. (Cit. on p. 44).

[198] Michał Karlicki, Stanisław Antonowicz, and Anna Karnkowska. "Tiara: deep learning-based classification system for eukaryotic sequences." In: *Bioinformatics* 38.2 (2022), pp. 344–350. (Cit. on p. 44).

[199] Qian Zhang, Se-Ran Jun, Michael Leuze, David Ussery, and Intawat Nookaew. "Viral phylogenomics using an alignment-free method: A three-step approach to determine optimal length of k-mer." In: *Scientific reports* 7.1 (2017), pp. 1–13. (Cit. on pp. 44, 45).

[200] Burkhard Morgenstern. "Sequence comparison without alignment: The SpaM approaches." In: *Multiple Sequence Alignment*. Springer, 2021, pp. 121–134. (Cit. on p. 45).

[201] Thomas Dencker, Chris-André Leimeister, Michael Gerth, Christoph Bleidorn, Sagi Snir, and Burkhard Morgenstern. "'Multi-SpaM': a maximum-likelihood approach to phylogeny reconstruction using multiple spaced-word matches and quartet trees." In: *NAR Genomics and Bioinformatics* 2.1 (Oct. 2019). lqz013. ISSN: 2631-9268. DOI: 10.1093/nargab/lqz013. eprint: https://academic.oup.com/nargab/article-pdf/2/1/lqz013/34054190/lqz013.pdf. (Cit. on p. 45).

[202] Benjamin J Garcia, Ramanuja Simha, Michael Garvin, Anna Furches, Piet Jones, Joao GFM Gazolla, P Doug Hyatt, Christopher W Schadt, Dale Pelletier, and Daniel Jacobson. "A k-mer based approach for classifying viruses without taxonomy identifies viral associations in human autism and plant microbiomes." In: *Computational and structural biotechnology journal* 19 (2021), pp. 5911–5919. (Cit. on p. 45).

[203] Lily He, Siyang Sun, Qianyue Zhang, Xiaona Bao, and Peter K Li. "Alignment-free sequence comparison for virus genomes based on location correlation coefficient." In: *Infection, Genetics and Evolution* 96 (2021), p. 105106. (Cit. on p. 45).

[204] Gordon K Smyth. "Statistical applications in genetics and molecular biology." In: *Linear models and empirical Bayes methods for assessing differential expression in microarray experiments* (2004). (Cit. on p. 45).

[205] Jennifer Lu and Steven L Salzberg. "Removing contaminants from databases of draft genomes." In: *PLoS computational biology* 14.6 (2018), e1006277. (Cit. on p. 48).

[206] Robert W Weisberg. *Creativity: Understanding innovation in problem solving, science, invention, and the arts*. John Wiley & Sons, 2006. (Cit. on p. 55).

[207] Aaron Hertzmann. "Can computers create art?" In: *Arts*. Vol. 7. 2. MDPI. 2018, p. 18. (Cit. on p. 55).

[208] Fahad Shahbaz Khan, Shida Beigpour, Joost Van de Weijer, and Michael Felsberg. "Painting-91: a large scale database for computational painting categorization." In: *Machine vision and applications* 25.6 (2014), pp. 1385–1397. (Cit. on pp. 55, 58, 60, 63).

[209] Siwei Lyu, Daniel Rockmore, and Hany Farid. "A digital technique for art authentication." In: *Proceedings of the National Academy of Sciences* 101.49 (2004), pp. 17006–17010. (Cit. on p. 55).

[210] Daniel Kim, Seung-Woo Son, and Hawoong Jeong. "Large-scale quantitative analysis of painting arts." In: *Scientific reports* 4.1 (2014), pp. 1–7. (Cit. on pp. 55, 56, 65, 67, 73).

[211] Hai Zhang, Stefano Sfarra, Karan Saluja, Jeroen Peeters, Julien Fleuret, Yuxia Duan, Henrique Fernandes, Nicolas Avdelidis, Clemente Ibarra-Castanedo, and Xavier Maldague. "Non-destructive investigation of paintings on canvas by continuous wave terahertz imaging and flash thermography." In: *Journal of Nondestructive Evaluation* 36.2 (2017), pp. 1–12. (Cit. on p. 55).

[212] Richard P Taylor, Adam P Micolich, and David Jonas. "Fractal analysis of Pollock's drip paintings." In: *Nature* 399.6735 (1999), pp. 422–422. (Cit. on p. 55).

[213] C Richard Johnson, Ella Hendriks, Igor J Berezhnoy, Eugene Brevdo, Shannon M Hughes, Ingrid Daubechies, Jia Li, Eric Postma, and James Z Wang. "Image processing for artist identification." In: *IEEE Signal Processing Magazine* 25.4 (2008), pp. 37–48. (Cit. on p. 56).

[214] Jia Li and James Ze Wang. "Studying digital imagery of ancient paintings by mixtures of stochastic models." In: *IEEE transactions on image processing* 13.3 (2004), pp. 340–353. (Cit. on p. 56).

[215] Marco Bressan, Claudio Cifarelli, and Florent Perronnin. "An analysis of the relationship between painters based on their work." In: *2008 15th IEEE International Conference on Image Processing*. IEEE. 2008, pp. 113–116. (Cit. on p. 56).

[216] Bruno A Olshausen and Michael R DeWeese. "The statistics of style." In: *Nature* 463.7284 (2010), pp. 1027–1028. (Cit. on p. 56).

[217] James M Hughes, Daniel J Graham, and Daniel N Rockmore. "Quantification of artistic style through sparse coding analysis in the drawings of Pieter Bruegel the Elder." In: *Proceedings of the National Academy of Sciences* 107.4 (2010), pp. 1279–1283. (Cit. on p. 56).

[218] David G Stork and Yasuo Furuichi. "Image analysis of paintings by computer graphics synthesis: an investigation of the illumination in Georges de la Tour's Christ in the carpenter's studio." In: *Computer image analysis in the study of art*. Vol. 6810. SPIE. 2008, pp. 176–187. (Cit. on p. 56).

[219] Martin Lettner and Robert Sablatnig. "Estimating the original drawing trace of painted strokes." In: *Computer image analysis in the study of art*. Vol. 6810. SPIE. 2008, pp. 113–122. (Cit. on p. 56).

[220] Morteza Shahram, David G Stork, and David Donoho. "Recovering layers of brush strokes through statistical analysis of color and shape: an application to van Gogh's" Self portrait with grey felt hat"." In: *Computer image analysis in the study of art*. Vol. 6810. SPIE. 2008, pp. 123–130. (Cit. on p. 56).

[221] S Blair Hedges. "Image analysis of Renaissance copperplate prints." In: *Computer image analysis in the study of art*. Vol. 6810. SPIE. 2008, pp. 82–101. (Cit. on p. 56).

[222] Vladimir M Petrov. "Entropy and stability in painting: An information approach to the mechanisms of artistic creativity." In: *Leonardo* 35.2 (2002), pp. 197–202. (Cit. on p. 56).

[223] António M Lopes and J A Tenreiro Machado. "Dynamics of the N-link pendulum: a fractional perspective." In: *Internat. J. Control* 90.6 (2017), pp. 1192–1200. ISSN: 0020-7179. DOI: 10.1080/00207179.2015.1126677.
URL: https://doi.org/10.1080/00207179.2015.1126677 (cit. on p. 56).

[224] Kuan-Chuan Peng and Tsuhan Chen. "Cross-layer features in convolutional neural networks for generic classification tasks." In: *2015 IEEE International Conference on Image Processing (ICIP)*. IEEE. 2015, pp. 3057–3061. (Cit. on p. 56).

[225] Hui Mao, Ming Cheung, and James She. "Deepart: Learning joint representations of visual arts." In: *Proceedings of the 25th ACM international conference on Multimedia*. 2017, pp. 1183–1191. (Cit. on pp. 56, 74).

[226] W Chu and Y Wu. "Image Style Classification Based on Learnt Deep Correlation Features." In: *IEEE Transactions on Multimedia* 20.9 (2018), pp. 2491–2502. (Cit. on pp. 56, 74).

[227] Joost Smiers. *Arts under pressure: promoting cultural diversity in the age of globalization*. Zed Books, 2003. (Cit. on p. 56).

[228] Paulo Jorge S G Ferreira and Armando J Pinho. "A method to detect repeated unknown patterns in an image." In: *International Conference Image Analysis and Recognition*. Springer. 2014, pp. 12–19. (Cit. on p. 56).

[229] Armando J Pinho and Paulo Jorge S G Ferreira. "Finding unknown repeated patterns in images." In: *2011 19th European Signal Processing Conference*. IEEE. 2011, pp. 584–588. (Cit. on p. 56).

[230] Diogo Pratas and Armando J Pinho. "On the detection of unknown locally repeating patterns in images." In: *International Conference Image Analysis and Recognition*. Springer. 2012, pp. 158–165. (Cit. on pp. 56, 122).

[231] Robert R Sokal. "A statistical method for evaluating systematic relationships." In: *Univ. Kansas, Sci. Bull.* 38 (1958), pp. 1409–1438. (Cit. on p. 57).

[232] Joseph B Kruskal. "On the shortest spanning subtree of a graph and the traveling salesman problem." In: *Proceedings of the American Mathematical society* 7.1 (1956), pp. 48–50. (Cit. on pp. 57, 71, 170).

[233] Stuart Lloyd. "Least squares quantization in PCM." In: *IEEE transactions on information theory* 28.2 (1982), pp. 129–137. (Cit. on p. 58).

[234] David S Taubman and Michael W Marcellin. "JPEG2000: Image compression fundamentals." In: *Standards and Practice* 11.2 (2002). (Cit. on p. 58).

[235] Lasse Collin. *XZ Utils*. https://tukaani.org/xz/. accessed May 16, 2020. (Cit. on p. 58).

[236] Morteza Hosseini, Diogo Pratas, and Armando J Pinho. "AC: A compression tool for amino acid sequences." In: *Interdisciplinary Sciences: Computational Life Sciences* 11.1 (2019), pp. 68–76. (Cit. on p. 58).

[237] John Cleary and Ian Witten. "Data compression using adaptive coding and partial string matching." In: *IEEE transactions on Communications* 32.4 (1984), pp. 396–402. (Cit. on p. 58).

[238] Matt Mahoney. *Data Compression Programs*. http://mattmahoney.net/dc/. accessed January 21, 2022. (Cit. on p. 58).

[239] Matthew V Mahoney. *Adaptive weighing of context models for lossless data compression*. Tech. rep. Florida Institute of Technology CS Department of the W University Blvd, 2005. (Cit. on pp. 59, 103).

[240] Jorma Rissanen and Glen G Langdon. "Arithmetic coding." In: *IBM Journal of research and development* 23.2 (1979), pp. 149–162. (Cit. on p. 59).

[241] Alistair Moffat, Radford M Neal, and Ian H Witten. "Arithmetic coding revisited." In: *ACM Transactions on Information Systems (TOIS)* 16.3 (1998), pp. 256–294. (Cit. on p. 59).

[242] *Best Artworks of All Time*. https://www.kaggle.com/ikarus777/best-artworks-of-all-time/data. accessed May 18, 2020. (Cit. on p. 60).

[243] *Diabetic Retinopathy Detection*. https://www.kaggle.com/c/diabetic-retinopathy-detection/overview. accessed May 18, 2020. (Cit. on p. 60).

[244] X Wang, Y Peng, L Lu, Z Lu, M Bagheri, and RM Summers. "ChestX-Ray8: Hospital-scale Chest X-ray Database and Benchmarks on Weakly-Supervised Classification and Localization of Common Thorax Diseases." In: *IEEE CVPR*. 2017, pp. 3462–3471. (Cit. on p. 60).

[245] *COCO - Common Objects in Context*. http://cocodataset.org/#download. accessed May 18, 2020. (Cit. on p. 60).

[246] Cecile Shapiro et al. "Abstract Expressionism: The politics of apolitical painting." In: *Prospects* 3 (1978), pp. 175–214. (Cit. on p. 65).

[247] H Rosenberg. *The Tradition Of The New*. Hachette Books, 1994. ISBN: 9780306805967. (Cit. on p. 65).

[248] Laura Garrard. *Colourfield painting: Minimal, Cool, Hard Edge, Serial and Post-painterly Abstract Art from the Sixties to the present*. Crescent Moon Publishing, 2007. (Cit. on p. 65).

[249] David Hockney. *Secret knowledge: Rediscovering the lost techniques of the old masters*. Thames & Hudson London, 2001. (Cit. on p. 72).

[250] Saboya Yang, Gene Cheung, Patrick Le Callet, Jiaying Liu, and Zongming Guo. "Computational modeling of artistic intention: Quantify lighting surprise for painting analysis." In: *2016 Eighth International Conference on Quality of Multimedia Experience (QoMEX)*. IEEE. 2016, pp. 1–6. (Cit. on p. 73).

[251] Lois Fichner-Rathus. *Foundations of art and design: An enhanced media edition*. Cengage Learning, 2011. (Cit. on p. 73).

[252] Loris Nanni, Stefano Ghidoni, and Sheryl Brahnam. "Handcrafted vs. non-handcrafted features for computer vision classification." In: *Pattern Recognition* 71 (2017), pp. 158–172. ISSN: 0031-3203. DOI: https://doi.org/10.1016/j.patcog.2017.05.025. (Cit. on p. 75).

[253] Dina Goldin and Peter Wegner. "The church-Turing thesis: breaking the myth." In: *New computational paradigms*. Vol. 3526. Lecture Notes in Comput. Sci. Springer, Berlin, [2005] ©2005, pp. 152–168. (Cit. on p. 79).

[254] George S Boolos, John P Burgess, and Richard C Jeffrey. *Computability and logic*. Fourth. Cambridge University Press, Cambridge, 2002, pp. xii+356. ISBN: 0-521-80975-4; 0-521-00758-5. DOI: 10.1017/CBO9781139164931.
URL: https://doi.org/10.1017/CBO9781139164931 (cit. on p. 80).

[255] George S Boolos, John P Burgess, and Richard C Jeffrey. *Computability and logic*. Cambridge university press, 2002. (Cit. on p. 80).

[256] Rajeev Motwani and Prabhakar Raghavan. *Randomized algorithms*. Cambridge University Press, Cambridge, 1995, pp. xiv+476. ISBN: 0-521-47465-5. (Cit. on pp. 81, 107).

[257] John Gill. "Computational complexity of probabilistic Turing machines." In: *SIAM J. Comput.* 6.4 (1977), pp. 675–695. ISSN: 0097-5397. (Cit. on pp. 81, 107).

[258] Hector Zenil, Narsis A Kiani, and Jesper Tegnér. "Algorithmic information dynamics of emergent, persistent, and colliding particles in the game of life." In: *From Parallel to Emergent Computing*. CRC Press, 2019, pp. 367–384. (Cit. on p. 94).

[259] Tibor Rado. "On non-computable functions." In: *Bell System Technical Journal* 41.3 (1962), pp. 877–884. (Cit. on p. 102).

[260] Kurt Gödel. "Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I." In: *Monatshefte für mathematik und physik* 38.1 (1931), pp. 173–198. (Cit. on p. 103).

[261] Gregory Landini. *Russell*. Routledge Philosophers. Routledge/Taylor & Francis Group, London, 2011, pp. xvi+468. ISBN: 978-0-415-39626-4; 978-0-415-39627-1; 978-0-203-84649-0. (Cit. on p. 103).

[262] Jorge M Silva, Diogo Pratas, Tânia Caetano, and Sérgio Matos. "Supporting data for "The complexity landscape of viral genomes"." In: (2022). DOI: http://dx.doi.org/10.5524/102241. (Cit. on p. 164).

Appendices

Appendix A

Appendix of Chapter 3

A.1 Content

Here, we depict the supplementary material of Chapter 3. The appendix is divided in tree main sections:

- Additional information of Chapter 3;
- Website;
- Software and hardware recommendations;
- Reproducibility.

A.2 Additional information of chapter 3

A.2.1 Data compressors and level selection benchmark

Herein, it is depicted the supplementary material to the Data compressors and Level selection benchmark.

Table A.1 describes the parameters used in the six custom build levels. The flag *"tm"* is the template of a target context model, the flag *"lr"* defines the learning rate, and the flag *"hs"* defines the number of hidden nodes for the neural network.

Table A.2 describes the parameters used in the template of a target context model. The template has the flag *"tm"* and follows the model "[NB_C]:[NB_D]:[NB_I]:[NB_-H]:[NB_G]/[NB_S]:[NB_E]:[NB_A]".

A.2.2 Viral genome analysis

We present the supplementary material discussed in the viral genome analysis of Chapter 3.

Table A.3 depicts the top NC values by taxonomic group. Three main groups separate the Table. The first represents the highest 10 NC values using standard settings NC (best

Table A.1: Depiction of the parameters used in the six custom levels.

Level	**Values**
1	-tm 1:1:0:0:0.7/0:0:0 -tm 12:20:1:1:0.97/1:1:0.97
2	-tm 1:1:0:0:0.7/0:0:0 -tm 12:20:1:1:0.97/2:1:0.97
3	-tm 1:1:0:0:0.7/0:0:0 -tm 12:50:1:1:0.97/0:0:0.97
4	-tm 1:1:0:0:0.7/0:0:0 -tm 12:20:1:1:0.97/0:0:0.97 -lr 0.05 -hs 40
5	-tm 1:1:0:0:0.7/0:0:0 -tm 12:20:1:1:0.97/0:0:0.97 -lr 0.15 -hs 40
6	-tm 1:1:0:0:0.7/0:0:0 -tm 12:20:1:1:0.97/0:0:0.97 -lr 0.3 -hs 40

performing model); the second group shows the top 10 lowest NC values obtained using the IR_2 subprogram. Finally, the third group shows the top 10 highest values of the difference between NC using IR_0 and IR_1 subprograms.

Tables A.4, A.5,a nd A.6 organize the top taxa (by taxonomic group) regarding their NC, normalized compression capacity (NCC) and difference. The tables also shows the genomes' average Sequence Length and GC-Content.

Table A.2: Depiction of the parameters used in the template of a target context model.

Parameter	Values	Description
[NB_C]	integer [1;20]	Order size of the regular context model. The higher the value of the regular context model, the more RAM it uses but, usually, are related to a better compression score.
[NB_D]	integer [1;5000]	Denominator to build alpha, which is a parameter estimator. Alpha is given by 1/[NB_D]. Higher values are usually used with higher [NB_C] and are related to sure bets. When [NB_D] is one, the probabilities assume a Laplacian distribution.
[NB_I]	integer {0,1,2}	Number to define if a sub-program that addresses the specific properties of DNA sequences (inverted repeats) is used or not. The number 2 turns ON this sub-program without the regular context model (only inverted repeats). The number 1 turns ON the sub-program using at the same time the regular context model. The number 0 does not contemplate its use (inverted repeats OFF). This sub-program increases the necessary time to compress, but it does not affect the RAM.
[NB_H]	integer [1;254]	Size of the cache-hash for deeper context models, namely for [NB_C] >14. When the [NB_C] <= 14 use, for example, 1 as a default. The RAM is highly dependent of this value (higher value stand for higher RAM).
[NB_G]	real [0;1)	Real number to define gamma. This value represents the decaying forgetting factor of the regular context model in the definition.
[NB_S]	integer [0;20]	The maximum number of editions allowed to use a substitutional tolerant model with the same memory model of the regular context model with an order size equal to [NB_C]. The value 0 stands for turning the tolerant context model off. When the model is on, it pauses when the number of editions is higher than [NB_C]. When it is turned on when a full match of size [NB_C] is seen again, this is a probabilistic-algorithmic model advantageous to handle the high substitutional nature of genomic sequences. When [NB_S] >0, the compressor used more processing time but used the same RAM and, usually, achieved a substantial higher compression ratio. The impact of this model is usually only noticed for [NB_C] >= 14.
[NB_E]	integer [1;5000]	Denominator to build alpha for substitutional tolerant context model. It is analogous to [NB_D]. However, it is only used in the probabilistic model for computing the statistics of the substitutional tolerant context model.
[NB_A]	real [0;1)	Real number to define gamma. This value represents the decaying forgetting factor of the substitutional tolerant context model in the definition. Its definition and use are analogous to [NB_G].

Table A.3: Depiction of the top NC values by viral taxonomic group. Three main groups separate the Table. The first represents the highest 10 NC values using standard settings NC (best performing model); the second group shows the top 10 lowest NC values obtained using the IR_2 subprogram. Finally, the third group shows the top 10 highest values of the difference between NC using IR_0 and IR_1 subprograms.

NC	**Top**	**Realm**	**Kingdom**	**Phylum**	**Class**	**Order**	**Family**	**Genus**
Best performing model	**1**	Ribozyviria	Shotokuvirae	Lenarviricota	Miaviricetes	Ourlivirales	Botourmiaviridae	Clostunsatellite
	2	Monodnaviria	Sangervirae	Cressdnaviricota	Arfiviricetes	Cirlivirales	Alphasatellitidae	Milvetsatellite
	3	Riboviria	Orthornavirae	Duplornaviricota	Chunqiuviricetes	Cremevirales	Tolecusatellitidae	Aumaivirus
	4	Duplodnaviria	Pararnavirae	Phixviricota	Magsaviricetes	Muvirales	Circoviridae	Virtovirus
	5	Varidnaviria	Loebvirae	Kitrinoviricota	Amabiliviricetes	Nodamuvirales	Genomoviridae	Mivedwarsatellite
	6	Adnaviria	Trapavirae	Cossaviricota	Duplopiviricetes	Wolframvirales	Nodaviridae	Babusatellite
	7	-	Heunggongvirae	Pisuviricota	Allassoviricetes	Durnavirales	Kolmioviridae	Fabenesatellite
	8	-	Bamfordvirae	Negarnaviricota	Repensiviricetes	Levivirales	Smacoviridae	Ourmiavirus
	9	-	Helvetiavirae	Artverviricota	Yunchangviricetes	Geplafuvirales	Qinviridae	Albetovirus
	10	-	Zilligvirae	Hofneiviricota	Insthoviricetes	Goujianvirales	Narnaviridae	Geminialphasatellitinae
Inverted repeats Subprogram (NC_{IR_2})	**1**	Adnaviria	Loebvirae	Peploviricota	Pokkesviricetes	Imitervirales	Mimiviridae	Betaentomopoxvirus
	2	Varidnaviria	Zilligvirae	Nucleocytoviricota	Herviviricetes	Chitovirales	Rudiviridae	Oryzopoxvirus
	3	Duplodnaviria	Helvetiavirae	Hofneiviricota	Maveriviricetes	Herpesvirales	Poxviridae	Vespertilionpoxvirus
	4	Monodnaviria	Bamfordvirae	Taleaviricota	Mouviricetes	Priklausovirales	Malacoherpesviridae	Simplexvirus
	5	Riboviria	Heunggongvirae	Dividoviricota	Faserviricetes	Polivirales	Plectroviridae	Cafeteriavirus
	6	Ribozyviria	Trapavirae	Uroviricota	Tokiviricetes	Ligamenvirales	Mononiviridae	Mardivirus
	7	-	Shotokuvirae	Saleviricota	Laserviricetes	Tubulavirales	Herpesviridae	Cervidpoxvirus
	8	-	Pararnavirae	Preplasmiviricota	Megaviricetes	Halopanivirales	Lavidaviridae	Varicellovirus
	9	-	Orthornavirae	Negarnaviricota	Naldaviricetes	Pimascovirales	Bidnaviridae	Ostreavirus
	10	-	Sangervirae	Cossaviricota	Milneviricetes	Lefavirales	Polydnaviridae	Vespertiliovirus

NC	Top	Realm	Kingdom	Phylum	Class	Order	Family	Genus
$NC_{IR_0} - NC_{IR_1}$	**1**	Adnaviria	Zilligvirae	Peploviricota	Herviviricetes	Herpesvirales	Malacoherpesviridae	Mardivirus
	2	Varidnaviria	Trapavirae	Taleaviricota	Mouviricetes	Polivirales	Herpesviridae	Ostreavirus
	3	Duplodnaviria	Bamfordvirae	Nucleocytoviricota	Tokiviricetes	Chitovirales	Rudiviridae	Iltovirus
	4	Monodnaviria	Heunggongvirae	Saleviricota	Pokkesviricetes	Ligamenvirales	Bidnaviridae	Leporipoxvirus
	5	Ribozyviria	Shotokuvirae	Cossaviricota	Quintoviricetes	Piccovirales	Poxviridae	Simplexvirus
	6	Riboviria	Helvetiavirae	Dividoviricota	Huolimaviricetes	Haloruvirales	Polydnaviridae	Varicellovirus
	7	-	Loebvirae	Hofneiviricota	Megaviricetes	Cirlivirales	Ampullaviridae	Aurivirus
	8	-	Sangervirae	Cressdnaviricota	Laserviricetes	Pimascovirales	Nudiviridae	Oryzopoxvirus
	9	-	Orthornavirae	Preplasmiviricota	Arfiviricetes	Algavirales	Parvoviridae	Vespertilionpoxvirus
	10	-	Pararnavirae	Duplornaviricota	Faserviricetes	Kalamavirales	Ascoviridae	Entnonagintavirus

Table A.4: Depiction of the viral taxonomic groups with the highest NC values. The Table shows each group's average NC, Sequence Length (SL) and GC-Content.

Taxonomic Group	Taxonomic Name	NC	SL	GC-Content
Super-Realm	Viruses	1.007	36067	0.460
Realm	Ribozyviria	1.080	1682	0.588
	Monodnaviria	1.046	4380	0.450
	Riboviria	1.016	9332	0.438
	Duplodnaviria	0.972	78102	0.500
	Varidnaviria	0.957	109560	0.448
	Adnaviria	0.948	33068	0.353
Kingdom	Shotokuvirae	1.049	4200	0.447
	Sangervirae	1.026	5518	0.435
	Orthornavirae	1.018	9472	0.438
	Pararnavirae	0.995	7787	0.433
	Loebvirae	0.994	7332	0.483
	Trapavirae	0.993	10151	0.564
	Heunggongvirae	0.972	78102	0.500
	Bamfordvirae	0.957	112955	0.441
	Helvetiavirae	0.949	24833	0.665
	Zilligvirae	0.948	33068	0.353
Phylum	Lenarviricota	1.094	2654	0.476
	Cressdnaviricota	1.067	3134	0.453
	Duplornaviricota	1.045	9418	0.456
	Phixviricota	1.026	5518	0.435
	Kitrinoviricota	1.018	8548	0.474
	Cossaviricota	1.013	6260	0.436
	Pisuviricota	1.012	10580	0.442
	Negarnaviricota	1.012	9620	0.397
	Artverviricota	0.995	7787	0.433
	Hofneiviricota	0.994	7332	0.483
Class	Miaviricetes	1.151	1792	0.514
	Arfiviricetes	1.085	2557	0.464
	Chunqiuviricetes	1.075	3870	0.503
	Magsaviricetes	1.073	3730	0.513
	Amabiliviricetes	1.072	2703	0.586
	Duplopiviricetes	1.066	3298	0.467
	Allassoviricetes	1.063	3753	0.493
	Repensiviricetes	1.063	3281	0.451
	Yunchangviricetes	1.061	3987	0.358
	Insthoviricetes	1.054	5784	0.425
Order	Ourlivirales	1.151	1792	0.514

Taxonomic Group	Taxonomic Name	NC	SL	GC-Content
	Cirlivirales	1.103	1864	0.471
	Cremevirales	1.078	2572	0.478
	Muvirales	1.075	3870	0.503
	Nodamuvirales	1.073	3730	0.513
	Wolframvirales	1.072	2703	0.586
	Durnavirales	1.066	3298	0.467
	Levivirales	1.063	3753	0.493
	Geplafuvirales	1.063	3281	0.451
	Goujianvirales	1.061	3987	0.358
Family	Botourmiaviridae	1.151	1792	0.514
	Alphasatellitidae	1.143	1296	0.418
	Tolecusatellitidae	1.116	1347	0.389
	Circoviridae	1.103	1864	0.471
	Genomoviridae	1.096	2201	0.517
	Nodaviridae	1.080	3368	0.514
	Kolmioviridae	1.080	1682	0.588
	Smacoviridae	1.078	2572	0.478
	Qinviridae	1.075	3870	0.503
	Narnaviridae	1.072	2703	0.586
Genus	Clostunsatellite	1.192	1008	0.423
	Milvetsatellite	1.186	1022	0.402
	Aumaivirus	1.185	1168	0.510
	Virtovirus	1.180	1150	0.442
	Mivedwarsatellite	1.179	1014	0.402
	Babusatellite	1.178	1104	0.437
	Fabenesatellite	1.176	1007	0.385
	Ourmiavirus	1.167	1605	0.519
	Albetovirus	1.167	1221	0.426
	Geminialphasatellitinae	1.131	1370	0.418

Table A.5: Depiction of the viral taxonomic groups with the highest normalized compression capacity (NCC) using only the inverted repeats subprogram IR_2. The top results were obtained by $NCC = 1 - NC_{IR_2} > 0$. Besides the normalized compression capacity, the Table shows each group's average Sequence Length (SL) and GC-Content.

Group	**Taxonomic Group**	$\mathbf{NCC = 1 - NC_{IR_2} > 0}$	**SL**	**GC-Content**
Super-Realm	Viruses	0.026	66796	0.462
Realm	Adnaviria	0.052	33068	0.353
	Varidnaviria	0.038	110591	0.447
	Duplodnaviria	0.028	82677	0.499
	Monodnaviria	0.022	6958	0.399
	Riboviria	0.015	13682	0.391
Kingdom	Loebvirae	0.053	7371	0.385
	Zilligvirae	0.052	33068	0.353
	Helvetiavirae	0.050	24833	0.665
	Bamfordvirae	0.038	114079	0.440
	Heunggongvirae	0.028	82677	0.499
	Trapavirae	0.021	12225	0.577
	Shotokuvirae	0.016	6184	0.378
	Pararnavirae	0.016	9610	0.378
	Orthornavirae	0.015	14012	0.393
	Sangervirae	0.005	4421	0.321
Phylum	Peploviricota	0.068	168832	0.534
	Nucleocytoviricota	0.063	210417	0.389
	Hofneiviricota	0.053	7371	0.385
	Taleaviricota	0.052	33068	0.353
	Dividoviricota	0.050	24833	0.665
	Uroviricota	0.026	79042	0.497
	Saleviricota	0.021	12225	0.577
	Preplasmiviricota	0.017	32147	0.483
	Negarnaviricota	0.016	12180	0.376
	Cossaviricota	0.016	6128	0.378
Class	Pokkesviricetes	0.072	190762	0.365
	Herviviricetes	0.068	168832	0.534
	Maveriviricetes	0.066	18227	0.290
	Mouviricetes	0.066	8377	0.299
	Faserviricetes	0.053	7371	0.385
	Tokiviricetes	0.052	33068	0.353
	Laserviricetes	0.050	24833	0.665
	Megaviricetes	0.046	248459	0.436
	Naldaviricetes	0.040	132022	0.410
	Milneviricetes	0.029	11079	0.349

Group	Taxonomic Group	$NCC = 1 - NC_{IR_2} > 0$	SL	GC-Content
Order	Imitervirales	0.109	899501	0.256
	Chitovirales	0.091	193551	0.356
	Herpesvirales	0.068	168832	0.534
	Priklausovirales	0.066	18227	0.290
	Polivirales	0.066	8377	0.299
	Ligamenvirales	0.055	34464	0.343
	Tubulavirales	0.053	7371	0.385
	Halopanivirales	0.050	24833	0.665
	Pimascovirales	0.043	162587	0.456
	Lefavirales	0.040	132022	0.410
Family	Mimiviridae	0.109	899501	0.256
	Rudiviridae	0.103	30804	0.299
	Poxviridae	0.091	193551	0.356
	Malacoherpesviridae	0.091	209479	0.427
	Plectroviridae	0.080	7045	0.248
	Mononiviridae	0.077	41178	0.275
	Herpesviridae	0.074	158421	0.539
	Lavidaviridae	0.066	18227	0.290
	Bidnaviridae	0.066	8377	0.299
	Polydnaviridae	0.055	306235	0.377
Genus	Betaentomopoxvirus	0.174	247441	0.195
	Oryzopoxvirus	0.164	185139	0.236
	Vespertilionpoxvirus	0.156	176688	0.236
	Simplexvirus	0.144	148626	0.694
	Cafeteriavirus	0.127	617453	0.233
	Mardivirus	0.121	177993	0.509
	Cervidpoxvirus	0.115	166259	0.262
	Varicellovirus	0.107	139331	0.560
	Ostreavirus	0.107	207439	0.387
	Vespertiliovirus	0.103	7970	0.228

Table A.6: Depiction of the viral taxonomic groups with the highest difference of values between $NC_{IR_0} - NC_{IR_1}$. The Table shows each group's average $difference = NC_{IR_0} - NC_{IR_1}$, Sequence Length (SL) and GC-Content.

Taxonomic Group	**Taxonomic Name**	$\mathbf{NC_{IR_0} - NC_{IR_1} > 0}$	**SL**	**GC-Content**
Super-Realm	Viruses	0.004	44293	0.451
Realm	Adnaviria	0.019	35299	0.322
	Varidnaviria	0.007	111364	0.443
	Duplodnaviria	0.007	78316	0.512
	Monodnaviria	0.005	5359	0.436
	Ribozyviria	0.002	1682	0.588
	Riboviria	0.001	9847	0.431
Kingdom	Zilligvirae	0.019	35299	0.322
	Trapavirae	0.009	16113	0.503
	Bamfordvirae	0.007	114249	0.437
	Heunggongvirae	0.007	78316	0.512
	Shotokuvirae	0.005	5124	0.434
	Helvetiavirae	0.004	27439	0.664
	Loebvirae	0.002	8519	0.453
	Sangervirae	0.001	4552	0.426
	Orthornavirae	0.001	10049	0.430
	Pararnavirae	0.001	8050	0.435
Phylum	Peploviricota	0.050	159507	0.557
	Taleaviricota	0.019	35299	0.322
	Nucleocytoviricota	0.013	210797	0.381
	Saleviricota	0.009	16113	0.503
	Cossaviricota	0.007	5450	0.433
	Dividoviricota	0.004	27439	0.664
	Hofneiviricota	0.002	8519	0.453
	Cressdnaviricota	0.002	4539	0.438
	Preplasmiviricota	0.002	32788	0.483
	Duplornaviricota	0.001	8140	0.389
Class	Herviviricetes	0.050	159507	0.557
	Mouviricetes	0.029	8377	0.299
	Tokiviricetes	0.019	35299	0.322
	Pokkesviricetes	0.017	193309	0.354
	Quintoviricetes	0.011	5164	0.446
	Huolimaviricetes	0.009	16113	0.503
	Megaviricetes	0.005	247791	0.441
	Laserviricetes	0.004	27439	0.664
	Arfiviricetes	0.004	5459	0.432
	Faserviricetes	0.002	8519	0.453

Taxonomic Group	Taxonomic Name	$NC_{IR_0} - NC_{IR_1} > 0$	SL	GC-Content
Order	Herpesvirales	0.050	159507	0.557
	Polivirales	0.029	8377	0.299
	Chitovirales	0.022	196072	0.341
	Ligamenvirales	0.019	35299	0.322
	Piccovirales	0.011	5164	0.446
	Haloruvirales	0.009	16113	0.503
	Cirlivirales	0.008	2114	0.476
	Pimascovirales	0.005	169619	0.458
	Algavirales	0.005	339710	0.413
	Kalamavirales	0.004	15181	0.459
Family	Malacoherpesviridae	0.062	209479	0.427
	Herpesviridae	0.050	155406	0.564
	Rudiviridae	0.035	30804	0.299
	Bidnaviridae	0.029	8377	0.299
	Poxviridae	0.022	196072	0.341
	Polydnaviridae	0.019	306235	0.377
	Ampullaviridae	0.012	23814	0.346
	Nudiviridae	0.012	127615	0.416
	Parvoviridae	0.011	5164	0.446
	Ascoviridae	0.010	172411	0.453
Genus	Mardivirus	0.103	177993	0.509
	Ostreavirus	0.072	207439	0.387
	Iltovirus	0.070	155856	0.546
	Leporipoxvirus	0.066	160815	0.415
	Simplexvirus	0.061	148626	0.694
	Varicellovirus	0.061	139331	0.560
	Aurivirus	0.052	211518	0.468
	Oryzopoxvirus	0.050	185139	0.236
	Vespertilionpoxvirus	0.046	176688	0.236
	Entnonagintavirus	0.036	29564	0.558

A.3 Website

The website of this chapter is available at https://asilab.github.io/canvas/. This site showcases, among other things, the pipeline of this study, the compressor's model selection, the detection of inverted repeats in synthetic genomic sequences, the viral genome characterization with regards to genome and type of taxonomic group, and the computed cladograms with a magnifier to allow a better observation of the normalized complexity results with illustrative examples of viruses. Snapshots of our code and other data further supporting this work are openly available in GigaDB[262].

A.4 Software and hardware recommendations

The experiences of Chapter 3 can be replicated using a laptop, desktop, or server computer running Arch linux or Linux Ubuntu (for example, 18.04 LTS or higher) with GCC (`https://gcc.gnu.org`), git and git LFS, Conda (`https://docs.conda.io`) and python version 3.6. The hardware must contain at least 16 GB of RAM and a 100 GB disk.

A.5 Reproducibility

A.5.1 Viral analysis and taxonomic classification

Creating and installing the project

The descriptions of how to replicate the experiments of Chapter 3 are depicted bellow, for more detail see https://github.com/jorgeMFS/canvas.

Install Git LFS:

```
mkdir -p gitLFS
cd gitLFS/
wget https://github.com/git-lfs/git-lfs/releases/download/v2.9.0/git-lfs-linux-amd64-v2.9.0.tar.gz
tar -xf git-lfs-linux-amd64-v2.9.0.tar.gz
chmod 755 install.sh
sudo ./install.sh
```

Get CANVAS project, create the docker and run it:

```
git clone https://github.com/jorgeMFS/canvas.git
cd canvas
docker-compose build
docker-compose up -d && docker exec -it canvas bash && docker-compose down
```

Inside the docker, give run permissions to the files and install tools using :

```
chmod +x *.sh
bash Make.sh;
```

Replication of the results

The code was created in order to allow independent replication and reproduction of each step, this was done due to the extensive processing time required to filter and rearrange viral DB and extract the features and taxonomic information of each viral sequence. If you wish to rebuild database and feature reports extracted, see the Database reconstruction subsection.

To obtain the Human Herpesvirus, plot run:

```
cd python || exit;
python compare_cmix_hhv.py
```

To obtain the Compression Benchmark plots, run:

```
cd python || exit;
python select_best_nc_model.py;
```

To perform the synthetic sequence test, run:

```
cd scripts || exit;
bash Stx_seq_test.sh;
```

To perform classification, run the following code:

```
cd python || exit;
python prepare_classification.py; #recreate classification dataset
python classifier.py; #perform classifications
```

To perform the complete IR analysis and create:

- boxplots;
- 2d scatter plots;
- 3d scatter plots;
- top taxonomic group lists;
- Occurrence of each Genus.

Execute this code:

```
cd python || exit;
python ir_analysis.py; # Performs complete IR analysis
```

To perform the Human Herpesvirus analysis and obtain the plots, run:

```
cd scripts || exit;
bash Herpesvirales.sh;
```

Database reconstruction

To run the pipeline and obtain all the Reports in the folder reports, use the following commands. Note that if you wish to recreate the features reports, you must perform the database reconstruction task first.

If you wish to reconstruct the viral database, run the following script:

```
cd scripts || exit;
bash Build_DB.sh;
```

To create the features for analysis and classification (very time consuming, can take several days), run:

```
cd scripts || exit;
bash Process_features.sh;
```

To recreate the compression reports used for benchmark (very time consuming, can take several hours), run:

```
cd scripts || exit;
bash Compress.sh;
```

Cladograms

The Cladograms require GUI application. As such, the reproduction of the cladograms has to be performed outside of the docker on the Ubuntu system on the /canvas folder:

```
chmod +x *.sh
bash so_dependencies.sh #install Ubuntu system dependencies required for the
    script to run and Anaconda
conda create -n canvas python=3.6
conda activate canvas
bash Make.sh #install python libs
bash Install_programs.sh #install tools using conda
```

Afterwards, to obtain the Cladogram plots, run:

```
cd python || exit;
python phylo_tree.py;
```

A.5.2 Archaea taxonomic classification

The descriptions of reproducion is depicted bellow, for more detail see https://github.com/jorgeMFS/Archaea2.

Installation

To perform the installation, get Archaea2 project, create the docker and run it:

```
git clone https://github.com/jorgeMFS/Archaea2.git
cd Archaea2
docker-compose build
docker-compose up -d && docker exec -it archaea2 bash && docker-compose down
```

Get Archaea project using:

```
chmod +x run.sh
bash run.sh
```

Dataset Download

To obtain the dataset, run:

```
cd scripts || exit
bash download_dataset.sh
```

Preprocess Dataset

To prepare the data for classification, run:

```
cd scripts || exit
bash prepare_and_classify_dataset.sh
```

Classification

Finally, to perform classification, run:

```
cd "../python_src/" || exit
python3 create_classification_dataset.py
python3 classification.py
```

Appendix B

Appendix of Chapter 4

B.1 Content

Here, we depict the supplementary material of Chapter 4. The appendix is divided in tree main sections:

- Additional information of chapter 4;
- Website;
- Software and hardware recommendations;
- Reproducibility.

B.2 Additional information of chapter 4

B.2.1 Comparison towards normalized images

The images provided by the dataset were not normalized. This section evaluates the effects and possible impact of the 8-bit images' normalization on the measures used. The images were normalized by forcing the brightest pixels to white (255), the darkest pixels to black (0), and spreading the ones in between. We computed each measure's average values per author for the normalized images, and then we measured the average difference and its standard deviation between them and the previously obtained results. Furthermore, we also computed the mean percentage difference (MPD) and standard deviation as

$$MPD = \sum_{i=0}^{n} \frac{|a_i - b_i|}{\frac{a_i + b_i}{2}} \times 100, \tag{B.1}$$

a and b are the average value of the measure for a given author for the normalized and non-normalized images, respectively, and i is the author.

Table B.1 describes the average variation between the measures taken directly from the dataset and those taken after normalization.

Table B.1: Author's Average difference and the percentage difference between normalized and non-normalized images for the $NBDM_1$,$NBDM_2$, NC, and α.

Measure	**Average ± Standard Deviation**	**Percentage Difference (%)**
$NBDM_1$	0.031 ± 0.001	3.978
$NBDM_2$	0.015 ± 0.001	3.977
NC	0.017 ± 0.002	2.531
α	0.001 ± 0.000	0.545

The results show that measures have low average variation and percentage differences, being the most affected the NBDM and the least affected the roughness exponent α. Therefore, we can conclude that they are resistant and that the normalization has no significant impact on the measures utilized.

To assess the impact of the image normalization on the Regional complexity, we performed the Mantel test and computed the average difference between the normalized and non-normalized images' distances. The results are shown in Table 4.1.

The Mantel test measures the correlation between two distance matrices. The results show a high correlation of 0.955 with a p-value of 0.001 and a low average difference of distance between authors of 3.2622 ± 2.820. Since the only difference between the two distances is that one was computed from normalized images and the other from non-normalized images, we conclude with the results obtained that this measure is also robust to normalization. Consequently, image normalization has a minimal impact and does not significantly influence the measure.

B.2.2 Kruskal minimum spanning tree

To verify the congruence of the UPGMA tree, we constructed another tree from the distance tree using the Kruskal minimum spanning tree algorithm [232]. This algorithm uses the connected graph created by the distance between authors and removes the edges' subset that forms a tree that includes every vertex, where the sum of the weights of all the edges in the tree is minimized. The resulting tree is shown in Figure B.1 and can also be viewed in more detail on the website described in Section B.3.

Despite being organized in a different and more sparse manner, the same connections are observed in the UPGMA tree. The tree Kruskal minimum spanning tree retains the relationships of influence between authors of different artistic movements (Titian and Diego Velazquez; Caravaggio and Francisco de Zurbarán; Frida Kahlo and Amedeo Modigliani; Sandro Botticelli and William Blake; Claude Lorrain and Joseph Mallord William Turner; and Peter Paul Rubens and Jean-Antoine Watteau), as well as shows the fingerprint's capacity of grouping some artists from the same artistic movements mutually. We can conclude from this that qualitatively the fingerprints are useful description tools of the artist's way of painting despite the algorithm used to represent the tree.

B.3 Website

A support website to this site can be accessed at http://panther.web.ua.pt/. This site showcases among other things, the pipeline of this study, the author's average NC and NBDM variation for different quantization levels, the results of combining the NC with the roughness exponent of HDC function (α), a complete catalogue of each author's fingerprints as well as several examples of each author's paintings, and the computed cladograms with a magnifier to allow a better observation of the results.

B.4 Software and hardware recommendations

The experiences of Chapter 4 can be replicated using a laptop, desktop, or server computer running Arch linux or Linux Ubuntu (for example, 18.04 LTS or higher) with GCC (`https://gcc.gnu.org`), git, Conda (`https://docs.conda.io`) and python version 3.6. The hardware must contain at least 16 GB of RAM and a 100 GB disk.

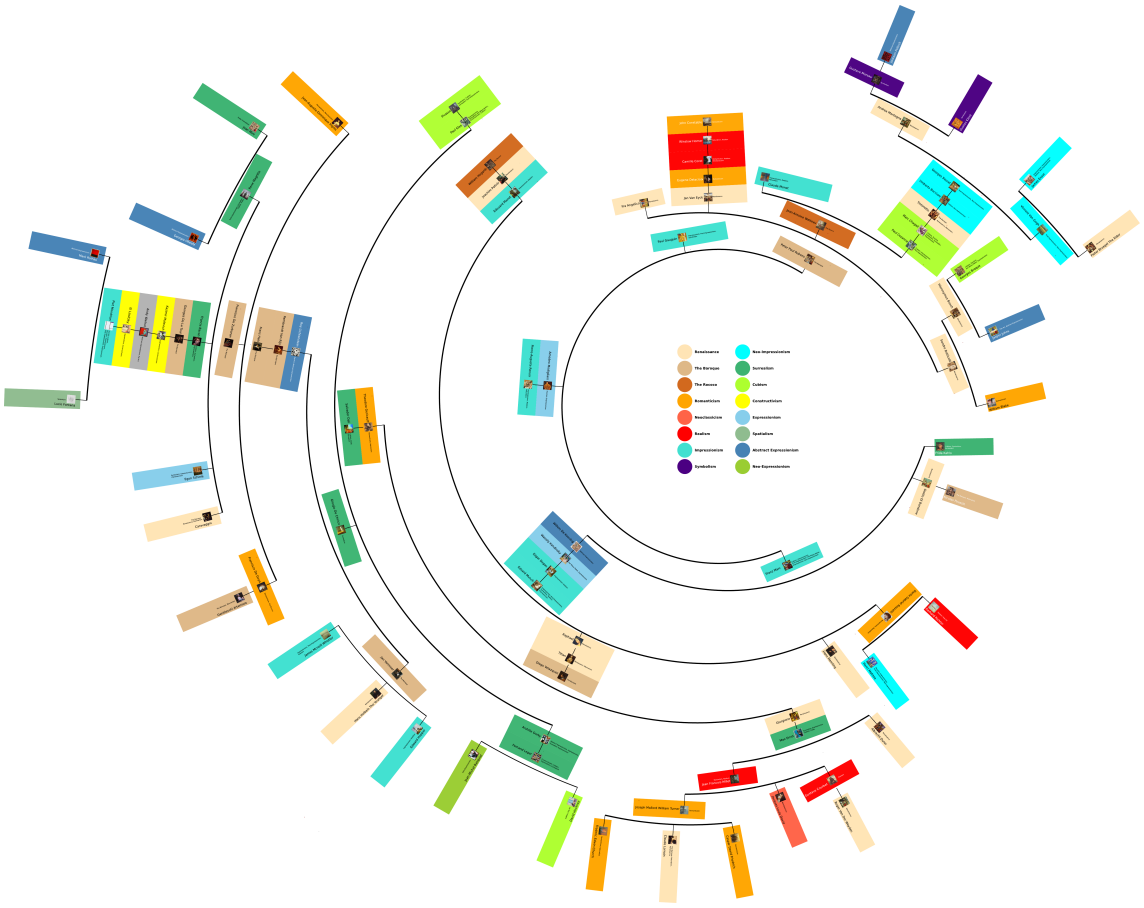


Figure B.1: Artists' cladogram computed recurring to Kruskal minimum spanning tree. Each artist has a painting and a color of a style (chosen based on nearest leaves) assigned to him, as well as a description of some styles usually associated with the author. To obtain an improved view of the tree, please see the website described in Section B.3.

B.5 Reproducibility

All the results presented in this chapter can be fully replicated, under a Linux machine, using the scripts provided at the repository https://github.com/asilab/panther.

These include the automatic installation of the tools, download the dataset, assessment, benchmarking, measurement, and visualization of the results.

B.5.1 Installation

First, there is the need to give execution access to the scripts using:

```
chmod +x *.sh
```

To perform automatic installation of the tools using

```
bash make.sh
pip3 install -r requirements.txt
```

Information-based Measures Assessment

To download and prepare the dataset, use script:

```
bash Dataset.sh
```

To reproduce the compression benchmark, use script:

```
bash Benchmark.sh
```

To perform all comparisons between NC, NBDM_1 and NBDM_2 use:

```
bash Compare.sh
```

To replicate the impact of increasing pseudo-random substitutions of pixels for the NC and different types of BDM normalizations (NBDM_1 and NBDM_2), use script:

```
bash  Pixel_Edition.sh
```

To test the different values of the NC and NBDM in different datasets use:

```
bash Diverse_Images.sh
```

If you desire to replicate the cellular automata objects run:

```
bash ca.sh
```

To replicate the super-sampling experience and the results of underestimation of BDM, run:

```
bash Side_Information_Test.sh
```

Information-based measures applied to artistic paintings

To perform all the pipeline execute:

```
bash Run.sh
```

To quantitize images run:

```
bash Quantize.sh
```

To trim and binarize use:

```
bash Trimm_and_Binarization.sh
```

To compute the average NC, $NBDM_1$, and $NBDM_2$ for each author use script:

```
bash Average_Complexity.sh
```

To compute the NC with the HDC results use scripts:

```
bash NC_HDC.sh
```

To recreate the reports of Regional complexity, use the following command:

```
 bash Region_Complexity.sh
```

To recreate the reports of author's Fingerprints, use the following command:

```
bash Average_Regional_Complexity.sh
```

To recreate the authors' fingerprints heat maps run:

```
bash Fingerprints.sh
```

To assess the author average variation and percentage difference between normalized and non-normalized measures, use the following command:

```
bash norm_vs_non_norm.sh
```

To perform the Mantel test and view the average variance between different distance matrices, use the following command:

```
bash Mantel_test_and_variation.sh
```

To recreate the cladogram, run:

```
bash Tree.sh
```

To make the feature file for author and style classification, run:

```
bash Create_classification_data.sh
```

To perform author classification, run the jupyter file:

```
Painting91_author_classification.ipynb
```

To perform style classification, run the jupyter file:

```
Painting91_style_classification.ipynb
```

Appendix C

Appendix of Chapter 5

C.1 Content

Here, we depict the supplementary material of Chapter 5. The appendix is divided in tree main sections:

- Additional information of chapter 5;
- Software and hardware recommendations;
- Reproducibility.

C.2 Additional Information of chapter 5

C.2.1 Probabilistic complexity patterns of Turing Machines

Figure C.1 shows the complete average rule complexity profiles of sampled TMs, inside and outside the regions identified.

C.2.2 Comparison between BDM and NC

Figure C.2 describes the overall behavior of the NC and BDM with different algorithmic generated tapes. Notice that, in Figure C.2, the results are presented after applying a low-pass filter. Furthermore, tape length was normalized by the maximum size obtained for its pair $(\#Q, \#\theta)$, and to observe BDM comparatively to NC, BDM results were scaled by a 10^2 factor. An example with non-scaled BDM is also presented in the image's bottom-right.

C.3 Software and hardware recommendations

The experiences of Chapter 5 can be replicated using a server computer running Arch linux or Linux Ubuntu (for example, 18.04 LTS or higher) with GCC (https://gcc.gnu.org) and git. The hardware must contain at least 64 GB of RAM and a 100 GB disk.

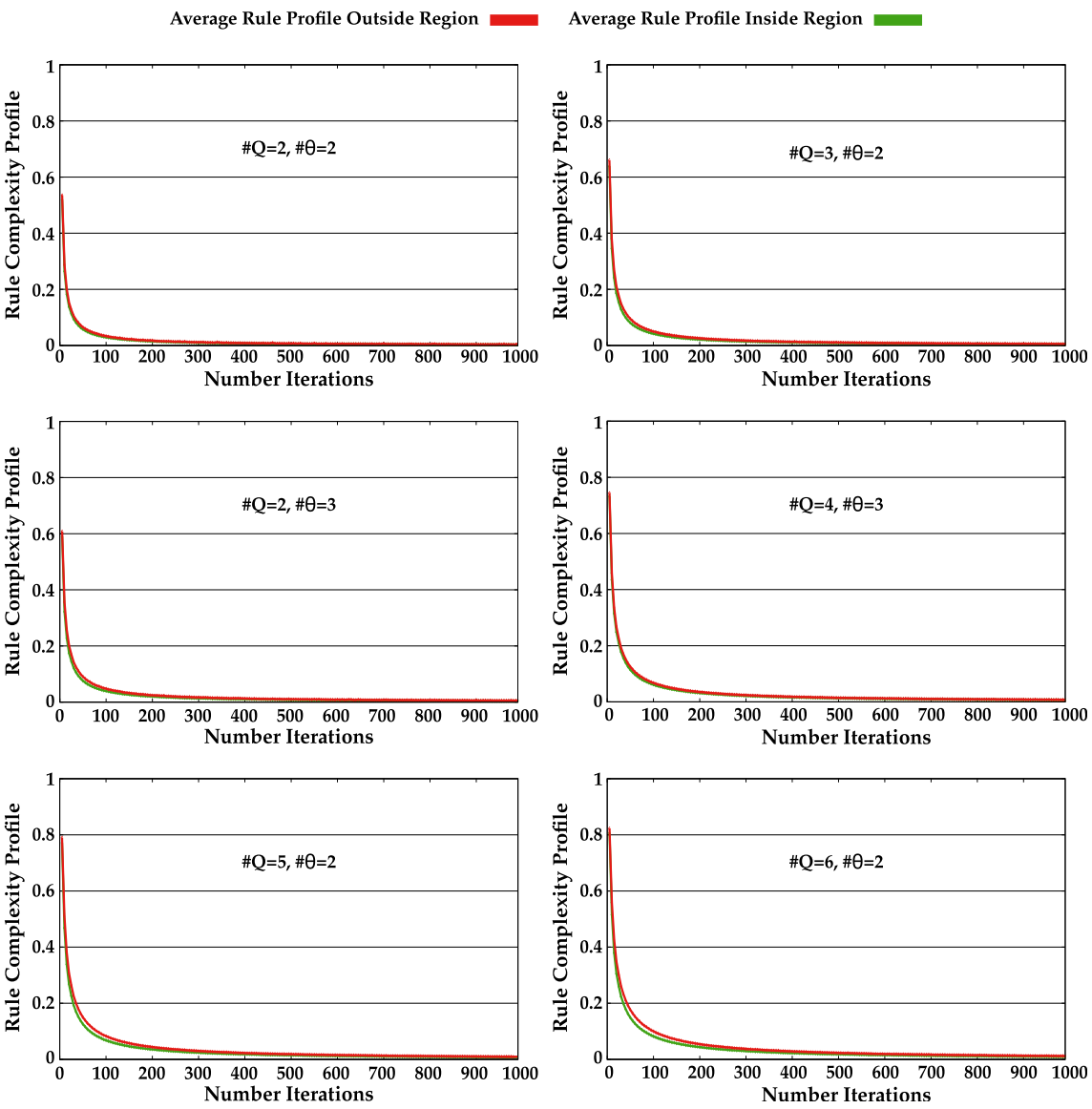


Figure C.1: Average rule complexity profiles obtained from pseudo-randomly selected TMs with $\#Q \in \{2, .., 6\}$ and $\#\theta = \{2, 3\}$ up to 1000 iterations.

C.4 Reproducibility

The descriptions of reproducion is depicted bellow, for more detail see https://github.com/asilab/TMCompression.git.

C.4.1 Creating Project and intalling tools

```
git clone https://github.com/jorgeMFS/TMCompression.git;
cd TMCompression;
make;
make ioStNormalize;
make ioAverage2;
make ioGrowthAverage;
```

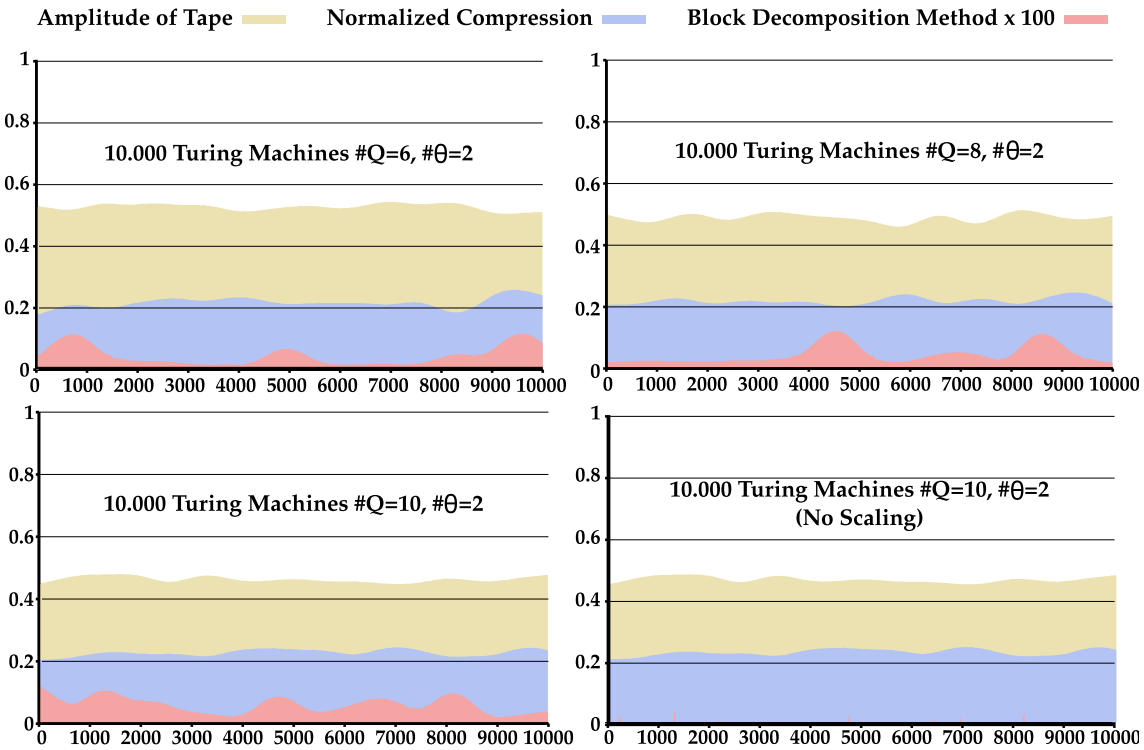


Figure C.2: Comparison between the NC and BDM for 10,000 TM that have run over 50,000 iterations. (**top-left**) TMs with $\#Q = 6, \#\theta = 2$; (**top-right**) TMs with $\#Q = 8, \#\theta = 2$; (**bottom-left**) TMs with $\#Q = 10, \#\theta = 2$; and (**bottom-right**) an example with non-scaled BDM results.

```
make bdmAvg;
make TMsimulator;
```

C.4.2 Recreate plots of Chapter 5

To recreate the plots shown in Chapter 5, start by running the script:

```
    chmod +x run.sh;
    ./run.sh;
```

Virus and string edition and permutation plots

To recreate edition and permutation plots, run the script:

```
# Command to create list:
echo "./resultText/Parvovirus_virus_genome.txt" | ./tm --mutateVirus;
# Recreate Edition and permutation plots...";
bash reprArticlePlot.sh 0 0 1 1 1 1;
```

TMs Plots

To recreate plots of cardinality growth and TMs, run the script:

```
    # Recreating plots of Cardinality Growth and TMs...;
    bash processResults.sh 0 1 1 1 1 1 1 1;
```

Profile Plots

To recreate plots of Normal and Dynamic Profiles, run the script:

```
bash reprArticlePlot.sh 1 1 0 0 0 0;
```

Inside and outside region plots

To recreate the inside and outside region plots, run the script:

```
# Recreating Plot of Inside vs Outside region of TM Tapes...;
bash getRegionTapeValues.sh;
# Recreating Plot of Inside vs Outside region of TM Rules...
bash getRegionRuleValues.sh 1 0 1 1;
```

Average rule complexity profiles plots

To recreate the average rule complexity profiles plots, run the script:

```
    # Obtain Average Rule Profiles...
    bash avg_rule_profile.sh 1 1;
```

Block Decomposition Method comparation with Normalized Compression plots

To recreate the BDM comparation with NC plots, run the script:

```
    #Compare BDM with NC for #Q={6,8,10}...
    bash bdm_NC_comparisson.sh 1 0 1 1;
```

Method I and II plots

To recreate the method I and II plots, run the script:

```
    # Command to create list:
    ./tm  --MethodI;
    ./tm  --MethodII;
    # Recreate Plots of Method I and II.
    bash evolve_tm_plot.sh Method_I_200;
    bash evolve_tm_plot.sh Method_II_2000;
```

Method II 3d plots

```
    # Command to create list:
    ./tm  --3DgraphMethodII;
    # Recreate 3D Plot of Method II...
    bash 3d_evolve_graph.sh;
```

Note: The previous commands, download the files containing the results from running all TMs for a givens state and alphabet cardinality, since this can take from several minutes to several days. If you wish to recreate the results use TM program. Example:

```
./tm --brief -s 2 -a 4 -i 50000 -k 2:10 -N 30000000 -j 20 -S monte_carlo > 2sts4alp.txt;

```

C.4.3 Run TMCompression

There are many ways to run this program see help for clarification:

```
./tm --help;

#Program that creates Turing Machines and Measures its Probabilistic complexity.


#Arguments to set flags:

--verbose       #Indicates programs that will receive inputs in a verbose form.

--brief         #Indicates programs that will receive inputs in a brief form.

--tmverbose     #Indicates programs that tm output will be send to the user.

--tmgrowth      #Indicates programs that output a list of Turing Machines with an increase in the number of states and a alphabet size of 2

--replicate     #Indicates programs that will replicate experiment to determine the best k and it for a given number of states and alphabet size.

--profile       #Indicates programs that will receive through the tm number through the flag tm and will create a profile of that turing

--dynprofile    #Indicates programs that will receive through the tm number through the flag tm and will create a dynamical temporal profile of that turing

--ruleProfile   #Indicates programs to create a Compression profile of the rules for a given Turing Machine

--ruleMetrics   #Indicates programs to compute the rule metrics of a given TMs

--StMatrix      #Indicates programs to print the StateMatrix of a given TMs

```

```
28    --mutate        #Indicates programs to print the nc of the mutation of a
      string (starting with all zeros and ending with all ones) and performing
      mutations until its 100% mutated
29
30    --mutateVirus   #Indicates programs perform the edition and permutation of
      a virus DNA sequence and print NC results
31
32    --MethodI       #Indicates programs to recreate similar list of results
      used in plots of the Chapter 5 using MethodI
33
34    --MethodII      #Indicates programs to recreate similar list of results
      used in plots of the Chapter 5 using MethodII
35
36    --3DgraphMethodII       #Indicates programs to recreate similar list of
      results used in 3D plots of Chapter 5 using MethodII
37
38    #Mandatory  Arguments:
39
40    -s, --number_states     #Number of States the Turing Machine has.
41
42    -a, --alphabet_size     #Alphabet size the Turing Machine considers.
43
44    -i, --iterations         #Number of iterations the Turing Machine makes in
      the tape.
45
46    -k, --context   #k indexes to consider as a context to fill the Markov
      Table.
47
48    -t, --tm        #Speciffy turing to obtain results, can only be activated
      with --profile flag.
49
50    #Other Optional Arguments:
51
52    -S, --strategy  #Turing Machine traversal strategy (default: sequential)
53
54    -N,     #Number of Turing Machines to traverse
55
56    -v, --version   #Outputs the version of the program.
57
58    -h, --help      #Describes program.
59
60
61    #Examples:
62
63    #Run all tms
64    ./tm -s 2 -a 2 -i 10 -k 1
65    ./tm --brief -s 2 -a 2 -i 10 -k 1
66    ./tm --verbose --number_states=2 --alphabet_size=2 --iterations=20 --
      context=2
67    #----------------
```

```
68  #Run all tms with multithreads
69  ./tm -s 2 -a 2 -i 10 -k 1 -j 7
70  ./tm --brief -s 2 -a 2 -i 10 -k 1 -j 7
71  #----------------
72  #Strategies of running TM Machines
73  #By default strategy is Sequential:
74  ./tm --brief -s 2 -a 2 -i 10 -k 1 -j 7
75  ./tm --brief -s 2 -a 2 -i 10 -k 1 -N 10 -j 4 -S sequential
76  #Monte Carlo:
77  ./tm --brief -s 2 -a 2 -i 10 -k 1 -N 10 -j 4 -S monte_carlo
78  #----------------
79  #Run specific TM and obtain profile:
80  ./tm --brief --profile -s 2 -a 2 -i 100 -k 2 -t 5
81  #----------------
82  #Run a specific TM and obtain their rules normal complexity profile
83  ./tm --brief --ruleProfile -s 2 -a 2 -i 100 -k 2 -t 5
84  #----------------
85  #Run specific tm and obtain dynamical temporal profile:
86  ./tm --brief --dynprofile -s 2 -a 2 -i 100 -k 2 -t 5
87  #----------------
88  # Run specific tm and obtain their Rule Compression Profile:
89  #----------------
90  ./tm --brief --ruleMetrics -s 2 -a 2 -i 100 -t 5
91  # ----------------
92  #Replicate k and it determination:
93  ./tm --brief --replicate -s 2 -a 2 -j 10
94  #----------------
95  #Turing machine growth with alphabet size of 2 and increase in state
    cardinality:
96  ./tm --tmgrowth
97  #----------------
98  #Run specific tm and print tape
99  ./tm --brief --printTape -s 2 -a 2 -i 100 -t 1
100 #----------------
101 #Run StMatrix of the tm
102 ./tm --brief --StMatrix -s 2 -a 2 -t 1
103 #----------------
104 #Obtain nc of the substitutions and permutations of a string
105 ./tm --mutate
106 #----------------
107 #Obtain nc of Virus genome sequence with substitutions and permutations
108 echo "./resultText/Parvovirus_virus_genome.txt" | ./tm --mutateVirus
109 #----------------
110 #Obtain list of graph of Method I
111 ./tm --MethodI
112 #----------------
113 #Obtain list of graph of Method II
114 ./tm --MethodII
115 #----------------
116 #Obtain list of 3d graphs of Method II
```

```
117     ./tm --3DgraphMethodII
118     #-----------------
```