Universidade de Aveiro
2020

Departamento de Electrónica,
Telecomunicações e Informática
Programa de Doutoramento em Informática
das Universidades de Aveiro, Minho e Porto

Morteza Hosseini

Modelos de compressão e ferramentas para dados ómicos

Compression models and tools for omics data

Universidade de Aveiro
2020

Departamento de Electrónica,
Telecomunicações e Informática
Programa de Doutoramento em Informática
das Universidades de Aveiro, Minho e Porto

Morteza Hosseini

Modelos de compressão e ferramentas para dados ómicos

Compression models and tools for omics data

Tese apresentada às Universidades de Aveiro, Minho e Porto para cumprimento dos requisitos necessários à obtenção do grau de Doutor em Informática, realizada sob a orientação científica do Doutor Armando José Formoso de Pinho, Professor Catedrático e co-orientação do Doutor Diogo Rodrigo Marques Pratas, Investigador Auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro.

Trabalho financiado pelas seguintes entidades:

To my family: Hamze, Ozra, Davoud, Masoud, Fateme, Mahdi, Sedighe, Somaye, Hasti, Tarannom and Delaram

o júri / the jury

presidente / president

João Carlos de Oliveira Matias
Full Professor, University of Aveiro

vogais / examiners committee

Mário Alexandre Teles de Figueiredo
Full Professor, University of Lisbon

Luís Filipe Coelho Antunes
Full Professor, University of Porto

Francisco Moreira Couto
Associate Professor with Aggregation, University of Lisbon

José Luís Guimarães Oliveira
Full Professor, University of Aveiro

Armando José Formoso de Pinho
Full Professor, University of Aveiro

agradecimentos / acknowledgements

First and foremost, I would like to express my sincere gratitude to my supervisor Armando J. Pinho, for the continuous support to my PhD study and research, for his patience, motivation, enthusiasm, and immense knowledge. His guidance helped me in all the time of research and writing of this thesis. It has been a privilege being his student.

I would like to give very special thanks to Diogo Pratas for being my co-supervisor and sharing his knowledge in computational biology. He has been all I could hope for, both from an academic and personal perspective.

I gratefully acknowledge financial support from Fundação para a Ciência e Tecnologia (FCT).

On a more personal note, I would like to thank Hooshiar Zolfagharnasab, João Carvalho, Mohammadreza Kasaei, Hamidreza Kasaei, Ehsan Shahri, Reza Parsamehr, Anabela Viegas, Ana Isabel Martins, Sónia Brandão, Raquel M. Silva, Cláudio Teixeira, Vahid Mokhtari, Abbas Abdolmaleki, Nima Shafii, Mohammadreza Kamali, Burkhard Morgenstern, Thomas Dencker, Peter Meinicke and Nafise Jafarzadeh, with whom I have shared unforgettable moments.

Last but not least, I owe so much thanks to my family, parents, brothers, sister, sisters-in-law and nieces who were continuously supporting me throughout my life.

Palavras-chave

Compressão relativa, Modelos de Markov, compressão de dados ómicos, encriptação de dados ómicos, metodo livre de alinhamentos, rearranjo genómico, genómica comparativa

Resumo

O crescente crescimento do desenvolvimento de tecnologias de sequenciamento de alto rendimento e, como consequência, a geração de um enorme volume de dados, revolucionou a pesquisa e descoberta biológica. Motivados por isso, nesta tese investigamos os métodos que fornecem uma representação eficiente de dados ómicos de maneira compactada ou criptografada e, posteriormente, os usamos para análise.

Em primeiro lugar, descrevemos uma série de medidas com o objetivo de quantificar informação em e entre sequências ómicas. Em seguida, apresentamos modelos de contexto finito (FCMs), modelos de Markov tolerantes a substituição (STMMs) e uma combinação dos dois, especializados na modelagem de dados biológicos, para compactação e análise de dados.

Para facilitar o armazenamento do dilúvio de dados acima mencionado, desenvolvemos dois compressores de dados sem perda para dados genómicos e um para dados proteómicos. Os métodos funcionam com base em (a) uma combinação de FCMs e STMMs ou (b) na combinação mencionada, juntamente com modelos de repetição e um modelo de previsão competitiva. Testados em vários dados sintéticos e reais mostraram a sua eficiência sobre os métodos do estado-de-arte em termos de taxa de compressão.

A privacidade dos dados genómicos é um tópico recentemente focado nos desenvolvimentos do campo da medicina personalizada. Propomos uma ferramenta capaz de representar dados genómicos de maneira criptografada com segurança e, ao mesmo tempo, compactando as sequências FASTA e FASTQ para um fator de três. Emprega criptografia AES acompanhada de um mecanismo de embaralhamento para melhorar a segurança dos dados. Os resultados mostram que é mais rápido que os algoritmos de uso geral e específico.

As técnicas de compressão podem ser exploradas para análise de dados ómicos. Tendo isso em mente, investigamos a identificação de regiões únicas em uma espécie em relação a espécies próximas, que nos podem dar uma visão das características evolutivas. Para esse fim, desenvolvemos duas ferramentas livres de alinhamento que podem encontrar e visualizar com precisão regiões distintas entre duas coleções de sequências de DNA ou proteínas. Testados em humanos modernos em relação a neandertais, encontrámos várias regiões ausentes nos neandertais que podem expressar novas funcionalidades associadas à evolução dos humanos modernos.

Por último, investigamos a identificação de rearranjos genómicos, que têm papéis importantes em desordens genéticas e cancro, empregando uma técnica de compressão. Para esse fim, desenvolvemos uma ferramenta capaz de localizar e visualizar com precisão os rearranjos em pequena e grande escala entre duas sequências genómicas. Os resultados da aplicação da ferramenta proposta, em vários dados sintéticos e reais, estão em conformidade com os resultados parcialmente relatados por abordagens laboratoriais, por exemplo, análise FISH.

Keywords Relative compression, Markov model, omics data compression, omics data encryption, alignment-free method, genomic rearrangement, comparative genomics

Abstract

The ever-increasing growth of the development of high-throughput sequencing technologies and as a consequence, generation of a huge volume of data, has revolutionized biological research and discovery. Motivated by that, we investigate in this thesis the methods which are capable of providing an efficient representation of omics data in compressed or encrypted manner, and then, we employ them to analyze omics data.

First and foremost, we describe a number of measures for the purpose of quantifying information in and between omics sequences. Then, we present finite-context models (FCMs), substitution-tolerant Markov models (STMMs) and a combination of the two, which are specialized in modeling biological data, in order for data compression and analysis.

To ease the storage of the aforementioned data deluge, we design two lossless data compressors for genomic and one for proteomic data. The methods work on the basis of (a) a combination of FCMs and STMMs or (b) the mentioned combination along with repeat models and a competitive prediction model. Tested on various synthetic and real data showed their outperformance over the previously proposed methods in terms of compression ratio.

Privacy of genomic data is a topic that has been recently focused by developments in the field of personalized medicine. We propose a tool that is able to represent genomic data in a securely encrypted fashion, and at the same time, is able to compact FASTA and FASTQ sequences by a factor of three. It employs AES encryption accompanied by a shuffling mechanism for improving the data security. The results show it is faster than general-purpose and special-purpose algorithms.

Compression techniques can be employed for analysis of omics data. Having this in mind, we investigate the identification of unique regions in a species with respect to close species, that can give us an insight into evolutionary traits. For this purpose, we design two alignment-free tools that can accurately find and visualize distinct regions among two collections of DNA or protein sequences. Tested on modern humans with respect to Neanderthals, we found a number of absent regions in Neanderthals that may express new functionalities associated with evolution of modern humans.

Finally, we investigate the identification of genomic rearrangements, that have important roles in genetic disorders and cancer, by employing a compression technique. For this purpose, we design a tool that is able to accurately localize and visualize small- and large-scale rearrangements between two genomic sequences. The results of applying the proposed tool on several synthetic and real data conformed to the results partially reported by wet laboratory approaches, e.g., FISH analysis.

Contents

1 Introduction **1**

1.1 Next-generation sequencing . . . 2

1.2 Storage of omics data . . . 3

1.3 Omics data compression . . . 6

1.4 Omics data encryption . . . 8

1.5 Omics data analysis . . . 9

1.6 Outline . . . 10

1.7 Contributions . . . 10

1.7.1 Publications . . . 11

1.7.2 Software . . . 12

2 Measures and models **13**

2.1 Measures for quantifying information . . . 13

2.1.1 Introduction . . . 14

2.1.2 Normalized information distance . . . 15

2.1.3 Normalized compression distance . . . 16

2.1.4 Normalized conditional compression distance . . . 17

2.1.5 Normalized relative compression . . . 17

2.1.6 Normalized compression . . . 18

2.2 Compression models . . . 18

2.2.1 Finite-context model (FCM) . . . 18

2.2.2 Substitution-tolerant Markov model (STMM) . . . 19

2.2.3 Cooperation of FCMs and STMMs . . . 20

2.2.4 FCMs compared to cooperation of FCMs and STMMs . . . 22

2.3 Application on quantifying inverted repeats . . . 24

2.4 Conclusions . . . 29

3 Compression of omics data **31**

3.1 Compression of genomic sequences . . . 32

3.1.1 Introduction . . . 32

3.1.2 GeCo2 . . . 34

3.1.3 Jarvis . . . 39

3.2 Compression of amino acid sequences . . . 48

3.2.1 Introduction . . . 48

3.2.2 Methods . . . 50

3.2.3 Results and discussion . . . 50

3.3 Conclusions . . . 55

4 Secure encryption of genomic data **57**

4.1 Introduction . . . 57

4.2 Methods . . . 58

4.2.1 Pack and unpack . . . 59

4.2.2 Shuffle and unshuffle . . . 60

4.2.3 Encrypt and decrypt . . . 65

4.3 Results and discussion . . . 66

4.3.1 Experiment setup . . . 66

4.3.2 Compare with compression and encryption methods . . . 69

4.3.3 Run with different number of threads . . . 73

4.3.4 Explore redundancy . . . 75

4.4 Conclusions . . . 77

5 Finding and visualization of distinct regions in omics sequences **79**

5.1 Genomic level . . . 80

5.1.1 Introduction . . . 80

5.1.2 Methods . . . 80

5.1.3 Results and discussion . . . 82

5.2 Proteomic level . . . 85

5.2.1 Introduction . . . 85

5.2.2 Methods . . . 85

5.2.3 Results and discussion . . . 86

5.3 Conclusions . . . 90

6 Detection and visualization of genomic rearrangements **93**

6.1 Introduction . . . 93

6.2 Methods . . . 95
6.2.1 Data modeling . . . 96
6.2.2 Storing models in memory . . . 96
6.2.3 Finding similar regions . . . 97
6.2.4 Computing complexity . . . 98
6.3 Results and Discussion . . . 100
6.3.1 Dataset . . . 101
6.3.2 Application on synthetic data . . . 101
6.3.3 Application on real data . . . 101
6.3.4 Comparison to Smash . . . 106
6.3.5 Robustness against fragmented data . . . 111
6.3.6 Benchmarking . . . 111
6.4 Conclusions . . . 115

7 Conclusions **117**

Bibliography **119**

List of Figures

1.1 Double helix structure of DNA. Credit: Nature Education [1]. 2

1.2 The cost of sequencing a human genome. Credit: National Human Genome Research Institute. 3

1.3 A sample FASTQ file. 5

1.4 A sample FASTA file. 5

1.5 A sample SAM file. 6

1.6 A sample VCF file. 7

1.7 The scheme of a data compressor. 7

1.8 The encoding process of arithmetic coder for the input $a_1 a_2 a_3$ [35]. 8

2.1 Relation between the Kolmogorov complexity, conditional Kolmogorov complexity, conjoint Kolmogorov complexity and algorithmic mutual information. 15

2.2 Algorithm for enabling/disabling an STMM. 20

2.3 (a) Cooperation of FCMs and STMMs. Note that each STMM needs to be associated with an FCM; (b) probability of an input symbol is estimated by employing the probability and weight values that have been obtained from processing previous symbols. 21

2.4 The relation between context-order sizes (k), forgetting factors and complexity (information content). For the experiment, we compressed 10 synthetic sequences with the sizes of 500 kb to 20 Mb, with different redundancies, and calculated the average information content. 22

2.5 Compression of a synthetic target sequence relatively to a reference using (a) FCMs; and (b) cooperation of FCMs and STMMs. 23

2.6 Compression of a synthetic target sequence relatively to a reference in presence of mutations, employing (a) FCMs; and (b) cooperation of FCMs and STMMs. 23

2.7 NRC values calculated by compression of chimpanzee and human chromosomes. (a), (b) IRs applied (IR = 0) and not applied (IR = 1); and (c), (d) the difference in NRCs between not applying and applying IRs ($NRC_{IR=0} - NRC_{IR=1}$). 26

2.8 NRC results concerned with compression of gorilla using human chromosomes as references. (a) left: IR = 0 and right: IR = 1; and (b) the difference between NRCs. 27

2.9 NRC values associated with the compression of turkey and chicken chromosomes. (a), (b) With and without IRs; and (c), (d) the difference between NRCs. 28

3.1 Cache-hash data structure. 35

3.2 An example of a competitive prediction between five weighted context models and three weighted stochastic repeat models. 39

3.3 Example of a repeat model. The base marked with ? is intended to be encoded. The dashed arrows show failure in prediction by the model. 40

3.4 The hash table constructed by the repeat model. 41

3.5 Example of a competitive prediction context model with the context order size of five. 42

3.6 Bit-rates, in bps, for compressing four sequences of HoSa, EnIn, AeCa and YeMi, when applying CPCM with different context order sizes. 43

3.7 Applying state-of-the-art compressors on 15 sequences (described in Table 3.2). (a) compression ratio; (b) compression speed, in kilobase per second. Note that CoGI was an outlier, therefore, it was removed from this figure. 47

3.8 Running Jarvis with different modes on (a) HoSa; (b) GaGa; and (c) DaRe. 47

3.9 Times of carrying out GeCo2 and Jarvis in all modes on a sequence including the human Y-chromosome and a very repetitive sequence. 48

3.10 (a) Bit-rates; (b) times; (c) memory usages; (d) bit-rates versus times of AC and other protein compressors, obtained by testing on 16 different sequences. 53

3.11 Normalized compression results of AC compressing 10,677 proteins, described in Table 3.13. . 54

3.12 Histogram of sizes for the datasets described in Table 3.13. 54

3.13 (a) Violin plot of NCs and (b) average NCs for the sequences described in Table 3.13. 55

4.1 The schema of Cryfa. (a) The process of compaction & encryption of a FASTA/FASTQ file. Dashed lines show that quality scores are not considered for FASTA files; (b) decryption & unpacking of a file that has already been compacted & encrypted by Cryfa. 59

4.2 An example of packing DNA bases. 60

4.3 An example to show the importance of applying shuffling in Cryfa, in which an attacker tries exhaustive password search to break the encryption. 61

4.4 An example to show the importance of shuffling in Cryfa. An attacker downloads the entire sequences in NCBI database and encrypts them. By comparing each encrypted sequence with the target file, the attacker tries to break the encryption. 62

4.5 An example of shuffling and unshuffling. The uniform pseudo-random number generators in shuffle and unshuffle blocks need to employ the same seed. 64

4.6 The authenticated encryption operation by AES method (GCM). 65

4.7 Total time and file size to encrypt / compact & encrypt and decrypt / decrypt & unpack the whole FASTA and FASTQ datasets by Cryfa and AES Crypt, a general-purpose encryption tool. **(a)** real time, **(b)** file size. CR stands for compression ratio. 71

4.8 Total time and file size for compaction & encryption and decryption & unpacking of the entire FASTA and FASTQ datasets obtained by different methods. (a) real times; (b) file sizes. CR stands for compression ratio. 76

4.9 Time and memory used by Cryfa when running with different number of threads, on a 338 MiB FASTA file (viruses.fasta, from viruses species) as well as a 1.2 GiB FASTQ file (DS-B1088_SR.fastq, from Denisova subspecies). (a) real times and CPU times, that is user time plus system time; (b) memory usage. 77

4.10 Normalized compression (NC) values obtained by running Cryfa, DELIMINATE and MFCompress on several genomic sequences. 78

5.1 Algorithm of the proposed method for finding and visualizing distinct regions between two collections of omics data. 82

5.2 Regions in modern human chromosomes that do not exist in Neanderthal genomes. "Enlarge" shows the number of times that a region is enlarged, for the visualization purpose. 83

5.3 Uniqueness ratios for different rates of mutation and different k-mer sizes applied to synthetic and real (Neanderthals) datasets. 87

5.4 (a) Distribution of uniqueness ratios; (b) total uniqueness ratios; (c) probability of a target word being seen in the reference, for different k-mers. 88

5.5 Modern human proteins with the most distinct regions against Altai, Sidron and Vindija Neanderthals. The format m / n shows that considering k-mer size of 7, m out of n amino acids are relatively unique. 90

6.1 The schema of Smash++. The process of finding similar regions in reference and target sequences and computing the redundancy in each region includes eight stages. Smash++ outputs a *.pos file that includes the positions of the similar regions, and can be then visualized as an SVG image. 95

6.2 The data structures used by Smash++ to store the models in memory. (a) table of 64 bit counters that uses up to 128 MiB of memory, (b) table of 32 bit counters that consumes at most 960 MiB of memory, (c) table of 8 bit approximate counters with memory usage of up to 1 GiB and (d) Count-Min-Log sketch of 4 bit counters which consumes up to $\frac{1}{2} w \times d$ B of memory, e.g., if $w = 2^{30}$ and $d = 4$, it uses 2 GiB of memory. 97

6.3 Approximate counting update and query. 98

6.4 Count-Min-Log Sketch update and query. 99

6.5 Finding similar regions in reference and target sequences. Smash++ finds, first, the regions in the target that are similar to the reference, and then, finds the regions in the reference similar to the detected target regions. This procedure is performed for both regular and inverted homologies. 100

6.6 Similarities between synthetic sequences with different sizes, detected by Smash++. The parameters used are k-mer size = 14 and number of substitutions in STMM = 5, which are the default parameters used by Smash++. For the threshold, the default value of 1.5 and 1.97 are used for panels a-d and e, respectively. (a) 1.5 kb sequences; (b) 100 kb sequences. No similarity is detected for part II of the reference, since it is mutated 90%. Parts III and IV of the reference and I and II of the target are joined, since there is no space between consecutive regions; (c) 5 Mb sequences; (d) 100 Mb sequences; (e) 60 kb sequences. Roughly 43% of mutation is detected. 103

6.7 Similarities in a real dataset, detected by Smash++. (a) *G. gallus* chr. 18 and *M. gallopavo* chr. 20. The parameters were k-mer size = 14, No. substitutions in STMM = 5, threshold = 1.9 and min block size (m) = 500,000, i.e., the regions smaller than 500,000 bases were not considered for further processing; (b) *G. gallus* chr. 14 and *M. gallopavo* chr. 16. The result is obtained by setting k = 14, No. substitutions = 5, threshold = 1.95 and m = 400,000; (c) *H. sapiens* chr. 12 and *P. troglodytes* chr. 12. The parameters were k = 14, without using STMM, threshold = 1.9 and m = 100,000; (d) *X. oryzae* pv. *oryzae* PXO99A and *X. oryzae* pv. *oryzae* MAFF 311018 (two rice pathogens). The result obtained by setting k = 13, threshold = 1.55 and m = 10,000. 104

6.8 Pair-wise comparison of *G. gallus* chr. 18 and *M. gallopavo* chr. 20. (a) Smash++, with $k = 14$ and 5 used by an FCM and an STMM, respectively. The blocks smaller than 500 kb are discarded; (b) progressiveMauve [280], with LCB (locally collinear block) weight of 18,692. Reverse complements are shown in lower level; (c) adopted from [297], which is confirmed by FISH analysis. The box shows a local rearrangement; (d) SynBrowser [284], with the resolution of 150 kb (minimum size of a reference block); (e) adopted from [133], which confirms an inversion rearrangement of size ~5 Mb by FISH analysis. 105

6.9 *G. gallus* chr. 14 compared to *M. gallopavo* chr. 16. (a) Smash++, employing an FCM and an STMM with $k = 14$ and 5, respectively. The blocks smaller than 400 kb are discarded; (b) progressiveMauve, with LCB weight of 27,424; (c) adopted from [297]. The box shows an inversion rearrangement; (d) SynBrowser, with the resolution of 150 kb. 106

6.10 Comparison of *H. sapiens* chr. 12 and *P. troglodytes* chr. 12. (a) Smash++, with $k = 14$ used by an FCM. The blocks smaller than 100 kb are discarded; (b) progressiveMauve, with LCB weight of 55,186; (c) Cinteny [285], with minimum length of synteny block = 1 kb, maximum gap between adjacent markers = 5 Mb and minimum number of markers = 1; (d) SynBrowser, with the resolution of 150 kb; (e) D-GENIES [298], in "strong precision" mode. 107

6.11 Pair-wise comparison of PXO99A and MAFF 311018. (a) Smash++, with $k = 13$ used by an FCM. The blocks smaller than 10 kb are discarded. In order to make the figure clearer, the shaded paths for connecting corresponding regions are not drawn; (b) progressiveMauve, with LCB weight of 3926; (c) adopted from [294], which employs an alignment-based method to obtain this dot plot. The blue and red colors shows regions of PXO99A that align to the same or opposite strand of MAFF 311018, respectively. 108

6.12 Comparison of Smash++ and Smash on the synthetic dataset. Using cooperation of an FCM and an STMM (in Smash++) produces more accurate results rather than using a single FCM (in Smash). 109

6.13 Comparison of Smash++ and Smash on the real dataset, including *S. cerevisiae* chr. VII and *S. paradoxus* chr. VII. The rearrangements maps clearly show the improvement made over Smash, using an FCM along with an STMM. 110

6.14 Similarity of a target sequence to a reference sequence, when the reference is permuted by blocks of different sizes of 450 kb, 30 kb, 1 kb and 30 b. To run Smash++, an FCM with k-mer size of 14 and the threshold of 1.5 was used. It shows the proposed method is resistant to fragmentation of sequence; for example, in the case of 30 b, although the reference is highly fragmented, Smash++ could detect the same three similar regions in the target as the original case. 112

6.15 (a) The peak memory consumption, in gigabytes; and (b) the elapsed (wall clock) time usage, in minutes, of Smash++ by running on all synthetic and real datasets described in Table 6.1. . 113

6.16 Comparison of Smash++ and Smash, in terms of (a) memory usage; and (b) time usage, running on real and synthetic data described in Table 6.1. To have a fair comparison, only one model (FCM) is used by Smash++, and also self-complexity is not computed. Diamonds indicate the mean, and bars, the ranges from minimum to maximum values. 114

List of Tables

1.1 Comparison of various high-throughput sequencing methods. . . . 4
1.2 Mandatory fields in SAM file format[1]. . . . 6
1.3 Mandatory fields in VCF file format[1]. . . . 7
1.4 List of software developed in the dissertation. . . . 12

2.1 Compression results for real dataset. . . . 24
2.2 Dataset used for studying the role of inverted repeats on DNA sequence similarity. . . . 25

3.1 Specification of 12 pre-computed modes of GeCo2. . . . 36
3.2 Dataset used for benchmarking genomic sequence compressors. . . . 37
3.3 Compressed file sizes, in bytes, obtained by GeCo2 and state-of-the-art genomic data compressors. . . . 38
3.4 Computation times, in seconds, of applying GeCo2 and other genomic sequence compressors on the dataset described in Table 3.2. . . . 38
3.5 Specification of 15 running modes of Jarvis. . . . 44
3.6 Compressed file sizes, in bytes, obtained by Jarvis and other state-of-the-art data compressors. 46
3.7 Computational time, in seconds, of applying Jarvis and other compressors on the dataset. . . . 46
3.8 Representation of amino acids and possible DNA codons associated with them, along with their distribution percentage in two large protein databases. . . . 49
3.9 Datasets used for comparing AC with other compressors. . . . 51
3.10 Bit-rates of AC and existing protein compressors, in bits per symbol (bps). . . . 51
3.11 Compression time of different compressors, in milliseconds. . . . 52
3.12 Memory usage of different protein compressors, in megabytes. . . . 52
3.13 Datasets used exclusively by AC. . . . 54

4.1 Datasets for compression and encryption experiments, in FASTA and FASTQ formats. . . . 68
4.2 Datasets used exclusively for encryption experiments. . . . 69
4.3 Datasets for redundancy exploration experiments. . . . 69
4.4 Comparing Cryfa and AES Crypt, as a general-purpose encryption tool, running on FASTA and FASTQ datasets. . . . 70

4.5 Cryfa compared to AES Crypt, running on VCF, SAM and BAM datasets. 72

4.6 Compression & encryption and decryption & decompression of FASTA datasets. Note that for all methods, except Cryfa, the compressed file is encrypted with AES Crypt method. 73

4.7 Compression & encryption and decryption & decompression of modern human FASTQ dataset. 74

4.8 Compression & encryption and decryption & decompression of Denisova and synthetic FASTQ datasets. 75

5.1 Genes present in at least half of the regions that exist in modern human chromosomes and do not exist in Neanderthal, considering the threshold of 0.90. 84

5.2 Datasets used by FRUIT, including synthetic and real data from Neanderthals and modern human. 87

5.3 The most unique proteins of modern human absent in the Neanderthals. 89

5.4 The most similar non-human primate exomes to modern human, that are listed in Table 5.3. . 91

6.1 Synthetic and real dataset used in the experiments. 102

6.2 Performance of Smash++, in terms of memory and time usage, running on all synthetic and real datasets described in Table 6.1. 114

6.3 The memory and time usage of Smash++ and Smash, running on synthetic and real dataset (Table 6.1). To have a fair comparison, Smash++ uses only one model (FCM), as Smash does. . 115

List of Abbreviations

AES	Advanced encryption standard
AF	Alignment-free
BAM	Binary alignment map
BGZF	Blocked GNU Zip format
BPS	Bits per symbol
CD	Compression distance
CMLS	Count-min-log sketch
CPCM	Competitive prediction context model
DNA	Deoxyribonucleic acid
FCM	Finite-context model
FISH	Fluorescence *in situ* hybridization
GCM	Galois/counter mode of AES
GGA	*Gallus gallus* (chicken)
GiB	Gigabyte
HS	*Homo sapiens* (human)
HTS	High-throughput sequencing
ID	Information distance
IR	Inverted repeat
kb	Kilobase
KiB	Kilobyte
KPA	Known-plaintext attack
LCB	Locally collinear block
Mb	Megabase
MGA	*Meleagris gallopavo* (turkey)
MiB	Megabyte
Mya	Million years ago
NC	Normalized compression
NCCD	Normalized conditional compression distance
NCD	Normalized compression distance
NGS	Next-generation sequencing
NID	Normalized information distance
NRC	Normalized relative compression

PCR	Polymerase chain reaction
PRNG	Pseudo-random number generator
PT	*Pan troglodytes* (chimpanzee)
SAM	Sequence alignment map
STMM	Substitution-tolerant Markov model
SVG	Scalable vector graphics
UI	User interface
VCF	Variant call format

Chapter 1

Introduction

DNA (deoxyribonucleic acid) sequencing is a process in which the physical order of nucleotides contained in a DNA molecule is determined. The canonical structure of DNA includes four nucleotides: adenine (A), cytosine (C), guanine (G) and thymine (T), which is shown in Fig. 1.1. By sequencing DNA of any organism, the sequence of individual genes, clusters of genes, full chromosomes, or entire genomes can be determined.

DNA sequencing has a broad application to the fields including, but not limited to, (a) molecular biology, to study genomes and the proteins encoded by them [2]; (b) evolutionary biology, to study the relation between different organisms and their evolution [3]; (c) medicine, to perform genetic testing for determining risk of genetic diseases [4] and also, determining a specific bacteria to provide a more precise antibiotics treatments [5]; (d) metagenomics, to study on identification of organisms that are present in samples from particular environments [6]; (e) forensics, for paternity testing [7] and forensic identification [8]; and (f) paleogenomics, to analyze the genomic information concerned with extinct species [9], [10].

In 1970 the fist method for determining the sequence of DNA was developed, by employing a location-specific primer extension strategy [11]. Sanger *et al.* in 1977 adopted this strategy and developed a faster method for sequencing with chain-terminating inhibitors, based on the DNA polymerase [12]. At the same time, Gilbert and Maxam developed a method for sequencing by chemical degradation [13]; however, it involved employing hazardous chemicals and had a more complex process rather than the Sanger method. Over the next years different sequencing methods were developed [14]–[16] and the first automated sequencing machine, ABI 370, was marketed by Applied Biosystems [16].

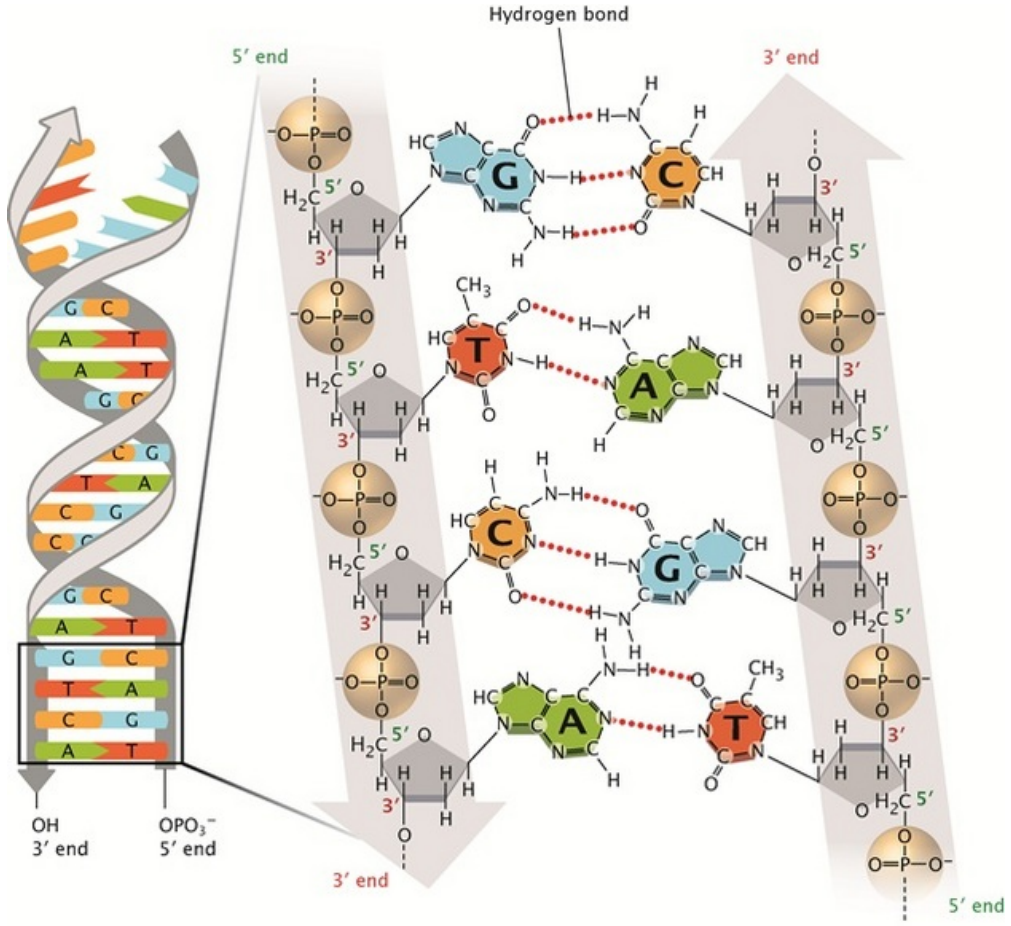

Figure 1.1: Double helix structure of DNA. Credit: Nature Education [1].

1.1 Next-generation sequencing

In the 1990s, various DNA sequencing methods were proposed and later employed in commercial sequencers. They were called "next-generation sequencing" (NGS) methods, since they had some characteristics that were absent in earlier methods mentioned above, e.g., being highly scalable and being able to sequence the entire genome at once.

In 1990, "base-by-base" sequencing method was patented that involved removable 3′ blockers on DNA arrays [17]. In 1996, pyrosequencing was introduced which was a real-time method based on "sequencing by synthesis" [18]. "DNA colony sequencing" method was patented in 1998, that involved random surface-polymerase chain reaction (PCR) arraying [19]. In 2000, the first commercial "next-generation" sequencing method, called massively parallel signature sequencing (MPSS), was introduced by Lynx Therapeutics [20].

Next-generation or "high-throughput" sequencing (HTS) technologies are able to parallelize the sequencing of billions of fragments of DNA, which has led to faster and cheaper sequencing rather than the first-generation sequencing [21], [22]. As a comparison, it took 13 years to the Human

Genome Project (HGP) to provide the final draft of an entire human genome in 2003 [23], while today, HTS have enabled us to perform such sequencing in less than one day [24]. Also, the cost of sequencing human genome drastically reduced from $40,000,000 in 2003 to $1000 in 2019 (see Fig. 1.2). A comparison of HTS methods is provided in Table 1.1.

1.2 Storage of omics data

The biological data produced by high-throughput sequencing technologies can be stored in different file formats, among which FASTQ, FASTA, SAM and VCF are the most common ones that represent data in an ASCII character-encoding scheme. The raw sequencing data, generated by a sequencing instrument, is stored in FASTQ format which contains multiple *reads*. In each read, there is a *nucleotide sequence*, a *quality score* associated with each base, that shows the sequencing quality, and some extra information. After generating the raw data, it is typically desirable to align the reads to a reference sequence, that is to find the location(s) in the reference sequence that correspond(s) to the nucleotide sequence of a read. These locations along with the mismatching information and some extra fields are stored in SAM format. From aligned data, FASTA files can be stored, which include only *headers* and nucleotide sequences. By analyzing the aligned data, stored in the SAM file, the variants (substitutions, insertions and deletions) between the original and the reference genome can be obtained. These variants along with their position and some extra information is stored in VCF format. In the followings, we describe in detail the mentioned formats.

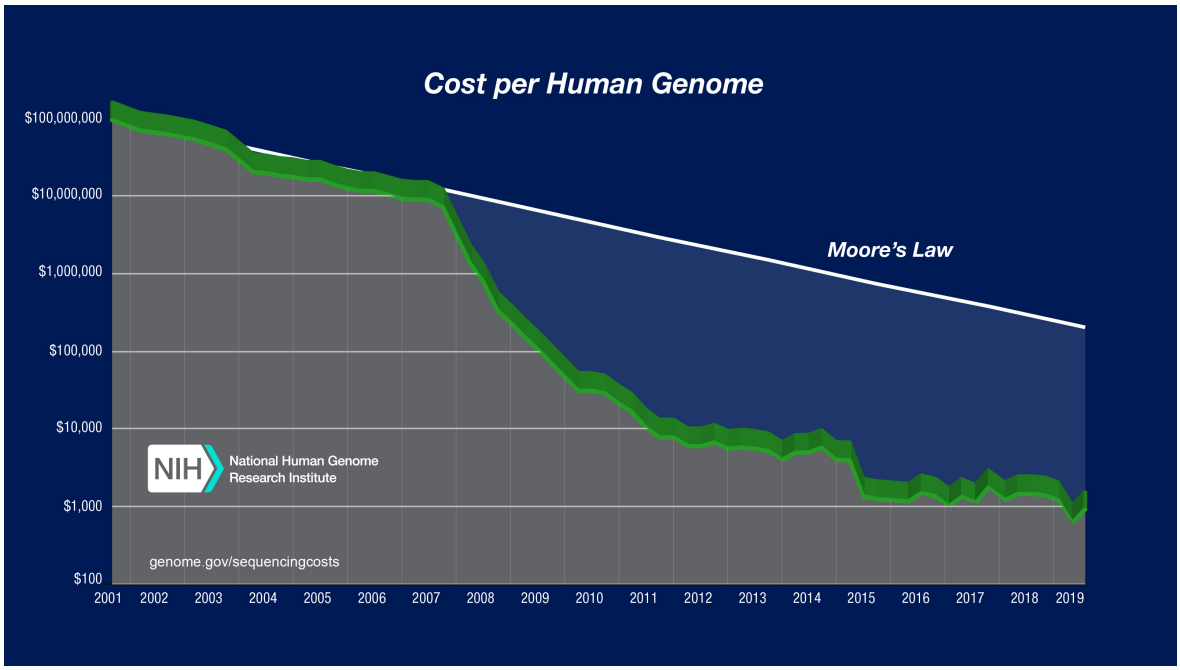

Figure 1.2: The cost of sequencing a human genome. Credit: National Human Genome Research Institute.

Table 1.1: Comparison of various high-throughput sequencing methods.

Platform	Read length	Accuracy	Reads per run	Time per run	Cost per 1 Gbp (US$)
Pacific Biosciences	30,000 b (N50) max > 100 kb[1,2,3]	87% raw-read [25]	4 M per Sequel 2 SMRT cell, 100–200 gigabases[1,4] [26]	30 min–20 h[1] [27]	$7.2–$43.3
Ion Torrent	< 600 b[5]	99.6%[6]	< 80 M	2 h	$66.8–$950
Roche 454	700 b	99.9%	1 M	24 h	$10,000
Illumina	MiniSeq, NextSeq: 75–300 b MiSeq: 50–600 b HiSeq 2500: 50–500 b HiSeq 3/4000: 50–300 b HiSeq X: 300 b	99.9% (Phred30)	MiniSeq/MiSeq: 1–25 M NextSeq: 130–400 M HiSeq 2500: 300 M–2 G HiSeq 3/4000: 2.5 G HiSeq X: 3 G	1–11 days [28]	$5–$150
cPAS-BGI/MGI	BGISEQ-50: 35–50 b[7] MGISEQ 200: 50–200 b[7] BGISEQ-500: 50–300 b[7] MGISEQ-2000: 50–300 b[7]	99.9% (Phred30)	BGISEQ-50: 160 M MGISEQ 200: 300 M BGISEQ-500: 1300 M per flow cell MGISEQ-2000: 375 M FCS flow cell, 1500 M FCL flow cell per flow cell.	1–9 days	$5–$120
SOLiD	50 + 35 or 50 + 50 b	99.9%	1.2–1.4 G	1–2 weeks	$60–$130
Nanopore	Dependent on library preparation, not the device, so user chooses read length (< 2,272,580 b [29])	~92–97% single read	dependent on read length selected by user	data streamed in real time. Choose 1 min–48 h	$7–$100
GenapSys	~150 b single-end	99.9% (Phred30)	1–16 M	~24 h	$667
Sanger	400–900 b	99.9%	N/A	20 min–3 h	$2,400,000

[1] www.pacb.com/blog/new-software-polymerase-sequel-system-boost-throughput-affordability
[2] www.genomeweb.com/sequencing/after-year-testing-two-early-pacbio-customers-expect-more-routine-use-rs-sequenc#.XwdmoHUzbEp
[3] www.globenewswire.com/news-release/2013/10/03/577891/10051072/en/Pacific-Biosciences-Introduces-New-Chemistry-With-Longer-Read-Lengths-to-Detect-Novel-Features-in-DNA-Sequence-and-Advance-Genome-Studies-of-Large-Organisms.html
[4] www.flxlexblog.wordpress.com/2013/07/05/de-novo-bacterial-genome-assembly-a-solved-problem
[5] www.thermofisher.com/order/catalog/product/A30670#/A30670
[6] www.web.archive.org/web/20180330075720/http://129.130.90.13/ion-docs/GUID-C6419130-57D8-4DE2-BCF8-47157CB3C9A2.html
[7] en.mgitech.cn/products

FASTQ is a text-based format which represents biological sequences and their corresponding quality scores with ASCII characters. This file format is considered as the *de facto* standard to store the output of high-throughput sequencing instruments [30]. A sequence in FASTQ format contains four lines: (a) the sequence identifier, that begins with a "@" character; (b) the raw sequence letters; (c) a "+" character, optionally followed by the same sequence identifier in (a); and (d) the quality scores for the sequence in (b), that has necessarily the same number of symbols as in the sequence (see Fig. 1.3). The ".fq" and ".fastq" file extensions are commonly used to store a FASTQ file.

FASTA is a text-based format which represents nucleotides or amino acids with single-letter codes [31]. A sequence in this format has two parts: (a) a single-line description, that starts with the ">" symbol; and (b) the sequence data. As shown in Fig. 1.4, a file in this format can contain concatenation of different single sequence files. Filename extensions associated with this format include, but are not limited to, ".fasta", ".fna", ".ffn", ".faa" and ".frn".

SAM (Sequence Alignment Map) is a text-based format to store biological sequences that are aligned to a reference sequence. As shown in Fig. 1.5, a sequence in this format contains two sections: (a) the header, that starts with an "@" character; and (b) the alignment section, that has 11 mandatory fields (see Table 1.2). It should be mentioned that the binary version of a SAM file is a "BAM" (Binary Alignment Map) file, that is generated by compressing the SAM file in BGZF (Blocked GNU Zip Format) [32]–[34]. SAM and BAM files can be saved in ".sam" and ".bam" formats, respectively.

VCF (Variant Call Format) is used to store genomic sequence variations in a text file. As Fig. 1.6 shows, a sequence in this format consists of three sections: (a) meta-information lines, that begin with "##"; (b) a header line, prefixed with "#"; and (c) data lines, that record sample(s) information

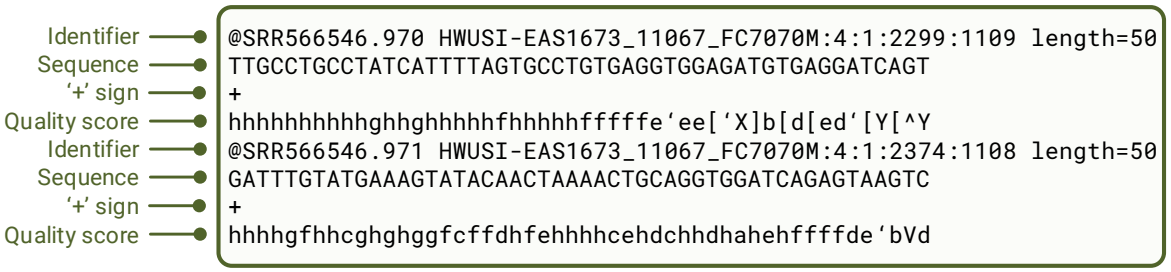

Figure 1.3: A sample FASTQ file.

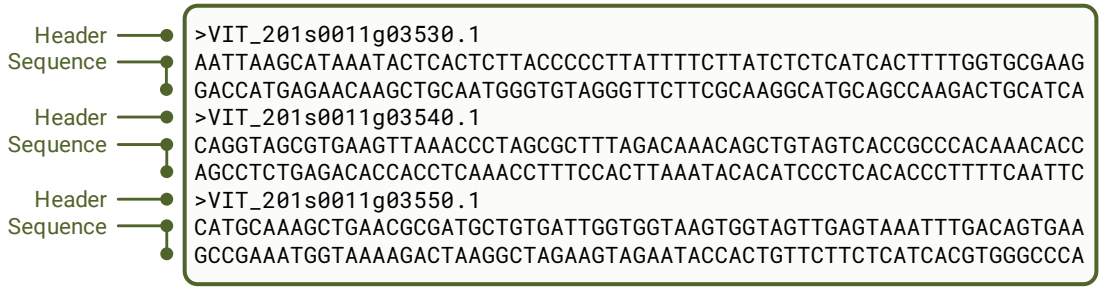

Figure 1.4: A sample FASTA file.

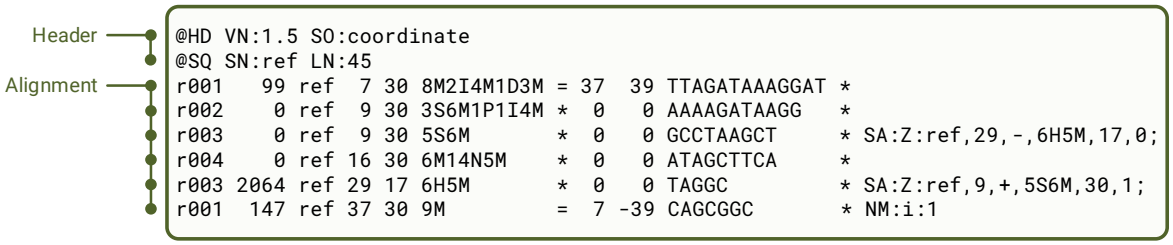

Figure 1.5: A sample SAM file.

Table 1.2: Mandatory fields in SAM file format[1].

Column	Field	Type	Description
1	QNAME	String	Query template NAME
2	FLAG	Integer	bitwise FLAG
3	RNAME	String	References sequence NAME
4	POS	Integer	1-based leftmost mapping POSition
5	MAPQ	Integer	MAPping Quality
6	CIGAR	String	CIGAR String
7	RNEXT	String	Ref. name of the mate/NEXT read
8	PNEXT	Integer	Position of the mate/NEXT read
9	TLEN	Integer	observed Template LENgth
10	SEQ	String	segment SEQuence
11	QUAL	String	ASCII of Phred-scaled base QUALity + 33

[1] samtools.github.io/hts-specs/SAMv1.pdf

and are tab-separated into at least eight columns (Table 1.3). VCF files can be saved in ".vcf" format.

1.3 Omics data compression

With the advancements in high-throughput sequencing technologies, a large amount of biological data is produced, that needs to be stored, processed and transmitted. *Data Compression* can tackle these challenges by reducing the size of storage and cost of processing and transmission. A compression algorithm can be either (a) lossless, in which the compressed data can be perfectly reconstructed to the original data, or (b) lossy, where the decompressed data is an approximation of the original data. Lossy compressors can, usually, provide better compression at the expense of loosing information. In this thesis we focus on the lossless approach, since the lossy one is not able to approximate the Kolmogorov complexity without overestimation.

A data compressor can be seen as the combination of a mathematical *model* and a *coder* (Fig. 1.7). To have an efficient compressor, the model should represent the data efficiently. In other words, the better is the model, the lower number of bits is required to store the compressed data [35]. There exist general-purpose algorithms, e.g., gzip, bzip2 and 7zip, that do not consider characteristics of biological sequences, e.g., inverted repeats (IRs); therefore, such methods cannot model the biological data efficiently and lead to low compression gains [36], [37]. Special-purpose algorithms, however,

Meta-info
Header
Data

```
##fileformat=VCFv4.3
##INFO=<ID=NS,Number=1,Type=Integer,Description="Number of Samples With Data">
##INFO=<ID=AF,Number=A,Type=Float,Description="Allele Frequency">
##INFO=<ID=AA,Number=1,Type=String,Description="Ancestral Allele">
##FORMAT=<ID=GT,Number=1,Type=String,Description="Genotype">
##FORMAT=<ID=GQ,Number=1,Type=Integer,Description="Genotype Quality">
##FORMAT=<ID=DP,Number=1,Type=Integer,Description="Read Depth">
#CHROM POS   ID        REF ALT QUAL FILTER INFO          FORMAT   NA00001
20     1437  rs6054257 G   A   29   PASS   NS=3;AF=0.5   GT:GQ:DP 0|0:48:1:51,51
20     1733  .         T   A   3    q10    NS=3;AF=0.017 GT:GQ:DP 0|0:49:3:58,50
20     11106 rs6040355 A   G,T 67   PASS   NS=2;AA=T     GT:GQ    1|2:21:6:23,27
20     12327 .         T   .   47   PASS   NS=3;AA=T     GT:GQ    0|0:54:7:56,60
```

Figure 1.6: A sample VCF file.

Table 1.3: Mandatory fields in VCF file format[1].

Column	Field	Type	Description
1	CHROM	String	sequence name (typically a CHROMosome) on which the variation is being called
2	POS	Integer	1-based POSition of the variation on the given sequence
3	ID	String	IDentifier of the variation
4	REF	String	REFerence base(s) at the given position on the given reference sequence
5	ALT	String	list of ALTernative alleles at this position
6	QUAL	String	Phred-scaled QUALity score for the call
7	FILTER	String	A flag indicating the FILTERs status and result
8	INFO	String	additional INFOrmation about the variation
9	FORMAT	String	an (optional) list for describing the samples
+	SAMPLEs	String	for each (optional) SAMPLE, values are associated with the fields listed in FORMAT

[1] samtools.github.io/hts-specs/VCFv4.3.pdf

provide higher compression gains by modelling better the biological data. In Chapter 3 we present a number of statistical models for the purpose of efficient representation of omics sequences.

The coder assigns a binary sequence to a source data, that can be achieved by (a) assigning a number of bits to each nucleotide in a DNA sequence or amino acid in a protein sequence, or (b) assigning a binary sequence to the entire DNA or protein sequence. The former approach is taken by Huffman [38], Golomb [39] and Shannon-Fano [40] algorithms, and the latter by *arithmetic coder* [41],

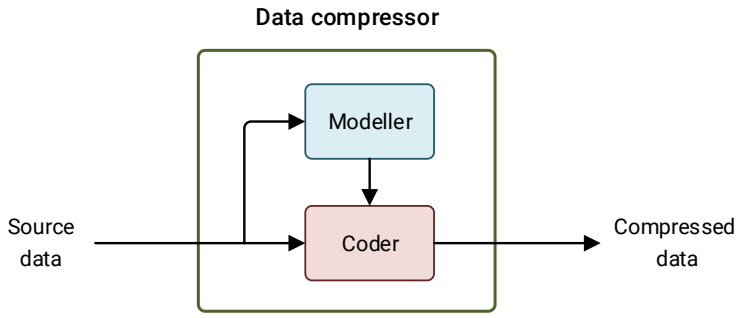

Figure 1.7: The scheme of a data compressor.

which is specifically useful to cope with the data with small alphabets, e.g., genomic sequences. It is also useful to keep separated the modelling and the coding parts of a data compressor [35].

Arithmetic coder stores the entire source data into an arbitrary-precision fraction within the range $[0.0, 1.0)$. As an example, considering a source sequence $S = a_1 a_2 a_3$ with $P(a_1) = 0.7$, $P(a_2) = 0.1$ and $P(a_3) = 0.2$, the encoded message is a fraction within the range $[0.5600, 0.5460)$, as shown in Fig. 1.8. Since arithmetic coder provides, in general, near-optimal output for any given source data, to have a better compression algorithm, a modeller is needed that can accurately predict the patterns of bases in a sequence.

1.4 Omics data encryption

Development of high-throughput sequencing technologies and as a consequence, generating a huge volume of data, has revolutionized personalized medicine in which, treatments are tailored to the individual patients based on their genetic contents [42], [43]. The advent of this field, however, raises issues concerned with preserving security of patients genetic information, which is highly sensitive by nature. To address these issues, a method is required that allows only authorized parties to have access to this private information [44]. *Authenticated encryption* has the potential to provide the three key criteria of *confidentiality*, *integrity* and *authenticity* for such information [45].

In information security, confidentiality is a property upon which the information is not disclosed or made available to unauthorized individuals [46]; integrity assures the data remains accurate and complete over its entire life cycle, hence, unauthorized individuals cannot modify it [47]; authenticity is a property upon which a claim of identity of a computer system user is verified [48].

Omics data files have specific properties; for example, VCF files essentially begin with the 16 character "##fileformat=VCF" or FASTQ files begin with the "@" character at the first line, fol-

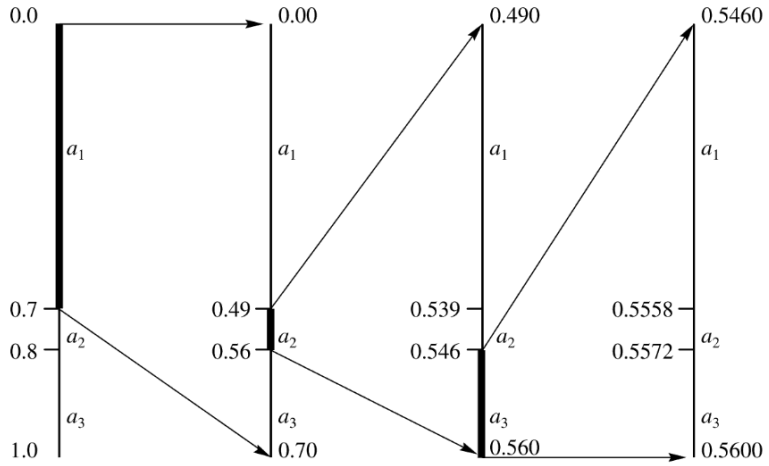

Figure 1.8: The encoding process of arithmetic coder for the input $a_1 a_2 a_3$ [35].

lowed by “A”, “C”, “G” and “T” bases in the second line and then, a “+” character in the third line. General-purpose encryption approaches do not consider these properties, hence, a special-purpose encryptor is needed. Such method should be able to prevent an adversary to break the encryption by security attacks such as *known-plaintext* attacks (KPA) [49]. KPA is defined as an attack model in which the adversary has access to the plaintext as well as the ciphertext (encrypted plaintext).

1.5 Omics data analysis

Data compression can be used for analyzing genomics or proteomics data. An application of such analysis method is to find unique regions in a species with respect to its close species, which can elaborate our understanding of evolutionary traits. Neanderthals are one of the closest groups to modern humans who have lived in Eurasia until 35,000 ± 5000 years ago and then went extinct for probably multiple reasons such as climate change [50]–[52], extermination by immigrating modern humans [53], [54] and disease [55], [56]. With availability of a complete Neanderthal genome [57], [58], we can find regions in modern human reference genome which are highly dissimilar to those of Neanderthals; it gives us an insight into human evolution.

Another application of compression-based data analysis is to find rearrangements, that are mutational changes in the genomes e.g., insertion, deletion, duplication and inversion, between a pair of genomic sequences. Relation to cancer, genetic disorders and chromosomal evolution makes detection of genomic rearrangements very important [59], [60]. Rearrangements can be detected by either wet laboratory methods e.g., Fluorescence *in situ* hybridization (FISH) [61] or computational (dry laboratory) algorithms, which has a major difference with the first approach that is using computers instead of chemicals. The computational algorithms for comparing a pair of sequences can be categorized into alignment-based and alignment-free methods.

Alignment-based methods, that arrange sequences to identify similar/conserved regions between them, have limitations such that (a) they assume the order of homology between the pair of sequences is maintained [62]; (b) their accuracy drops drastically if the identity of sequences is below a certain point, which is quite possible in protein sequences [63]; and (c) they depend on assumptions about evolution of the sequences [64]. Alignment-free methods, however, do not have these limitations; moreover, they are faster, in general.

Alignment-free methods for finding sequence similarity can be classified into methods based on (a) information theory [65]–[68]; (b) word frequency [69]–[73]; (c) number of matching (spaced) words [74]–[78]; (d) length of matching words [79]–[82]; and (e) graphical representation [83], [84]. Among the mentioned classes, information theory-based ones can find local sequence similarity, by computing the amount of information shared between sequences [64].

1.6 Outline

The present thesis comprises of seven Chapters. Chapter 1 is introduction. In Chapter 2 we describe a number of measures to calculate the amount of information in and within omics sequences. Then, we present a number of mathematical models for omics data compression. Finally, we show an application of the mentioned measures and models on the role of inverted repeats on the similarity between a pair of genomic sequences.

Chapters 3 and 4 focus on an efficient representation of omics data by means of compression and encryption tools. In Chapter 3 we introduce two genomic data compressors: GeCo2, that uses a combination of finite-context models and substitution-tolerant Markov models; and Jarvis, that employs weighted context models, weighted stochastic repeat models and a competitive prediction model. We also introduce a proteomic data compressor, namely AC, that uses the same mathematical models as GeCo2, with the difference that they consider the characteristics of protein sequences.

In Chapter 4 we introduce a special-purpose tool, namely Cryfa, for secure encryption of genomic data. It is also able to compact FASTA and FASTQ files. Cryfa provides confidentiality, integrity and authenticity of genomic data by employing AES encryption along with a shuffling mechanism that enhances the security.

Chapters 5 and 6 show applications of the compression models and measures, introduced in Chapter 2, on the analysis of omics data. In Chapter 5 we focus on identifying and visualizing unique regions in a species sequences with respect to the sequences of its close species, which can increase our understanding of evolutionary traits. For this purpose, we introduce CHESTER for genomic sequences and FRUIT for proteomic sequences, and apply them on the sequences of modern human and Neanderthal, as one of the closest hominins to humans.

In Chapter 6 we propose an alignment-free tool, namely Smash++, for detection and visualization of genomic rearrangements, that play an important role on cancer and genetic disorders. This information theory-based algorithm employs data compression for the purpose of finding rearrangement between a pair of DNA sequences. Chapter 7 concludes the thesis and provides future research directions.

1.7 Contributions

Author's contributions to the present thesis are divided into two sections of publications, including journal articles and conference papers, and software, that are publicly available.

1.7.1 Publications

Part of this dissertation has been published in the following manuscripts:

1. M. Hosseini, D. Pratas, B. Morgenstern, and A. J. Pinho, "Smash++: An alignment-free and memory-efficient tool to find genomic rearrangements," *GigaScience*, vol. 9, no. 5, giaa048, 2020.

2. M. Hosseini, D. Pratas, and A. J. Pinho, "AC: A compression tool for amino acid sequences," *Interdisciplinary Sciences: Computational Life Sciences*, vol. 11, no. 1, pp. 68–76, 2019.

3. D. Pratas, M. Hosseini, J. M. Silva, and A. J. Pinho, "A reference-free lossless compression algorithm for DNA sequences using a competitive prediction of two classes of weighted models," *Entropy*, vol. 21, no. 11, p. 1074, 2019.

4. M. Hosseini, D. Pratas, and A. J. Pinho, "A probabilistic method to find and visualize distinct regions in protein sequences," in *The 27th European Signal Processing Conference* (*EUSIPCO*), IEEE, 2019, pp. 1–5.

5. D. Pratas, M. Hosseini, and A. J. Pinho, "Visualization of similar primer and adapter sequences in assembled archaeal genomes," in *The 13th International Conference on Practical Applications of Computational Biology & Bioinformatics* (*PACBB*), Springer, 2019, pp. 129–136.

6. D. Pratas, M. Hosseini, and A. J. Pinho, "GeCo2: An optimized tool for lossless compression and analysis of DNA sequences," in *The 13th International Conference on Practical Applications of Computational Biology & Bioinformatics* (*PACBB*), Springer, 2019, pp. 137–145.

7. M. Hosseini, D. Pratas, and A. J. Pinho, "Clustering DNA sequences by relative compression," in *The 25th Portuguese Conference on Pattern Recognition* (*RECPAD*), Oct. 2019.

8. M. Hosseini, D. Pratas, A. Amorim, and J. Carneiro, "Improving the detection of mtDNA rearrangements using a fast and accurate algorithm," in *XV Encontro Nacional de Biologia Evolutiva*, Nov. 2019.

9. M. Hosseini, D. Pratas, and A. J. Pinho, "Cryfa: A secure encryption tool for genomic data," *Bioinformatics*, vol. 35, no. 1, pp. 146–148, 2018.

10. D. Pratas, M. Hosseini, G. Grilo, A. J. Pinho, R. M. Silva, T. Caetano, J. Carneiro, and F. Pereira, "Metagenomic composition analysis of an ancient sequenced polar bear jawbone from Svalbard," *genes*, vol. 9, no. 9, p. 445, 2018.

11. D. Pratas, M. Hosseini, and A. J. Pinho, "Compression of amino acid sequences," in *The 12th International Conference on Practical Applications of Computational Biology & Bioinformatics* (*PACBB*), Springer, 2018, pp. 105–113.

12. M. Hosseini, D. Pratas, and A. J. Pinho, "On the role of inverted repeats in DNA sequence similarity," in *The 11th International Conference on Practical Applications of Computational Biology & Bioinformatics* (*PACBB*), Springer, 2017, pp. 228–236.

13. D. Pratas, M. Hosseini, and A. J. Pinho, "Substitutional tolerant Markov models for relative compression of DNA sequences," in *The 11th International Conference on Practical Applications of Computational Biology & Bioinformatics* (*PACBB*), Springer, 2017, pp. 265–272.

14. D. Pratas, M. Hosseini, and A. J. Pinho, "Cryfa: A tool to compact and encrypt FASTA files," in *The 11th International Conference on Practical Applications of Computational Biology & Bioinformatics* (*PACBB*), Springer, 2017, pp. 305–312.

15. D. Pratas, M. Hosseini, R. M. Silva, A. J. Pinho, and P. J. S. G. Ferreira, "Visualization of distinct DNA regions of the modern human relatively to a Neanderthal genome," in *The 8th Iberian Conference on Pattern Recognition and Image Analysis* (*IbPRIA*), Springer, 2017, pp. 235–242.

16. M. Hosseini, D. Pratas, and A. J. Pinho, "A survey on data compression methods for biological sequences," *information*, vol. 7, no. 4, p. 56, 2016.

The following paper was published in collaboration with department of bioinformatics at university of Göttingen in Germany, during the author's visit:

17. S. Röhling, A. Linne, J. Schellhorn, M. Hosseini, T. Dencker, and B. Morgenstern, "The number of k-mer matches between two DNA sequences as a function of k and applications to estimate phylogenetic distances," *PLoS ONE*, vol. 15, no. 2, e0228070, 2020.

1.7.2 Software

Associated with every method proposed in this dissertation is a software developed and publicly available under GNU GPLv3 license. The source codes can be accessed by the URLs in Table 1.4.

Table 1.4: List of software developed in the dissertation.

Software	Languages	URL
Smash++	C++, Python	www.github.com/smortezah/smashpp
Cryfa	C++, Bash	www.github.com/cobilab/cryfa
GeCo2	C, Bash	www.github.com/cobilab/geco2
AC	C, Bash	www.github.com/cobilab/ac
CHESTER	C, Bash	www.github.com/cobilab/chester
FRUIT	C++, Bash	www.github.com/cobilab/fruit
Jarvis	C, Bash	www.github.com/cobilab/jarvis

Chapter 2

Measures and models

In this chapter is described and introduced a number of measures, including NID, NCD, NCCD, NRC and NC, in order to quantify the amount of information between and in omics sequences. It is discussed that NRC, among others, is the most light-weight comparative measure in terms of time and memory usage.

Then, different mathematical models, including finite-context models, substitution-tolerant Markov models and a cooperation of them, is presented for compressing genomic and proteomic sequences. Applying on various synthetic and real data showed that employing a combination of FCMs and STMMs provides better results than solely FCMs.

Finally, an application of the compression models and the compression-based measures is presented on the role of inverted repeats in DNA sequences similarity. Tested on several genomic sequences from various species, it is found that some sequences are more similar to each other when IRs are considered in the compression process. This raises the possibility to detect chromosomal rearrangements. The results obtained conform to the ones previously reported using FISH approach or other computational methods; however, we unveiled a number of undocumented similarities.

2.1 Measures for quantifying information

Kolmogorov complexity can be used in order to quantify the information within a sequence; however, it is not computable [101]. For this purpose, here we present a number of measures which are based on compression, as a computable approximation of the Kolmogorov complexity.

2.1.1 Introduction

Kolmogorov complexity

In 1965, Kolmogorov introduced three approaches for quantitative definition of "information": combinatorial, probabilistic and algorithmic approaches [102], among which the algorithmic one is more explored to date. To define algorithmic entropy or Kolmogorov complexity, let x denote a binary string of finite length, where $x = x_1 x_2 \dots x_N \in \Theta^N$ and the alphabet $\Theta = \{0, 1\}$. Its Kolmogorov complexity, $K(x)$, is the length of the shortest binary program x^* that computes x in a universal Turing machine and halts [103]. Therefore, $K(x) = |x^*|$ represents the minimum number of bits from which x can be computationally retrieved [104].

The conditional Kolmogorov complexity of x given y, $K(x|y)$, denotes the length of the shortest binary program that on input y, outputs x and halts. Note that if the input is an empty object, $y = \lambda$, then $K(x|y) = K(x|\lambda) = K(x)$. The conjoint Kolmogorov complexity of x and y, $K(x, y)$, denotes the length of the shortest binary program that, without auxiliary information to the computation, computes x along with y and halts. Disregarding correcting terms, that asymptotically become irrelevant [104], the relation between the three mentioned Kolmogorov complexity functions can be shown by the chain rule [105]:

$$K(x, y) = K(x) + K(y|x). \tag{2.1}$$

Given the symmetric property of the conjoint Kolmogorov complexity, that is $K(x, y) = K(y, x)$, and Eq. 2.1, we can define algorithmic mutual information, $I(x : y)$, as [104], [106]

$$\begin{aligned} I(x : y) &= K(x) - K(x|y) && \text{(2.2a)} \\ &= K(y) - K(y|x) && \text{(2.2b)} \\ &= K(x) + K(y) - K(x, y) && \text{(2.2c)} \\ &= K(x, y) - K(x|y) - K(y|x). && \text{(2.2d)} \end{aligned}$$

Note that it has the symmetric property, i.e., $I(x : y) = I(y : x)$. Fig. 2.1 illustrates the relation between the Kolmogorov complexity functions and the algorithmic mutual information.

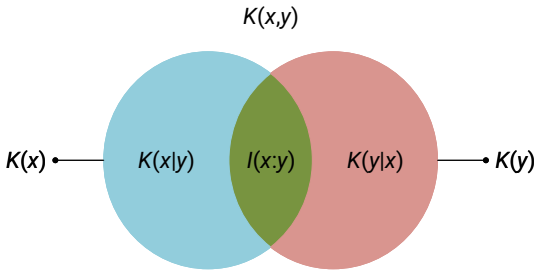

Figure 2.1: Relation between the Kolmogorov complexity, conditional Kolmogorov complexity, conjoint Kolmogorov complexity and algorithmic mutual information.

Distance of information

A distance, D, on a set X is a function that maps $X \times X$ to $[0, \infty)$, where $[0, \infty)$ is the set of non-negative real numbers. D satisfies the following conditions for all $x, y, z \in X$:

$$D(x, y) \geq 0 \qquad \text{(non-negativity)} \tag{2.3a}$$

$$D(x, y) = 0 \iff x = y, \qquad \text{(identity)} \tag{2.3b}$$

$$D(x, y) = D(y, x), \qquad \text{(symmetry)} \tag{2.3c}$$

$$D(x, y) + D(y, z) \geq D(x, z). \qquad \text{(triangle inequality)} \tag{2.3d}$$

In the following sections, different measurements for the distance of information are described.

2.1.2 Normalized information distance

Two decades ago, Bennett *et al.* proposed information distance (ID) with the idea that if two strings are closely related according to any "admissible distance", then they will also be close according to the information distance [107]. ID is defined as

$$\mathrm{ID}(x, y) = \max\{K(x|y),\ K(y|x)\}, \tag{2.4}$$

up to an additive logarithmic term. The normalized version of ID, namely normalized information distance (NID), is defined as [108]

$$\mathrm{NID}(x, y) = \frac{\max\{K(x|y),\ K(y|x)\}}{\max\{K(x),\ K(y)\}}. \tag{2.5}$$

It is worth mentioning that since NID employs Kolmogorov complexity in its definition, it inherits non-computability.

2.1.3 Normalized compression distance

Since Kolmogorov complexity is not computable, a number of computable alternatives have been proposed that are based on compression algorithms. By substituting K by C, a compressor, the metric "compression distance" (CD) which is inspired by ID (Eq. 2.5), can be defined as [108]

$$\mathrm{CD}(x, y) = \max\{C(x|y),\ C(y|x)\}, \tag{2.6}$$

up to an additive logarithmic term. $C(x|y)$ denotes the number of bits required by a (lossless) compression program that on input y, outputs x. Since many of compressors could not handle the conditional compression, Eq. 2.7 was proposed as an alternative, inspired by the chain rule (Eq. 2.1):

$$\mathrm{CD}(x, y) = \max\{C(x, y) - C(x),\ C(y, x) - C(y)\}, \tag{2.7}$$

up to an additive logarithmic term. $C(x)$, $C(y)$ and $C(x, y)$ denote the number of bits required by a (lossless) compressor to represent x, y and concatenation of x and y, and the information needed for splitting them, respectively.

The normalized version of CD, that is normalized compression distance (NCD), is defined as [109]

$$\begin{aligned}\mathrm{NCD}(x, y) &= \frac{\max\{C(x, y) - C(x),\ C(y, x) - C(y)\}}{\max\{C(x),\ C(y)\}} \\ &= \frac{C(xy) - \min\{C(x),\ C(y)\}}{\max\{C(x),\ C(y)\}},\end{aligned} \tag{2.8}$$

up to an additive logarithmic term. Values of NCD fall within the interval of 0 to 1, that is $0 \leq \mathrm{NCD}(x, y) \leq 1$. The closer its value is to 0, the more similar are x and y, and the closer its value is to 1, the more dissimilar are x and y.

In order for a compression program to achieve an admissible NCD, it must be *normal*, that is it must satisfy the following conditions [110]:

$$C(xx) = C(x) \text{ and } C(\lambda) = 0,\ \lambda \text{ is an empty object,} \qquad \text{(idempotency)} \tag{2.9a}$$

$$C(xy) \geq C(x), \qquad \text{(monotonicity)} \tag{2.9b}$$

$$C(xy) = C(yx), \qquad \text{(symmetry)} \tag{2.9c}$$

$$C(xy) + C(z) \leq C(xz) + C(yz), \qquad \text{(distributivity)} \tag{2.9d}$$

up to an additive $O(\log n)$ term, where n is the maximal length of an element involved in the (in)equality concerned.

2.1.4 Normalized conditional compression distance

In the definition of normalized compression distance, the chain rule (Eq. 2.1) has been used to substitute the conditional compression with the conjoint compression, that is substituting $C(x|y)$ with $C(x, y) - C(y)$, since at the time of presenting NCD there were not many compression programs that could do the conditional compression. The simplest way to calculate $C(x, y)$ is compressing the direct concatenation of x and y, however, all forms of x and y combined can be considered. In [110] it is shown that employing the concatenation can affect the efficiency of NCD. To overcome the limitations, normalized conditional compression distance (NCCD) has been defined as [111], [112]

$$\mathrm{NCCD}(x, y) = \frac{\max\{C(x|y),\ C(y|x)\}}{\max\{C(x),\ C(y)\}}. \tag{2.10}$$

In the case the individual algorithmic complexities of strings x and y are known, Eq. 2.10 turns into

$$\mathrm{NCCD}(x, y) = \begin{cases} \dfrac{C(x|y)}{C(x)}, & C(x) \geq C(y) \\ \dfrac{C(y|x)}{C(y)}, & C(x) < C(y). \end{cases} \tag{2.11}$$

It is worth noting that the conditional compression used in NCCD needs less computation rather than conjoint compression used in NCD, since $C(x|y)$ only needs to load models of y, however $C(x, y)$ needs to compress y in addition to loading its models. Also, if conditional compression of multiple sequences, e.g., $C(x|y), C(z|y), \ldots$ is intended, models of y need to be loaded only once and then, the sequences $x, z, \ldots$ can be compressed in parallel using the model.

2.1.5 Normalized relative compression

The relative compression of x given y, denoted by $C(x\|y)$, is defined as size of a compressed sequence x given *exclusively* the information from a sequence y [111]. It has the following properties:

1. $C(x\|y) \approx 0$ iff y contains x,
2. $C(x\|y) \approx |x|$ iff $C(x|y) \approx C(x)$.

In the relative compressor $C(x\|y)$, the self-similarities that might occur in x are not possible to be used, since it exclusively employs the information from the model of y.

Normalized version of the relative compression, namely normalized relative compression (NRC), is defined as

$$\mathrm{NRC}(x, y) = \frac{C(x\|y)}{|x|}. \tag{2.12}$$

NRC is considered as a "semi-quasi distance" [113], [114], since it relaxes the symmetry and triangle inequality axioms (see Eq. 2.3a). Note that $0 < \mathrm{NRC} \leq 1$, since $0 < C(x\|y) \leq |x|$.

The main advantage of NRC rather than NCCD is consuming less computation time and memory, since it does not include self-similarity compression terms, i.e., $C(x)$ and $C(y)$, and depends only on one compression term, $C(x\|y)$. In addition, unlike NCCD that needs setting parameters for both target and reference sequences, x and y, NRC only needs to set parameters for the reference sequence y. A comprehensive study on the comparison of the two measures has been done in [115] by calculating them for various synthetic and real DNA sequences.

2.1.6 Normalized compression

The measures of NID, NCD, NCCD and NRC are *comparative*, in that they are defined between two sequences. Here, we describe normalized compression (NC), that is inspired by the concept of Kolmogorov complexity and is defined for a single sequence x as

$$\mathrm{NC}(x) = \frac{C(x)}{|x| \log_2 |\Theta|}, \tag{2.13}$$

where Θ is the alphabet of symbols in the sequence x. As an example, for a genomic sequence we have $\Theta = \{\mathrm{A}, \mathrm{C}, \mathrm{G}, \mathrm{T}\}$ and therefore $|\Theta| = 4$.

2.2 Compression models

We consider sequences over the nucleotide alphabet Θ; our goal is to measure the degree of local similarity between two such sequences. More specifically, we consider a reference sequence $S = s_1, \ldots, s_N$ over Θ, and we want to measure the local *information content* of a target sequence, given this reference sequence. To this end, we employ a combination of finite-context models and substitution-tolerant Markov models to derive different probability measures for observing a nucleotide θ in a sequence, given the *context* of the previous k nucleotides; these probabilities are then mixed to provide the final probability of observing the nucleotide θ. The following sections describe in detail the models.

2.2.1 Finite-context model (FCM)

We consider the probability of observing a certain nucleotide, given the previous k nucleotides, by using the relative frequency of this event in the reference sequence S. For $\theta \in \Theta$ and a k-mer $Q \in \Theta^k$, let $N(\theta|Q)$ be the number of occurrences of Q in S that are followed by nucleotide θ, and let $N(Q)$ be the number of occurrences of Q in S. As in [35], [86], [116], we then define

$$P_{\mathrm{FCM}}(\theta|Q) = \frac{N(\theta|Q) + \alpha}{N(Q) + |\Theta|\alpha}, \tag{2.14}$$

where $|\Theta|$ is the size of alphabet Θ and α is a pseudo-count parameter. For $\alpha = 1$, Eq. 2.14 turns into the *Laplace estimator*. Note that an FCM has the Markov property, in which the conditional probability distribution of observing a nucleotide depends only on the state of preceding k-mer.

2.2.2 Substitution-tolerant Markov model (STMM)

Given the reference sequence S, we use the aforementioned probability distribution P_{FCM} to define a sequence $S' = s'_{-k}, s'_{-k+1}, \dots, s'_N$ recursively by

$$s'_i = \begin{cases} \text{A} & \text{if } i < 1, \\ \arg\max\limits_{\theta \in \Theta} P_{\text{FCM}}(\theta | s'_{i-k}, \dots, s'_{i-1}) & \text{if } i \geq 1, \end{cases} \tag{2.15}$$

in which "A" represents adenine nucleobase. For $\theta \in \Theta$ and a k-mer $Q \in \Theta^k$, we then define $N'(\theta|Q)$ as the number of occurrences of Q followed by θ and $N'(Q)$ as the number of occurrences of Q, respectively, in the sequence S'. Finally, we define

$$P_{\text{STMM}}(\theta|Q) = \frac{N'(\theta|Q) + \alpha}{N'(Q) + |\Theta|\alpha}. \tag{2.16}$$

STMMs, that are probabilistic-algorithmic models [86], [97], can be used along with FCMs when confronted with nucleotide substitutions in genomic sequences. These models can be disabled, to reduce the number of mathematical calculations, and consequently, increase the performance of the proposed method. Such operation is automatically performed using an array of size k (the context size), named history, which preserves the past k hits/misses. Observing a symbol in the sequence, the memory is checked for the symbol with the highest number of occurrences. If they are equal, a hit is saved in the history array; otherwise, a miss is inserted into the array. Before getting to store a hit/miss in the array, it is checked for the number of misses and in the case they are more than a predefined threshold t, the STMM will be disabled and also the history array will be reset. This process (Fig. 2.2) is performed for each nucleotide in the sequence.

The following example shows the distinction between an FCM and an STMM. Assume that the current context at a certain position is AGACGTAC, and the number of occurrences of symbols saved in memory is 10, 6, 15 and 8 for A, C, G and T, respectively; also, the symbol to appear in the sequence is T. An FCM considers the next context as GACGTAC**T**, while an STMM considers it as GACGTAC**G**, since the nucleotide G is the most probable symbol, based on the number of occurrences stored in memory.

```
1: function GetBestId(array)
2:     return index of max element in array
3: end function

4: function Miss(history)
5:     for i ← 0 to k − 1 do
6:         if history[i] ≠ 0 then
7:             numFail ← numFail + 1
8:         end if
9:     end for
10:    if numFail > threshold then
11:        on ← 0                          ▷ Set STMM off
12:    else
13:        insert 1 into history           ▷ Add one fail
14:    end if
15: end function

16: function Hit(history)
17:    insert 0 into history               ▷ Add zero fail
18: end function

19: function CorrectSTMM( )
20:    best ← 0
21:    if on = 0 then
22:        on ← 1                          ▷ Set STMM on
23:        reset history                   ▷ Set all elements to 0
24:    else
25:        best ← GetBestId(freqs)         ▷ freqs is an array of counters for bases
26:        if best = currentBase then
27:            Hit(history)
28:        else
29:            Miss(history)
30:            update seqBuffer with best  ▷ seqBuffer is a buffer to keep bases
31:        end if
32:    end if
33:    update seqBuffer
34: end function
```

Figure 2.2: Algorithm for enabling/disabling an STMM.

2.2.3 Cooperation of FCMs and STMMs

Fig. 2.3 provides an overview of cooperation of FCMs and STMMs. When these models are in cooperation, the probability of observing a nucleotide $\theta \in \Theta$ in a sequence S can be estimated as

$$P(\theta) = \sum_{i=1}^{m} P_{\mathrm{FCM}_i}(\theta|Q)\ w_i + \sum_{j=1}^{n} P_{\mathrm{STMM}_j}(\theta|Q)\ w_j, \tag{2.17}$$

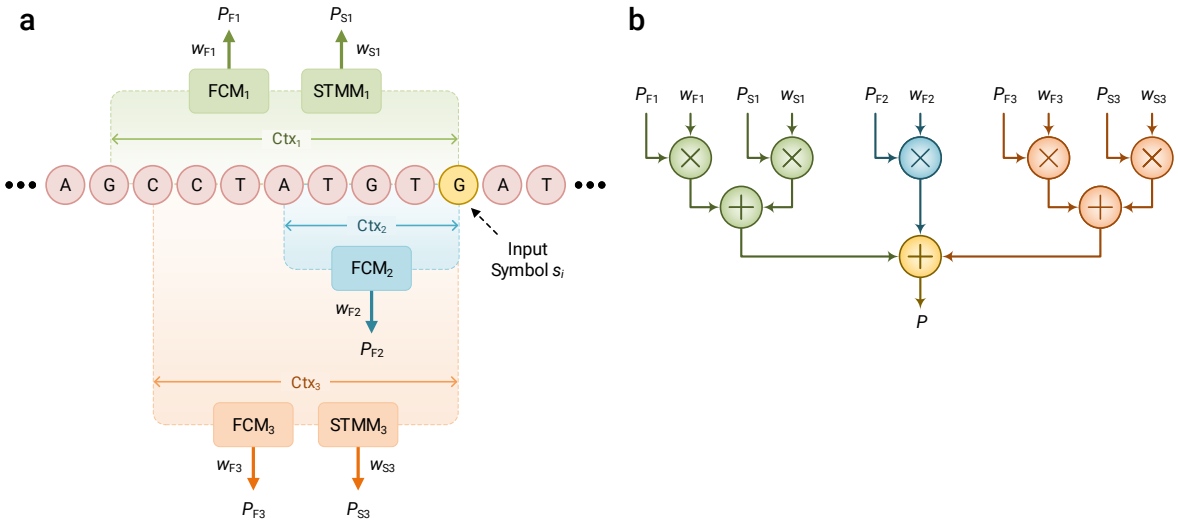

Figure 2.3: (a) Cooperation of FCMs and STMMs. Note that each STMM needs to be associated with an FCM; (b) probability of an input symbol is estimated by employing the probability and weight values that have been obtained from processing previous symbols.

in which m and n denote the number of FCMs and STMMs, respectively, and w_i and w_j are weights assigned to each FCM and STMM, respectively, based on its performance. We have

$$\begin{aligned} w_{i_u} &\propto (w_{i_{u-1}})^{\gamma_i} P_{\text{FCM}}(\theta|Q_{u-1}), \quad 1 \le i \le m, \quad 1 \le u \le |S|, \\ w_{j_v} &\propto (w_{j_{v-1}})^{\gamma_j} P_{\text{STMM}}(\theta|Q_{v-1}), \quad 1 \le j \le n, \quad 1 \le v \le |S'|, \end{aligned} \tag{2.18}$$

where u and v denote certain positions in the sequences S and S', respectively, and γ_i and $\gamma_j \in [0, 1)$ are forgetting factors predefined for each model. Also,

$$\sum_{i=1}^{m} w_i + \sum_{j=1}^{n} w_j = 1. \tag{2.19}$$

By experimenting different forgetting factors for models of genomic or proteomic sequences, we have found that higher factors should be assigned to models that have higher context-order sizes (less complexity) and vice versa (see Fig. 2.4 for more details).

Bit-rate, in "bits per symbol" (bps), denotes the average number of bits that each symbol will take up in the compressed sequence, and can be calculated for $s_1, s_2, \ldots, s_N \in S$ as

$$\text{Bit-rate} = -\frac{1}{N} \sum_{u=1}^{N} \log_2 P(s_u). \tag{2.20}$$

Note that the probability of observing a symbol in a sequence can be calculated by either of Equations 2.14, 2.16 or 2.17, depending on the usage of FCMs, STMMs or a cooperation of them, respectively. Note also that the better the compression model is, the smaller the bit-rate value is [117].

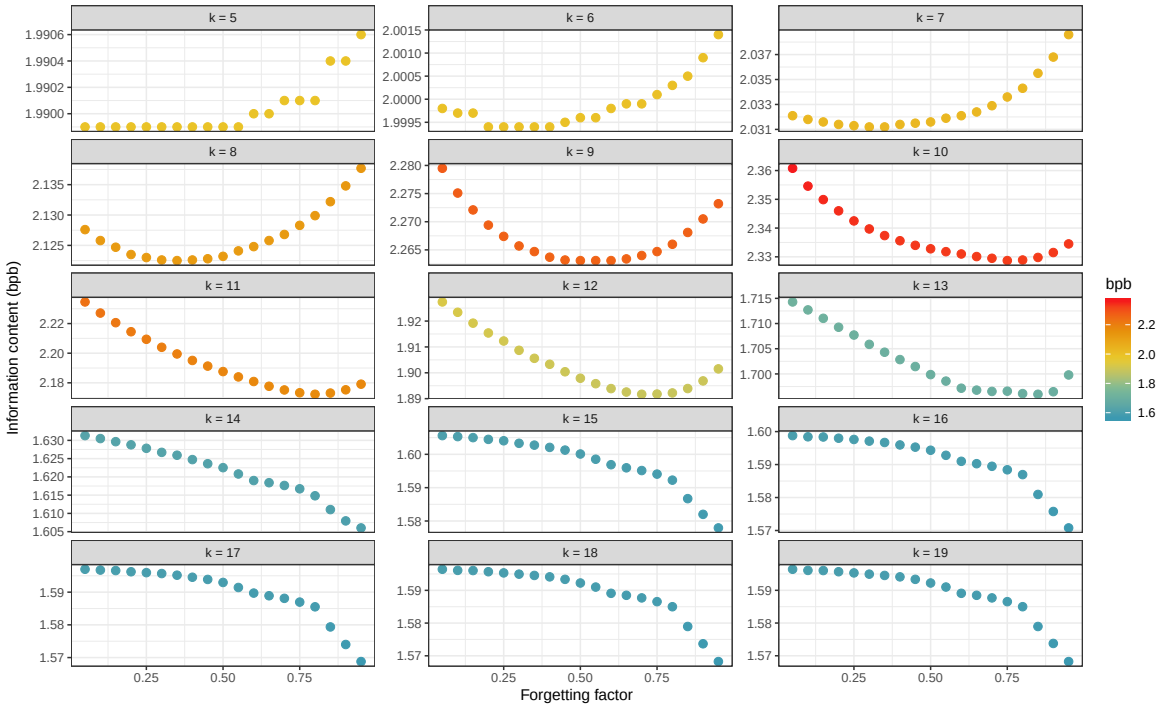

Figure 2.4: The relation between context-order sizes (k), forgetting factors and complexity (information content). For the experiment, we compressed 10 synthetic sequences with the sizes of 500 kb to 20 Mb, with different redundancies, and calculated the average information content.

It is mentioned in Section 2.1.1 that there are three approaches of combinatorial, probabilistic and algorithmic to define information quantitatively. FCMs follow the probabilistic approach by computing the probability of a given symbol in a sequence, considering the past k symbols, while STMMs follow the algorithmic approach by getting enabled/disabled based on their performance. Hence, we follow a combination of probabilistic and algorithmic approaches by employing a cooperation of FCMs and STMMs, although it has a closer connection to the probabilistic one.

2.2.4 FCMs compared to cooperation of FCMs and STMMs

In order to compare the effect of employing solely FCMs with employing FCMs along with STMMs, we applied them on synthetic and real data, including white-tailed eagle, bald eagle, human, chimpanzee, orangutan and marmoset species. The experiments were run on one core of a 3.40 GHz Intel® Core™ i7-6700 CPU with 32 GiB RAM and a solid-state hard drive.

Synthetic dataset

We have performed experiments on two different datasets. For the first experiment, we made a synthetic reference sequence of size 200 b; then, duplicated it and inserted edits on positions 50, 100, 102, 150, 152 and 154, to make the target sequence. Fig. 2.5 illustrates in terms of bit-rates (bps) the

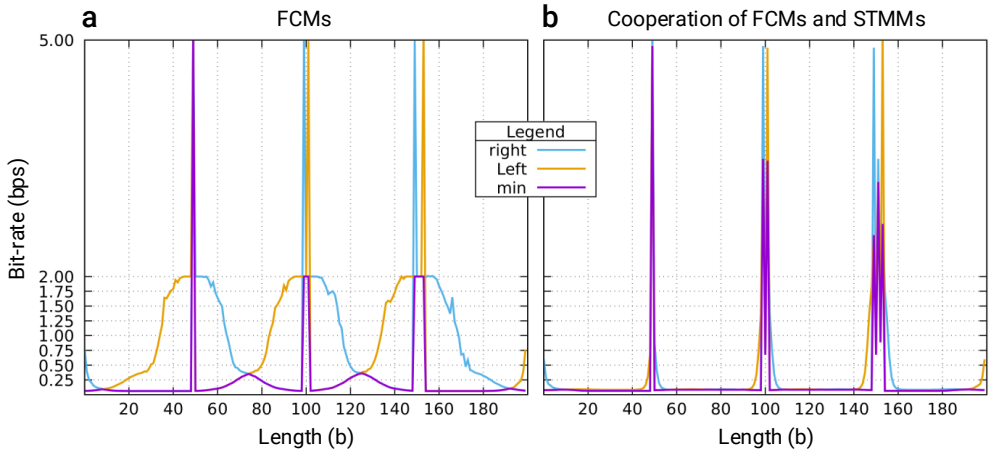

Figure 2.5: Compression of a synthetic target sequence relatively to a reference using (a) FCMs; and (b) cooperation of FCMs and STMMs.

result of compressing the target relatively to the reference, when using only FCMs (Fig. 2.5a) and cooperation of FCMs and STMMs (Fig. 2.5b). In the figure, right direction means traversing the target sequence from the first nucleotide to the last one (left to right), and left direction means traversing it from the last nucleotide to the first one (right to left); also, min is the minimum of left and right directions for each base in the target sequence. As can be seen, cooperation of FCMs and STMMs led to a better modeling of the data, since it could provide less bit-rate values. On the contrary to the cooperation, the FCMs could not address efficiently the data modeling after occurring a substitution. The cooperation has an almost strict decay to a low complexity value.

For the second experiment, we made a 100 kb reference sequence; then, duplicated it 12 times, applied some degree of random mutations to each of them and concatenated them all to make the target sequence. Fig. 2.6 shows bit-rates obtained by compressing the target relatively to the reference, in presence of mutations. In the figure, min and max represent minimum and maximum

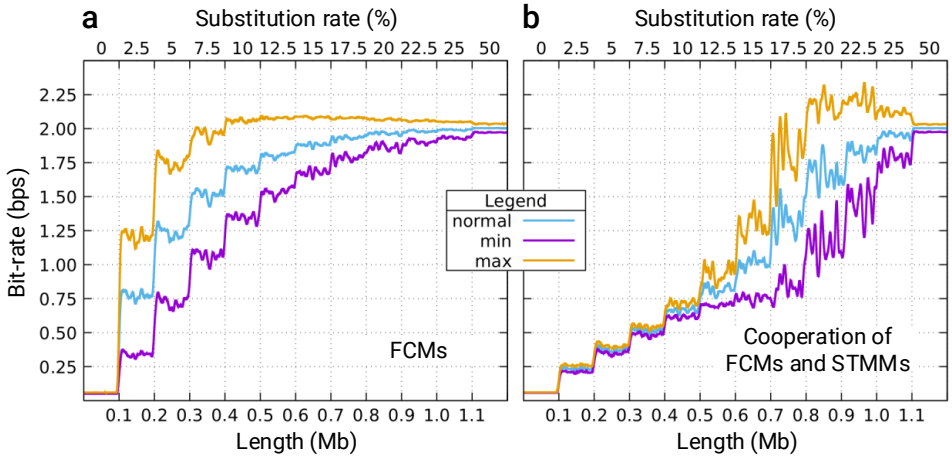

Figure 2.6: Compression of a synthetic target sequence relatively to a reference in presence of mutations, employing (a) FCMs; and (b) cooperation of FCMs and STMMs.

bit-rates of left and right directions for each 100 kb chunks, respectively. Using FCMs, the minimum bit-rate reaches to 1 bps when there is 7.5% mutation, while employing the cooperation of FCMs and STMMs, it reaches to 1 bps when occurring 20% mutation.

Real dataset

We have downloaded from NCBI non-assembled sequences of white-tailed eagle (*Haliaeetus albicilla* with 26X coverage) and bald eagle (*Haliaeetus leucocephalus* with 88X coverage) [118] and also, reference genomes of human, chimpanzee, gorilla, orangutan and marmoset. The compression was performed employing four FCMs with k-mer sizes $= 4, 6, 13, 20$ and $\alpha = 1, 1, 0.5, 0.005$, respectively, along with an STMM with $k = 20$, $\alpha = 0.5$ and $t = 5$.

Table 2.1 provides in detail results of compressing the targets relatively to the references. In the columns "Compress. gain" and "Time loss", percentage of compression improvement and computation time added, respectively, are reported when using a cooperation of FCMs and STMMs over using only FCMs. Note that both approaches consume the same amount of RAM. The "Div." column shows divergence time of the reference and the target species, in million years ago (Mya). As seen in the table, there is a trade-off between compression improvement and extra computation time.

2.3 Application on quantifying inverted repeats

Genomic sequences have specific properties such as the presence of inverted repeats [122], which may play an important role in chromosomal rearrangements [123]. IRs, as sub-sequences of genomic sequences, are reversed and complemented copies of some other sub-sequences [124]. The compression methods, aside from providing efficient storage, processing and transmission, can help investigating properties of IRs. We exploit a finite-context model, introduced in Section 2.2, to carry out a reference-based compression on genomic sequences.

To handle inverted repeats in the mathematical model, when a base is intended to be encoded, in addition to the counter associated with the preceding context, we update another counter in the

Table 2.1: Compression results for real dataset.

Reference	Target	Compressed size (b)		Compress. gain (%)	Time loss (%)	RAM (GiB)	Div. (Mya)	
		FCM	FCM+STMM					
White-tailed eagle	Bald eagle	34,864,683	31,561,247	10.0	10.0	14	1	[119]
Human	Chimpanzee	274,450,972	210,691,987	23.0	22.5	26	4.5–3	[120]
Human	Gorilla	262,271,376	199,204,749	24.0	19.8	26	9–5	[120]
Human	Orangutan	418,481,411	299,316,387	28.5	19.2	26	10	[120]
Human	Marmoset	562,916,901	488,238,361	13.3	18.8	26	40	[121]

following way. Let "G" be the base to be coded and "TCTA" be the associated context. First, we concatenate the context and the base to form "TCTAG"; then, we calculate its reverse as "GATCT", and complement the reversed string as "CTAGA" (A $\leftrightarrow$ T, C $\leftrightarrow$ G). Here, we update the counter associated with the 4-mer "CTAG', observing "A" as the base to be encoded.

To investigate the properties of IRs in DNA data, a measure is also required. For this purpose, different measures have been proposed, such as normalized compression distance, normalized conditional compression distance and normalized relative compression, which rely on the notion of Kolmogorov complexity (Section 2.1). Here, we use the normalized relative compression, which is light-weight regarding computational time and memory, to measure the dissimilarity between genomic data. Testing this approach on various species, including human, chimpanzee, gorilla, chicken and turkey genomes, we unveil unreported results that may support several evolution insights.

We implemented the method using the C++ language, considering as input FASTA or SEQ format with the alphabet $\Theta = \{A, C, G, T, N\}$. The machine used for the tests had a 16-core Intel® Xeon® CPU E7320 with 2.13 GHz frequency, and 256 GiB RAM. For all tests, we set the parameters $\alpha = 0.01$ and context-order size $(k) = 20$. As dataset, several genome sequences with different species origins and lengths were considered (described in Table 2.2), that can be downloaded from NCBI[1]. Hereinafter, the notation X.i refers to chromosome i of sequence X; moreover, MT, UL, UP, AL and LG refer to mitochondrial DNA, unlocalized sequence, unplaced sequence, alternate locus and a linkage group, that is not assigned to a chromosome, respectively.

In Fig. 2.7 is plotted heatmaps of the NRC values regarding the compression of human and chimpanzee chromosomes, in an all-to-all scheme. Squares show similarity between the reference and target sequences. The less the NRC value is, the more similar the corresponding chromosomes are, since less bits are used for their relative compression. Figures 2.7a and 2.7b show NRC values with and without considering IRs. As can be seen, there is higher similarity between the reference and target chromosomes with the same numbers, rather than others, except for HS.2, HS.MT, HS.UL and HS.UP corresponding to PT.2A & PT.2B, PT.MT, PT.UL and PT.UP, respectively. Also, HS.AL is not similar to any PT chromosome. Note that HS.2 (corresponding to PT.2A and PT.2B) is presumed to contain an ancestral chromosome fusion [125]; also, HS.Y (as a target) is highly correlated to PT.X, since they had possibly exchanged information in recombination processes [126].

[1] ftp.ncbi.nlm.nih.gov/genomes

Table 2.2: Dataset used for studying the role of inverted repeats on DNA sequence similarity.

Target	Species	Size (GiB)	Reference	Species	Size (GiB)
Human (HS)	*H. sapiens*	3 3	Chimpanzee (PT)	*P. troglodytes*	3 3
Chimpanzee (PT)	*P. troglodytes*	3 3	Human (HS)	*H. sapiens*	3 3
Gorilla (GG)	*G. gorilla*	4 4	Human (HS)	*H. sapiens*	3 3
Chicken (GGA)	*G. gallus*	1 3	Turkey (MGA)	*M. gallopavo*	1 2
Turkey (MGA)	*M. gallopavo*	1 2	Chicken (GGA)	*G. gallus*	1 3

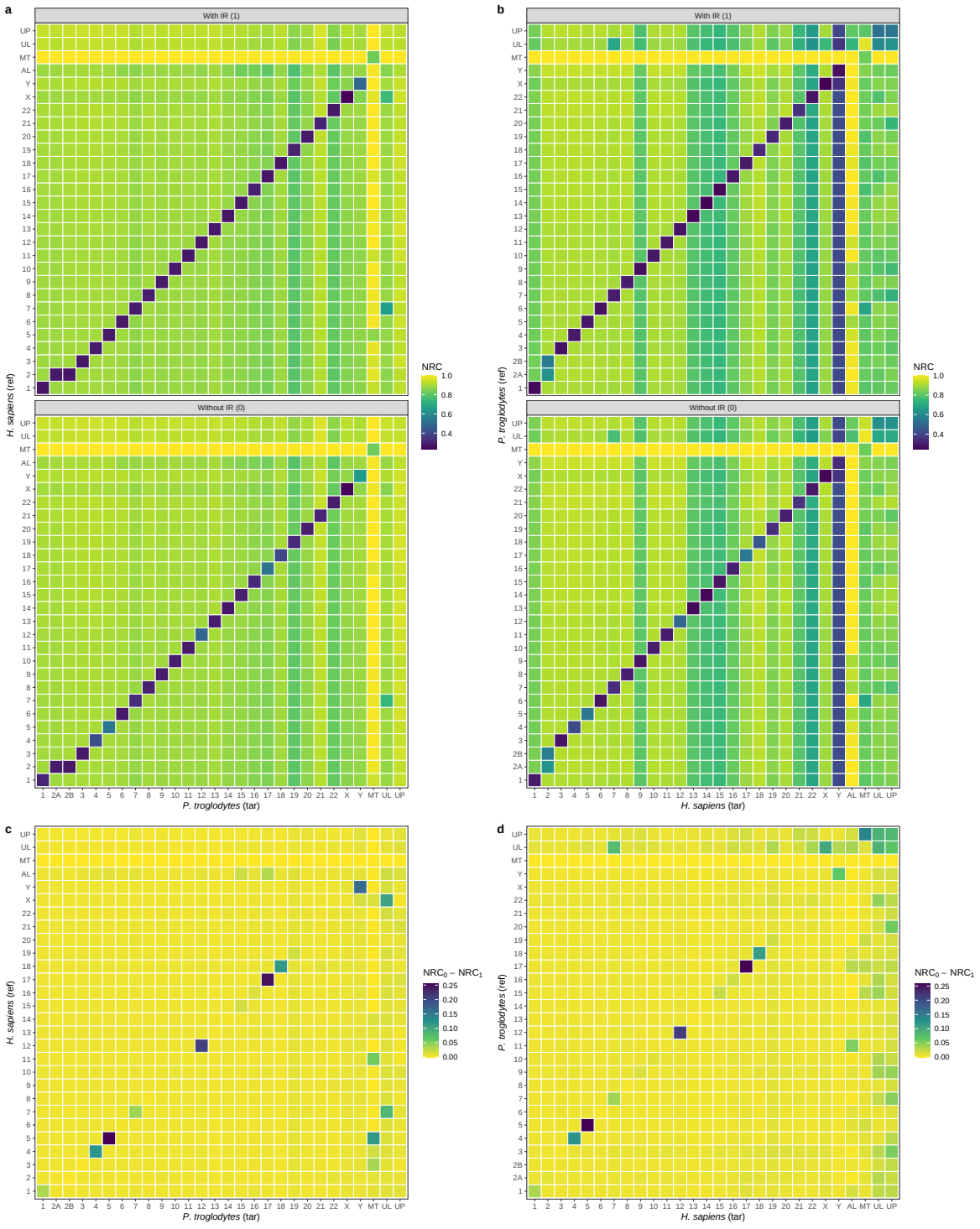

Figure 2.7: NRC values calculated by compression of chimpanzee and human chromosomes. (a), (b) IRs applied (IR = 0) and not applied (IR = 1); and (c), (d) the difference in NRCs between not applying and applying IRs ($NRC_{IR=0} - NRC_{IR=1}$).

Figures 2.7c and 2.7d show the difference in NRC between considering and not considering IRs in the compression. The larger the difference of NRC values for two sequences is, the more similar those sequences are when considering IRs, and consequently, possibly the higher the probability of

chromosomal rearrangement would be. As can be seen, chromosomes 4, 5, 12, 17 and 18 of HS and PT have more similarity than others, when considering IRs. This conforms with the results reported in [127], [128], in which pericentric inversions were detected by FISH analysis. Also, HS.Y and PT.Y are correlated, which conforms to [129]. Additionally, we have found a high correlation between HS.MT and PT.UP (as the reference), when considering IRs.

The NRC results regarding compression of gorilla chromosomes using human as reference are shown in Fig. 2.8. Considering IRs, a similarity between GG.17 and HS.5 is seen, which is justified by a chromosomal translocation [130]–[132]. Moreover, GG.2B and HS.2 are similar with and without

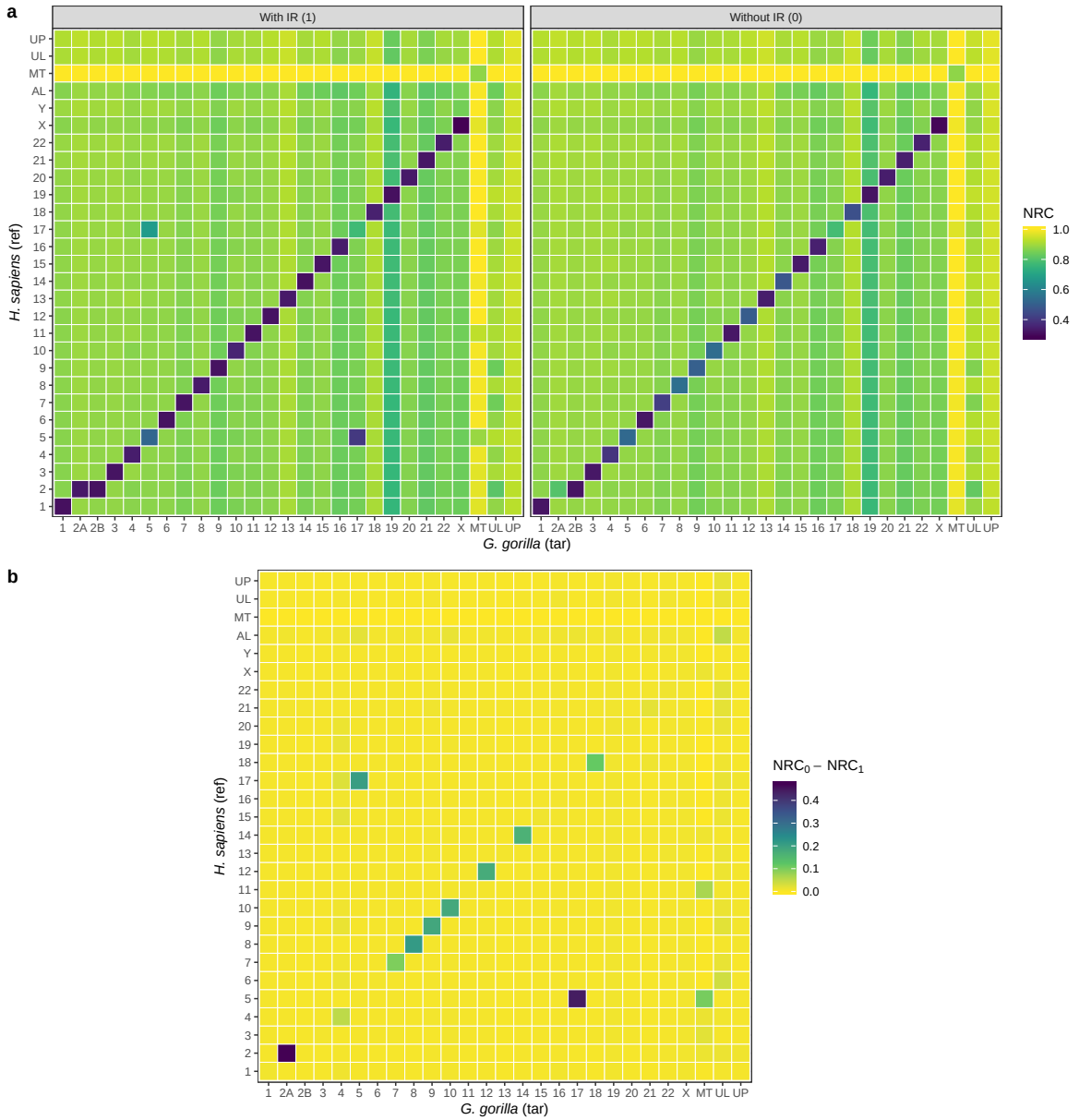

Figure 2.8: NRC results concerned with compression of gorilla using human chromosomes as references. (a) left: IR = 0 and right: IR = 1; and (b) the difference between NRCs.

considering IRs. However, there is a remarkable difference between considering and not considering IRs in the compression of GG.2A using HS.2 as reference; thus, these two chromosomes are similar only when IRs are considered.

In Fig. 2.9, the NRC results of compressing chicken and turkey chromosomes are plotted. Many

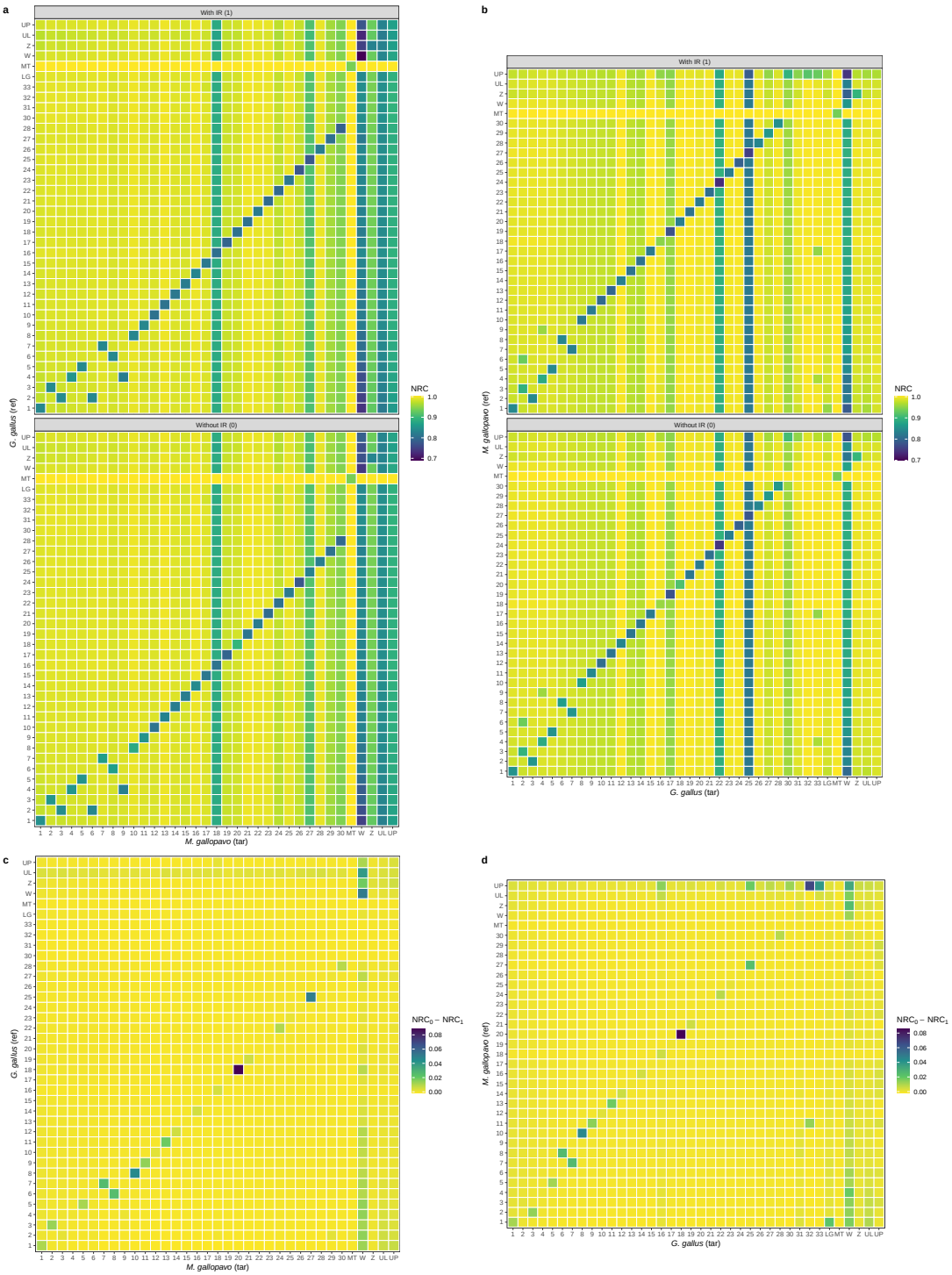

Figure 2.9: NRC values associated with the compression of turkey and chicken chromosomes. (a), (b) With and without IRs; and (c), (d) the difference between NRCs.

different similarities can be seen, such as in chromosomes 3 and 2, 6 and 8, 8 and 10, 11 and 13, and 18 and 20 of GGA and MGA, respectively, which were reported in [133] as chromosomal rearrangements. We have found similarities between MGA.W and, GGA.1 and GGA.UL, and also GGA.W and, MGA.1, MGA.Z and MGA.UP. Moreover, we have found that MGA.27 and GGA.25, and also GGA.32 and MGA.UP are highly similar when IRs are considered.

2.4 Conclusions

We described a number of measures in order for quantification of information in and between omics sequences. Among them are compression-based measures that employ a compression method to approximates the Kolmogorov complexity. Also, we presented different models, including FCMs, STMMs and a cooperation of FCMs and STMMs, for the purpose of compressing genomic and proteomic data. Tested on several sequences showed that the cooperation produces better results than using solely the FCMs. Finally, we presented an application of exploiting the introduced compression models along with the measures on the role of inverted repeats in similarity of DNA sequences. Applied on various DNA dataset showed that some sequences are more similar to each other when IRs are considered during the compression procedure.

Chapter 3

Compression of omics data

Advancements in high-throughout sequencing technologies has led to generating a huge volume of omics data. Development of efficient compressors for such data is crucial, to reduce the storage and bandwidth for transmission, and also for analysis purposes. In this chapter are presented two genomic sequence compressors, namely GeCo2 and Jarvis, and a protein data compressor, AC.

GeCo2 performs lossless compression by employing a combination of FCMs and STMMs with which is associated a specific decay factor. It can use models considering only inverted repeats. The results of testing our tool on several DNA sequences from various domains and kingdoms shows an improvement over state-of-the-art methods using less computational resources (time and memory).

Jarvis works based on weighted context models, including FCMs and STMMs, and weighted stochastic repeat models from which the best class of model is used for each nucleotide, using a competitive prediction model. The two classes of models use specific sub-programs to handle inverted repeats efficiently. The results of applying Jarvis on the same dataset used by GeCo2 shows that it attains a higher compression ratio than state-of-the-art approaches, including GeCo2.

AC is a state-of-the-art method for lossless compression of amino acid sequences, which works based on a cooperation of FCMs and STMMs. Compared to several general-purpose and special-purpose protein compressors, our method provides the best bit-rates. AC can also compress the sequences nine times faster than its competitor, paq8l. In addition, by employing the proposed method we analyze the compressibility of a large number of sequences from different domains. The results show that viruses and eukaryota are the most and the least complex sequences to compress, respectively, and also archaea and bacteria are the second most complex ones.

3.1 Compression of genomic sequences

3.1.1 Introduction

With the development of high-throughput sequencing technologies, huge volumes of sequencing data are produced everyday, fast and cheaply, that can lead to upgrading the scientific knowledge of genome sequence information as well as facilitating the diagnosis and therapy. Along with these advantages, three challenges exist to deal with this data deluge: storage, processing and transmission. Also, the costs associated with them are higher compared to sequence generation, that makes the situation more complicated [134]. *Compression* is able to overcome these challenges by reducing the storage size and processing costs, such as I/O bandwidth, along with increasing transmission speed [33], [135], [136].

Genomic (or DNA) sequences are codified messages from an alphabet of four symbols (A, C, G and T) that contain instructions, structure and historical marks of all known cellular organisms [137]. Initially, genomic sequences were compressed with general-purpose tools, such as gzip[1], bzip2[2] or lzma[3]. Since the emergence of biocompress [138] (described in the following), the development of special-purpose compression algorithms for these sequences revolutionized the field.

Over the past three decades, many genomic sequence compressors have been proposed in the literature, that can fall into two categories of reference-free and reference-based methods [87], [117], [138]–[174]. The basic idea of reference-free compression is to exploit structural properties, e.g., inverted repeats, along with statistical properties of the sequences in order for compression [175]. In the following a selection of these methods are described. The first algorithm specifically proposed for compression of genomic sequences is biocompress [138], which detects factors (repeats) and complementary factors (inverted repeats) in sequences and encodes them by the Ziv and Lempel compression method [176], using the length and position of their earliest occurrences.

POMA is an adaptive particle swarm optimization-based memetic algorithm [147] which works based on comprehensive learning particle swarm optimization (CLPSO) [177] and an adaptive intelligent single particle optimizer (AdpISPO)-based local search. In this method, an approximate repeat vector (ARV) codebook is designed and then optimized by CLPSO and AdpISPO for compressing the sequence. The approximate repeats that have the fewest base variations employ the candidate ARV codebooks, encoded as particles, to achieve the optimal solution in POMA. Then, the weighted fitness values are used to select the leader particles in the swarm. Finally, an AdpISPO-based local search is exploited to fine-tune the leader particles.

DNA-COMPACT (DNA COMpression based on a Pattern-Aware Contextual modeling Tech-

[1] www.gzip.org
[2] www.sourceware.org/bzip2
[3] www.7-zip.org/sdk.html

nique) [148] employs complementary contextual models and consists of two phases. First, the exact repeats and inverted repeats are searched and then represented by a compact quadruplet. Second, the non-sequential contextual models are introduced in order to exploit the features of DNA sequences; then, the predictions of these models are synthesized using the logistic regression model, which is shown to lead to less biased results rather than Bayesian averaging. Note that DNA-COMPACT can handle both reference-free and reference-based genome compression.

The CoGI (Compressing Genomes as an Image) method [150] initially transforms the genomic sequence into a binary image (or bitmap). Then, it uses rectangular partition coding algorithm [178] to compress the image. Finally, it employs entropy coding for further compression of the encoded image along with mismatches.

GeCo [152] uses a soft-blending cooperation of finite-context models and substitution-tolerant Markov models with one specific forgetting factor. The probability values calculated by the models are then passed to an arithmetic encoder. The GeCo method has sub-programs to deal with inverted repeats and uses cache-hash for high-order context models, which is a middle point between a dictionary and a probabilistic model.

The SPRING method [155], that supports both lossless and lossy compression of FASTQ sequences, provides its best results when the reads are reordered (lossy compression). For short reads (up to 511 bp) it employs HARC [179], while for long reads it uses BSC[1] (block sorting compression). To compress quality scores, it performs quantization by means of QVZ [180], binary thresholding or Illumina's binning[2] (quantizing to eight values) and then, encodes the quantized values by BSC. Identifiers in lossless mode are tokenized and compressed based on the difference with the previous identifier and finally encoded by an adaptive arithmetic encoder, and in lossy mode are discarded (the pairing information is saved).

The key idea of reference-based genome sequence compression is to employ similarity between a target sequence and a (set of) reference sequence(s). This type of compressors achieve to, generally, higher compression ratios than reference-free compressors. In the following a selection of reference-based genomic compressors is introduced.

The GRS method [160], which is based on the "diff" utility in Unix, finds the longest subsequences that are identical in the reference and the target sequences. Then, it calculates a similarity measure; if that is greater than a threshold, the difference between the reference and target sequences are compressed with Huffman encoding [38]; otherwise, the reference and the target are divided into smaller sub-sequences and the same strategy, i.e., comparing the value of the similarity measure with an identified threshold, is used to compress the sub-sequences.

FRESCO (Framework for REferential Sequence COmpression) [166] uses a compressed suffix

[1] www.github.com/IlyaGrebnov/libbsc

[2] www.illumina.com/documents/products/whitepapers/whitepaper_datacompression.pdf

tree [181] of the reference sequence to match the prefixes of an input string with sub-strings of the reference sequence. In addition, three techniques are used to improve the compression performance: (a) selecting a good reference; (b) rewriting the reference sequence; and (c) second-order compression, that is reference-based compressing of an already compressed sequence.

GDC2 (Genome Differential Compressor) [168] considers multiple genomes of the same species. It uses a number of extra reference phrases, that are extracted from other sequences, along with an LZ-parsing for detection of approximate repeats. GDC2 considers short matches after some longer ones, and supports random access to an individual sequence.

The iDoComp method [170] includes: (a) mapping generation, that employs the suffix arrays to parse the target sequence relatively to the reference sequence; (b) post-processing of the mapping, that finds the consecutive matches which can be merged together to form an approximate match, and uses them to reduce the size of mapping; and (c) entropy encoding, that uses an adaptive arithmetic encoder to further compress the mapping and then generates the compressed file.

ParRefCom [173] is a scalable parallel algorithm for paired-end (PE) genomics reads that comprises three steps: (a) a special alignment of PE reads to standard reference, in a sense that reads are reordered while PEs are kept together; (b) classification of PE reads based on the number of ends aligned; and (c) developing custom compression strategies for reads in different categories.

3.1.2 GeCo2

In this section we describe the GeCo2 tool, which is an improved version of GeCo [152]. In the proposed tool we enhance the mixture of FCMs and STMMs, where each model has now a specific decay factor. Additionally, specific cache-hash sizes and the ability to run only with inverted repeats is developed. A new command line interface, several pre-computed modes of compression and several optimizations in the code are included.

Methods

GeCo2 is an improved version of GeCo [152] and uses a cooperation of FCMs and STMMs, described in Section 2.2, with a specific forgetting factor for each model, followed by arithmetic encoding. We have found, experimentally, that lower forgetting factors should be used for more complex models.

To save models in memory, we use cache-hash data structure introduced in [152]. It keeps only the last hashed entries, rendering a flexible and predictable amount of memory, that is independent of sequence sizes. Fig. 3.1 shows the schema of this data structure. In order to save a context (index), we divide it into “INDEX A” and “INDEX B”. As an example, considering contexts of size 20, 40 bits is needed to save them. We divide these 40 bits into 24 bits for INDEX A, assuming we need a

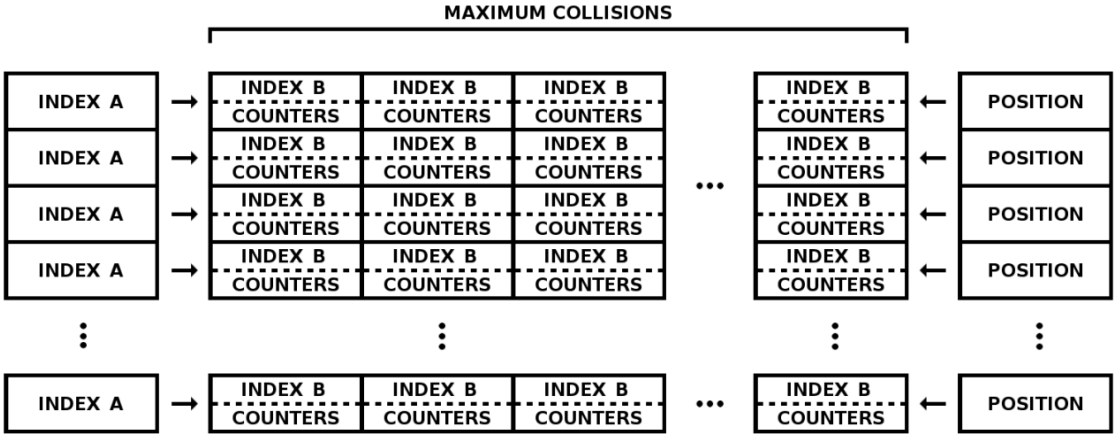

Figure 3.1: Cache-hash data structure.

hash table with 2^{24} rows, and 16 bits for INDEX B. Associated with each INDEX A is "POSITION", a circular buffer with the size of "MAXIMUM COLLISIONS", that holds the position of the last entry in that INDEX A row. MAXIMUM COLLISIONS is the maximum number of entries associated with each INDEX A, that can be defined by user. Choosing a large number for it will make the cache-hash sparse; it is 10 by default. Each entry in an INDEX A row includes INDEX B and "COUNTERS", that stores frequencies of occurring A, C, G and T bases after a specific context with 4 bit precision. COUNTERS are 16-bit long, since they contain four 4-bit values. When a counter reaches 15, all the counters (associated with the four bases) are divided by two. To deal with new contexts, if their rightmost 16 bits already exists in an entry, the counters of the associated entry will be updated, otherwise a new entry will be inserted after the last entry. Since in genomic sequences similar regions tend to be grouped or close to each other, in order to search for an entry, we move from the last entry toward the first one, using the positions saved in the POSITION buffer. Note that the cache-hash data structure uses a single hash function.

Compared with GeCo, we have made the following improvements: (a) in GeCo, each model with a context order size of 14 or higher uses a hash table that keeps only the latest hash collisions, while in GeCo2, each model has its own cache size. This enables to explore different repetitive natures without collision of models; (b) in GeCo, the inverted repeats model could only be added to an existing model, but in GeCo2 we added a mode for running exclusively the IR model. This allows to use GeCo2 in IRs studies, namely for the detection of rearrangements of inverted nature; (c) a new interface layout is created which enables more flexibility and permits easier optimization of parameters; (d) new functions, e.g., a new approximate power function, were added to make the program faster. Several other functions were optimized; and (e) 12 pre-computed modes for reference-free compression are added, that can be enabled with "-l <LEVEL>" flag. The modes and associated parameters (see Section 2.2) are described in detail in Table 3.1. "k" denotes the context order size. "α" is a parameter that allows to keep a balance between the maximum likelihood estimator and the uniform distribution in a model, and is taken from user as $1/\alpha$. To consider inverted repeats, the

Table 3.1: Specification of 12 pre-computed modes of GeCo2.

Mode	k	FCM				STMM		
		$1/\alpha$	IR	c	γ	Edit	$1/\alpha$	γ
1	1	1	0	–	0.7		–	
	12	20	1	–	0.97		–	
2	2	1	0	–	0.78		–	
	4	1	1	–	0.78		–	
	11	80	1	–	0.96		–	
3	3	1	0	–	0.8		–	
	4	1	1	–	0.84		–	
	12	50	1	–	0.94	2	15	0.95
4	4	1	0	–	0.8		–	
	6	1	1	–	0.84		–	
	13	50	1	–	0.94	2	15	0.95
5	4	1	0	–	0.82		–	
	6	1	1	–	0.72		–	
	13	50	1	–	0.95	2	15	0.95
6	4	1	0	–	0.88		–	
	6	1	1	–	0.76		–	
	13	50	1	–	0.95	2	15	0.95
7	4	1	1	–	0.9		–	
	6	1	1	–	0.79		–	
	8	1	1	–	0.91		–	
	13	10	1	–	0.94	1	20	0.94
	16	200	1	5	0.95	4	15	0.95
8	4	1	1	–	0.9		–	
	6	1	1	–	0.8		–	
	13	10	1	–	0.95	1	20	0.94
	16	100	1	5	0.95	3	15	0.95
9	4	1	1	–	0.91		–	
	6	1	1	–	0.82		–	
	13	10	1	–	0.94	1	20	0.94
	17	100	1	8	0.95	3	15	0.95
10	1	1	0	–	0.9		–	
	3	1	0	–	0.9		–	
	6	1	1	–	0.82		–	
	9	10	0	–	0.9		–	
	11	10	0	–	0.9		–	
	13	10	1	–	0.9		–	
	17	100	1	8	0.89	5	10	0.9
11	4	1	1	–	0.91		–	
	6	1	1	–	0.82		–	
	13	10	1	–	0.95	1	20	0.94
	17	100	1	15	0.95	3	15	0.95
12	1	1	0	–	0.9		–	
	3	1	0	–	0.9		–	
	6	1	1	–	0.85		–	
	9	10	0	–	0.9		–	
	11	10	0	–	0.9		–	
	13	50	1	–	0.9		–	
	17	100	1	20	0.9	3	10	0.9

user should set IR = 1. "c" is the number of maximum collisions in cache-hash. "γ" is forgetting factor in a model. "Edit" denotes the number of edits in a substitution-tolerant Markov model.

Results and discussion

We tested GeCo2, state-of-the-art genomic data compressors and one general-purpose compressor on 15 DNA sequences with the total size of ~534 Mb (Table 3.2), described in [182], from various domains and kingdoms, namely viruses, archaea, bacteria and eukaryota. The machine used for tests had a single core Intel® Xeon® CPU E7320 at 2.13 GHz frequency. GeCo2 is implemented in the C language and is publicly available[1] under GNU GPLv3 license.

Tables 3.3 and 3.4 demonstrate compressed sizes and computation times, respectively, for applying GeCo2 and other compressors on the dataset. We carried out paq8 (paq8kx variant) with the "-8" (best option), GeCo with "-tm 1:1:0:0/0 -tm 3:1:0:0/0 -tm 6:1:0:0/0 -tm 9:10:0:0/0 -tm 11:10:0:0/0 -tm 13:50:1:0/0 -tm 18:100:1:3/10 -c 30 -g 0.9" option and XM using 50 copy experts.

GeCo2 is able to compress the complete dataset 0.2% more than GeCo, while performing in 11.8% less time. The memory usage of GeCo and GeCo2 is ~4.8 GiB and ~3.8 GiB, respectively. Therefore, comparing with GeCo (version 1), we improved the compression and computational time and also, saved ~1 GiB of RAM.

GeCo2 can compress better than XM all sequences except DaRe, OrSa and EnIn. Overall, XM uses rather than GeCo2 6.5 times more computational time and substantially larger RAM (at least 3 times more). Note that compression/decompression of GeCo, XM and GeCo2 are approximately symmet-

[1] www.github.com/cobilab/geco2

Table 3.2: Dataset used for benchmarking genomic sequence compressors.

Name	Species	Category	Size (b)
HoSa	*Homo sapiens*	Eukaryota, animalia	189,752,667
GaGa	*Gallus gallus*	Eukaryota, animalia	148,532,294
DaRe	*Danio rerio*	Eukaryota, animalia	62,565,020
OrSa	*Oriza sativa*	Eukaryota, plant	43,262,523
DrMe	*Drosophila miranda*	Eukaryota, animalia	32,181,429
EnIn	*Entamoeba invadens*	Eukaryota, amoebozoa	26,403,087
ScPo	*Schizosaccharomyces pombe*	Eukaryota, fungi	10,652,155
PlFa	*Plasmodium falciparum*	Eukaryota, protozoan	8,986,712
EsCo	*Escherichia coli*	Bacteria	4,641,652
HaHi	*Haloarcula hispanica*	Archaea	3,890,005
HePy	*Helicobacter pylori*	Bacteria	1,667,825
AeCa	*Aeropyrum camini*	Archaea	1,591,049
YeMi	*Yellowstone lake* mimivirus	Virus, mimivirus	73,689
AgPh	*Aggregatibacter* phage S1249	Virus, phage	43,970
BuEb	*Bundibugyo ebolavirus*	Virus	18,940
Total			534,263,017

Table 3.3: Compressed file sizes, in bytes, obtained by GeCo2 and state-of-the-art genomic data compressors.

Name	paq8	GeCo	XM	GeCo2 (mode[1])
HoSa	40,517,624	38,877,294	38,940,458	38,845,642 (12)
GaGa	34,490,967	33,925,250	33,879,211	33,877,671 (11)
DaRe	12,628,104	11,520,064	11,302,620	11,488,819 (10)
OrSa	9,280,037	8,671,732	8,470,212	8,646,543 (10)
DrMe	7,577,068	7,498,808	7,538,662	7,481,093 (10)
EnIn	5,761,090	5,196,083	5,150,309	5,170,889 (9)
ScPo	2,557,988	2,536,457	2,524,147	2,518,963 (8)
PlFa	1,959,623	1,944,036	1,925,841	1,925,726 (7)
EsCo	1,107,929	1,109,823	1,110,092	1,098,552 (6)
HaHi	904,074	906,991	913,346	902,831 (5)
HePy	385,096	381,545	384,071	375,481 (4)
AeCa	380,273	385,640	387,030	380,115 (5)
YeMi	16,835	17,167	16,861	16,798 (3)
AgPh	10,754	10,882	10,711	10,708 (2)
BuEb	4668	4774	4642	4686 (1)
Total	117,582,130	112,986,546	112,558,213	112,744,517

[1] It represents the compression level of GeCo2.

ric, since they perform decompression using approximately the same computational resources, i.e., time and memory.

Compared with paq8 (general-purpose compressor), GeCo2 is able to compress the dataset 4.1% better and perform 138 times less computation. The memory usage of paq8 did not exceed 2 GiB.

Table 3.4: Computation times, in seconds, of applying GeCo2 and other genomic sequence compressors on the dataset described in Table 3.2.

Name	paq8	GeCo	XM	GeCo2 (mode)[1]
HoSa	85,269.1	648.6	5589.8	652.4 (12)
GaGa	64,898.9	503.2	3633.9	494.7 (11)
DaRe	29,907.7	215.9	785.2	198.8 (10)
OrSa	20,745.1	192.4	489.7	138.3 (10)
DrMe	14,665.8	114.6	362.6	102.4 (10)
EnIn	11,183.6	95.8	279.8	82.5 (9)
ScPo	4619.1	45.2	96.5	34.2 (8)
PlFa	4133.9	39.7	84.4	35.3 (7)
EsCo	1973.9	26.4	36.8	5.1 (6)
HaHi	1738.1	23.7	39.1	4.4 (5)
HePy	715.1	17.2	11.2	1.9 (4)
AeCa	675.3	17.0	10.3	1.9 (5)
YeMi	32.6	12.3	0.9	0.1 (3)
AgPh	20.1	12.1	0.9	0.1 (2)
BuEb	9.1	12.2	0.7	0.1 (1)
Total	240,587.4	1976.3	11,421.8	1742.2

[1] The compression level (mode) used in GeCo2 for each sequence is the same as in Table 3.3.

3.1.3 Jarvis

In this section we describe Jarvis, a new lossless compressor for DNA sequences that uses a competitive prediction model to estimate for each symbol the best class of models, among context and repeat models, to be used for compression. The probability value associated with the chosen class will be then redirected to an arithmetic encoder.

Methods

Jarvis works based on a competitive prediction between two classes of models: weighted context models and weighted stochastic repeat models. The context models are combined through a weighted set of contexts and substitution-tolerant contexts [97], [152] using a specific forgetting factor for each model, while the stochastic repeat models use a common forgetting factor. By setting different number of context models and repeat models and their parameters, the proposed method provides a flexibility to address compression of different types of DNA sequences.

Fig. 3.2 shows an example of a competitive prediction between five weighted context models (represented by prefix C) and three weighted stochastic repeat models (represented by prefix R). Each model has weights (W) and probabilities (P) that are calculated based on the counts present at the respective data structure saved in memory (M). The suffices (1 to 5) show model numbers. The fifth context model includes an FCM (CW4, CP4) and an STMM (CW5, CP5) that share the same data structure saved in memory (CM4), since they have the same context order size. After calculating probabilities and weights for different models, they are redirected to the competitive prediction model in order to find the model with the highest probability (predicted), which is then redirected to an arithmetic encoder to provide the compressed sequence.

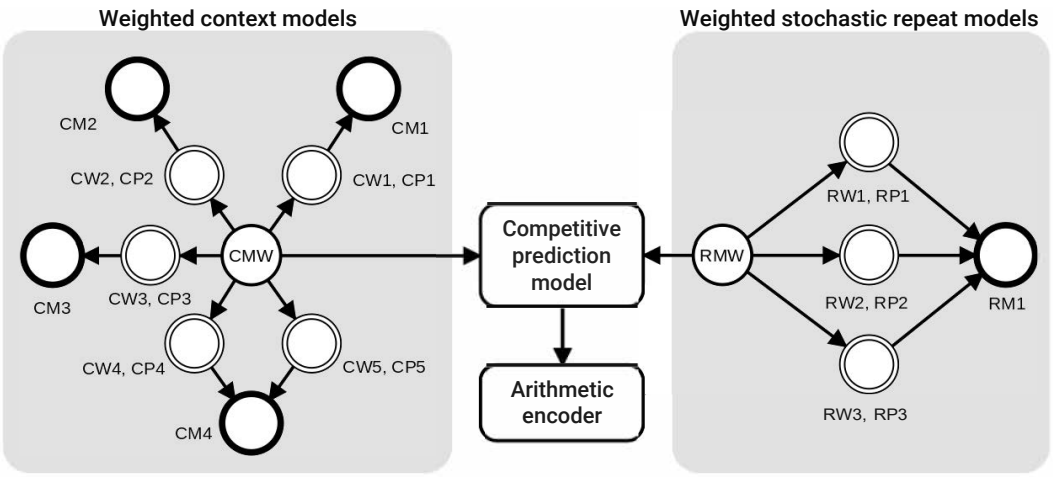

Figure 3.2: An example of a competitive prediction between five weighted context models and three weighted stochastic repeat models.

In the following sections we describe the weighted context models, weighted stochastic repeat models and competitive prediction model. For the purpose, we assume that there is a source generating symbols from a finite alphabet $\Theta = \{A, C, G, T\}$.

Weighted context models

Context models include finite-context models and substitution-tolerant Markov models, described in detail in Section 2.2, which generate weight and probability values that are updated for each base. In order to save these models in memory, we use cache-hash data structure that is described in Section 3.1.2. Note that a user can set the maximum number of collisions in the cache-hash to constrain the peak memory (RAM) usage by Jarvis.

Weighted stochastic repeat models

The repeat model, that is inspired by the copy expert of the XM omics data compressor [145], relies on a pointer to a position in the reference sequence that by a "good chance" contains a character identical to that being encoded (in the target sequence) [163]. As we move through the target sequence, the pointer may be repositioned to different locations of the reference sequence; as its consequence, all parameters of the model will be reset. In a repeat model, two counters are maintained: t, that stores the number of times that the model was used after the previous repositioning, and h, that keeps the number of times the model predicted correctly the nucleotide.

Fig. 3.3 shows an example of a repeat model, with the most recent repositioning occurred at position 192872 and 178514 of the reference and the target sequences, respectively. The nucleotide that is going to be encoded (marked with ?) is predicted by the model to be "G" (the base under the "Current position" arrow), with which associated counters are $t = 10$ and $h = 8$. The bases linked by the dashed arrows indicate prediction errors (the predicted nucleotides were G and A, whereas the correct ones were T and C).

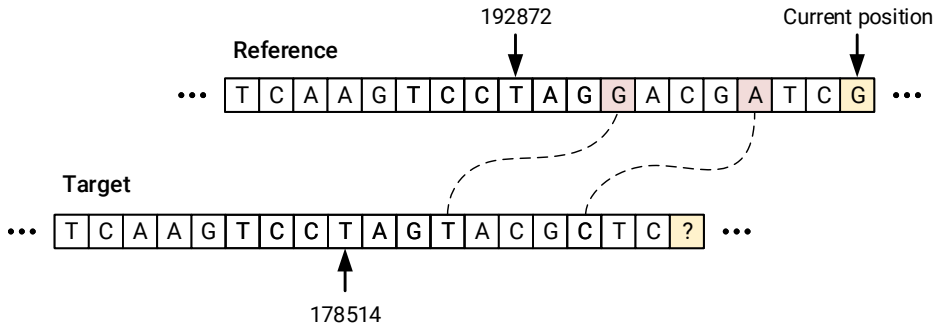

Figure 3.3: Example of a repeat model. The base marked with ? is intended to be encoded. The dashed arrows show failure in prediction by the model.

In order to compute the probabilities, that will be then passed to an arithmetic encoder, let x be the nucleotide to be encoded, y be the predicted base, h be the number of times that the model successfully predicted the base and t be the number of times that the model was used after the previous repositioning. The estimated probability of a nucleotide x can be calculated as

$$P(x) = \frac{h+1}{t+2}. \tag{3.1}$$

A repeat model constructs a hash table with hashed k-mers (of a given size) as *keys*, and k-mers and the positions of their occurrences in the reference sequence as *values*. Fig. 3.4 depicts an example where $k = 7$ and k-mers "TATGTCC" and "CGATCGT" have been hashed into the same key, i.e., 3123807. By saving k-mers in the hash table, direct comparison can be done to disambiguate between them in the encoding phase. Note that using the hash table, the nucleotides that come right after all occurrences of a given k-mer can be found.

Before a new nucleotide in the target sequence is encoded, performance of the repeat model is checked. If the number of prediction failures, $t - h$, is greater than a preset threshold, the repeat model will be stopped. To restart the model, the positions of the preceding k-mer (before the current nucleotide in the target sequence) is located in the hash table. In case there are more than one position at that index (for that k-mer), the one closest to the encoding position would be chosen. Note that if no record is found in the hash table, the current repeat model would not participate in the process of compressing the current nucleotide.

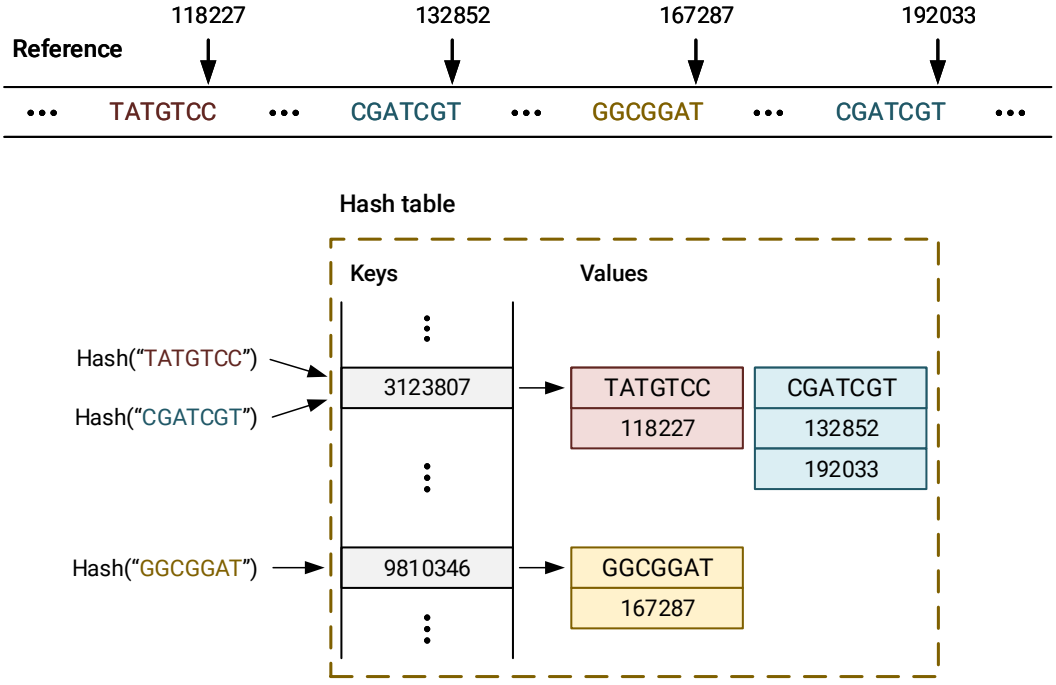

Figure 3.4: The hash table constructed by the repeat model.

Jarvis exploits multiple repeat models, with a maximum value (*RPN*) that can be set by a user. When the *RPN* is larger than the available number of positions, the number of effective models will be narrowed down to that number. The repeat models are combined in the same way as weighted context models, considering the fact that associated with each model is a weight value that is updated based on its performance. The decaying factors (γ) should be small for such models, since the weights need to be adapted quickly. Note that the reason to use the term "stochastic" in naming these models is that in the hash table, all positions associated with a k-mer have the same probability (*chance*) of being used.

Competitive prediction context model

The competitive prediction context model (CPCM) uses a finite-context model, with an order size defined as a parameter, to choose the best class between weighted context models and weighted stochastic repeat models. For this purpose, a sequence of zeros and ones, Z, is generated.

Figure 3.5 depicts an example of a CPCM with a context order size of five ($k = 5$). To predict the best class of models at the position $i + 1$, the probability of $P(Z_{i+1} = S|Z_{i-4}Z_{i-3}Z_{i-2}Z_{i-1})$ is calculated by Eq. 2.14; in this case, $S = 0$ as the next symbol. Note that the alphabet is $\Theta = \{0, 1\}$ in a CPCM. The probability values obtained by CPCM will then be redirected to an arithmetic encoder.

We evaluated the impact of context order sizes in CPCM on compression of HoSa, EnIn, AeCa and YeMi sequences. The results shown in Fig. 3.6 denote that there is a direct relation between the context order size and the length of a sequence (according to the respective redundancy). For example, HoSa, as the largest sequence, has the least bit-rate when compressed with a context size of 16 (in level 12) and YeMi, as the shortest sequence, presents the best compression with a context size of 5 (in level 2). Note that the accuracy of prediction can be improved by using multiple CPCMs; however, it will increase the computational time.

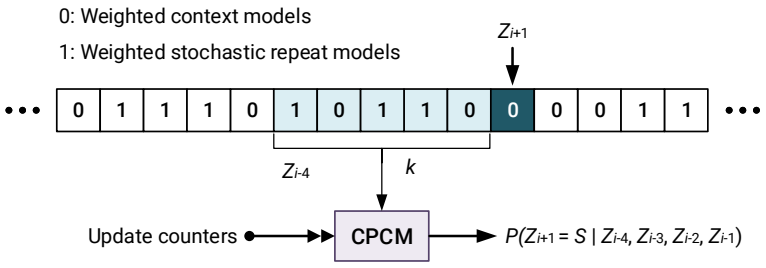

Figure 3.5: Example of a competitive prediction context model with the context order size of five.

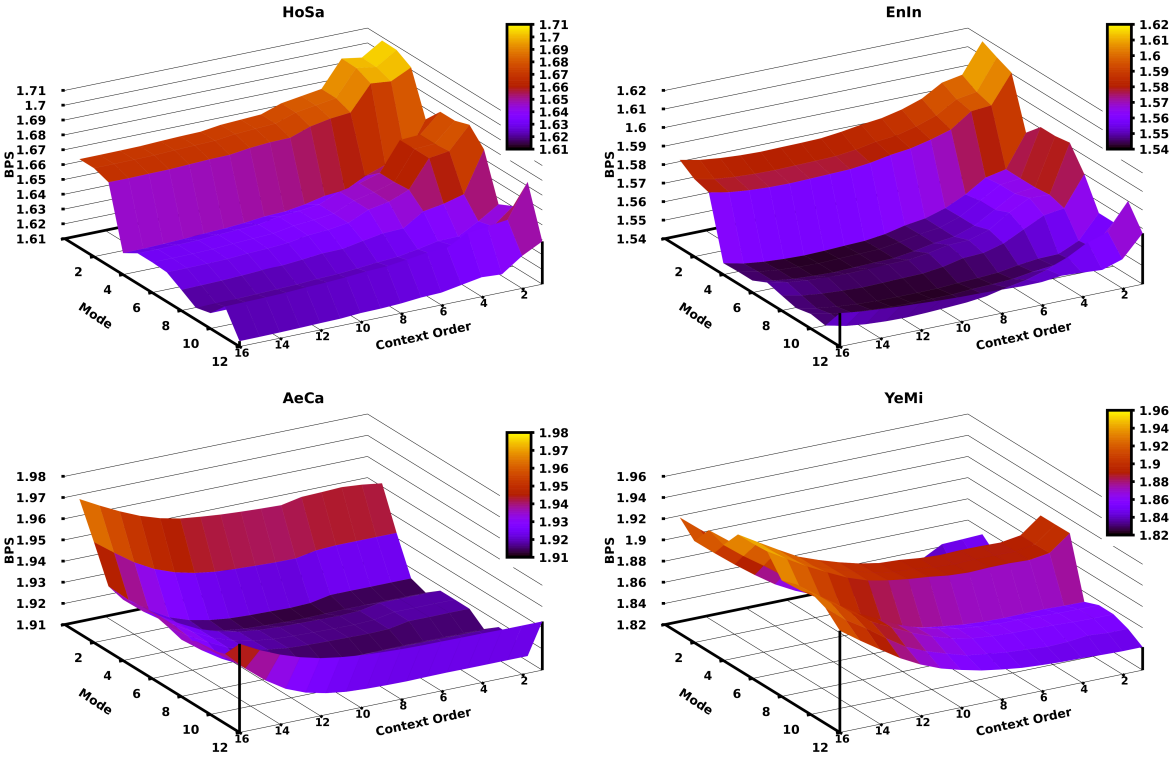

Figure 3.6: Bit-rates, in bps, for compressing four sequences of HoSa, EnIn, AeCa and YeMi, when applying CPCM with different context order sizes.

Decompression

Jarvis is a lossless compressor, therefore, it decompresses the compressed sequence to the original sequence without any loss. During the compression, some side information is added to the beginning of the compressed sequence that will be used by the decompressor to generate the exact same original sequence. For instance, the seed used by stochastic repeat models is saved in the header to ensure the exact beginning in the stochastic process.

The compression/decompression process in Jarvis is symmetric, meaning that weighted context models, weighted stochastic repeat models and competitive prediction model are synchronized in the same order, by the same characteristics, on both sides of compression and decompression.

Implementation

Jarvis is written in the C language and is publicly available[1] under GNU GPLv3 license. As arithmetic encoder, we use a slightly modified version of the one proposed in [183]. Our tool includes 15 pre-computed running modes with the parameters described in Table 3.5. Except for a few modes, lower modes use less computational resources (time and memory) and are more appropriate for

[1] www.github.com/cobilab/jarvis

Table 3.5: Specification of 15 running modes of Jarvis.

Mode	Context model (FCM)					Context model (STMM)				Repeat model							z
	k	$1/\alpha$	IR	c	γ	Edit	$1/\alpha$	IR	γ	*RPN*	k	α	β	lim	γ	IR	
1	2	1	1	–	0.7			–		50	13	1	0.9	5	0.2	1	4
2	3	1	1	–	0.7			–		50	13	1	0.9	5	0.2	1	6
3	3	1	0	–	0.95			–		50	13	1	0.9	6	0.1	1	8
	6	1	1	–	0.7			–									
4	3	1	0	–	0.93			–		1000	15	0.1	0.9	6	0.1	1	12
	6	1	1	–	0.7			–									
	12	10	1	–	0.95			–									
5	6	1	1	–	0.92			–		1000	15	0.1	0.9	6	0.1	1	12
6	6	1	1	–	0.92			–		1000	15	0.1	0.9	6	0.1	1	14
7	6	1	1	–	0.92			–		1000	16	0.1	0.9	6	0.1	1	14
8	3	1	0	–	0.93			–		1000	16	0.1	0.9	6	0.1	1	14
	6	1	1	–	0.7			–									
	13	10	1	–	0.95			–									
9	1	1	0	–	0.9			–		2000	16	0.1	0.9	6	0.1	1	14
	4	1	0	–	0.93			–									
	6	1	0	–	0.7			–									
	10	10	0	–	0.94			–									
	12	20	0	–	0.95			–									
	14	50	1	–	0.95	1	1	0	0.95								
	18	200	1	30	0.95	1	10	1	0.95								
10	1	1	0	–	0.9			–		5000	16	0.1	0.9	6	0.1	1	14
	4	1	0	–	0.93			–									
	6	1	0	–	0.7			–									
	12	10	0	–	0.95			–									
	14	50	1	–	0.95	1	1	0	0.95								
	20	500	1	50	0.95	2	20	1	0.95								
11	3	1	0	–	0.93			–		5000	14	0.1	0.9	6	0.1	1	14
	6	1	1	–	0.7			–									
	13	10	1	–	0.95			–									
12	1	1	0	–	0.8			–		200	16	1	0.9	6	0.2	1	14
	4	1	0	–	0.93			–									
	7	1	0	–	0.93			–									
	10	10	0	–	0.94			–									
	12	20	0	–	0.95			–									
	14	50	1	–	0.95	1	1	0	0.95								
	18	500	1	20	0.95	1	10	1	0.95								
13	1	1	0	–	0.8			–		1000	16	1	0.9	6	0.2	1	14
	4	1	0	–	0.93			–									
	7	1	0	–	0.93			–									
	10	10	0	–	0.94			–									
	12	20	0	–	0.95			–									
	14	50	1	–	0.95	1	1	0	0.95								
	18	500	1	20	0.95	1	10	1	0.95								

Table 3.5 (continued)

Mode	Context model (FCM)					Context model (STMM)				Repeat model							z
	k	$1/\alpha$	IR	c	γ	Edit	$1/\alpha$	IR	γ	*RPN*	k	α	β	lim	γ	IR	
14	3	1	0	–	0.93			–		3000	16	1	0.9	6	0.2	1	14
	13	20	0	–	0.95			–									
	18	500	1	20	0.95	2	10	1	0.95								
15	4	1	0	–	0.93			–		5000	16	1	0.9	6	0.2	1	14
	7	1	0	–	0.93			–									
	14	50	1	–	0.95	1	1	0	0.95								
	18	500	1	20	0.95	1	10	1	0.95								

shorter sequences, while higher modes perform better for larger sequences. Nevertheless, model configurations can be manually done by passing parameters to the program. In the table, "k" denotes k-mer size. "α" is a parameter that allows to keep a balance between the maximum likelihood estimator and the uniform distribution in a model. For context models and repeat models, it is taken from user as $1/\alpha$ and α, respectively. To consider inverted repeats, the user should set IR = 1. "c" is the number of maximum collisions in cache-hash. "γ" denotes forgetting factor in a model. "Edit" denotes the number of edits in a substitution-tolerant Markov model. "*RPN*" is the maximum number of repeats models. "β" is a parameter for discarding or maintaining a certain repeat model. "lim" is a threshold value associated with β that may or may not accept a certain repeat model. Finally, "z" is the context order size for the competitive prediction context model.

Results and discussion

We tested Jarvis, state-of-the-art genomic data compressors and two general-purpose compressors on 15 DNA sequences (Table. 3.2) with various sizes and from different domains and kingdoms of biological organisms. The machine used for tests had a single core Intel® Xeon® CPU E7320 at 2.13 GHz frequency.

Tables 3.6 and 3.7 depict in detail compressed sizes and computational times, respectively, for applying Jarvis and other data compresses on the dataset. The following settings were used to run the methods: lzma with "-9" flag that provides the best compression, paq8 (paq8kx variant) with "-8" option that provides the least compression size, GeCo with "-tm 1:1:0:0/0 -tm 3:1:0:0/0 -tm 6:1:0:0/0 -tm 9:10:0:0/0 -tm 11:10:0:0/0 -tm 13:50:1:0/0 -tm 18:100:1:3/10 -c 30 -g 0.9", GeCo2 with the modes shown in Table 3.3 (described in Table 3.1), XM with 50 copy experts, and Jarvis with the modes shown in Table 3.6.

Fig. 3.7 illustrates compression ratios and speeds (in kilobase per second), respectively, for general-purpose and special-purpose methods. Compression ratios are calculated as total size of

Table 3.6: Compressed file sizes, in bytes, obtained by Jarvis and other state-of-the-art data compressors.

Name	lzma	paq8	CoGI	GeCo	GeCo2	XM	Jarvis (mode)
HoSa	42,292,440	40,517,624	51,967,817	38,877,294	38,845,642	38,940,458	38,660,851 (7)
GaGa	36,179,650	34,490,967	40,846,177	33,925,250	33,877,671	33,879,211	33,699,821 (6)
DaRe	12,515,717	12,628,104	17,084,450	11,520,064	11,488,819	11,302,620	11,173,905 (5)
OrSa	9,348,183	9,280,037	11,999,580	8,671,732	8,646,543	8,470,212	8,448,959 (5)
DrMe	8,016,544	7,577,068	8,939,690	7,498,808	7,481,093	7,538,662	7,490,418 (5)
EnIn	5,785,343	5,761,090	7,210,867	5,196,083	5,170,889	5,150,309	5,087,286 (4)
ScPo	2,722,233	2,557,988	2,921,247	2,536,457	2,518,963	2,524,147	2,517,535 (4)
PlFa	2,097,979	1,959,623	2,411,342	1,944,036	1,925,726	1,925,841	1,924,430 (4)
EsCo	1,185,704	1,107,929	1,307,943	1,109,823	1,098,552	1,110,092	1,095,606 (4)
HaHi	985,096	904,074	1,124,483	906,991	902,831	913,346	899,464 (3)
AeCa	413,886	380,273	454,357	385,640	380,115	387,030	380,507 (3)
HePy	415,161	385,096	457,859	381,545	375,481	384,071	374,362 (3)
YeMi	19,262	16,835	19,805	17,167	16,798	16,861	16,861 (2)
AgPh	12,183	10,754	12,243	10,882	10,708	10,711	10,745 (2)
BuEb	5441	4668	5291	4774	4686	4642	4690 (1)
Total	121,994,822	117,582,130	146,763,151	112,986,546	112,744,517	112,558,213	111,785,440

original sequences (shown in Table 3.2) divided by total size of compressed sequences (shown in Table 3.6), and compression speeds are calculated as total size of compressed sequences divided by total run times. As seen in Tables 3.6 and 3.7 and Fig. 3.7, Jarvis compresses the dataset 5.2% more than paq8, while it is 140 times faster. CoGI is the fastest method and can compress the dataset 2.3% better than gzip. Although our tool is 28 times slower than CoGI, it is able to provide 31% higher compression ratios. Jarvis shows an improvement of 1.1% and 0.9% in compression ratios over GeCo and GeCo2, respectively, although it is slightly slower than these two methods. Compared to XM that is the second-best tool in compression ratios, Jarvis improves the compression

Table 3.7: Computational time, in seconds, of applying Jarvis and other compressors on the dataset.

Name	lzma	paq8	CoGI	GeCo	GeCo2	XM	Jarvis
HoSa	552.5	85,269.1	25.2	648.6	652.4	5589.8	814.8
GaGa	468.7	64,898.9	19.9	503.2	494.7	3633.9	412.3
DaRe	170.0	29,907.7	8.2	215.9	198.8	785.2	284.9
OrSa	112.9	20,745.1	5.8	192.4	138.3	489.7	234.5
DrMe	85.6	14,665.8	4.3	114.6	102.4	362.6	66.7
EnIn	66.0	11,183.6	3.7	95.8	82.5	279.8	101.1
ScPo	23.0	4619.1	1.5	45.2	34.2	96.5	28.7
PlFa	18.3	4133.9	1.2	39.7	35.3	84.4	25.4
EsCo	8.1	1973.9	0.6	26.4	5.1	36.8	10.9
HaHi	6.9	1738.1	0.5	23.7	4.4	39.1	7.1
AeCa	2.2	675.3	0.2	17.0	1.9	10.3	2.2
HePy	2.3	715.1	0.2	17.2	1.9	11.2	2.7
YeMi	0.1	32.6	0.0	12.3	0.1	0.9	0.2
AgPh	0.0	20.1	0.0	12.1	0.1	0.9	0.1
BuEb	0.0	9.1	0.0	12.2	0.1	0.7	0.1
Total	1516.6	240,587.4	71.3	1976.3	1742.2	11,421.8	1991.7

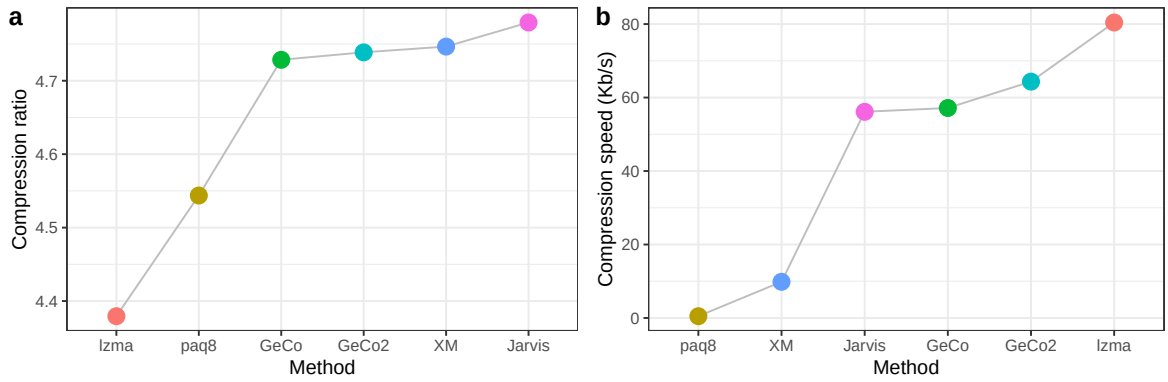

Figure 3.7: Applying state-of-the-art compressors on 15 sequences (described in Table 3.2). (a) compression ratio; (b) compression speed, in kilobase per second. Note that CoGI was an outlier, therefore, it was removed from this figure.

by 0.6%, while it is 5.7 times faster and uses half RAM. As an example, it uses ~7 GiB RAM for the largest sequence. Overall, special-purpose methods could provide better compression ratios than general-purpose ones, including lzma and paq8.

Jarvis can run on 15 different modes. Fig. 3.8 provides a comparison of these modes for the three largest sequences, i.e., HoSa, GaGa and DaRe. As an instance, applying our tool on HoSa sequence in mode 12 achieves a bit-rate of 1.6 bps, which is 1% better than mode 7. As another instance, running on DaRe, mode 1 is the fastest mode and mode 9 provides the best bit-rate. There is a trade-off between time and memory usage, when running on different modes. Depending on the needs of a user, the "best" mode can be chosen, which depends on a combination of number of models, depths and estimator parameters, among others.

In order to test the idea that repetitive regions in a sequence are better modeled by stochastic repeat models than by context models, we added a very repetitive sequence (exogenous from the benchmarking dataset) to the assembled human Y-chromosome and run GeCo2 and Jarvis on the sequence. The result, illustrated in Fig. 3.9, denotes that Jarvis performs better than GeCo2 in all modes. As an example, Jarvis in mode 12 compresses the sequence 5.4% better than GeCo2 in

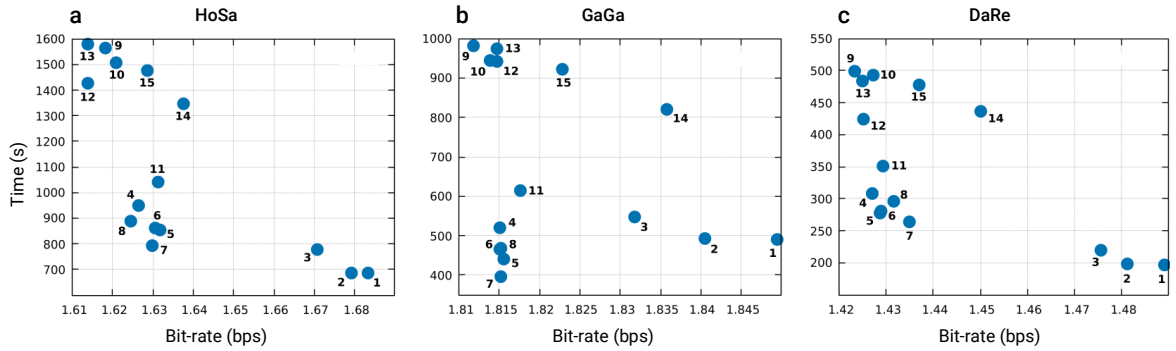

Figure 3.8: Running Jarvis with different modes on (a) HoSa; (b) GaGa; and (c) DaRe.

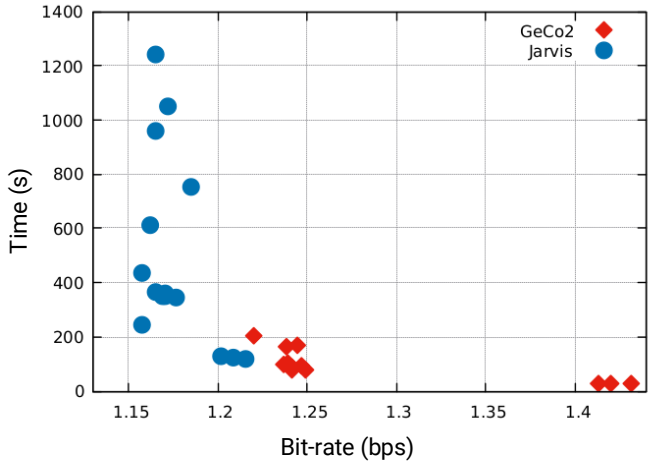

Figure 3.9: Times of carrying out GeCo2 and Jarvis in all modes on a sequence including the human Y-chromosome and a very repetitive sequence.

mode 15, using approximately the same computational time, which shows that adding repeat models to context models is important in such cases.

3.2 Compression of amino acid sequences

3.2.1 Introduction

Proteins play a key role in all organisms, since they are involved in the control of the development of those organisms [145], [184], [185]. Recently, the importance of protein sequences has increased, namely after the start of the Human Proteome Project (HPP) [186]–[189]. Amino acid sequences are successions of letters, usually from an alphabet of 20 symbols, which indicate the order and nature of amino acids within a protein. Each amino acid corresponds to one or more DNA/RNA codons, that are represented by a 4-symbol alphabet of {A, C, G, T/U}.

Possible DNA codons associated with amino acids, and also their distribution percentage in two large datasets of DB1 (uniprot_sprot.fasta.gz) and DB2 (uniprot_trembl.fasta.gz) [190], that are available online[1], are listed in Table 3.8. As can be seen, from a specific amino acid, we cannot retrieve a single codon. The W and M, which is the start codon, are exceptions. As the result of DNA/RNA sequencing process, there might be extra symbols in the sequences, including but not limited to B, Z and X. Appearance of such symbols show that the identity of a residue cannot be determined by a crystallographic or chemical analysis of a protein [191].

Compression of amino acid sequences, with intrinsically disordered nature, is known to be com-

[1] ftp://ftp.uniprot.org/pub/databases/uniprot/current_release/knowledgebase/complete

Table 3.8: Representation of amino acids and possible DNA codons associated with them, along with their distribution percentage in two large protein databases.

Sym	Amino Acid	DNA codon(s)	DB1%	DB2%
A	Alanine	GCT, GCC, GCA, GCG	8.26	9.10
C	Cysteine	TGT, TGC	1.37	1.21
D	Aspartic acid	GAT, GAC	5.46	5.45
E	Glutamic acid	GAA, GAG	6.73	6.16
F	Phenylalanine	TTT, TTC	3.86	3.92
G	Glycine	GGT, GGC, GGA, GGG	7.08	7.26
H	Histidine	CAT, CAC	2.27	2.19
I	Isoleucine	ATT, ATC, ATA	5.93	5.70
K	Lysine	AAA, AAG	5.82	4.99
L	Leucine	TTA, TTG, CTT, CTC, CTA, CTG	9.65	9.87
M	Methionine	ATG	2.41	2.38
N	Asparagine	AAT, AAC	4.06	3.88
P	Proline	CCT, CCC, CCA, CCG	4.72	4.85
Q	Glutamine	CAA, CAG	3.93	3.79
R	Arginine	CGT, CGC, CGA, CGG, AGA, AGG	5.53	5.71
S	Serine	TCT, TCC, TCA, TCG, AGT, AGC	6.61	6.69
T	Threonine	ACT, ACC, ACA, ACG	5.35	5.57
V	Valine	GTT, GTC, GTA, GTG	6.86	6.88
W	Tryptophan	TGG	1.09	1.29
Y	Tyrosine	TAT, TAC	2.92	2.93
B	D or N	GAC, GAT, AAC, AAT	0.00	5.03
Z	E or Q	GAA, GAG, CAA, CAG	0.00	0.00
X	<any>	<any>	0.00	0.04

Note: the symbol X in a sequence can be interpreted as any DNA codon [95], [191].

plex [192]–[195]. Nevill-Manning and Witten, in [196], even propose that these sequences are approximately incompressible, given the marginal compression gains obtained by their proposed method. We give another example. Considering a sequence with the cardinality of 20, a compressor should not need more than $\log_2 20 = 4.322$ bits, theoretically, to save each amino acid. However, gzip, as one of the most known compressors, needs more than 4.322 bits to save each symbol, due to having a short memory model, among others.

A few protein compressors have been proposed in the literature [100], [140], [145], [184], [193], [196]–[200]. In [145], [196] statistical models that use probabilities of contexts are exploited. The first protein compression algorithm, CP (Compress Protein) [196], employs probability to put weight on all contexts with the maximum of a certain length, based on their similarity to the current context. The XM (eXpert Model) method [145] encodes each symbol by estimating its probability distribution based on previous occurrences of that symbol. The probability values are finally redirected to an arithmetic encoder.

"Approximate repeats" are utilized in the methods presented in [140], [197]. Matsumoto *et al.* consider in [140] palindromes and approximate repeats as two characteristic structures. In this method, an LZ77-type algorithm and the CTW algorithm [201] encode long and short approxi-

mate repeats, respectively. Heuristics are also used to improve compression ratios. The ProtComp method [197] models approximate repeats and builds an optimal substitution probability matrix for the purpose of exploring medium and long-range correlations.

Employing a "dictionary" is considered in [184], [198], [199]. The ProtCompSecS method [198] combines ProtComp [197] with a dictionary based method that uses DSSP (Dictionary of Protein Secondary Structure) database [202]. CaBLASTP, proposed in [199], introduces a course database to store unique data that comes from the original protein sequence database. Then, sequence segments that align to previously seen sequences are added to a link index. This method combines a dictionary-based compression algorithm and a sequence alignment algorithm. In [184], a dictionary-based scheme is proposed that works based on random or repeated protein sequence reduction and ASCII replacement.

In [200] a heuristic method is introduced that exploits protein domain compositions. It builds a hyper-graph for the proteome based on evolutionary mechanisms of gene duplication and fusion; then, based on a minimum spanning tree, a cost function is minimized to compress the proteome.

Compression of amino acid sequences, besides reducing storage space, can be exploited to predict and uncover structure of proteins [195], [203], namely through fusions and duplications [204], which are possibly linked with new functionalities [200]. Protein classification [205] and domain identification [206] are other examples.

3.2.2 Methods

AC works based on cooperation between finite-context models and substitution-tolerant Markov models with several depths, which is described in detail in Section 2.2. The mixture weights associated with each model are updated based on its performance, that is related to the specific forgetting function of each model. Note that in protein sequences we confront with an alphabet of, usually, 20 amino acids (cardinality = 20) instead of 4 in genomic sequences.

3.2.3 Results and discussion

The AC tool is implemented in the C language and is publicly available[1] under GNU GPLv3 license. Besides enabling users to customize the parameters of the compression models, this tool provides seven predefined compression levels, which are obtained by authors experience. The machine used for tests had a 4-core 3.40 GHz Intel® Core™ i7-6700 CPU with 32 GiB RAM. In order to compare AC with existing methods, we have used 16 datasets that are described in Table 3.9 and are available online[2,3]. We have selected the datasets with different cardinalities and lengths and from different

[1] www.github.com/cobilab/ac

[2] ftp://ftp.uniprot.org/pub/databases/uniprot/current_release/knowledgebase/reference_proteomes

[3] sweet.ua.pt/pratas/datasets/AminoAcidsCorpus.zip

Table 3.9: Datasets used for comparing AC with other compressors.

Name	Species	Domain	Kingdom	Length (b)	Cardin.[1]
BT	*Bos taurus*	Eukaryota	Animalia	12,845,466	24
HS	*Homo sapiens*	Eukaryota	Animalia	3,295,751	19
SC	*Saccharomyces cerevisiae*	Eukaryota	Fungi	2,900,352	20
HT	*Haloterrigena turkmenica*	Archaea	Euryarchaeota	1,473,976	20
EC	*Escherichia coli*	Bacteria	Eubacteria	1,308,765	21
LC	*Lactobacillus casei*	Bacteria	Eubacteria	809,301	20
SA	*Staphylococcus aureus*	Bacteria	Eubacteria	796,785	20
HI	*Haemophilus influenzae*	Bacteria	Eubacteria	509,519	20
MJ	*Methanococcus jannaschii*	Archaea	Euryarchaeota	448,779	20
DA	*Desulfurococcus amylolyticus*	Archaea	Crenarchaeota	400,476	20
AP	*Acanthamoeba polyphaga*	Viruses	dsDNA viruses	341,649	21
HA	*Hadesarchaea archaeon*	Archaea	Euryarchaeota	218,643	21
FM	*Fomitiporia mediterranea*	Eukaryota	Fungi	161,738	20
FV	*Fowlpox virus*	Viruses	dsDNA viruses	80,735	21
XV	*Xanthomonas virus Xp10*	Viruses	dsDNA viruses	13,372	20
EP	*Enterococcus phage*	Viruses	dsDNA viruses	4184	20

[1] Cardinality demonstrates number of different amino acids in each sequence.

domains and kingdoms.

AC and several other compression methods were tested on the collection of files, described in Table 3.9. The results are shown in Tables 3.10, 3.11 and 3.12, in terms of bit-rate, time and memory usage, respectively. Note that all methods were carried out with their best options, e.g. for lzma and paq8l, "-9" and "-8" options were used, respectively, which provide the best bit-rates.

The box-plot in Fig. 3.10a shows that AC provides, on average, the best bit-rates. The largest alphabet cardinality of the datasets that we have used is 24, therefore, a compressor needs theoreti-

Table 3.10: Bit-rates of AC and existing protein compressors, in bits per symbol (bps).

Dataset	gzip	bzip2	7zip	lzma	paq8l	AC
BT	4.52	4.25	3.21	3.21	3.15	3.05
HS	4.61	4.26	4.03	4.03	3.90	3.79
SC	4.64	4.30	4.13	4.13	3.94	3.88
HT	4.53	4.25	4.06	4.06	3.91	3.83
EC	4.68	4.35	4.27	4.27	4.08	4.04
LC	4.66	4.30	4.27	4.27	4.08	4.06
SA	4.65	4.30	4.26	4.25	4.06	4.06
HI	4.67	4.32	4.29	4.27	4.10	4.10
MJ	4.59	4.27	4.21	4.20	4.00	4.00
DA	4.62	4.28	4.22	4.22	4.03	4.03
AP	4.59	4.27	4.14	4.14	3.97	3.99
HA	4.66	4.32	4.23	4.22	4.09	4.08
FM	4.42	4.10	3.54	3.53	3.60	3.43
FV	4.67	4.31	4.19	4.18	4.07	4.06
XV	4.68	4.37	4.31	4.26	4.15	4.14
EP	4.69	4.49	4.59	4.43	4.30	4.32

Table 3.11: Compression time of different compressors, in milliseconds.

Dataset	gzip	bzip2	7zip	lzma	paq8l	AC
BT	500	1052	6274	9182	778,461	97,446
HS	135	265	912	1685	195,154	22,828
SC	119	240	703	1396	172,661	17,935
HT	66	121	243	577	86,516	10,812
EC	54	105	212	474	79,108	7175
LC	32	67	138	257	47,578	4495
SA	32	70	124	256	46,833	4436
HI	21	41	88	153	30,196	1875
MJ	20	36	71	132	26,502	1703
DA	17	32	63	122	23,661	1537
AP	15	27	75	103	20,265	2186
HA	10	17	35	68	13,106	968
FM	7	15	31	57	9931	1709
FV	5	7	16	39	5165	485
XV	1	2	5	26	1107	34
EP	1	1	3	24	440	16

cally $\log_2 24 = 4.58$ bits, at most, to save each amino acid symbol. This is plotted with a red line. As it is shown, for most datasets, gzip needs more than 4.58 bits to save each symbol. This shows that gzip is not appropriate for amino acid sequences, although it is a widely used compressor. Fig. 3.10b shows that AC is nine times faster than its main competitor, paq8l. The memory usage of different tools is shown in Fig. 3.10c. AC and paq8l use ~1 GiB of memory, which is available on present-day standard computers. Fig. 3.10d shows the bit-rates obtained by different tools as a function of their time usages. Since amino acid sequences are known to be complex [192]–[196], even a small reduction in bit-rate can be considered as a significant improvement. As another example, improvements in the lossless compression of amino acid sequences is an indication that shows that the model used

Table 3.12: Memory usage of different protein compressors, in megabytes.

Dataset	gzip	bzip2	7zip	lzma	paq8l	AC
BT	4	8	151	177	1200	4443
HS	4	8	46	95	1035	3445
SC	4	8	42	91	1023	1745
HT	4	7	26	79	990	1606
EC	4	7	25	78	986	1101
LC	4	7	18	73	979	960
SA	4	7	18	73	978	959
HI	4	5	15	71	973	508
MJ	4	5	14	70	970	496
DA	4	4	14	70	969	492
AP	4	4	13	69	969	863
HA	1	4	10	68	965	481
FM	1	4	9	68	962	1120
FV	1	2	7	67	934	409
XV	1	1	3	47	617	58
EP	1	1	3	46	359	58

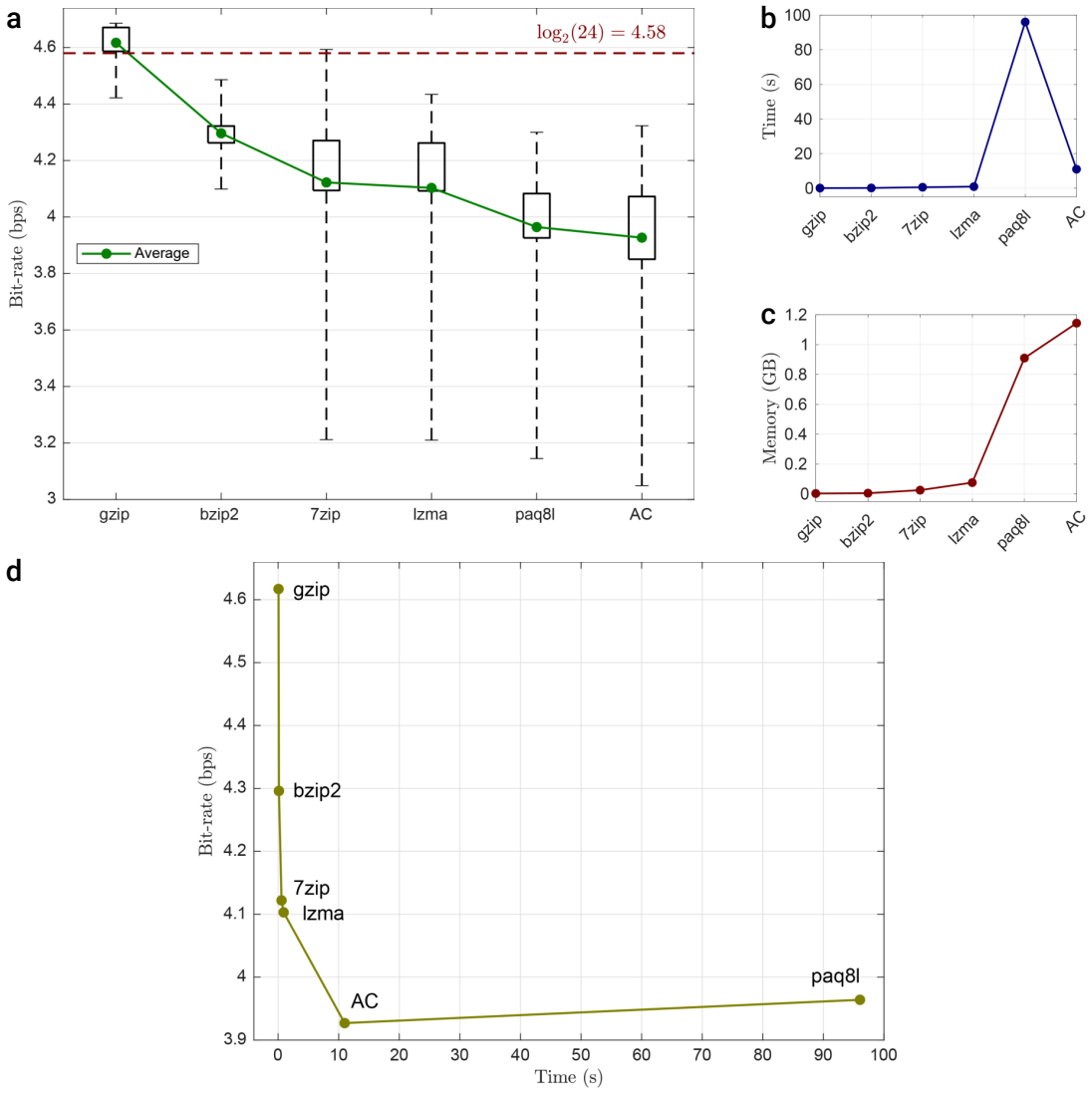

Figure 3.10: (a) Bit-rates; (b) times; (c) memory usages; (d) bit-rates versus times of AC and other protein compressors, obtained by testing on 16 different sequences.

is more reliable and may serve as an extra expert for predicting the structure of proteins [207]; this can lead us to the development of better diagnostics and therapeutics. As it is shown, AC provides the lowest bit-rate in a time comparable to well-known general-purpose tools.

To analyze the behavior of AC in more detail, we have compressed 10,677 sequences from different domains, which are described in Table 3.13 and are available online[1]. Normalized compression results are shown in Fig. 3.11. The NC shows the average number of bits needed to represent each amino acid in a sequence. Note that the harder a sequence is to be compressed by AC, the greater the value of NC is.

In Fig. 3.12, histogram of sizes are demonstrated. The widest range of sizes belongs to Eukaryota

[1]ftp://ftp.uniprot.org/pub/databases/uniprot/current_release/knowledgebase/reference_proteomes

Table 3.13: Datasets used exclusively by AC.

Domain	No. seq	Min size (b)	Max size (b)	Total size (b)
Bacteria	8586	16,117	4,652,819	8,336,646,144
Eukaryota	1064	161,738	43,726,990	6,455,205,888
Archaea	465	33,380	1,528,994	256,815,104
Viruses	562	201	341,649	7,790,592
Total	10,677	201	43,726,990	15,056,457,728

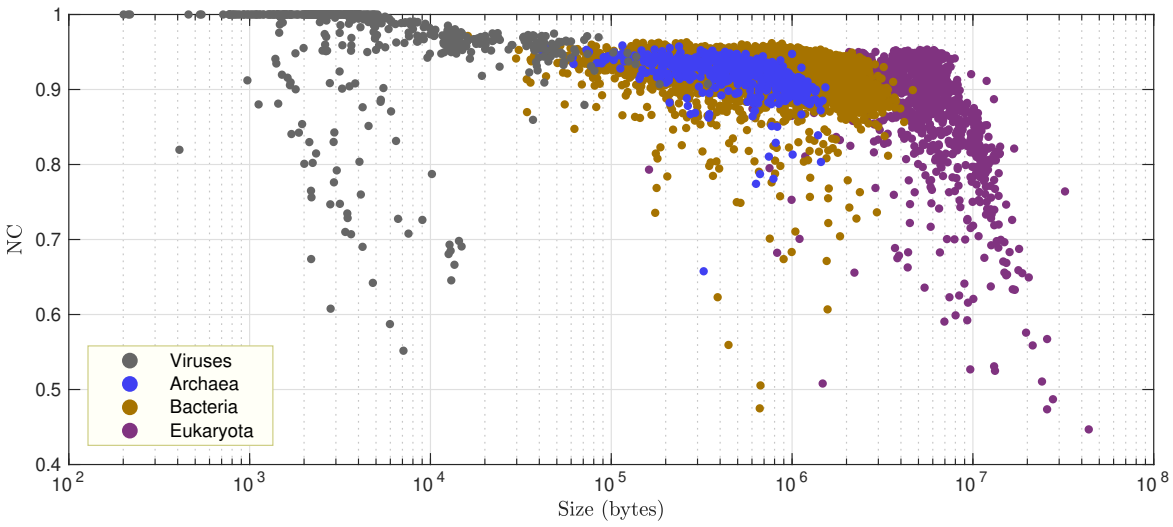

Figure 3.11: Normalized compression results of AC compressing 10,677 proteins, described in Table 3.13.

datasets. In Fig. 3.13a, violin plot of NCs for the datasets described in Table 3.13 is illustrated. In Fig. 3.13b is shown the average NC values for the 10,677 protein sequences from different domains. Viruses have the greatest average NC values, which means they are the most complex sequences to be compressed by AC. On the contrary, Eukaryota datasets are the least complex ones. This result conforms to the similar study on nucleotide level [208], with the only exception that in DNA, archaea are slightly more complex than bacteria.

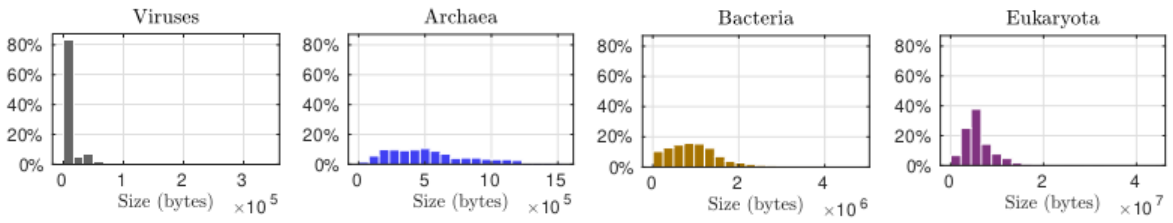

Figure 3.12: Histogram of sizes for the datasets described in Table 3.13.

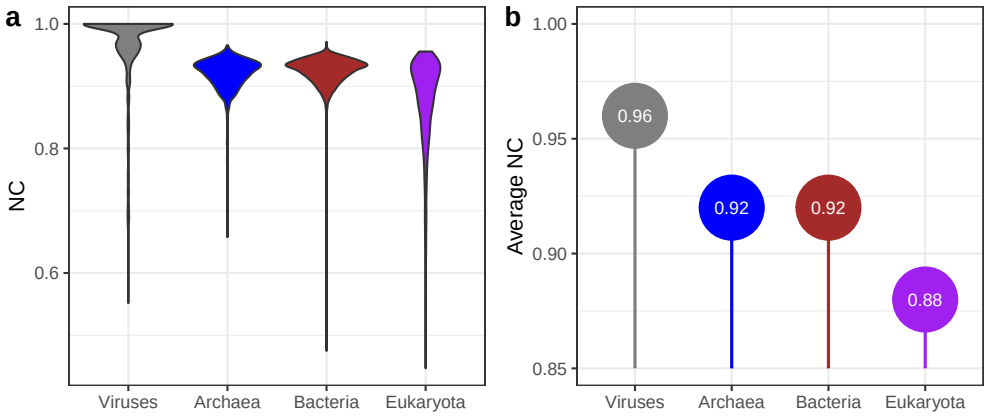

Figure 3.13: (a) Violin plot of NCs and (b) average NCs for the sequences described in Table 3.13.

3.3 Conclusions

With the ever-increasing production of genomic and proteomic sequences, due to the development of omics sequencing technologies, there is a need to store such data in an efficient way. In this chapter we introduced two genomic data compressors, GeCo2 and Jarvis, and an amino acid sequence compressor, AC.

In GeCo2, which is an improved version of GeCo, we enhanced mixture of the context models, added specific cache-hash sizes, provided the ability of considering solely the inverted repeats, a new command line interface, several pre-computed compression modes, and several code optimizations. The results of applying our tool on several DNA sequences showed an improvement over state-of-the-art compressors using less computational resources.

Jarvis is a lossless genomic data compressor which employs two classes of models, namely weighted context models (including FCMs and STMMs) and weighted stochastic repeat models, and uses a competitive prediction model to estimate for each nucleotide the best class among them. Testing on DNA sequences from various domains and kingdoms showed that our tool achieves a higher compression ratio than state-of-the-art approaches, including GeCo2.

AC is a novel method for lossless compression of amino acid sequences, which uses a combination of FCMs and STMMs. Experimental results show that this method provides the best bit-rates compared to several existing protein compressors. Also, AC performs nine times faster than its competitor, paq8l. Furthermore, we have employed AC to analyze compressibility of 10,677 sequences from different domains, using the normalized compression measure. The results show that viruses are the hardest sequences to be compressed. Archaea and bacteria are the second hardest ones, and eukaryota are the easiest sequences to be compressed.

Chapter 4

Secure encryption of genomic data

The ever-increasing growth of high-throughput sequencing technologies has led to a great acceleration of medical and biological research and discovery. As these platforms advance, the amount of information for diverse genomes increases at unprecedented rates. Confidentiality, integrity and authenticity of such genomic information should be ensured due to its extremely sensitive nature. In this chapter is proposed Cryfa, a fast secure encryption tool for genomic data, namely in FASTA, FASTQ, VCF, SAM and BAM formats, which is also capable of reducing the storage size of FASTA and FASTQ files. The proposed tool uses AES encryption combined with a shuffling mechanism, which leads to a substantial enhancement of the security against low-data complexity attacks. Compared to AES Crypt, a general-purpose encryption tool, Cryfa is an industry-oriented tool that is able to provide confidentiality, integrity and authenticity of data at three times more speed; in addition, it can reduce the file sizes to 1/3. Due to the absence of a method similar to our tool, we have simulated its behavior with a combination of encryption and compression tools, for comparison purpose. For instance, Cryfa is nine times faster than its fastest competitor in FASTA files. Also, it has a very low memory usage (only a few megabytes), which makes it feasible to run on any computer.

4.1 Introduction

The rapid advancement in HTS sequencing technologies has triggered a revolution in personalized medicine, biotechnology and ancient DNA studies [209], [210]. However, it raises critical issues regarding preserving security of the genomic data, which is highly sensitive due to its nature.

Authenticated encryption has the potential to address the issues of genomic data security, by allowing only authorized parties to access such data. This cryptographic scheme is able to appropriately ensure the key criteria of integrity, confidentiality and authenticity of the genomic data, under a single and easy-to-use programming interface [211], [212].

General-purpose encryption methods, albeit directly applicable, do not take into consideration specific properties of genomic data files; for example, FASTA files contain headers, beginning with the ">" character, and DNA bases, including "A", "C", "G", "T" and "N" symbols. As another example, FASTQ files comprise headers, beginning with the "@" character, bases, "+" separators and quality scores. Also, VCF files are required to begin with "##fileformat=VCF" string, which contains 16 characters and must be at the first line. Therefore, a special-purpose encryption approach is required, that is able to protect the genomic data against known-plaintext attacks (KPA) [49], as well as low-data complexity attacks [213].

In this chapter, we present the Cryfa tool, that follows industry recommendations for upholding security of in-transit and at-rest genomic data. This tool addresses secure encryption of such data, along with compacting FASTA and FASTQ sequences by a fixed-block transformation, followed by shuffling the transformed information and ultimately, performing a fast authenticated encryption on the shuffled content. The encryption is performed by the advanced encryption standard (AES), announced by the U.S. National Institute of Standards and Technology (NIST), which is a symmetric-key algorithm. It processes data blocks, using cipher keys, based on a substitution-permutation network [214]. Operating the shuffling before encryption is crucial, since it prevents an adversary to break the encryption by low-data complexity or KPA attacks.

Applying the AES method distributes information uniformly, therefore it is needed to perform the compacting phase before encryption. In this case, however, there are a few compression side-channel attacks, such as CRIME- and BREACH-based attacks, which explore redundancy in a compressed text by observing the size of its cipher text [215]. These methods can attack a compression method which utilizes redundancy of a text and produces blocks of compressed text with variable sizes. Regarding genomic data, they can exploit conserved sequences [216] to identify the encrypted species by a brute-force attack to the beginning of the sequences. To make Cryfa resistant against such security exploitations, we perform a fixed-size compacting, i.e., we pack equally sized blocks of symbols in FASTA or FASTQ files, independently of their redundancy.

4.2 Methods

Fig. 4.1 shows the schema of the Cryfa method. Fig 4.1a illustrates compaction & encryption of a FASTA/FASTQ file, that includes splitting headers, bases and quality scores, compacting each one of them, shuffling the joined pack and encrypting the shuffled content. Fig. 4.1b illustrates decryption & unpacking of a file that has already been compacted & encrypted by Cryfa. It includes decrypting the compacted & encrypted file, unshuffling the decrypted content and unpacking the unshuffled content. Facing a genomic file other than FASTA/FASTQ, the "split" and "compact" phases in encryption, and "unpack" in decryption sides will be bypassed. In the following sections, packing/unpacking, shuffling/unshuffling and encryption/decryption processes are described in detail.

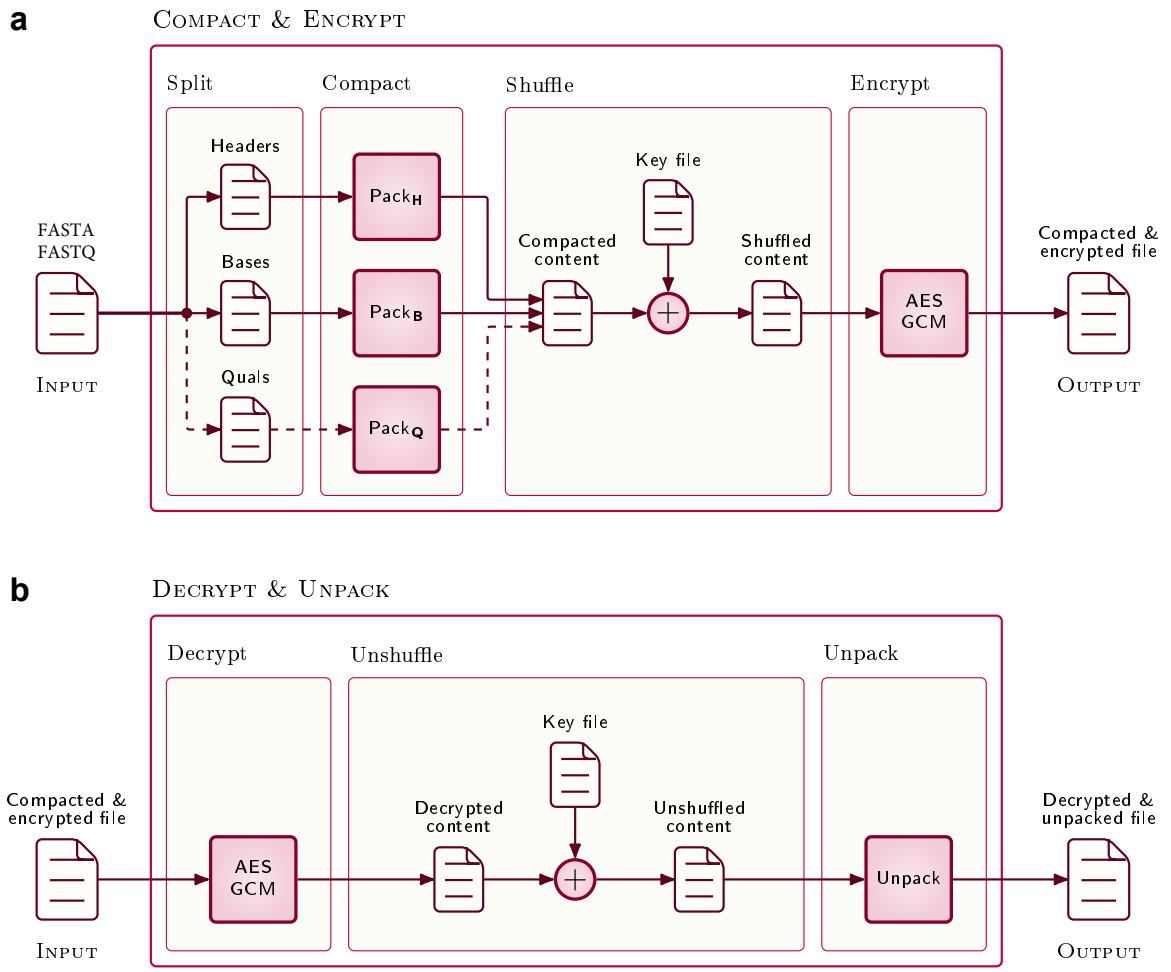

Figure 4.1: The schema of Cryfa. (a) The process of compaction & encryption of a FASTA/FASTQ file. Dashed lines show that quality scores are not considered for FASTA files; (b) decryption & unpacking of a file that has already been compacted & encrypted by Cryfa.

4.2.1 Pack and unpack

The operations of packing headers, bases and quality scores are similar to each other, with the only difference being the number of symbols considered for each tuple varies. To clarify the process, an example of packing DNA bases is provided, in which each tuple contains three symbols. To pack DNA bases, a hash table is used to map each triplet of bases into an integer that represents the ASCII value of a character that falls within the range $[0, 6^3 - 1]$. The number of integers in this interval is $6^3 = 216$, which corresponds to the number of all possible triplets that can be built by six symbols of A, C, G, T, N and X. We consider the X symbol based on the fact that, in DNA sequences, we might face a number of bases other than A, C, G, T and N; since they are rarely presented in a sequence, we pay a slight penalty dealing with them. This penalty would be to convert each of such bases to an X symbol and insert an extra byte to point out which base it had been.

Fig. 4.2 shows an example of the packing process. All triplets of the sequence, except the last

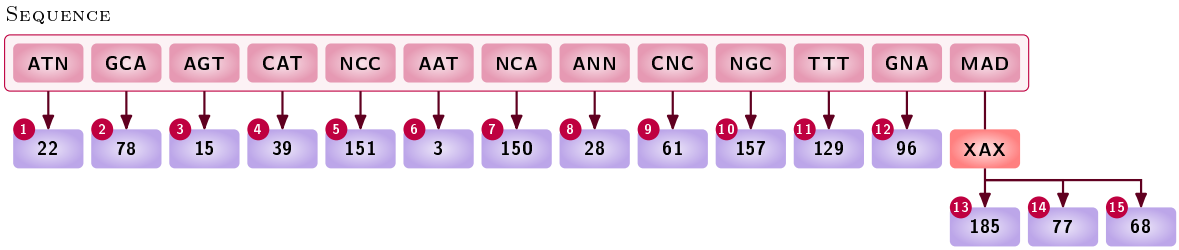

Figure 4.2: An example of packing DNA bases.

one, are directly mapped to an integer between 0 and 215. Since the two bases M and D of the last triplet, MAD, are absent from the alphabet {A, C, G, T, N}, they are first converted to X symbols; then, the obtained triplet, XAX, is mapped into three integers of which two ones are penalties. The first integer, 185, corresponds to the XAX triplet, while the second and the third integers, 77 and 68, are ASCII codes for M and D symbols, respectively. Size of the packed sequence is calculated as

$$\text{Size of packed sequence (bit)} = \left\lceil \frac{|S|}{n} \right\rceil \left\lceil \log_2 |\Theta|^n \right\rceil + 8x, \tag{4.1}$$

in which $|S|$ is the original sequence size in bits, n is the number of bases per pack, $|\Theta|$ is the alphabet size, 8 is the number of bits per ASCII character and x is the number of penalty symbols, that is Xes. In fact, $\lceil |S|/n \rceil$, $\left\lceil \log_2 |\Theta|^n \right\rceil$ and $8x$ show the number of packs, number of bits per pack and the penalty, respectively. $|\Theta|$ is size of the alphabet {A, C, G, T, N, X}, that is 6. In this example, size of the original sequence is 13 triplets × 3 bases × 8 bits = 312 bits, while the size of the packed sequence is $\lceil 39/3 \rceil \left\lceil \log_2 6^3 \right\rceil + 8 \times 2 = 120$ bits. Therefore, the sequence is compacted by a factor of 312/120 = 2.6.

The Unpacking process is the inverse of the packing process, in a sense that a lookup table is employed to map each integer, representing the ASCII value of each character, to a tuple of symbols. If that tuple includes any X symbols, that X is replaced by the character corresponding to the next integer read from the content being unpacked. The unpacking operations that are performed on headers, bases and quality scores are identical.

4.2.2 Shuffle and unshuffle

The shuffling phase of Cryfa is of crucial importance. With the following examples, we show the difference between applying and not applying the shuffling. Note that in these examples, the attacker is called Chuck. Fig. 4.3 demonstrates the first example. Assume that a genomic file in VCF format has been encrypted; Chuck has access to this encrypted sequence and also, knows that AES cipher has been employed for the encryption. AES is a block cipher, therefore, he decides to decrypt the first block of encrypted sequence, which includes 128 bits. Assume, the number of characters required for a password is 8. For the purpose of decryption, he performs an exhaustive search on passwords, in the sense that he guesses a password and tries to decrypt that block of data using that password; if

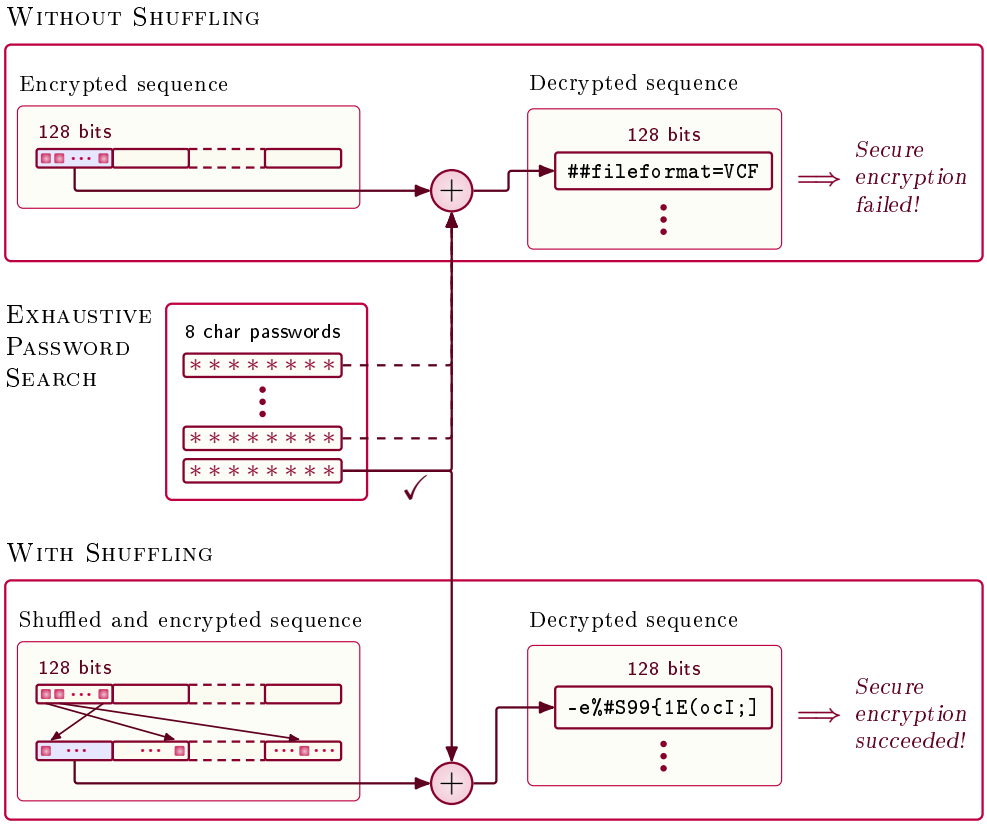

Figure 4.3: An example to show the importance of applying shuffling in Cryfa, in which an attacker tries exhaustive password search to break the encryption.

he could not obtain a meaningful set of characters, he guesses and tries the next password. Assume Chuck knows that the original file is in VCF format. Note that since every VCF file must begin with "##fileformat=VCF" string, this string is accounted as a "meaningful set of characters" for Chuck. Therefore, whenever he could find a password by which he could break the encryption and obtain "##fileformat=VCF", with 128 bit length, he terminates the exhaustive search process. Since the exhaustive attack to passwords with 8 characters is affordable with the modern computers, Chuck would be able to break the encryption in a feasible time t_0. Note that if Chuck did not know that the original file format was VCF, he could have searched for sets of characters meaningful in other file formats; for instance, FASTQ files have an "@" character followed by a sequence identifier and an optional description, in their first lines. Most of the times, the identifier is known.

Cryfa shuffles the sequence before encrypting it; this way, the first block of an encrypted file does not correspond to the first block of the original file. Therefore, Chuck is not able to decrypt the first 128 bit block and obtain the "##fileformat=VCF" string by an exhaustive attack to passwords. In order for Chuck to break the encryption, he needs to guess a password and use it to decrypt the entire file, since shuffling changes the positions of all characters. Then, he has to unshuffle it, employing the password he has guessed earlier. If the resulting file has the format of a VCF file, he would reach his goal, otherwise he has to guess another password and repeat the same steps.

Thus, to recover the original sequence, Chuck needs to perform the exhaustive attack to passwords and decrypt the entire sequence, instead of only 128 bits of it, and also he needs to unshuffle the entire decrypted file. As an instance, the size of one real dataset we employed in our experiments, HS-ERR031905_2, is ~11 GiB (see Section 4.3.1); performing the aforementioned steps on such a file takes a huge time $t_1 \gg t_0$.

The following scenario, along with Fig. 4.4, provide another example to demonstrate the importance of shuffling in Cryfa. Let S_0 be the original genomic sequence, C_0 be an efficient compressor other than Cryfa and E_0 be an encryptor. By compressing S_0 with C_0, the compressed sequence S_{C_0} is obtained, and by encrypting S_{C_0} with E_0, the compressed and encrypted sequence S_{CE_0} is obtained; these operations are performed in the compression & encryption side. The sequence S_{CE_0}, the compressor C_0 and the encryptor E_0 are available to an attacker, called Chuck.

Assume the file to be protected resides in NCBI database. Assume also that Chuck's goal is to recover the original sequence S_0 by breaking the encryption of S_{CE_0}. For this purpose, he downloads

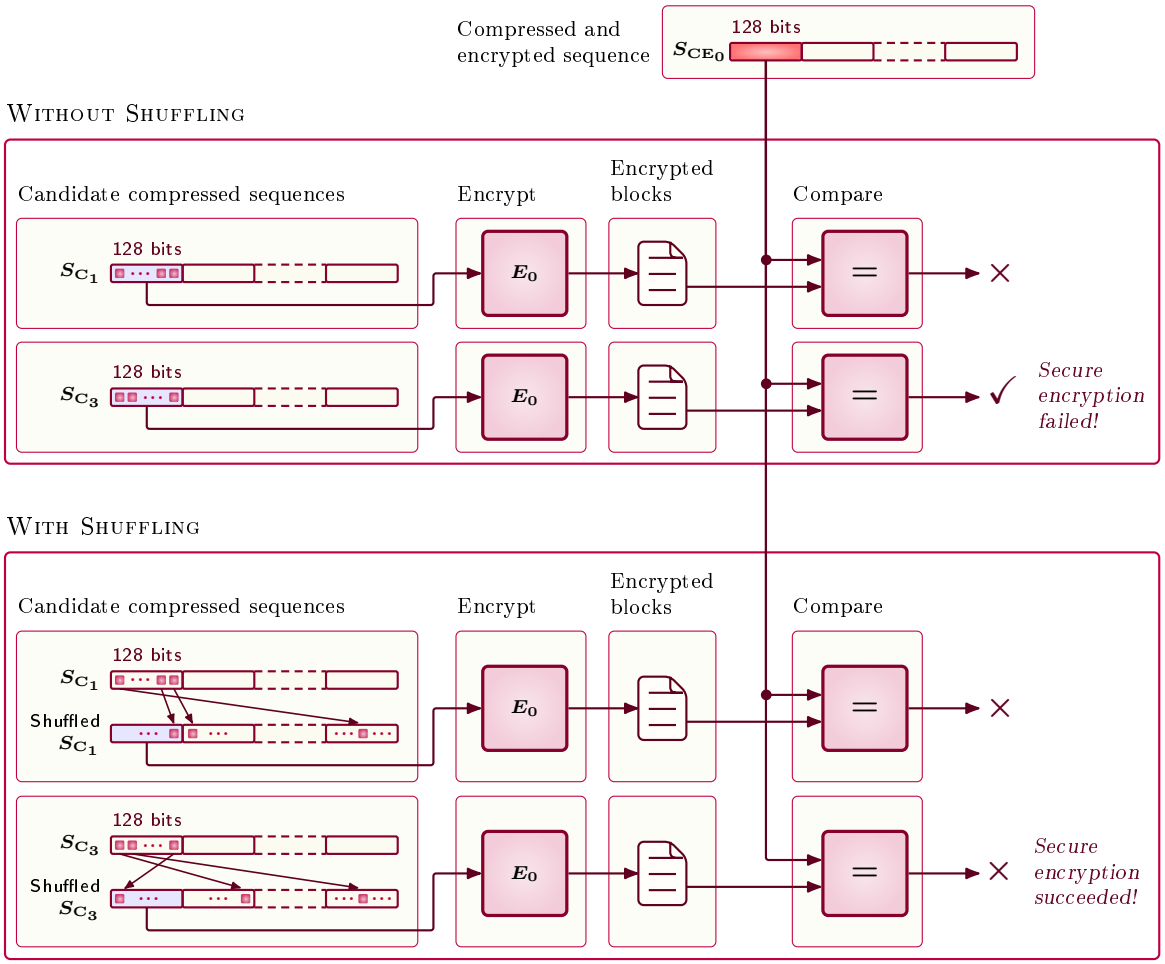

Figure 4.4: An example to show the importance of shuffling in Cryfa. An attacker downloads the entire sequences in NCBI database and encrypts them. By comparing each encrypted sequence with the target file, the attacker tries to break the encryption.

from the NCBI database all sequences $S_1, S_2, \ldots, S_n$ from different species, such as viruses, fungi, bacteria, and so on; then, he compresses them by C_0 and obtains $S_{C_1}, S_{C_2}, \ldots, S_{C_n}$. Note that compressing these sequences with current compressors takes from a few hours to a few days (at maximum). An encryptor, which is a tool that performs AES block cipher in the case of this study, may add during its encryption only a number of bits to a file. Hence, the size of the encrypted file is approximately the same as the size of the not-yet-encrypted file. Having this in mind, Chuck compares the sizes of each of S_{C_1}, S_{C_2}, ..., S_{C_n} with the one from S_{CE_0}, and selects those sequences with approximately the same size as S_{CE_0}. Let S_{C_i} be each one of these sequences, in which $1 \leq i \leq n$. A single S_{C_i} is the compressed version of the original sequence that he is looking for. In the normal case, each of S_{C_i} sequences needs to be encrypted completely, then the results are compared with S_{CE_0}; if the two files are exactly the same, Chuck has reached his goal. However, on one hand, we know AES is a block cipher that encrypts a file using fixed data blocks of 128 bits. On the other hand, C_0 is a compressor that does not shuffle the data, during the compression. Therefore, the first 128 bit block of an S_{C_i} is encrypted to the first 128 bit block of S_{CE_i}; the same happens to the second block, and so forth. Thus, Chuck does not need to encrypt the whole sequence S_{C_i} and compare the result with S_{CE_0}; 128 bits is sufficient. This way, the size of encryption problem is considerably reduced. Hence, he can encrypt the first 128 bit block of an S_{C_i} sequence by guessing the password by exhaustive search. If the result file is equal to the first 128 bits of S_{CE_0}, he has reached his goal, that is to recover the original sequence S_0 by decompressing that S_{C_i}. Note that Chuck excludes the header bits that were introduced to the file in order to support AES decryption.

Regarding Cryfa, Chuck has access to the ciphertext and his goal is to decrypt it. Note that he is not capable of obtaining the result of a tampered input. Cryfa shuffles the data after compaction and passes it to the encryption phase. This way, the first block of an encrypted sequence will not correspond to the first block of the compacted sequence, before shuffling. Therefore, to recover the original sequence S_0, the entire sequence of shuffled S_{C_i} needs to be encrypted and compared with the entire sequence of S_{CE_0}. When dealing with big files, like the ~11 GiB one mentioned in the previous example, this recovery can take an enormous time greater than the case when we do not perform shuffling.

Another aspect of the importance of shuffling before encryption is that it prevents an adversary to break the encryption by low-data complexity attacks [217]–[219] or known-plaintext attacks [220]–[222]. Applying the shuffling, by employing uniform pseudo-random number generators, leads to permuting the plaintext uniformly and consequently, transforming the contents into pseudo high-data complexity. This makes Cryfa resistant against low-data complexity attacks.

An example of shuffling and unshuffling processes is depicted in Fig. 4.5. To shuffle the packed content, obtained by the packing process, a uniform pseudo-random number generator is employed to rearrange each symbol. This generator guarantees the appearance of each possible permutation of all symbols is equally likely. As an example, the symbols associated with the indices 0, 1, 2, 3 and 4

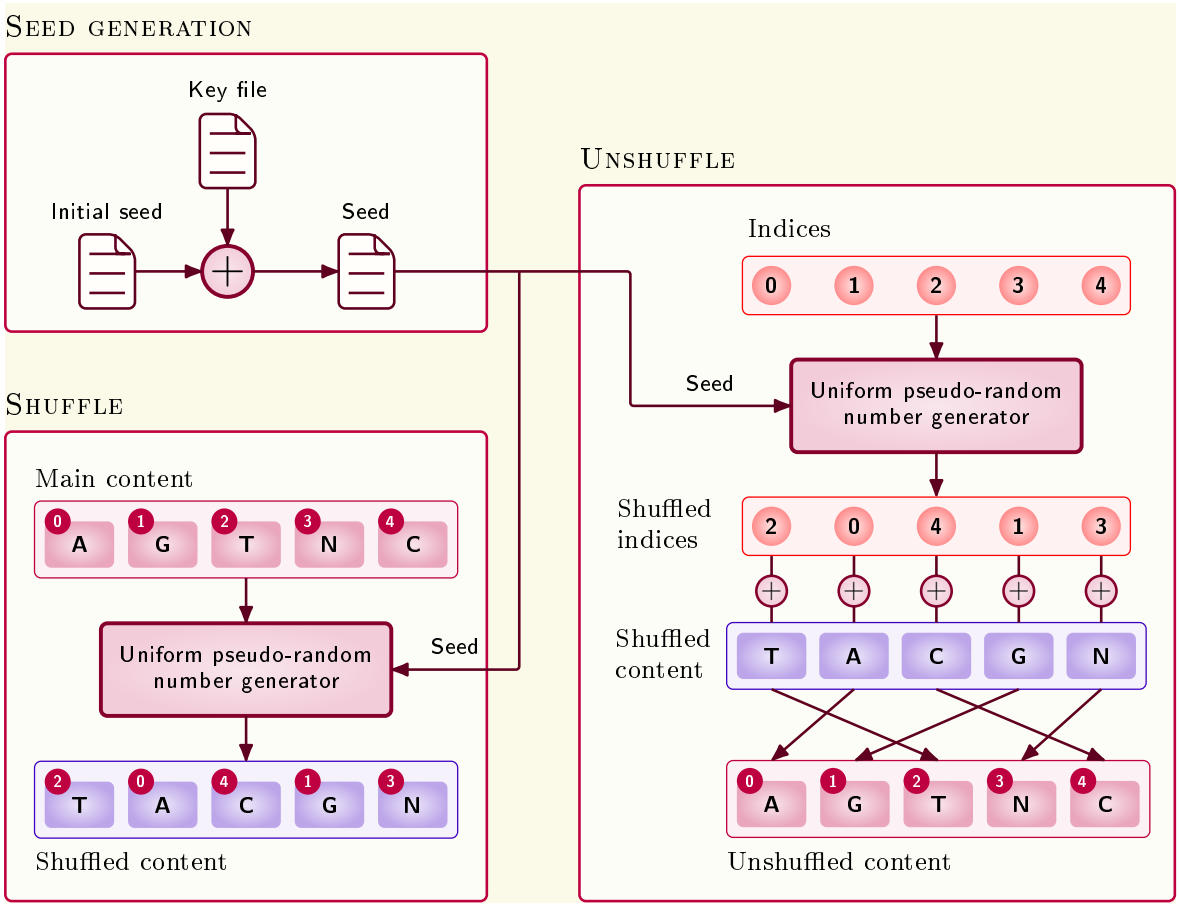

Figure 4.5: An example of shuffling and unshuffling. The uniform pseudo-random number generators in shuffle and unshuffle blocks need to employ the same seed.

are swapped with those of randomly picked indices 2, 0, 4, 1 and 3, respectively. The pseudo-random number generator is fed by a seed, which is provided by pseudo-randomizing the content of the key file. This file includes the password that is set by a user.

For unshuffling, the same uniform pseudo-random number generator and seed used in shuffling are employed. The generator shuffles a vector of indices with the values of $0, 1, 2, \ldots, n$, in which n is the number of elements minus one. Since the generator and the seed fed to it are the same as the ones employed in shuffling, the identical orders of elements will be obtained. In this way, positions that each symbol of the shuffled content will have in output unshuffled content is determined by elements of the vector of shuffled indices. As an example, the symbol T of the shuffled content will be placed at the 2^{nd} position in the unshuffled content. Similarly, the symbol A will be placed at the 0^{th} position, and so forth.

4.2.3 Encrypt and decrypt

In order for encryption and decryption, we have employed the advanced encryption standard method (AES) in Galois/counter mode of operation (GCM) [223], which is shown in Fig. 4.6. GCM is an authenticated encryption algorithm, meaning that it can provide data authenticity (integrity) and confidentiality, simultaneously. Therefore, any malicious changes to data or unauthorized access attempt will be detected. This mode of operation uses block ciphers with a block size of 128 bits. Note that GCM can exploit parallel processing by using a hardware pipeline efficiently [224].

To encrypt the shuffled content, first, two pseudo-random seeds are generated employing the password set by the user. Then, one of these seeds is used to make a key and another one is used to make the initialization vector (IV), which are required by GCM. IV, a unique binary sequence, guarantees that even if a plaintext is independently encrypted multiple times using the same key, the ciphertexts produced will be distinct [225]. As shown in Fig. 4.6, to produce a ciphertext, blocks

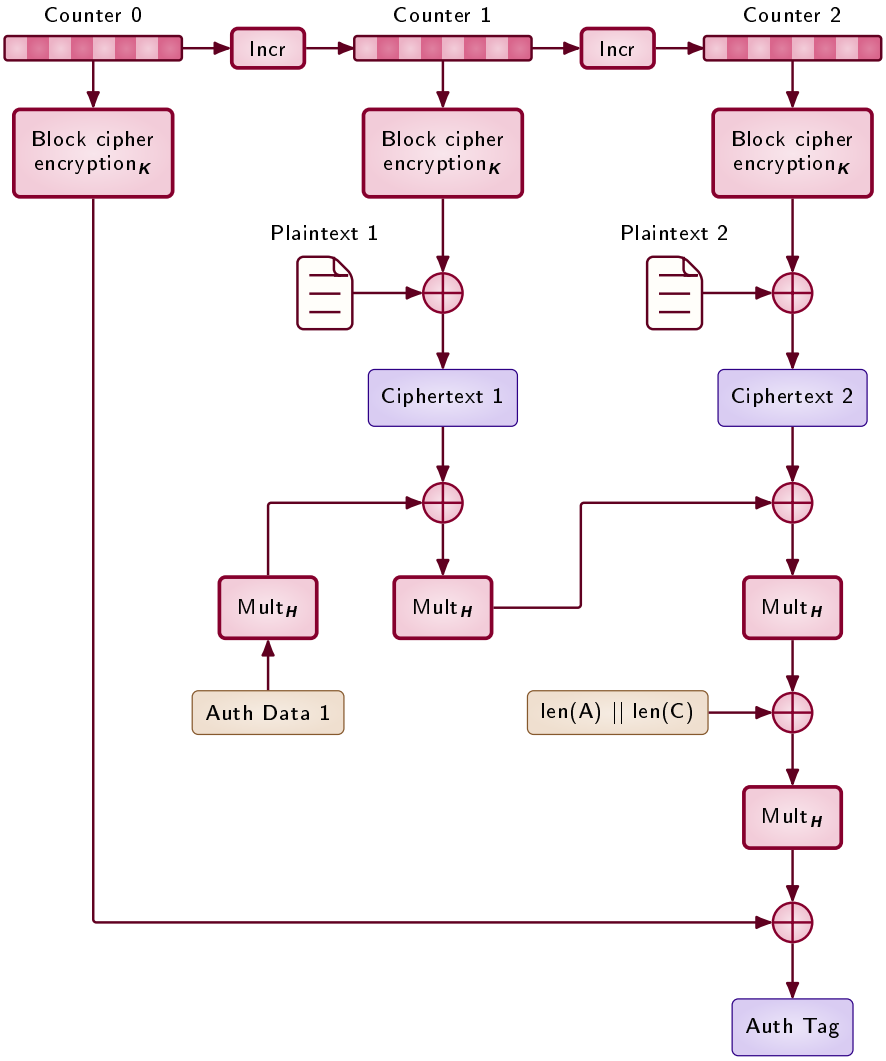

Figure 4.6: The authenticated encryption operation by AES method (GCM).

are numbered sequentially; then, these numbers are encrypted by AES, using a key K, and finally, the encryption results are XORed with the plaintext. For the purpose of verifying the data integrity, a block of additional authenticated data, "Auth Data 1", is fed to a Galois Mult block, and its output is used to generate the final authentication tag. Galois Mult performs multiplication in Galois Field, $GF\left(2^{128}\right)$, using the hash key H. Note that "Incr" denotes the function of counter increment. Ultimately, the encrypted text includes IV, ciphertext and authentication tag.

In order for decryption, the same cryptographic key as the one used in encryption side is employed, since AES is a symmetric-key algorithm. As mentioned before, this key is made using the password set by the user. GCM hashes the cryptographic key to automatically produce the authentication key. The operation of authenticated decryption is similar to the encryption operation, with the only difference that in encryption, first, encrypting the block numbers are performed, then, the hash step (Galois Mult) is taken, but in decryption, this order is reversed. To decrypt in GCM, authentication tag is computed, and then, compared to the tag associated with the ciphertext; if they match, the ciphertext is XORed with the result of encrypting block numbers, which is obtained similarly to the authenticated encryption process, to produce the plaintext.

4.3 Results and discussion

Cryfa and several other compression and encryption methods have been carried out on a collection of FASTA, FASTQ, VCF, SAM and BAM files, that will be described next. Then, the proposed tool is compared with a general-purpose encryption tool as well as several FASTA and FASTQ compressors. Next, the behavior of Cryfa, as a multi-threaded tool, is evaluated when it is running with a different number of threads. Finally, to evaluate redundancy exploration by different methods, we have tested the proposed tool and a number of FASTA compressors on thousands of genomic sequences and reported the results.

4.3.1 Experiment setup

Cryfa securely encrypts and compacts genomic data, leading to making the data resistant against low-data complexity AES attacks [217] and making redundancy of an output file almost inexplorable. Since no method has been proposed that performs similar to what Cryfa does, in the sense of simultaneous compaction & encryption, we have considered two situations in order to have a comparison between our proposed method and the other methods: first, we compare Cryfa with a general-purpose encryption tool, and second, we compress the datasets with different state-of-the-art compression methods, then, encrypt the compressed files. A description of the methods and datasets used for this purpose is provided in Section 4.3.1. In addition, the methods with which we have compared Cryfa, in the sense of redundancy exploration, and the datasets used are described

in Section 4.3.1. The machine used for the tests had a 4-core 3.40 GHz Intel® Core™ i7-6700 CPU with 32 GiB RAM. Source codes and binaries are publicly available[1] under GNU GPLv3 license.

Compression and encryption

A collection of methods for compressing FASTA files, another collection for compressing FASTQ files and a single tool for encrypting FASTA, FASTQ, VCF, SAM and BAM files are considered, that are described in the following.

Dataset

Table 4.1 provides with details of the datasets used for compression and encryption. Regarding VCF, SAM and BAM files, Cryfa can exclusively encrypt them securely. Table 4.2 shows the datasets with such formats which are used for encryption experiments. The datasets shown in Tables 4.1 and 4.2 are selected from different species/subspecies including modern human, viruses, Denisova and Neanderthal. Also, a number of synthetic data are considered that are generated by XS simulator [226].

Encryption methods

We have compared Cryfa with a general-purpose encryption software, AES Crypt[2], that uses the industry standard AES to encrypt files. This software is open-source and available on several operating systems including Linux, Windows and macOS. Note that for the purpose of comparing Cryfa and AES Crypt, we performed these two tools on all FASTA, FASTQ, VCF, SAM and BAM datasets.

FASTA compression and encryption methods

FASTA files are, first, compressed with two general-purpose compressors, gzip[3] and bzip2[4], and two special-purpose compressors, MFCompress [116] and DELIMINATE [227]; then, the output files are encrypted with AES Crypt.

FASTQ compression and encryption methods

For the purpose of compressing FASTQ files, the same general-purpose compressors considered for FASTA files are employed, that are gzip and bzip2, along with four special-purpose compressors,

[1] www.github.com/cobilab/cryfa
[2] www.aescrypt.com
[3] www.gzip.org
[4] www.bzip.org

Table 4.1: Datasets for compression and encryption experiments, in FASTA and FASTQ formats.

Category	Dataset	Size (GiB)	Description
FASTA			
Human	HS	3.1	Reference human genome, GRCh38.p7, obtained by concatenating all assembled chromosomes of modern human[1].
Viruses	viruses	0.3	All viruses genomes from NCBI[2].
Synthetic	SynFA-1	1.9	Generated by XS simulator [226]. Headers are simulated as the ones provided by Illumina sequencing technology. All base symbols, A, C, G, T and N, have identical occurrence probabilities of 0.2. Reads have static length of 100,000 bases.
Synthetic	SynFA-2	0.9	Generated by XS simulator. Headers are simulated as the ones by Ion Torrent technology. The bases A, C, G, and T have occurrence probabilities of 0.24 and N base has occurrence probability of 0.04. Reads length vary from 75,000 to 100,000 bases.
FASTQ			
Human	HS-ERR013103_1	4.6	Taken from a Finnish male in Finland[3].
Human	HS-ERR015767_2	1.3	Taken from a Puerto Rican female in Puerto Rico[4].
Human	HS-ERR031905_2	11.1	Taken from a southern Han Chinese female[5].
Human	HS-SRR442469_1	0.5	Taken from an African Caribbean female in Barbados[6].
Human	HS-SRR707196_1	8.4	Taken from a British male in England and Scotland[7].
Denisova	DS-B1087_SR	1.7	High-coverage genome sequence[8].
Denisova	DS-B1088_SR	1.2	High-coverage genome sequence[9].
Synthetic	SynFQ-1	5.6	Generated by XS simulator. Headers are simulated as an output of Ion Torrent sequencing technology. The bases A, C, G and T have occurrence probabilities of 0.23, while occurrence probability of base N is 0.08. Reads lengths vary from 70 to 150 bases.
Synthetic	SynFQ-2	0.5	Generated by XS simulator. Headers are simulated as an output of Illumina sequencing technology. There is an identical occurrence probabilities, 0.2, for all bases. Reads lengths vary from 70 to 120 bases.

[1] ftp://ftp.ncbi.nlm.nih.gov/genomes/H_sapiens/Assembled_chromosomes/seq
[2] ftp://ftp.ncbi.nlm.nih.gov/genomes/Viruses
[3] ftp://ftp.sra.ebi.ac.uk/vol1/fastq/ERR013/ERR013103/ERR013103_1.fastq.gz
[4] ftp://ftp.sra.ebi.ac.uk/vol1/fastq/ERR015/ERR015767/ERR015767_2.fastq.gz
[5] ftp://ftp.sra.ebi.ac.uk/vol1/fastq/ERR031/ERR031905/ERR031905_2.fastq.gz
[6] ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR442/SRR442469/SRR442469_1.fastq.gz
[7] ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR707/SRR707196/SRR707196_1.fastq.gz
[8] cdna.eva.mpg.de/denisova/raw_reads/B1087_SR.txt.gz
[9] cdna.eva.mpg.de/denisova/raw_reads/B1088_SR.txt.gz

fqzcomp [36], Quip [228], DSRC 2 [229] and FQC [230]. Then, the AES Crypt tool is used to encrypt the compressed files.

Explore redundancy

We have run on a set of data described in the following Cryfa, MFCompress and DELIMINATE, as FASTA file compressors, to evaluate the redundancy exploration of these methods.

Table 4.2: Datasets used exclusively for encryption experiments.

Category	Dataset	Size (GiB)	Description
VCF			
Denisova	DS-22	6.3	High-coverage genome sequence[1].
Neanderthal	N-n	0.8	High-quality genome sequence[2].
SAM			
Modern human	HS-n	0.5	Sample from a British male in England and Scotland, aligned to the human genome[3].
Neanderthal	N-y	1.5	High-quality genome sequence, aligned to the human genome[4].
BAM			
Modern human	HS-11	0.6	Sample from a British male in England and Scotland, aligned to the human genome[5].
Neanderthal	N-21	1.3	High-coverage genome sequence, aligned to the human genome[6].

[1] cdna.eva.mpg.de/denisova/VCF/human/HGDP0456.hg19_1000g.22.mod.vcf.gz
[2] cdna.eva.mpg.de/neandertal/altai/ModernHumans/vcf/SS6004472.hg19_1000g.nonchrom.mod.vcf.gz
[3] ftp://ftp.1000genomes.ebi.ac.uk/vol1/ftp/phase3/data/HG00096/alignment/HG00096.unmapped.ILLUMINA.bwa.GBR.low_coverage.20120522.bam
[4] cdna.eva.mpg.de/neandertal/altai/AltaiNeandertal/bam/AltaiNea.hg19_1000g.Y.dq.bam
[5] ftp://ftp.1000genomes.ebi.ac.uk/vol1/ftp/phase3/data/HG00096/alignment/HG00096.chrom11.ILLUMINA.bwa.GBR.low_coverage.20120522.bam
[6] cdna.eva.mpg.de/neandertal/Vindija/bam/Vi33.19.chr21.indel_realn.bam

Dataset

We downloaded the whole NCBI database for archaea, bacteria, fungi, plants and viruses, with the size of ~250 GiB. Then, we employed the GOOSE toolkit[1] to extract the sequences labeled as "complete genome", since incomplete genomes may lead to errors or misleading values. Finally, we selected the files smaller than 2 MiB. The filtered datasets are described in Table 4.3.

4.3.2 Compare with compression and encryption methods

To compare Cryfa with other methods, we first, compare it with a general-purpose encryption tool, and second, compress the datasets with different compression methods and encrypt the output with

[1] www.github.com/pratas/goose

Table 4.3: Datasets for redundancy exploration experiments.

Category	No. sequences	Tot. size (MiB)	Ave. size (KiB)	Min size (KiB)	Max size (KiB)
Archaea	104	553	1503	99	1943
Bacteria	760	10,154	1352	52	1950
Fungi	190	140	515	49	1763
Plants	2243	369	163	59	1812
Viruses	1333	231	135	49	1919
Total	4630	11,447	395	49	1950

an encryption method, then compare the results with the ones from Cryfa. For the first case, we have encrypted and decrypted the entire datasets with AES Crypt, as an encryption software. For FASTA and FASTQ files, we have compacted & encrypted and decrypted & unpacked the datasets with Cryfa. Regarding VCF, SAM and BAM datasets, we have exclusively encrypted and decrypted them by Cryfa, since it does not provide compaction for these file formats.

Table 4.4 shows compression ratios, time and memory usages by AES Crypt and Cryfa, running on FASTA and FASTQ datasets. In this table, the times taken by Cryfa to compact, encrypt, decrypt and unpack the datasets are reported separately. The total memory used by Cryfa to compact & en-

Table 4.4: Comparing Cryfa and AES Crypt, as a general-purpose encryption tool, running on FASTA and FASTQ datasets.

Dataset	Method	Comp. Time (min)	Encr. Time (min)	Total Time (min)	Total Mem (MiB)	Decr. Time (min)	Unpack Time (min)	Total Time (min)	Total Mem (MiB)	CR[1]
FASTA										
HS	AES Crypt	–	1.61	1.6	1	1.51	–	1.5	1	1.0
	Cryfa	0.27	0.06	0.3	9	0.06	0.45	0.5	25	2.9
viruses	AES Crypt	–	0.22	0.2	1	0.22	–	0.2	1	1.0
	Cryfa	0.09	0.02	0.1	9	0.02	0.09	0.1	25	2.9
SynFA-1	AES Crypt	–	1.01	1.0	1	1.03	–	1.0	1	1.0
	Cryfa	0.14	0.05	0.2	8	0.05	0.17	0.2	16	3.0
SynFA-2	AES Crypt	–	0.40	0.4	1	0.40	–	0.4	1	1.0
	Cryfa	0.09	0.02	0.1	9	0.02	0.11	0.1	12	3.0
FASTQ										
HS-ERR013103_1	AES Crypt	–	2.51	2.5	1	2.23	–	2.2	1	1.0
	Cryfa	1.47	0.34	1.8	9	0.18	0.85	1.0	23	1.9
HS-ERR015767_2	AES Crypt	–	0.72	0.7	1	0.62	–	0.6	1	1.0
	Cryfa	0.31	0.10	0.4	10	0.07	0.24	0.3	20	1.9
HS-ERR031905_2	AES Crypt	–	6.44	6.4	1	5.41	–	5.4	1	1.0
	Cryfa	1.66	0.76	2.4	11	0.46	2.36	2.8	29	1.8
HS-SRR442469_1	AES Crypt	–	0.20	0.2	1	0.22	–	0.2	1	1.0
	Cryfa	0.07	0.03	0.1	10	0.02	0.09	0.1	22	1.9
HS-SRR707196_1	AES Crypt	–	5.01	5.0	1	4.13	–	4.1	1	1.0
	Cryfa	1.24	0.54	1.8	10	0.30	1.69	2.0	30	1.8
DS-B1087_SR	AES Crypt	–	1.01	1.0	1	0.80	–	0.8	1	1.0
	Cryfa	0.33	0.07	0.4	14	0.07	0.32	0.4	31	2.3
DS-B1088_SR	AES Crypt	–	0.62	0.6	1	0.58	–	0.6	1	1.0
	Cryfa	0.22	0.06	0.3	12	0.04	0.17	0.2	31	2.3
SynFQ-1	AES Crypt	–	2.63	2.6	1	2.69	–	2.7	1	1.0
	Cryfa	0.85	0.16	1.0	6	0.15	1.16	1.3	7	2.4
SynFQ-2	AES Crypt	–	0.22	0.2	1	0.21	–	0.2	1	1.0
	Cryfa	0.08	0.02	0.1	8	0.02	0.07	0.1	14	2.1

[1] CR: Compression Ratio
Note: AES Crypt does not compact/unpack the files.

crypt the datasets is the maximum of compaction memory usage and encryption memory usage values. Also, the total decrypt & unpack memory usage is the maximum of decryption memory usage and unpacking memory usage values. Since Cryfa uses ~1 MiB to encrypt and also decrypt each file and this is less than the amount used for compaction or unpacking phases, we have only mentioned the total memory used in the table. The compression ratios are obtained by

$$\text{Compression ratio} = \frac{\text{Original file size}}{\text{Compressed file size}}. \tag{4.2}$$

For AES Crypt, this value is 1.0 for all files, since it does not compress the data.

Fig. 4.7 shows total encryption/compaction & encryption time usage and total decryption/decryption & unpacking time usage by AES Crypt and Cryfa, and also, total compressed files sizes obtained by these two methods. In Fig. 4.7a is shown the aggregate time of running AES Crypt and Cryfa on all FASTA and FASTQ files (Table 4.4). As can be seen, even though Cryfa compacts, shuffles and encrypts the files, it is 4.6 times faster than AES Crypt at compaction & encryption and is also, 3.4 times faster at decryption & unpacking of FASTA files. For FASTQ files, Cryfa is 2.3 times faster than AES Crypt at compaction & encryption and 2 times faster at decryption & unpacking. In Fig. 4.7b, the total sizes of original FASTA files (6.2 GiB) and FASTQ files (34.7 GiB) along with the total sizes of compressed files are shown. The compression ratio (CR) values are also denoted. In addition, the reduced file sizes, obtained by subtracting compressed file sizes from original file sizes, are demonstrated. By this measure, we can evaluate the amount of storage space that can be saved,

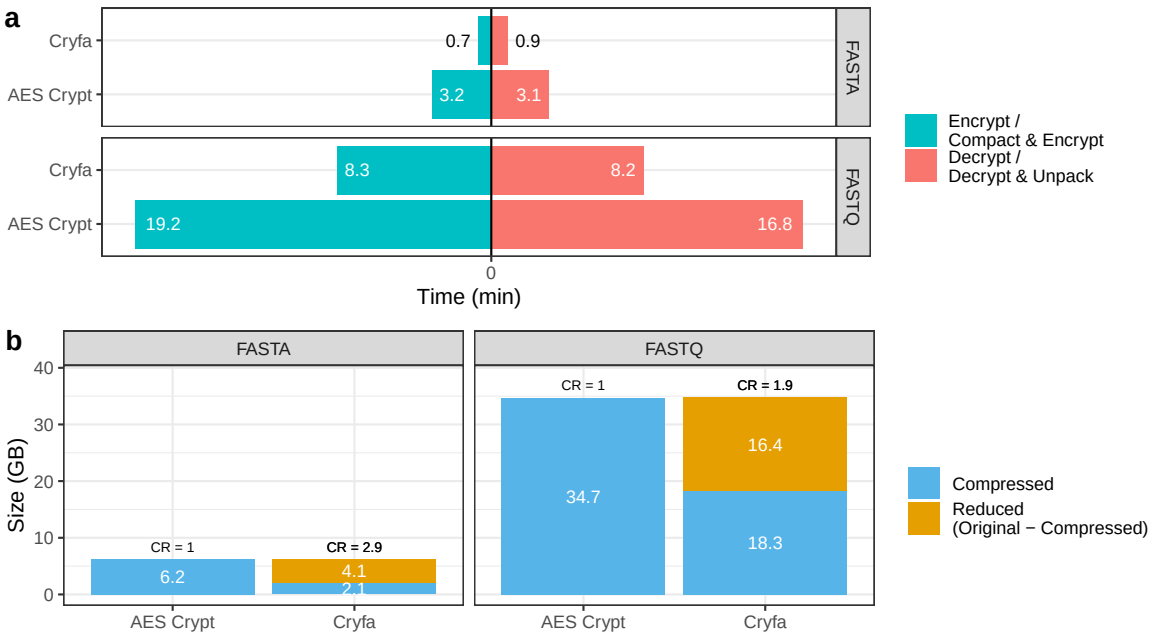

Figure 4.7: Total time and file size to encrypt / compact & encrypt and decrypt / decrypt & unpack the whole FASTA and FASTQ datasets by Cryfa and AES Crypt, a general-purpose encryption tool. **(a)** real time, **(b)** file size. CR stands for compression ratio.

using different methods. With AES Crypt, the reduced file size is zero, since it does not compress the data. However, using Cryfa, 47% to 66% of the space required to store a file can be decreased.

The time and memory used by AES Crypt and Cryfa, carrying out on VCF, SAM and BAM datasets, are reported in Table 4.5. While Cryfa shuffles plus encrypts the files, it is 1.7 to 3 times faster than AES Crypt. The RAM usage of Cryfa is 1 MiB for all of the datasets in this table. Note that when Cryfa is supposed to exclusively encrypt/decrypt a file, whether it is of FASTA/FASTQ format or VCF/SAM/BAM, it uses only 1 MiB of memory. This makes Cryfa feasible to run on any standard computer, including single-board computers.

For the case of compressing and then encrypting the datasets, we have carried out different FASTA compression methods along with AES Crypt on all FASTA files. The results are shown in Table 4.6. In the same way, the results for FASTQ files are shown in Tables 4.7 and 4.8. Regarding FASTA datasets, all methods successfully compressed the files losslessly. However, regarding FASTQ files, some methods were not able to compress a portion of the files losslessly. These cases are shown in the "Equal" column of Tables 4.7 and 4.8, with the word "No".

Fig. 4.8 shows total compression & encryption time and total decryption & decompression time usage by different methods, and also, total compressed files sizes obtained by these methods. As can be seen in Fig. 4.8a, although Cryfa performs compacting, shuffling and encrypting the FASTA files, it is 10.6 times faster than the second fastest method at compression & encryption, DELIMINATE plus AES Crypt, and also, 1.7 times faster than the second-ranked method at decryption & decompression, gzip plus AES Crypt. Also, it is 1.3 times faster than the second fastest method at compression & encryption of FASTQ files, DSRC 2 plus AES Crypt, and is also 1.2 times faster than the second-ranked method at decryption & decompression, gzip plus AES Crypt. In Figs 4.8b, the total sizes of original FASTA and FASTQ files, the total sizes of compressed files and the reduced file sizes are demonstrated. Cryfa is able to reduce the file sizes by a factor of up to 2.9, on average. Note that since there was no method performing similarly to what the proposed tool does, we had to simulate its behavior with a combination of compression and encryption methods.

Table 4.5: Cryfa compared to AES Crypt, running on VCF, SAM and BAM datasets.

Format	Dataset	Encrypt				Decrypt			
		AES Crypt		Cryfa		AES Crypt		Cryfa	
		Time (min)	Mem (MiB)	Time (min)	Mem (MiB)	Time (min)	Mem (MiB)	Time (min)	Mem (MiB)
VCF	DS-22	29	1	18	1	29	1	18	1
	N-n	03	1	01	1	04	1	02	1
SAM	HS-n	02	1	01	1	02	1	01	1
	N-y	07	1	02	1	07	1	03	1
BAM	HS-11	03	1	01	1	03	1	01	1
	N-21	06	1	02	1	06	1	03	1

Table 4.6: Compression & encryption and decryption & decompression of FASTA datasets. Note that for all methods, except Cryfa, the compressed file is encrypted with AES Crypt method.

C. Method	Compress Time (min)	Encrypt Time (min)	Total		Decrypt Time (min)	Decompr. Time (min)	Total		C. Ratio
			Time (min)	Mem (MiB)			Time (min)	Mem (MiB)	
HS (3.1 GiB)									
gzip	5.02	0.39	5.4	1	0.42	0.30	0.7	1	3.6
bzip2	4.54	0.36	4.9	8	0.36	1.79	2.2	1	3.9
MFCompress	7.58	0.28	7.9	518	0.28	5.95	6.2	521	5.1
DELIMINATE	3.88	0.28	4.2	3	0.29	2.95	3.2	2	5.1
Cryfa	0.27	0.06	0.3	9	0.06	0.45	0.5	25	2.9
viruses (0.3 GiB)									
gzip	0.80	0.05	0.9	1	0.05	0.03	0.1	1	3.3
bzip2	0.47	0.04	0.5	8	0.05	0.19	0.2	1	3.5
MFCompress	1.27	0.03	1.3	559	0.03	1.17	1.2	550	5.0
DELIMINATE	0.27	0.03	0.3	3	0.03	0.30	0.3	7	5.0
Cryfa	0.09	0.02	0.1	9	0.02	0.09	0.1	25	2.9
SynFA-1 (1.9 GiB)									
gzip	4.37	0.29	4.7	2	0.31	0.18	0.5	1	3.0
bzip2	2.65	0.27	2.9	8	0.28	1.22	1.5	1	3.2
MFCompress	3.50	0.27	3.8	554	0.25	2.69	2.9	552	3.4
DELIMINATE	1.73	0.25	2.0	3	0.27	2.57	2.8	7	3.4
Cryfa	0.14	0.05	0.2	8	0.05	0.17	0.2	16	3.0
SynFA-2 (0.9 GiB)									
gzip	1.83	0.14	2.0	1	0.14	0.09	0.2	1	3.1
bzip2	1.32	0.13	1.4	8	0.13	0.62	0.8	1	3.3
MFCompress	1.66	0.12	1.8	516	0.12	1.29	1.4	510	3.7
DELIMINATE	0.73	0.12	0.9	3	0.12	1.05	1.2	7	3.6
Cryfa	0.09	0.02	0.1	9	0.02	0.11	0.1	12	3.0

4.3.3 Run with different number of threads

Cryfa has been implemented as a multi-threaded program. In order to evaluate its cost-effectiveness when running on machines with a different number of cores, including the ones serving in the cloud, we have carried it out on two datasets, viruses (FASTA) and DS-B1088_SR (FASTQ), with one up to eight threads. The results are demonstrated in Fig. 4.9, in terms of real-time, CPU time and memory usage. To obtain real-time and CPU time, which is an aggregation of user time and system time, we have used "time" command, that is provided by default in Unix systems.

Running Cryfa with eight threads compared to using other number of threads, takes the least real-time, however, it takes the most CPU time and uses the most memory. On the contrary, running it with one thread uses the least memory. In this case, Cryfa takes the most real-time and the least CPU time. Fig. 4.9a shows that by using two threads compared to one thread, the compaction & encryption real-time decreases by a factor of ~2. Also, using eight threads compared to one thread, makes Cryfa 3 to 4 times faster in compaction & encryption and ~2 times faster in decryption & un-

Table 4.7: Compression & encryption and decryption & decompression of modern human FASTQ dataset.

C. Method	Comp. Time (min)	Encr. Time (min)	Total		Decr. Time (min)	Decompr. Time (min)	Total		C. Ratio	Equal
			Time (min)	Mem (MiB)			Time (min)	Mem (MiB)		
HS-ERR013103_1 (4.6 GiB)										
gzip	5.09	0.82	5.9	1	0.82	0.57	1.4	1	2.7	Yes
bzip2	5.70	0.64	6.3	8	0.65	2.60	3.2	1	3.4	Yes
fqzcomp	1.64	0.48	2.1	62	0.49	1.34	1.8	59	4.4	Yes
Quip	1.18	0.48	1.7	394	0.49	2.35	2.8	391	4.4	Yes
DSRC 2	0.92	0.49	1.4	3715	0.50	1.05	1.6	5286	4.2	Yes
FQC	6.93	0.48	7.4	4	0.50	5.97	6.5	4	4.3	Yes
Cryfa	1.47	0.34	1.8	9	0.18	0.85	1.0	23	1.9	Yes
HS-ERR015767_2 (1.3 GiB)										
gzip	1.54	0.16	1.7	1	0.16	0.12	0.3	1	3.8	Yes
bzip2	1.47	0.14	1.6	8	0.14	0.62	0.8	1	4.5	Yes
fqzcomp	0.42	0.10	0.5	60	0.11	0.29	0.4	58	6.1	Yes
Quip	0.35	0.10	0.5	391	0.10	0.67	0.8	390	6.0	Yes
DSRC 2	0.44	0.10	0.5	1780	0.11	0.30	0.4	2041	5.7	Yes
FQC	1.50	0.10	1.6	4	0.10	1.07	1.2	4	6.3	Yes
Cryfa	0.31	0.10	0.4	10	0.07	0.24	0.3	20	1.9	Yes
HS-ERR031905_2 (11.1 GiB)										
gzip	13.19	1.85	15.0	2	1.85	1.34	3.2	1	2.9	Yes
bzip2	13.63	1.52	15.2	8	1.54	6.19	7.7	1	3.5	Yes
fqzcomp	3.90	1.14	5.0	61	1.17	3.02	4.2	59	4.6	No
Quip	3.36	1.16	4.5	393	1.15	5.93	7.1	391	4.5	Yes
DSRC 2	2.28	1.23	3.5	2825	1.26	2.19	3.5	6374	4.3	Yes
FQC	14.03	1.35	15.4	4	1.14	13.64	14.8	4	4.8	Yes
Cryfa	1.66	0.76	2.4	11	0.46	2.36	2.8	29	1.8	Yes
HS-SRR442469_1 (0.5 GiB)										
gzip	0.63	0.07	0.7	1	0.07	0.05	0.1	1	3.0	Yes
bzip2	0.59	0.06	0.7	7	0.06	0.28	0.3	1	3.8	Yes
fqzcomp	0.20	0.04	0.2	61	0.05	0.14	0.2	59	4.9	No
Quip	0.14	0.05	0.2	385	0.05	0.24	0.3	383	4.8	Yes
DSRC 2	0.25	0.04	0.3	734	0.05	0.25	0.3	755	4.9	Yes
FQC	0.48	0.05	0.5	4	0.05	0.41	0.5	4	4.9	Yes
Cryfa	0.07	0.03	0.1	10	0.02	0.09	0.1	22	1.9	Yes
HS-SRR707196_1 (8.4 GiB)										
gzip	10.12	1.21	11.3	1	1.24	0.86	2.1	1	3.2	Yes
bzip2	11.22	0.99	12.2	8	1.01	4.87	5.9	1	4.1	Yes
fqzcomp	2.69	0.76	3.5	61	0.77	2.16	2.9	59	5.3	No
Quip	2.07	0.75	2.8	394	0.75	4.38	5.1	393	5.3	Yes
DSRC 2	1.72	0.78	2.5	2752	0.80	1.65	2.5	5763	5.1	Yes
FQC	9.77	0.66	10.4	4	0.68	8.10	8.8	4	6.0	Yes
Cryfa	1.24	0.54	1.8	10	0.30	1.69	2.0	30	1.8	Yes

packing. Fig 4.9b shows that Cryfa has a very low memory usage; even with eight threads, it uses up to 12 MiB RAM for compaction & encryption and up to 31 MiB for decryption & unpacking. Hence, the proposed tool can be feasibly carried out on present-day standard computers.

Table 4.8: Compression & encryption and decryption & decompression of Denisova and synthetic FASTQ datasets.

C. Method	Comp. Time (min)	Encr. Time (min)	Total		Decr. Time (min)	Decompr. Time (min)	Total		C. Ratio	Equal
			Time (min)	Mem (MiB)			Time (min)	Mem (MiB)		
DS-B1087_SR (1.7 GiB)										
gzip	1.27	0.17	1.4	1	0.18	0.14	0.3	1	4.2	Yes
bzip2	2.63	0.13	2.8	7	0.13	0.77	0.9	1	5.6	Yes
fqzcomp	0.43	0.10	0.5	57	0.10	0.28	0.4	53	7.6	No
Quip	0.40	0.13	0.5	391	0.13	0.58	0.7	388	5.9	No
DSRC 2	0.36	0.10	0.5	1923	0.10	0.25	0.4	2374	7.3	Yes
FQC	1.48	0.10	1.6	4	0.10	1.31	1.4	4	7.5	Yes
Cryfa	0.33	0.07	0.4	14	0.07	0.32	0.4	31	2.3	Yes
DS-B1088_SR (1.2 GiB)										
gzip	1.10	0.13	1.2	1	0.13	0.11	0.2	1	4.2	Yes
bzip2	2.01	0.10	2.1	7	0.10	0.49	0.6	1	5.6	Yes
fqzcomp	0.42	0.07	0.5	57	0.08	0.21	0.3	54	7.6	No
Quip	0.47	0.09	0.6	389	0.10	0.43	0.5	388	5.9	No
DSRC 2	0.32	0.08	0.4	1413	0.08	0.22	0.3	1736	7.2	Yes
FQC	1.16	0.07	1.2	4	0.08	0.96	1.0	4	7.4	Yes
Cryfa	0.22	0.06	0.3	12	0.04	0.17	0.2	31	2.3	Yes
SynFQ-1 (5.6 GiB)										
gzip	7.46	1.49	9.0	1	0.55	1.28	1.8	1	2.4	Yes
bzip2	7.30	0.93	8.2	8	0.87	3.54	4.4	1	3.0	Yes
fqzcomp	1.55	0.66	2.2	62	0.36	2.34	2.7	63	3.4	No
Quip	2.20	0.54	2.7	387	0.88	3.38	4.3	386	3.0	No
DSRC 2	1.12	0.62	1.7	5730	0.24	1.66	1.9	6370	3.4	Yes
FQC	5.60	0.51	6.1	4	0.98	4.97	6.0	4	3.5	Yes
Cryfa	0.85	0.16	1.0	6	0.15	1.16	1.3	7	2.4	Yes
SynFQ-2 (0.5 GiB)										
gzip	0.69	0.12	0.8	1	0.03	0.16	0.2	1	2.2	Yes
bzip2	0.63	0.09	0.7	8	0.06	0.31	0.4	1	2.8	Yes
fqzcomp	0.15	0.05	0.2	71	0.06	0.11	0.2	61	3.2	No
Quip	0.22	0.09	0.3	392	0.09	0.29	0.4	392	2.7	Yes
DSRC 2	0.21	0.08	0.3	908	0.10	0.26	0.4	904	3.2	No
FQC	0.45	0.06	0.5	4	0.11	0.59	0.7	4	3.4	Yes
Cryfa	0.08	0.02	0.1	8	0.02	0.07	0.1	14	2.1	Yes

4.3.4 Explore redundancy

In [208], it has been mentioned that the concept of Kolmogorov complexity (described in Section 2.1) can be employed by a compressor to differentiate species. On the other hand, it is important for a biological sequence encryptor not to allow a security attacker to do such a species differentiation. If there is a method which does not explore redundancy in a genomic sequence, it can resist against such attacks, since redundancy in a sequence has an inverse relation to its Kolmogorov complexity, meaning that the less a sequence is complex, the more it is redundant.

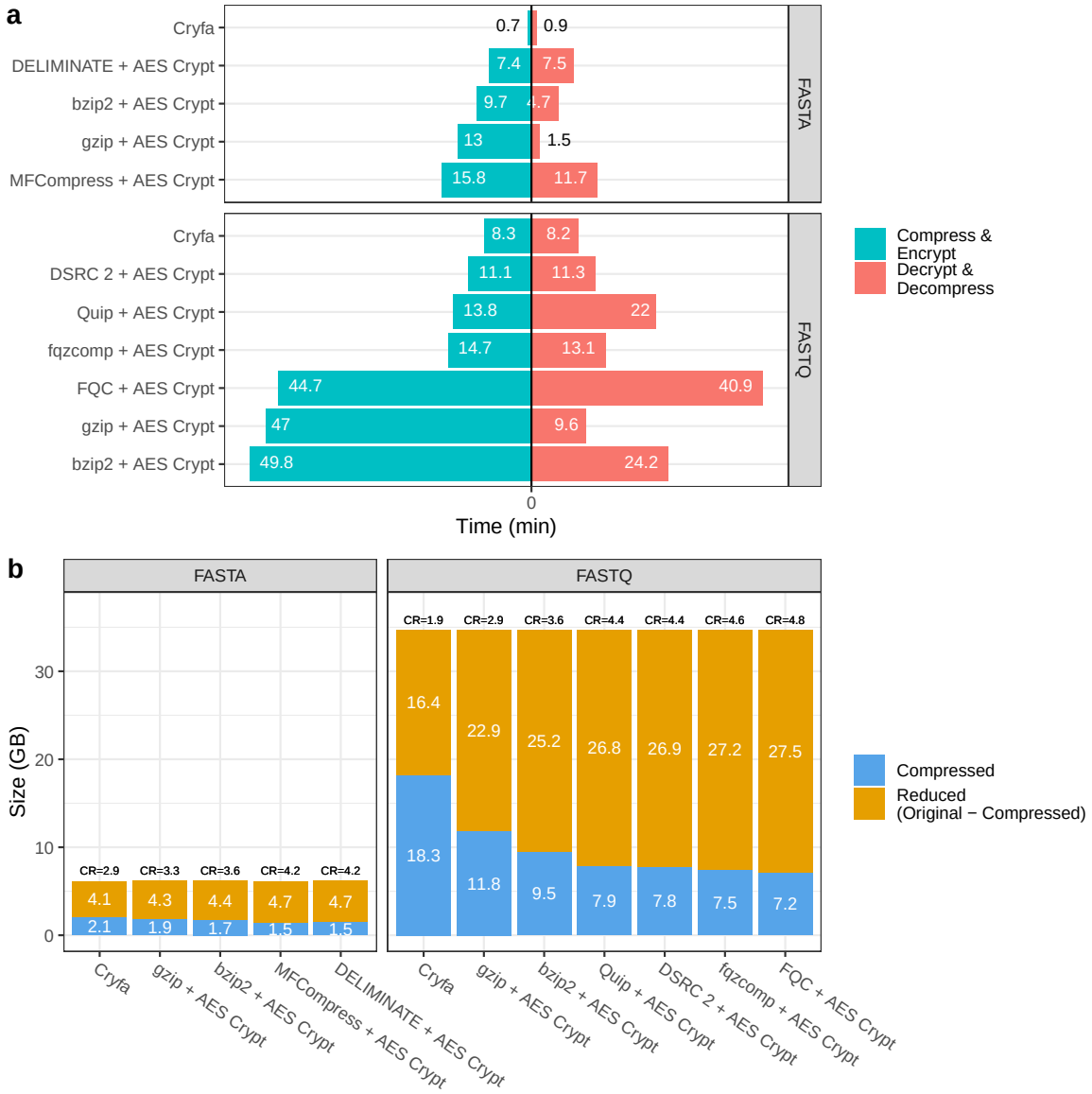

Figure 4.8: Total time and file size for compaction & encryption and decryption & unpacking of the entire FASTA and FASTQ datasets obtained by different methods. (a) real times; (b) file sizes. CR stands for compression ratio.

Cryfa securely encrypts genomic sequences by not exploring their redundancies. To evaluate this, we have compacted a collection of genomic sequences from archaea, bacteria, fungi, plants and viruses species (Table 4.3) by Cryfa, DELIMINATE and MFCompress, and computed the normalized compression (described in Section 2.1) for each sequence.

The NC results are represented in Fig. 4.10. The left pane provides a scatter plot of values and the right pane shows minimum, maximum and average (in a diamond shape) of NC values associated with each species. As can be seen in the right pane, DELIMINATE and MFCompress have broad ranges of values for different species. Looking into the average NC results obtained by these two methods, viruses, archaea, bacteria, fungi and plants can be easily differentiated, which means

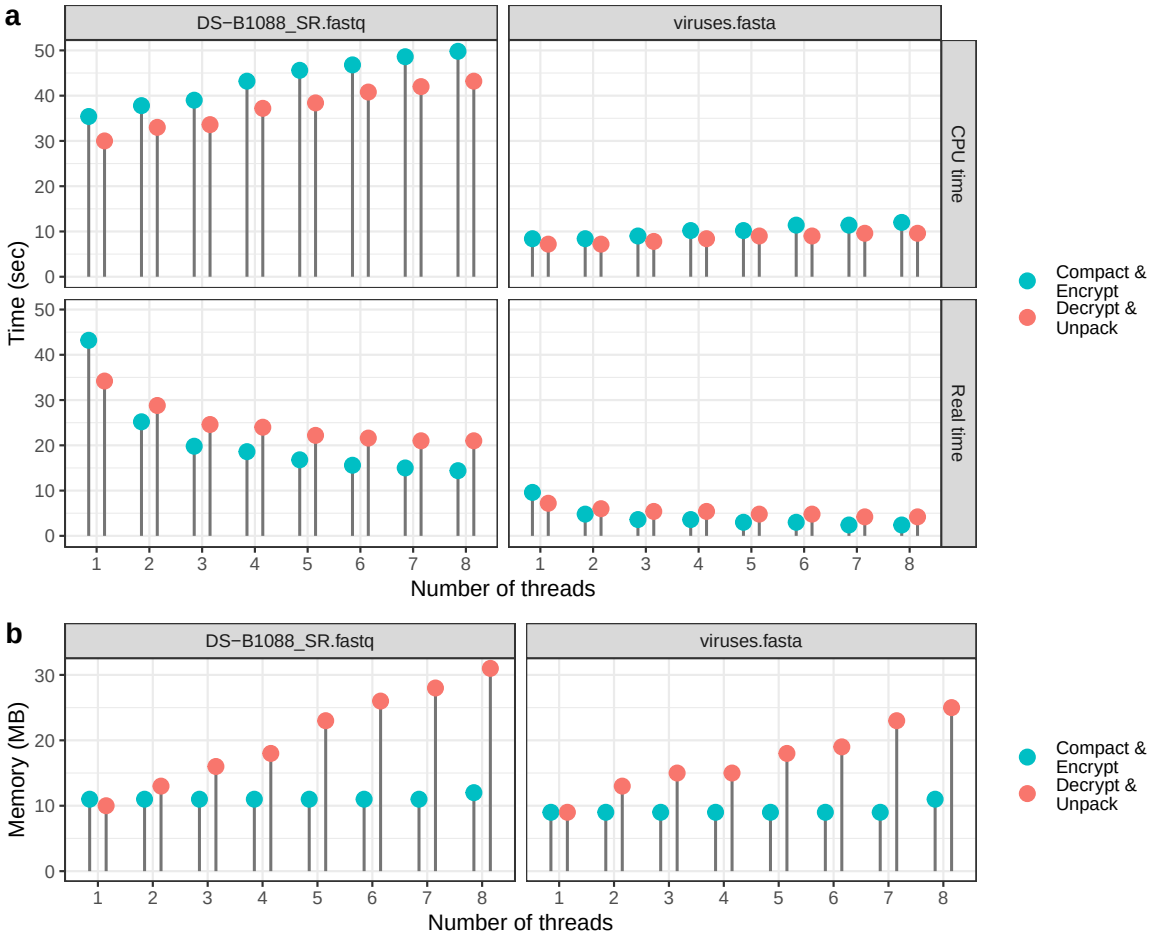

Figure 4.9: Time and memory used by Cryfa when running with different number of threads, on a 338 MiB FASTA file (viruses.fasta, from viruses species) as well as a 1.2 GiB FASTQ file (DS-B1088_SR.fastq, from Denisova subspecies). (a) real times and CPU times, that is user time plus system time; (b) memory usage.

these methods explore redundancy in genomic sequences. Consequently, they approximate the complexity values concerned with the sequences, which is a way to differentiate and authenticate species [86], [95], [208], [231]. On the other side, NC values of Cryfa is almost equal for all species. This equality proves our proposed method does not explore redundancy in genomic sequences, hence it can be employed to securely encrypt biological sequences.

4.4 Conclusions

We have developed Cryfa, an industry-oriented tool to securely encrypt genomic data and, also, compact FASTA and FASTQ files. The security of such data is substantially improved by a straightforward mechanism, shuffling. We further preserve the security of genomic data by not exploring complexity in those files. Therefore, Cryfa cannot be exploited for species differentiation. Running on several real and synthetic datasets showed that the proposed tool is faster than the fastest

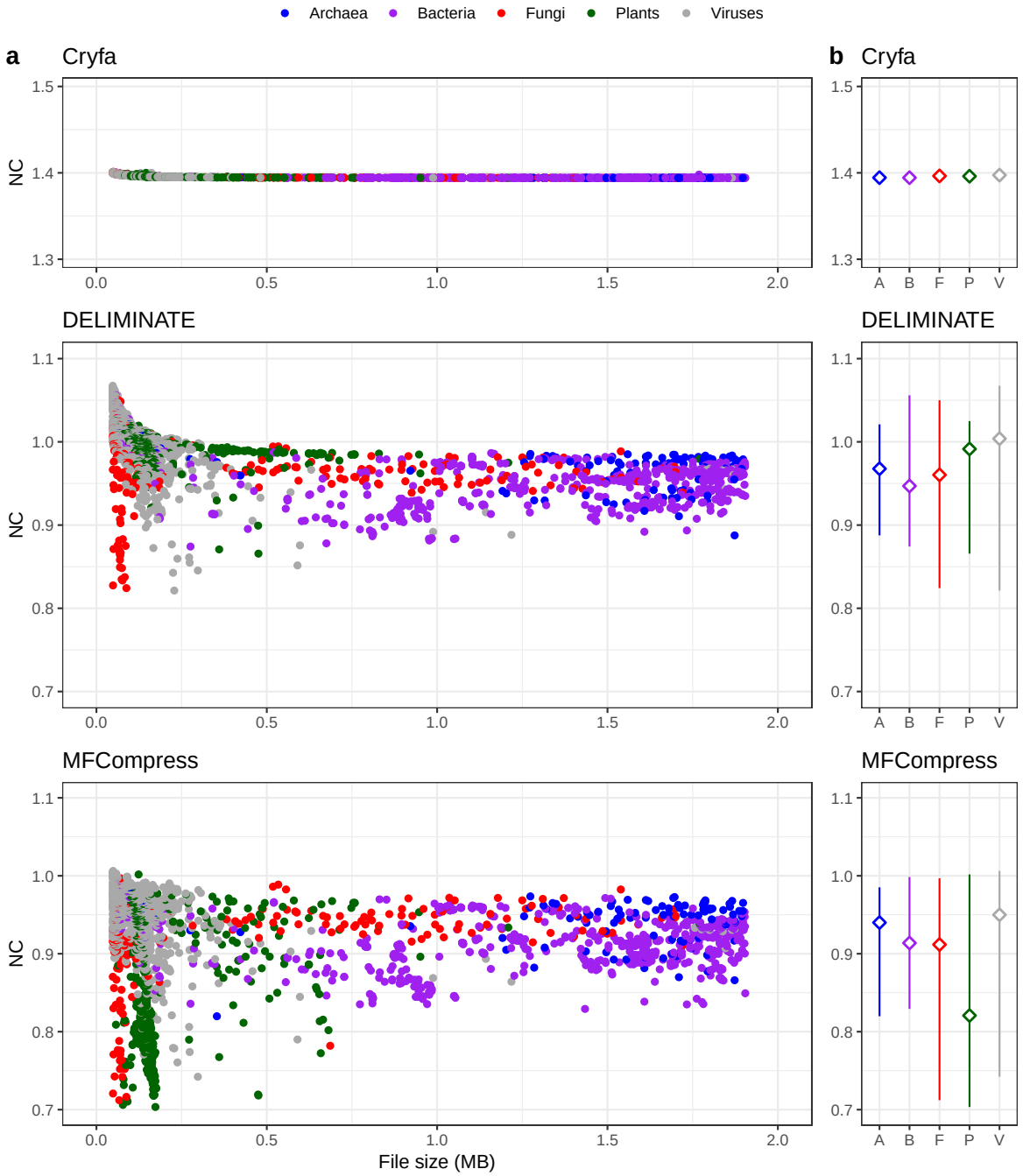

Figure 4.10: Normalized compression (NC) values obtained by running Cryfa, DELIMINATE and MFCompress on several genomic sequences.

state-of-the-art compression plus encryption tools, including general-purpose and special-purpose ones. Cryfa not only runs in high-speed and has a high level of security, as also has a very low memory usage, that is only a few megabytes.

Chapter 5

Finding and visualization of distinct regions in omics sequences

Studies on identification of species-specific genomic and proteomic regions, i.e., unique or highly dissimilar regions with respect to close species, will lead us to understanding of evolutionary traits, which can be related to novel functionalities or diseases [232]–[234]. In this chapter are proposed two alignment-free tools that are able to find and visualize distinct regions between two collections of omics data, including genomic and proteomic sequences.

In genomic level, CHESTER is presented. Applying this tool on whole genome of modern human and Neanderthal (high-quality genome sequence obtained from toe phalanx bone of a Siberian woman with 50,000 years age), shows several regions in modern human chromosomes that are absent in Neanderthal genome. These regions include documented genes that are associated with brain (neurotransmiters and synapses), hearing, blood, fertility and immune system, and undocumented regions that may express new functions linked with evolution of modern human.

In proteomic level, FRUIT is introduced. This tool is applied on multiple synthetic and real datasets to analyze its behavior when different rates of mutation occur. Testing with different k-mer sizes shows that the higher the mutation rate, the higher the relative uniqueness. FRUIT is also employed to find and visualize distinct regions in modern human proteins relatively to the proteins of Altai, Sidron and Vindija Neanderthals. The results show that four of the most distinct proteins, named ataxin-8, 60S ribosomal protein L26, NADH-ubiquinone oxidoreductase chain 3 and cytochrome c oxidase subunit 2 are involved in SCA8, DBA11, LS and MT-C1D, and MT-C4D diseases, respectively. There is also Interferon-induced transmembrane protein 3, among others, which is part of the immune system. Besides, we report the most similar primate exomes to the found modern human one, in terms of identity, query cover and length of sequences. The reported results can give us an insight into the evolution of proteomes.

5.1 Genomic level

In this section we present CHESTER, a probabilistic and alignment-free tool to map and visualize distinct regions between two collections of genomes.

5.1.1 Introduction

Up to a few years ago, it was only possible to obtain DNA sequences from present-day species. Due to the works of, mostly, Pääbo's group on methods and techniques for retrieving DNA sequences from archaeological and paleontological remains [235], [236], the first *time travel* to ancient DNA hominins became possible [237]–[239].

One of the closest hominid groups to modern humans is Neanderthal, that populated Eurasia from 350,000 ± 50,000 to 35,000 ± 5000 years ago and has extinct at current age. Availability of their sequences emerged as pieces [237]–[239], complete mitochondrial [240] and genome draft sequences [241]. The first public *complete* Neanderthal genome, acquired from a woman toe phalanx bone with ~50,000 years age that was found in Denisova Cave in Altai mountains of Siberia, was released on 2010 [57], albeit sequenced with a low coverage. This finding made it possible to analyze the Neanderthal whole genome at a computational level. The high coverage version (~30-fold) was released in 2014 [58].

We use high coverage whole Neanderthal genome (raw data) to localize and visualize distinct regions of modern human DNA, using a modern human reference assembly, GRCh37[1]. Based on recent reports which have suggested that modern humans interbred with Neanderthals when they arrived to Europe [242], and the similarity between Neanderthal and human genomes [243], [244], we can deduce that any unique region in the DNA of modern humans may be of very limited extent.

Using ancient DNA, detection of distinct regions between modern human and Neanderthal is difficult, since (a) a large volume of data is involved in the analysis (> 418 GiB); (b) it is needed to deal with raw data, because ancient DNA is not assembled and has random order; (c) there is contamination is DNA samples [245]; and (d) there is a high degree of substitutions in the DNA data, mostly caused by PCR amplifications [246] and postmortem degradation [247]. Aware of these, we propose in the next section an unsupervised probabilistic method that is able to compromise between precision and space/time resources, maintaining a reasonable precision.

5.1.2 Methods

We tackle the problem of finding the regions in target sequences which do not exist in reference sequences. For this purpose, a model is required that can search the references for all words of a

[1] www.ncbi.nlm.nih.gov/grc

certain size, k, in the targets, and is also able to report the positions of unique words. Such model is described next.

Model

For the purpose of checking the existence of all k-mers of a target in a reference sequence, it is impractical to use a binary vector which considers all the possible situations. We give an example; assume the cardinality of the alphabet representing a DNA sequence is 4 and the k-mer size is 20. This needs 4^{20} byte or 1 terabyte of memory. To tackle this problem, we use a space-efficient probabilistic data structure, named Bloom filter [248], [249]. In this data structure, false negative matches are impossible but false positives are not, i.e., it enables to test whether the k-mers of a target sequence are definitely not members of a reference, or they possibly are. A Bloom filter, with optimal number of hash functions and an appropriate size, can provide the results only slightly different than the deterministic approach.

An empty Bloom filter is a bit vector of size m. This model requires h different hash functions which map, separately, each k-mer to one of the bit vector positions; this will generate a uniform random distribution. The number of hash functions, h, which minimizes the false positive probability, p, is proportional to the number of bases in a reference sequence, n, and is obtained by

$$h = \frac{m}{n} \ln 2. \tag{5.1}$$

It is proven in [250] that considering the optimal value of h, the false positive probability is at most

$$\left(1 - e^{-\frac{h(n+0.5)}{m-1}}\right)^h. \tag{5.2}$$

For example, if $h = 6$, $n = 22{,}829{,}171$ and $m = 197{,}613{,}190$, the false positive probability will be at most 0.016 or 1.6%.

To hash k-mers, we use universal hashing. For mapping k-mers from some universe U to m bins, labeled as $[m] = \{0, 1, \ldots, m-1\}$, we need to randomly select a function from a family of hash functions. A family of functions $F = \{f : U \longrightarrow [m]\}$ is called a universal family if,

$$\forall x, y \in U,\ x \neq y : \quad \Pr_{f \in F}[f(x) = f(y)] \leq \frac{1}{m}. \tag{5.3}$$

$1/m$ is the probability of collision when a hash function maps a key to a truly random element. To obtain a family of universal hash functions, we pick a prime $p \geq m$ and define

$$f_{a,b}(x) = ((ax + b) \bmod p) \bmod m, \tag{5.4}$$

in which a and b are random integers modulo p with $a \neq 0$. The algorithm of the proposed method

is shown in Fig. 5.1.

Implementation

CHESTER is implemented in the C language and is publicly available[1] under GNU GPLv3 license. Similar to FRUIT, CHESTER contains three programs of CHESTER-map, CHESTER-filter and CHESTER-visual, which map relatively singular (distinct) regions between collections of references and targets, filter the regions and save the positions, and visualize positions of the relatively singular regions, respectively. The presented tool accepts FASTA, FASTQ and SEQ (including solely DNA bases, e.g., A, C, G, T, N) formats for references, and FASTA and SEQ for targets.

5.1.3 Results and discussion

We have applied the proposed method on modern human and Neanderthal whole genomes, running it on a 2.13 GHz Intel® Xeon® CPU. For the experiment, we have used a Bloom filter with the size of

[1] www.github.com/cobilab/chester

```
1: Initialize size of Bloom filter (BF), m, and k-mer size, k

2: for each r in reference sequences do
3:     calculate h, using (5.1)                ▷ h: optimal number of hash functions
4:     calculate p, using (5.2)                ▷ p: false positive probability
5:     for each k-mer string, s, in r do
6:         for each h_i in hash function family (HF) do
7:             hash s to position o_{s,i} in BF, using (5.4)
8:             update BF on position o_{s,i}
9:         end for
10:    end for
11:    for each t in target sequences do
12:        for each k-mer string, s, in t do
13:            for each h_i in HF do           ▷ HF: hash function family
14:                hash s to position o_{s,i} in BF
15:                query BF for position o_{s,i}, save Boolean result to file u_{r_i,t_j}
16:            end for
17:        end for
18:    end for
19: end for

20: for each t in target sequences do
21:    bit-wise OR on all Boolean elements of files u_{r_i,t}, save result to file u_t
22:    filter u_t by a window of size w, save relatively unique positions in file o_t
23: end for

24: Illustrate in an SVG image the positions saved in o_t files
```

Figure 5.1: Algorithm of the proposed method for finding and visualizing distinct regions between two collections of omics data.

64 GiB and false positive probability of 0.008205, and also a k-mer size of 30. The script to reproduce the results is available on the CHESTER's Github repository under "ancient" directory and is named "runNeanderthalGRC37.sh".

After ~51 hours of running (without parallelization), the full map of regions in modern human chromosomes that do not exist in Neanderthal genomes is produced (Fig. 5.2). Comparison to the supplementary information 19b of [58] shows that we have found more distinct regions, using a filter with the threshold of 0.95. Note that Prüfer *et al.* focused on the regions that had 20-fold excess over randomized rank assignments, but we used the whole regions.

We found 125, 170 and 2169 regions for thresholds of 0.85, 0.90 and 0.95, respectively. Table. 5.1

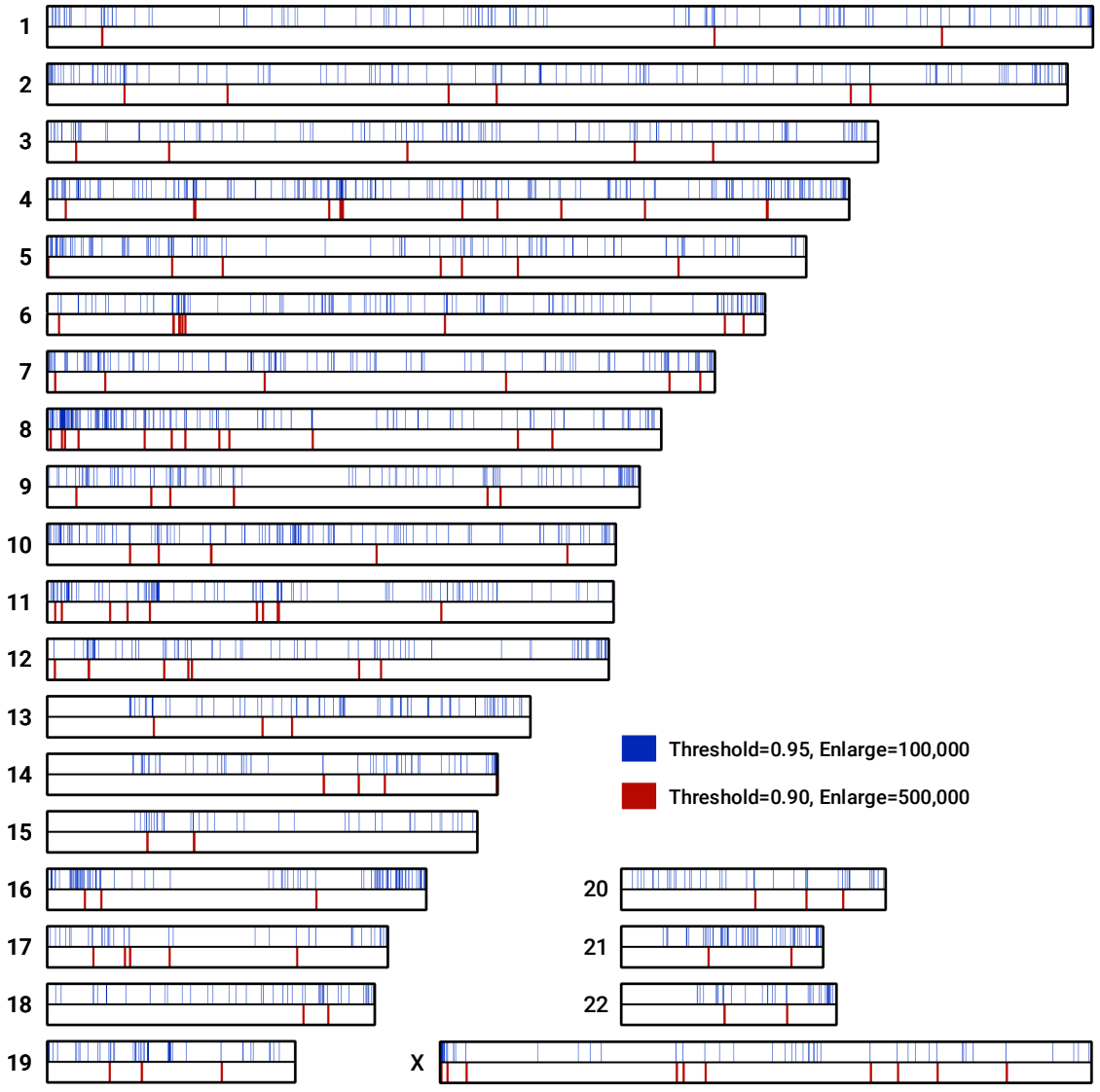

Figure 5.2: Regions in modern human chromosomes that do not exist in Neanderthal genomes. "Enlarge" shows the number of times that a region is enlarged, for the visualization purpose.

Table 5.1: Genes present in at least half of the regions that exist in modern human chromosomes and do not exist in Neanderthal, considering the threshold of 0.90.

Gene ID	Full name	Chr.	Gene type
149643	spermatogenesis associated 45	1	protein coding
57504	metastasis associated 1 family member 3	2	protein coding
54842	major facilitator superfamily domain containing 6	2	protein coding
253559	cell adhesion molecule 2	3	protein coding
64084	calsyntenin 2	3	protein coding
100287290	*LOC100287290*	3	protein coding
285555	sperm tail PG-rich repeat containing 2	4	protein coding
93627	TBC1 domain containing kinase	4	protein coding
84109	pyroglutamylated RFamide peptide receptor	4	protein coding
285600	*KIAA0825*	5	protein coding
721	complement C4B (Chido blood group)	6	protein coding
26047	contactin associated protein-like 2	7	protein coding
64478	CUB and Sushi multiple domains 1	8	protein coding
340441	POTE ankyrin domain family member A	8	protein coding
158038	leucine rich repeat and Ig domain containing 2	9	protein coding
340895	MAM and LDL receptor class A domain containing 1	10	protein coding
53904	myosin IIIA	10	protein coding
83938	chromosome 10 open reading frame 11	10	protein coding
283303	MRGPRG antisense RNA 1	11	ncRNA
5140	phosphodiesterase 3B	11	protein coding
23085	ELKS/RAB6-interacting/CAST family member 1	12	protein coding
408186	OVOS	12	protein coding
341346	single-pass membrane protein with coiled-coil domains 2	12	protein coding
6857	synaptotagmin 1	12	protein coding
399671	HEAT repeat containing 4	14	protein coding
9369	neurexin 3	14	protein coding
3492	immunoglobulin heavy locus	14	protein coding
79091	methyltransferase like 22	16	protein coding
92017	sorting nexin 29	16	protein coding
645027	envoplakin like	17	protein coding
54828	BCAS3, microtubule associated cell migration factor	17	protein coding
79839	coiled-coil domain containing 102B	18	protein coding
596	BCL2, apoptosis regulator	18	protein coding
9524	trans-2,3-enoyl-CoA reductase	19	protein coding
8537	breast carcinoma amplified sequence 1	20	protein coding
80161	ASMTL antisense RNA 1	X	ncRNA
57502	neuroligin 4, X-linked	X	protein coding
2182	acyl-CoA synthetase long-chain family member 4	X	protein coding
644717	sarcoma antigen 2 and pseudogene	X	pseudo

provides list of genes present in at least half of the regions when the threshold is 0.90. Majority of the detected regions are protein coding. From these regions, we highlight the following genes: (a) spermatogenesis associated 45, that has an important role in reproductive efficacy and success [251]; (b) myosin IIIA, by which the protein encoded plays an important role in hearing in humans. Three different recessive, loss of function mutations in the encoded protein have been shown to cause nonsyndromic progressive hearing loss [252]; (c) ELKS/RAB6-interacting/CAST family member 1, by which the protein encoded is a member of a family of RIM-binding proteins. RIMs are active

zone proteins that regulate neurotransmitter release. Changes in the gene have been associated with autism [253]; and (d) synaptotagmin 1, which encodes a protein that participates in triggering neurotransmitter release at the synapses [254]. It is worth mentioning that amplification of the sequencing process might create several mutations, namely C $\longrightarrow$ T and G $\longrightarrow$ A [255]. Therefore, there is a potential for future work to study if the differences between these regions may or may not be given by these characteristics, and if yes, to assess its impact.

5.2 Proteomic level

In this section we propose FRUIT, an alignment-free and probabilistic tool that is able to find and visualize distinct regions between two collections of protein sequences.

5.2.1 Introduction

Palaeoproteomics is an emerging field that focuses on the study of ancient proteomes and intersects evolutionary biology, archaeology and anthropology. It has the potential to provide researchers with the information about new or existing phylogenetic trees, species identification and past migrations [256]–[260].

Proteins can last longer than DNA, since they have more stable bonds for connecting them, are deposited in greater volumes and have more degradation-proof molecular structures. This makes them appropriate for recovering information from much longer periods back in time [261]. Even with the best preservation conditions, the oldest DNA samples date back to 0.4–1.5 million years ago, while the oldest proteins are hundreds of millions years old [262], [263].

The sequences of Neanderthals, as one of the closest hominins to modern humans, have been provided in the literature as complete genome [57], [58] and complete exome [264]. We use the complete exomes of a ~50,000 year old Neanderthal from Denisova Cave in the Altai Mountains in Siberia, a ~49,000 year old one from El Sidron Cave in Spain and a ~44,000 year old one from Vindija Cave in Croatia [264]. In the following sections, we use Altai, Sidron and Vindija Neanderthals proteins to find and visualize distinct regions of the reference proteome of modern human.

5.2.2 Methods

The model that we use to find regions in target sequences that are absent in reference sequences is described in Section 5.1.2, with a difference that to obtain a family of universal hash functions we use the multiply-add-shift scheme [265]:

$$f_{a,b}(x) = ((ax + b) \bmod 2^w) \operatorname{div} 2^{w-M}, \tag{5.5}$$

where w is the number of bits in a machine word, e.g., 64, M is $\log_2 m$, assuming the number of bins, i.e., m, is a power of two, a is a random positive integer less than 2^w and b is a random non-negative integer less than 2^{w-M}. Such family of functions can be implemented in the C++ language by

$$h_{a,b}(x) = \texttt{(uint64\_t) (a*x+b) >> (w-M)}.$$

Note that there are differences in implementing such model for genomic and proteomic sequences, namely (a) in genomes we face with sequences with cardinality (number of bases) of usually 4, while in proteomes the cardinality is usually 20; and (b) sizes of genomic sequences are in general larger; therefore, we have to use larger Bloom filters to keep the same false positive probability (see Equations 5.1 and 5.2).

Implementation

FRUIT has been implemented in the C++ language and the executables are publicly available[1] under GNU GPLv3 license. The implemented tool contains three programs: (a) fruit-map, to map relatively unique regions; (b) fruit-filter, to filter the regions and save the positions; and (c) fuit-visual, to visualize positions of the relatively unique regions. The three file formats of FASTA, FASTQ and SEQ (including solely amino acid letter codes, e.g., M, A, R, D) can be fed to this tool.

5.2.3 Results and discussion

To analyze the behavior of FRUIT for different rates of mutations, we applied it to real and synthetic datasets, shown in Table 5.2, and measured uniqueness ratios. The datasets are available at the Github repository associated with the proposed tool. We tested the proposed tool on a machine which had an 4-core 3.40 GHz Intel® Core™ i7-6700 CPU with 32 GiB RAM.

The uniqueness ratio falls within the range $[0.0, 1.0]$, and is obtained by size of the relatively unique region divided by size of the sequence. It shows the portion of a target file which does not exist in the reference, with respect to the k-mer size. Fig. 5.3 demonstrates uniqueness ratios versus mutation rates, for k-mer sizes of 5 to 10, for a synthetic dataset plus three samples of Altai, Sidron and Vindija Neanderthals. Note that when $k = 1, 2, 3, 4$, the uniqueness ratio is 0 for all mutation rates, i.e., all words of size up to 4 in the targets are found in the references. Fig. 5.3 shows that the higher the mutation rate, the higher the uniqueness ratio, and also, the greater the k, the greater is the uniqueness ratio. As an example, with 50% of mutation, given a uniform distribution, we expect a target to be highly dissimilar to the reference. This can be seen for all datasets, when $k \geq 7$.

For the next experiment, we picked as targets all the 20,412 proteins of modern human[2], and cal-

[1] www.github.com/cobilab/fruit
[2] www.uniprot.org/uniprot

Table 5.2: Datasets used by FRUIT, including synthetic and real data from Neanderthals and modern human.

	Dataset	Species	No. amino acids	No. reads	Cardin.[1]
Ref	Synthetic	–	5,000,000	–	20
Tar	Mutated[2]	–	5,000,000	–	20
Ref	Altai	*H. neanderthalensis*	22,829,171	42,394	21
Tar	Mutated[2]	–	22,829,171	42,394	21
Ref	Sidron	*H. neanderthalensis*	22,829,205	42,394	21
Tar	Mutated[2]	–	22,829,205	42,394	21
Ref	Vindija	*H. neanderthalensis*	22,829,173	42,394	21
Tar	Mutated[2]	–	22,829,173	42,394	21
Ref	Altai	*H. neanderthalensis*	22,829,171	42,394	21
	Sidron	*H. neanderthalensis*	22,829,205	42,394	21
	Vindija	*H. neanderthalensis*	22,829,173	42,394	21
Tar	Modern human[3]	*H. sapiens*	11,374,527	20,412	21

[1] Cardinality shows number of different amino acids in a protein sequence.

[2] To make this data, we duplicated the reference file 50 times, and mutated with "mutate" tool (`www.github.com/cobilab/fruit`) the first file by 1%, the second file by 2%, up to the 50th file by 50%.

[3] Reviewed reference proteome–manually annotated. The modern human multi-FASTA file is divided into 20,412 FASTA files. Each of which is considered as a target, and the three samples of Altai, Sidron and Vindija Neanderthals are considered as references altogether.

culated their unique regions relatively to the Altai, Sidron and Vindija Neanderthals[1], as references. For this purpose, we first used fruit-map to map amino acids of the targets to the files showing

[1] `cdna.eva.mpg.de/neandertal/exomes/proteins`

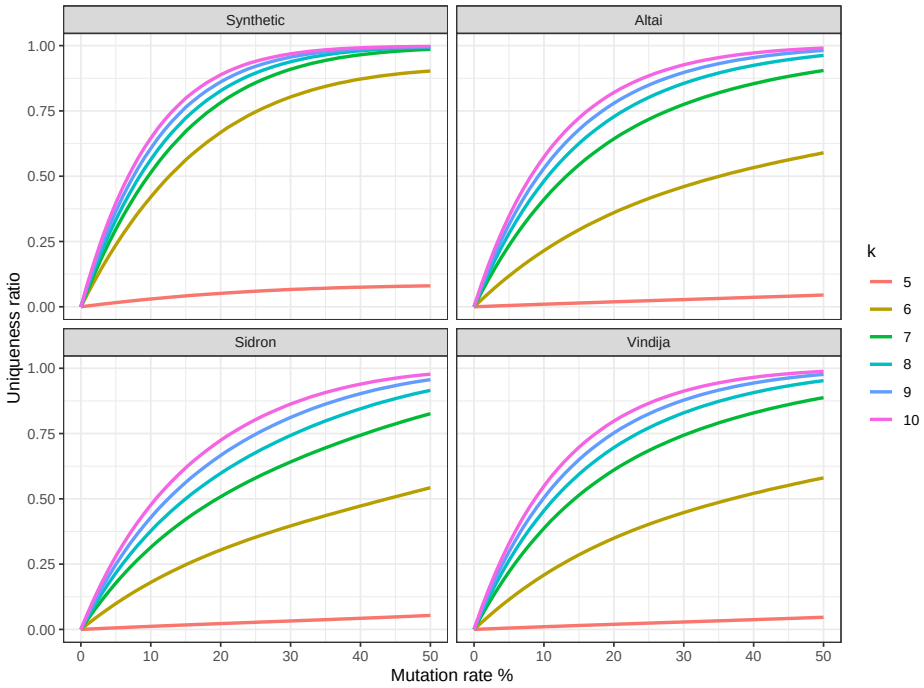

Figure 5.3: Uniqueness ratios for different rates of mutation and different k-mer sizes applied to synthetic and real (Neanderthals) datasets.

their existence in the references. In this phase, we considered as data structure a highly accurate Bloom filter with the false positive probability of 0.00001. Then, we used fruit-filter to filter the results of fruit-map and find the positions of relatively unique regions. Finally, we used fruit-visual to visualize the relative positions found in the previous step.

Fig. 5.4a shows distributions of uniqueness ratios for modern human sequences relatively to Neanderthals sequences, for different k-mer sizes. As can be seen, the shapes of distributions changes for $k = 5, 6, 7$, but thereafter, it changes only slightly with k. Fig. 5.4b demonstrates the total uniqueness ratios for different k values. For $k = 7$, the sign of the second derivative changes from positive to negative, meaning that it is the lowest upper-bound that can be chosen for our purpose. Fig. 5.4c shows the probability of a target word being seen in the reference, considering different k-mer sizes. As shown, its value is ~0 for $k \geq 4$. The probabilities of the k-mers are considered to be $1/21^k$, in which 21 is the maximum cardinality of alphabet representing the target and references.

The top ten unique modern human proteins relatively to the Neanderthals proteins are described in Table 5.3, in detail, and illustrated in Fig. 5.5, using fruit-visual. Each color in the figure represents a continuous relatively unique region. As an example, 2 out of 145 amino acids in 60S ribosomal

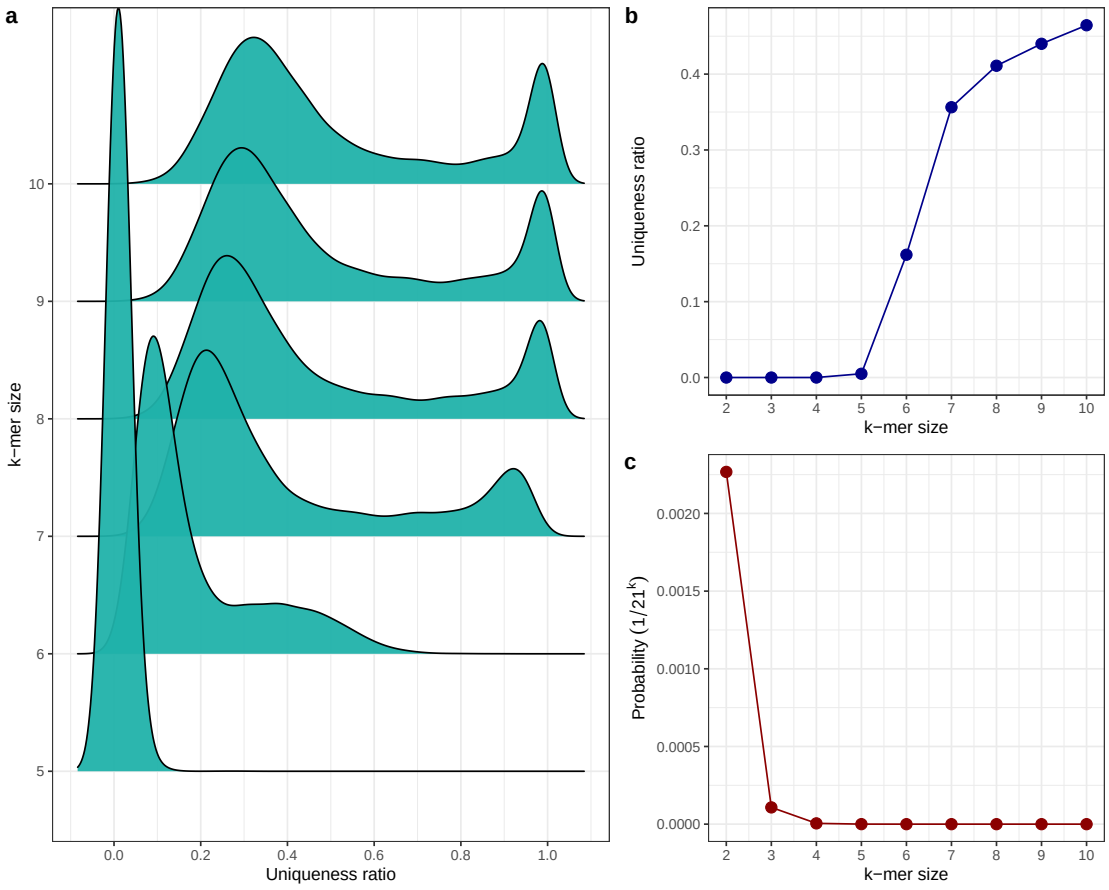

Figure 5.4: (a) Distribution of uniqueness ratios; (b) total uniqueness ratios; (c) probability of a target word being seen in the reference, for different k-mers.

Table 5.3: The most unique proteins of modern human absent in the Neanderthals.

Accession[1]	Modern human protein	Len. (b)	U. ratio	Description
Q3SY05	putative uncharacterized protein encoded by *LINC00303*	128	0 99	product of a dubious CDS prediction. Level of evidence is "uncertain"; therefore, it is either (a) derived from the erroneous translation of a pseudogene or non-coding RNA, that should be removed from protein database, in case the evidence of pseudogenization is overwhelming for instance, or (b) it should be upgraded to the certain level, which has happened to e.g., E.coli pseudogene ymiA that has now been found to produce a protein product.
Q5QFB9	protein PAPPAS	102	0 99	product of a dubious CDS prediction.
Q156A1	ataxin-8	80	0 99	involved in spinocerebellar ataxia 8 (SCA8) disease, that is caused by mutations affecting the *ATXN8* gene. It is unknown whether this protein exists in non-SCA8 individuals.
Q5VT33	putative uncharacterized protein encoded by *LINC01545*	79	0 99	protein predicted.
P61254	60S ribosomal protein L26	145	0 99	component of the large ribosomal subunit. It is involved in Diamond-Blackfan anemia 11 (DBA11) disease, that is caused by mutations affecting the *RPL26* gene.
A8MTZ7	uncharacterized protein C12orf71	265	0 99	protein predicted.
Q01628	interferon-induced transmembrane protein 3	133	0 99	IFN-induced antiviral protein which disrupts intracellular cholesterol homeostasis.
H3BRN8	uncharacterized protein C15orf65	121	0 98	experimental evidence at transcript level.
P03897	NADH-ubiquinone oxidoreductase chain 3	115	0 98	core subunit of the mitochondrial membrane respiratory chain NADH dehydrogenase (Complex I) that is believed to belong to the minimal assembly required for catalysis. It is involved in Leigh syndrome (LS) and mitochondrial complex I deficiency (MT-C1D) diseases.
P00403	cytochrome c oxidase subunit 2	227	0 98	Cytochrome c oxidase is the component of the respiratory chain that catalyzes the reduction of oxygen to water. Subunits 1-3 form the functional core of the enzyme complex. Subunit 2 transfers the electrons from cytochrome c via its binuclear copper A center to the bimetallic center of the catalytic subunit 1. It is involved in mitochondrial complex IV deficiency (MT-C4D) disease.

[1] A unique identifier of an entry in the UniProtKB database (www.uniprot.org/uniprot).

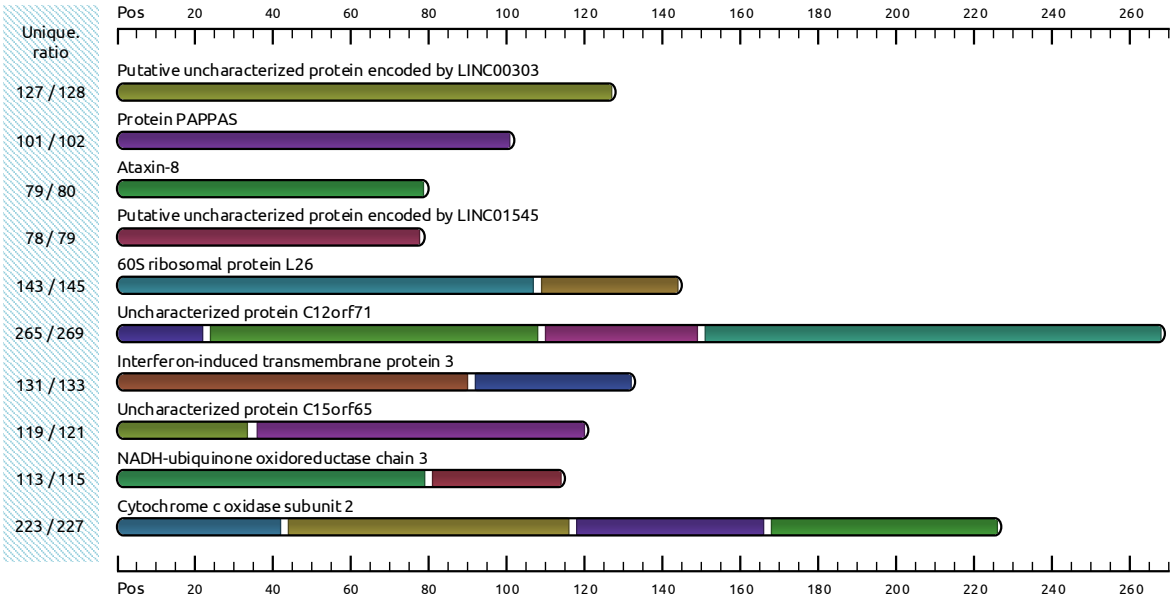

Figure 5.5: Modern human proteins with the most distinct regions against Altai, Sidron and Vindija Neanderthals. The format m / n shows that considering k-mer size of 7, m out of n amino acids are relatively unique.

protein L26 are not relatively unique, leading to the separation of the protein into two regions, each one represented by a distinct color.

Table 5.4 describes the most similar exomes of non-human primates to the ones listed in Table 5.3. This table shows that the found modern human exomes exist, fully or partially, in the listed primates, but not in the Neanderthals. This can have multiple reasons, such as ambiguity of computational models in prediction of proteins, inclusion of contaminant exogenous sources [266] and ancient DNA damage [267].

5.3 Conclusions

We introduced two unsupervised alignment-free tools, CHESTER and FRUIT, that are able to find and visualize relatively unique (distinct) regions between two collections of omics data, including genomic and proteomic sequences. In genomic level, we tested CHESTER on modern human chromosomes and Neanderthal high-quality genome (> 418 GiB of raw data). The results showed several regions that are associated with brain, namely neurotransmiters and synapses, hearing, blood, fertility and immune system, among others. These regions may now be studied according to their expression and meaning in the evolution path. Other regions have also been detected that, although undocumented, may reveal unique functionalities in Neanderthal or modern human. In proteomic level, we tested FRUIT on synthetic and real (Neanderthals) datasets to analyze the impact of different rates of mutations on uniqueness ratios. We also employed FRUIT to map, filter and visualize unique proteins in modern humans relatively to Altai, Sidron and Vindija Neanderthals. The top ten

Table 5.4: The most similar non-human primate exomes to modern human, that are listed in Table 5.3.

Modern human protein	Similar primate protein	Accession[1]	Species	Len. (b)	Iden. %[2]	Q. cov. %[3]
Putative uncharacterized protein encoded by *LINC00303*	predicted putative uncharacterized protein encoded by *LINC00303*	XP_004088138.1	*N. leucogenys*	128	92.19	100
Protein PAPPAS	predicted protein PAPPAS	XP_015292165.1	*M. fascicularis*	105	94.06	99
Ataxin-8	_[4]					
Putative uncharacterized protein encoded by *LINC01545*	putative uncharacterized protein encoded by *LINC01545*	XP_008960198.1	*P. paniscus*	79	100.00	100
60S ribosomal protein L26	60S ribosomal protein L26 isoform X1	XP_008059327.2	*C. syrichta*	149	100.00	100
Uncharacterized protein C12orf71	uncharacterized protein C12orf71 homolog	XP_520810.1	*P. troglodytes*	269	99.26	100
Interferon-induced transmembrane protein 3	predicted interferon-induced transmembrane protein 3 isoform X2	XP_004050385.1	*G. gorilla gorilla*	133	100.00	100
Uncharacterized protein C15orf65	uncharacterized protein C15orf65 homolog isoform X2	XP_003314728.1	*P. troglodytes*	121	99.17	100
NADH-ubiquinone oxidoreductase chain 3	NADH dehydrogenase subunit 3	ABU47841.1	*P. troglodytes*	115	95.65	100
Cytochrome c oxidase subunit 2	cytochrome oxidase subunit II (mitochondrion)	ACJ63818.1	*G. gorilla gorilla*	227	99.56	100

[1] A unique identifier of an entry in the NCBI database (www.ncbi.nlm.nih.gov).

[2] Identity describes the percentage of identical characters in proteins.

[3] Query cover describes how much of the primate protein is covered by the modern human protein.

[4] Using QuickBLASTP [268], no similar protein was found. Using DELTA-BLAST [269], which yields better homology detection, we found ataxin-8, partial protein from *Varroa destructor* species with the length of 89, 100.00% identity and 98% query cover, but it does not belong to a primate.

distinct proteins were reported in this chapter, of which some are associated with diseases, including SCA8, DBA11, LS, MT-C1D and MT-C4D, and some others are putative uncharacterized proteins, that are products of dubious CDS prediction. Furthermore, we have listed and described the most similar primate exomes to the found modern human ones.

Chapter 6

Detection and visualization of genomic rearrangements

The development of high-throughput sequencing technologies and, as its result, the production of huge volumes of genomic data, has accelerated biological and medical research and discovery. Study on genomic rearrangements is crucial due to their role in chromosomal evolution, genetic disorders and cancer. In this chapter is presented Smash++, an alignment-free and memory-efficient tool to find and visualize small- and large-scale genomic rearrangements between two DNA sequences. This computational solution extracts information contents of the two sequences, exploiting a data compression technique, in order to find rearrangements. We also present Smash++ visualizer, a tool that allows the visualization of the detected rearrangements along with their self- and relative complexity, by generating an SVG (Scalable Vector Graphics) image. Tested on several synthetic and real DNA sequences from bacteria, fungi, Aves and mammalia, the proposed tool was able to accurately find genomic rearrangements. The detected regions complied with previous studies which took alignment-based approaches or performed FISH analysis. The maximum peak memory usage among all experiments was ~1 GiB, which makes Smash++ feasible to run on present-day standard computers.

6.1 Introduction

With the ever-increasing development of HTS technologies, a massive amount of genomic information is produced at much higher speed and lower cost than was possible before [270]. Analysis of such information has led to the advancement of our understanding of biology and disease, over the past decade [271], [272]. Computational solutions play a key role in dry-lab analysis of the deluge of HTS data by using efficient and fast algorithms.

Genome rearrangements are mutations that alter the arrangement of genes on a genome, and usually occur in the presence of errors in cell division following meiosis or mitosis. These structural abnormalities in chromosomes include, but are not limited to, insertions, deletions, duplications, translocations, inversions, fissions ans fusions, mostly occur as an accident in the sperm or egg cell and hence are present in every cell of the body [273], [274].

Studies on chromosomal aberrations, which underlie many genetic diseases and cancer, are crucial for diagnostics, prognostics and targeted therapeutics [275], [276]. Examples of such diseases are the Wolf–Hirschhorn syndrome (WHS), that is caused by a partial deletion from human chromosome location 4p16.3 [277], the Charcot–Marie–Tooth disease (CMT), that is most commonly caused by duplication of the gene encoding peripheral myelin protein 22 (PMP22) on human chromosome 17 [278], and the acute myeloid leukemia (AML), that may be caused by translocations between human chromosome 8 and 21 [279].

Various computational methods have been proposed in the literature that perform alignment, that is aligning regions which are conserved in two (or more) genomic sequences, for the purpose of detecting chromosomal rearrangements (comparing sequences) [280]–[285]. Alignment-based methods cannot be solely employed to detect rearrangements, since they follow the assumption that order of homology is maintained between the sequences to be compared [62], [286]. Alignment-free (AF) approaches, on the other hand, do not have this limitation; in addition, they offer computational speedup advantages over alignment-based algorithms [64].

Among AF methods are information theory-based ones, that measure the amount of shared information within the sequences to quantify the similarity/dissimilarity between them. Information theory-based approaches have a broad range of applications, including but not limited to global and local characterization of DNA, e.g., prediction of transcription factor binding sites and classification of motifs, and gene mapping [65]. In the year 2015 another application of such approaches, namely finding rearrangements between DNA sequences, was introduced [131]. Here, we provide a significant improvement over the mentioned method. It should be noted that the two alignment-based and alignment-free approaches can be accompanied by genomic data visualization tools, that provide researchers with facilities to explore and analyze the genomic data [287].

We present Smash++, an alignment-free tool that finds chromosomal rearrangements between two DNA sequences based on their information content, which is obtained by a data compression technique. This computational solution follows a combination of probabilistic and algorithmic approaches for having a quantitative definition of information, although it can be seen as more of a probabilistic one [86]. Associated with Smash++ is a visualizer that is capable of visualizing as SVG images informationally similar regions between two genomic sequences. This tool also provides self- and relative redundancy (complexity) for the similar regions.

Smash++ is the improved version of Smash [131], featuring (a) improved accuracy, obtained by

using multiple finite-context models along with substitution-tolerant Markov models to find fine-grained and coarse-grained chromosomal rearrangements, (b) presenting self-complexity (redundancy) and relative redundancy of informationally similar regions between two DNA sequences, (c) improved user interface (UI) in command line, by adding several options to customize the tool for running, and resulting SVG image, by adding markers for positions of DNA bases and also plotting self- and relative redundancy, and (d) improved performance, in terms of memory usage.

6.2 Methods

The schema of the proposed method is illustrated in Fig. 6.1. Smash++ takes as inputs a reference and a target sequence and produces as output a position file, including local similarities of the two sequences, which can then be used by the Smash++ visualizer to produce an SVG image illustrating the similarities. This process has eight major stages: (1) compression of the original target file, based on the model of the original reference file, (2) filtering the information profile, which is the output of stage 1, and segmenting the target sequence after filtering, (3) reference-free compression of the segmented sequences, obtained by the previous stage, (4) compression of the original reference file, based on the model of segmented sequences, which are obtained by stage 2, (5) filtering the information profile and segmenting the reference sequence, (6) reference-free compression of the segmented sequences, (7) aggregating positions, that are generated by stages 3 and 6, and (8) visualizing the positions. The following sections describe the process in detail.

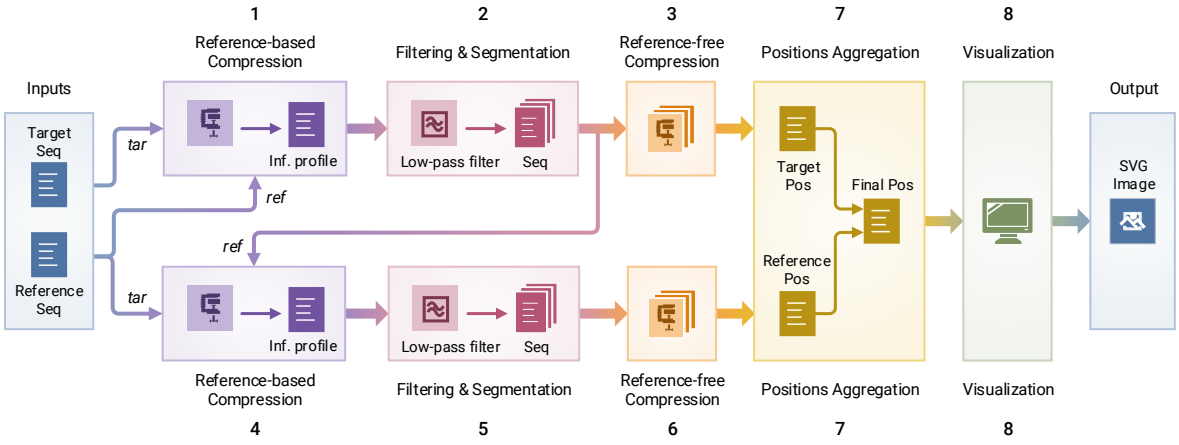

Figure 6.1: The schema of Smash++. The process of finding similar regions in reference and target sequences and computing the redundancy in each region includes eight stages. Smash++ outputs a *.pos file that includes the positions of the similar regions, and can be then visualized as an SVG image.

6.2.1 Data modeling

For the purpose of modeling genomic data, we exploit a combination of finite-context models and substitution-tolerant Markov models that are described in detail in Section 2.2.

6.2.2 Storing models in memory

The FCMs and STMMs include count values which need to be saved in memory. For this purpose, four different data structures have been employed considering the context-order size k, as follows:

- table of 64 bit counters, for $1 \leq k \leq 11$,
- table of 32 bit counters, for $k = 12, 13$,
- table of 8 bit approximate counters, for $k = 14$, and
- Count-Min-Log sketch of 4 bit counters, for $k \geq 15$.

The table of 64 bit counters, that is shown in Fig. 6.2a, simply saves the number of events for each context. The table of 32 bit counters saves in each position the number of times that the associated context is observed. When a counter reaches the maximum value $2^{32} - 1 = 4{,}294{,}967{,}295$, all the counts will be renormalized by dividing by two, as shown in Fig. 6.2b.

Approximate counting is a method that employs probabilistic techniques to count large number of events, while using a small amount of memory [288]. Fig. 6.3 shows the algorithm for two major functions associated with this method, UPDATE and QUERY. In order to update the counter, a pseudo-random number generator (PRNG) is used the number of times of the counter's current value to simulate flipping a coin. If it comes up 0/Heads each time or 1/Tails each time, the counter will be incremented. Fig. 6.2c shows the difference between arithmetic and approximate counting, and also the values which are actually stored in memory. Note that since an approximate counter represents the actual count by an order of magnitude estimate, one only needs to save the exponent. For example, if the actual count is 8, we store in memory $\log_2 8 = 3$.

Count-Min-Log Sketch (CMLS) is a probabilistic data structure to save frequency of events in a table by means of a family of independent hash functions [289]. The algorithm for updating and querying the counter is shown in Fig. 6.4. In order to update the counter, its current value is hashed with d independent hash functions; then, if the increment condition as in approximate counting (described above) is satisfied, the minimum hashed value (out of d values) will be updated, as shown in Fig. 6.2d. For the purpose of producing the family of hash functions we employ universal hashing, which is described in Chapter 5.

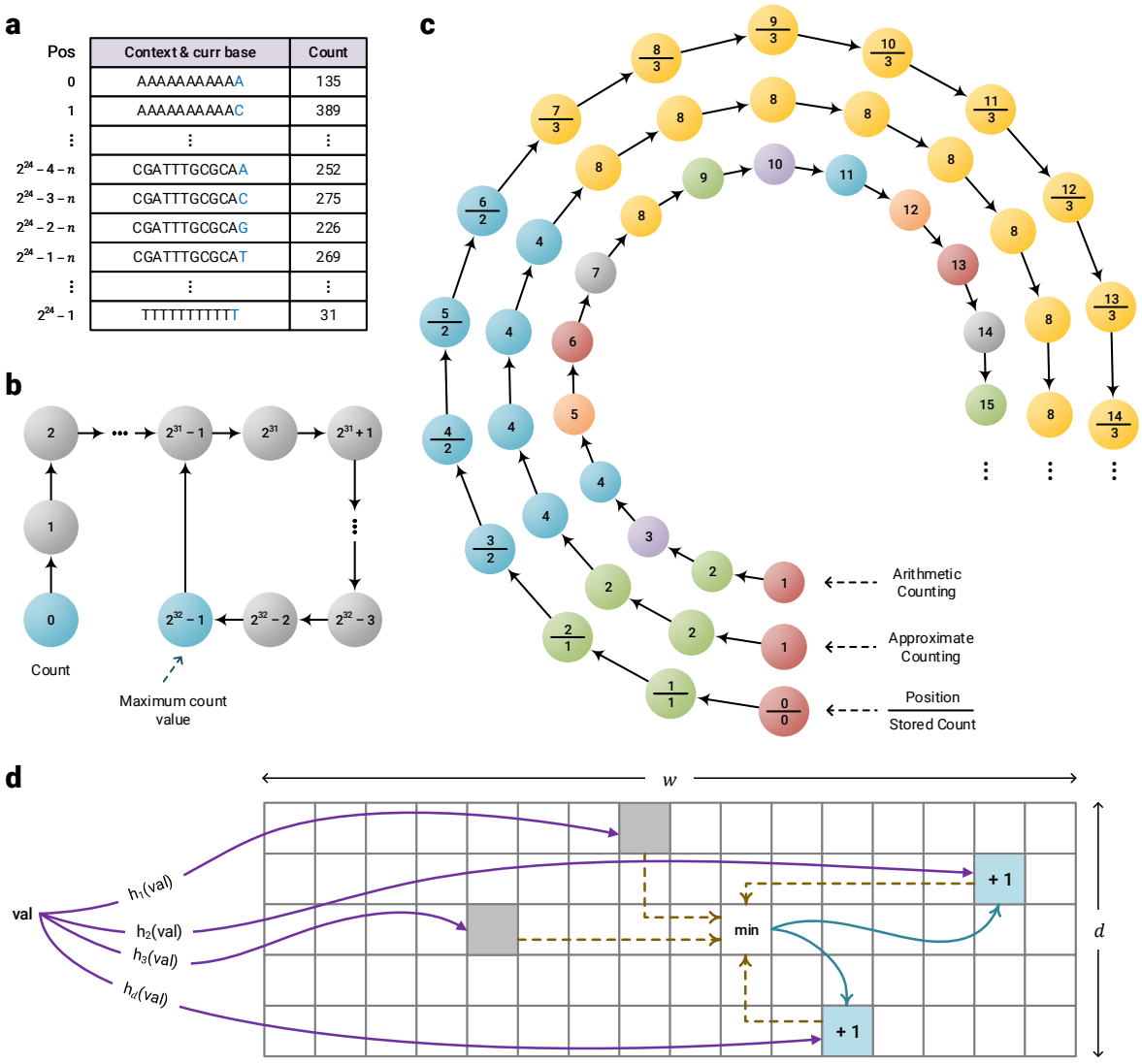

Figure 6.2: The data structures used by Smash++ to store the models in memory. (a) table of 64 bit counters that uses up to 128 MiB of memory, (b) table of 32 bit counters that consumes at most 960 MiB of memory, (c) table of 8 bit approximate counters with memory usage of up to 1 GiB and (d) Count-Min-Log sketch of 4 bit counters which consumes up to $\frac{1}{2}w \times d$ B of memory, e.g., if $w = 2^{30}$ and $d = 4$, it uses 2 GiB of memory.

6.2.3 Finding similar regions

To find similar regions in reference and target sequences, a quantity is required to measure the similarity. We use "per symbol information content", in bpb (bit per base), that can be calculated as

$$I(x) = -\log_2 P(x), \quad \forall x \in S, \tag{6.1}$$

where $P(x)$ denotes the probability of observing a nucleotide x in the sequence S (Eq. 2.17).

The information content is the amount of information required to represent a symbol in the target sequence, based on the model of the reference sequence. The less the value of this measure

```
1: function IncreaseDecision(x)
2:     return True with probability 1/2^x, else False
3: end function

4: function Update(x)
5:     c ← table[x]
6:     if IncreaseDecision(c) = True then
7:         table[x] ← c + 1
8:     end if
9: end function

10: function Query(x)
11:     c ← table[x]
12:     return 2^c − 1
13: end function
```

Figure 6.3: Approximate counting update and query.

is for two regions, the more amount of information is shared between them, and, therefore, the more similar the two regions are. Note that a version of this measure has been introduced in [131], which employs a single FCM to calculate the probabilities. Here, however, we exploit a cooperation between multiple FCMs and STMMs for highly accurate calculation of such probabilities.

The procedure of finding similar regions in a reference and a target sequence, illustrated in Fig. 6.5, is as follows: after creating the model of the reference, the target is compressed based on that model and the information content is calculated for each symbol in the target. Then, the content of the whole target sequence is smoothed by a Hann window [290], which is a discrete window function given by $w[n] = 0.5 - 0.5\ \cos\left(\frac{2\pi n}{N}\right)$, where $0 \leq n \leq N$ and length of the window is $N + 1$. Next, the smoothed information content is segmented considering a predefined threshold, meaning that the regions with the content greater than the threshold are filtered out. This is carried out for both regular and inverted repeat homologies and, at the end, the result would be the regions in the target sequence that are similar to the reference sequence (Fig. 6.5a). The described phase repeats for all of the target regions found, in the way that after creating the model for each region, the whole reference sequence is compressed to find those regions in the reference that are similar to each of the target regions (Fig. 6.5b). The final result would have the form of Fig. 6.5c.

6.2.4 Computing complexity

After finding the similar regions in reference and target sequences, we evaluate redundancy in each region, knowing that it is inversely related to Kolmogorov complexity (see Section 2.1.1), i.e., the more complex a sequence is, the less redundant it will be [93]. The Kolmogorov complexity is not computable, hence, an alternative is required to compute it approximately. It has been shown in the literature that a compression algorithm can be employed for this purpose [291]–[293]. Here,

Require: sketch width w, sketch depth d, m bins, prime $p \geq m$, randomly chosen integers $a_{1..d}$ and $b_{1..d}$ modulo p with $a \neq 0$

```
1:  function Hash(k, x)                    ▷ Universal hash family
2:      return ((a_k x + b_k) mod p) mod m
3:  end function

4:  function MinCount(x)
5:      minimum ← 15                       ▷ Biggest 4 bit number
6:      for k ← 1 to d do
7:          h ← Hash(k, x)
8:          if sketch[k][h] < minimum then
9:              minimum ← sketch[k][h]
10:         end if
11:     end for
12:     return minimum
13: end function

14: function IncreaseDecision(x)
15:     return True with probability 1/2^x, else False
16: end function

17: function Update(x)
18:     c ← MinCount(x)
19:     if IncreaseDecision(c) = True then
20:         for k ← 1 to d do
21:             h ← Hash(k, x)
22:             if sketch[k][h] = c then
23:                 sketch[k][h] ← c + 1
24:             end if
25:         end for
26:     end if
27: end function

28: function Query(x)
29:     c ← MinCount(x)
30:     return 2^c − 1
31: end function
```

Figure 6.4: Count-Min-Log Sketch update and query.

we employ a reference-free compressor to approximate the complexity and, consequently, the redundancy of the found similar regions in the reference and the target sequences. This compressor works based on cooperation of FCMs and STMMs, which has been previously described in detail. Note that the difference between reference-based and reference-free version of such compressor is that, in the former mode, a model is first created for the reference sequence and, then, the target sequence is compressed based on that model, while in the latter mode, the model is progressively created at the time of compressing the target sequence.

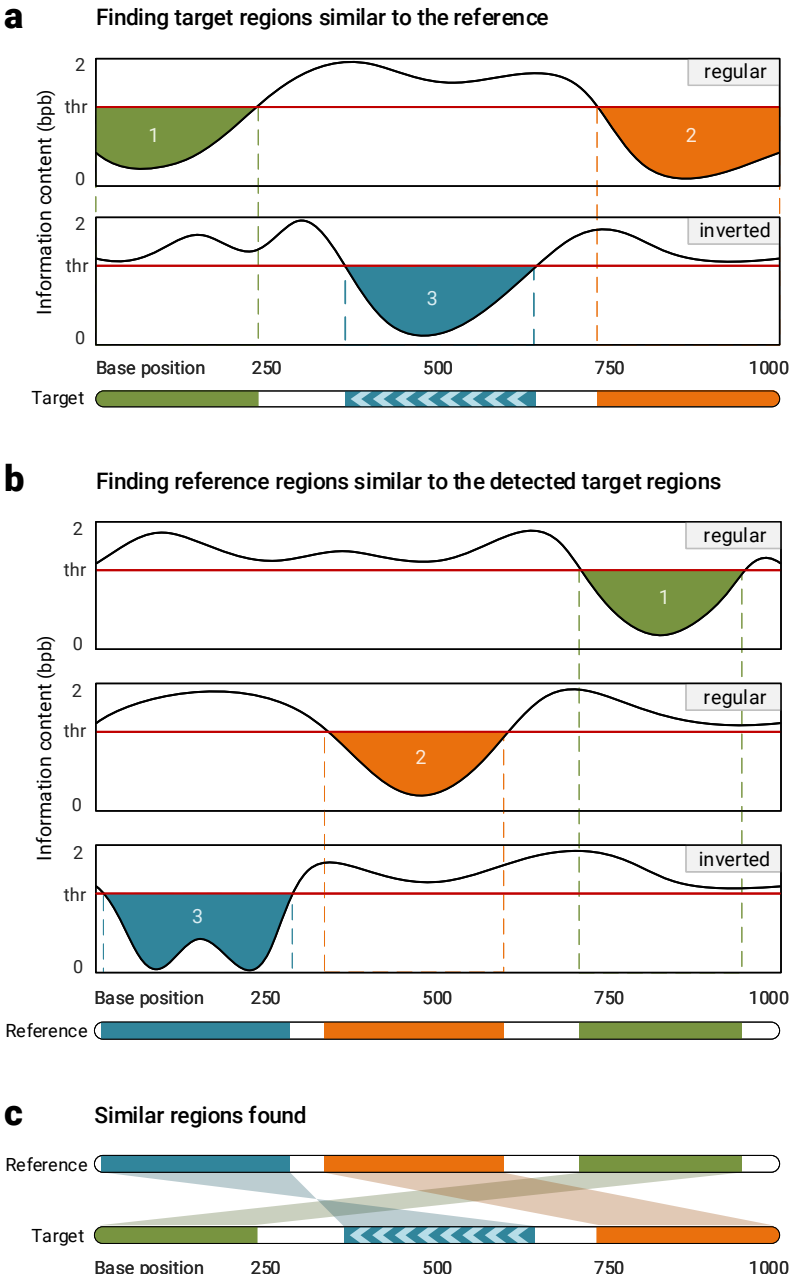

Figure 6.5: Finding similar regions in reference and target sequences. Smash++ finds, first, the regions in the target that are similar to the reference, and then, finds the regions in the reference similar to the detected target regions. This procedure is performed for both regular and inverted homologies.

6.3 Results and Discussion

Smash++ is implemented in the C++ language and is publicly available[1] under GNU GPLv3 license. The tool comes with a visualizer, that can be called in the command line with a flag called "-viz". Similar regions in reference and target sequences are shown with the same color, that are chosen randomly using HSV color model. The machine used for the tests had a 4-core 3.40 GHz Intel® Core™ i7-

[1] www.github.com/smortezah/smashpp

6700 CPU and 32 GiB of RAM.

6.3.1 Dataset

Smash++ and several other methods have been tested on a collection of synthetic and real sequences, that are described in Table 6.1. We used the GOOSE toolkit[1] to make the synthetic sequences of which the sizes vary from 1.5 kb to 100 Mb. We applied mutations and reversely complemented parts of the sequences. For a real dataset, we chose different sequences from bacteria, Aves, mammalia and fungi, with the sizes of ~1 Mb to ~127 Mb.

6.3.2 Application on synthetic data

Fig. 6.6 illustrates the result of running Smash++ and the associated visualizer on a synthetic dataset. The top sections show how we have built the reference and the target sequences. For example, to build the reference sequence in Fig. 6.6a, we generated three random sequences of size 500 b, using GOOSE, and concatenated them. For building the target sequence, we made reverse complements of parts I and III from the reference, and also mutated part II 2%, then we concatenated the parts in the order shown in the Figures 6.6b, c and d follow the same procedure. To build the target in Fig. 6.6e, we mutated the first 1 kb block of the reference 1%, the second block 2%, the third block 3%, up until the 60^{th} block that we mutated 60%.

The bottom sections of Fig. 6.6 show the output of the Smash++ visualizer, detecting similar regions regardless of their sizes. Note that for each detected region, the average value of redundancy and relative redundancy is illustrated. In Fig. 6.6b, part II of the reference is mutated 90%, i.e., nine out of every ten bases is mutated, on average. As expected, Smash++ does not recognize similarity between this pair of regions. Also, in the case of parts III and IV of the reference, since we detect similarity between part III of the reference and I of the target, and also part IV of the reference and II of the target, and there is no space between these regions, we join them and consider them as a bigger region of size 50 kb. Fig. 6.6e shows that Smash++ is able to detect ~43% of mutation, that has been made possible by the usage of STMMs (see Section 2.2.2). Fig. 6.6 shows that Smash++ can be employed to detect small-scale and large-scale similarities between DNA sequences.

6.3.3 Application on real data

Fig. 6.7 shows similarities between real sequences, found by Smash++. Figures 6.7a and 6.7b show similarities of chromosomes 18 and 14 of *Gallus gallus* (chicken) with orthologous chromosomes 20 and 16 of *Meleagris gallopavo* (turkey), respectively. These avian species, that are of great agricultural

[1] www.github.com/pratas/goose

Table 6.1: Synthetic and real dataset used in the experiments.

Sequence	Group	Length (b)	Description
GGA18	Aves	11,373,140	Access. CM000110 – *Gallus gallus* chromosome 18.
MGA20	Aves	10,730,484	Access. CM000981 – *Meleagris gallopavo* isolate NT-WF06-2002-E0010 breed Aviagen turkey brand Nicholas breeding stock chromosome 20.
GGA14	Aves	16,219,308	Access. CM000106 – *Gallus gallus* chromosome 14.
MGA16	Aves	14,878,991	Access. CM000977 – *Meleagris gallopavo* isolate NT-WF06-2002-E0010 breed Aviagen turkey brand Nicholas breeding stock chromosome 16.
HS12	Mammalia	133,275,309	Access. NC_000012 – *Homo sapiens* chromosome 12, GRCh38.p13 Primary Assembly.
PT12	Mammalia	130,995,916	Access. NC_036891 – *Pan troglodytes* isolate Yerkes chimp pedigree #C0471 (Clint) chromosome 12.
PXO99A	Bacteria	5,238,555	Access. CP000967 – *Xanthomonas oryzae* pv. *oryzae* causes the major disease of bacterial blight of rice (*Oryza sativa L.*). *X. oryzae* pv. *oryzae* PXO99A strain is virulent toward a large number of rice varieties representing diverse genetic sources of resistance [294].
MAFF 311018	Bacteria	4,940,217	Access. AP008229 – *X. oryzae* pv. *oryzae* MAFF 311018 is a Japanese race 1 strain [295].
ScVII	Fungi	1,090,940	Access. NC_001139 – *Saccharomyces cerevisiae* S288C chromosome VII.
SpVII	Fungi	1,105,967	Access. CP020299 – *Saccharomyces paradoxus* strain UFRJ50816 chromosome VII.
RefS	Synthetic	1500	It consists of three segments of 500 base size.
TarS	Synthetic	1500	To build TarS, segment I is mutated 2%, II is inversely repeated and III is duplicated.
RefM	Synthetic	100,000	It has four segments of 25 kilobase size.
TarM	Synthetic	100,000	For building TarM, segment I of RefM (out of total four) is inversely repeated, II is mutated 90%, III is duplicated and IV is mutated 3%.
RefL	Synthetic	5,000,000	It includes two segments, 2,500,000 bases each.
TarL	Synthetic	5,000,000	Segment I is inversely repeated and II is mutated 2% for building TarL.
RefXL	Synthetic	100,000,000	It is made of four segments, 25,000,000 bases each.
TarXL	Synthetic	100,000,000	Segment I is mutated 1%, segments II and III are inversely repeated and segment IV is duplicated to make TarXL.
RefMut	Synthetic	60,000	It includes 60 segments of 1 kilobase size.
TarMut	Synthetic	60,000	To build TarMut, the first segment (I) is mutated 1%, the second segment is mutated 2%, the third one is mutated 3%, and so on.
RefComp	Synthetic	1,000,000	It consists of 10 segments of 100 kilobases.
TarComp	Synthetic	1,000,000	For building it, the first segment (I) of RefComp is duplicated, the second, third and fourth segments are mutated 1%, 2% and 3%, respectively. The segments V, VI and VII of RefComp are inversely repeated, then mutated 4%, 5% and 6%, respectively. Finally, the segments VIII, IX and X are mutated 7%, 8% and 9%, respectively.
RefPerm	Synthetic	3,000,000	It includes three segments of 1 megabase size. In addition to the original sequence, it is permuted, using GOOSE toolkit, by blocks of sizes 450 kb, 30 kb, 1 kb and 30 b.
TarPerm	Synthetic	3,000,000	To build TarPerm, the first segment is mutated 1%, the second segment is inversely repeated and the third one is mutated 2%.

Note: the real dataset can be download from NCBI via accession number (access.) provided in the descriptions.

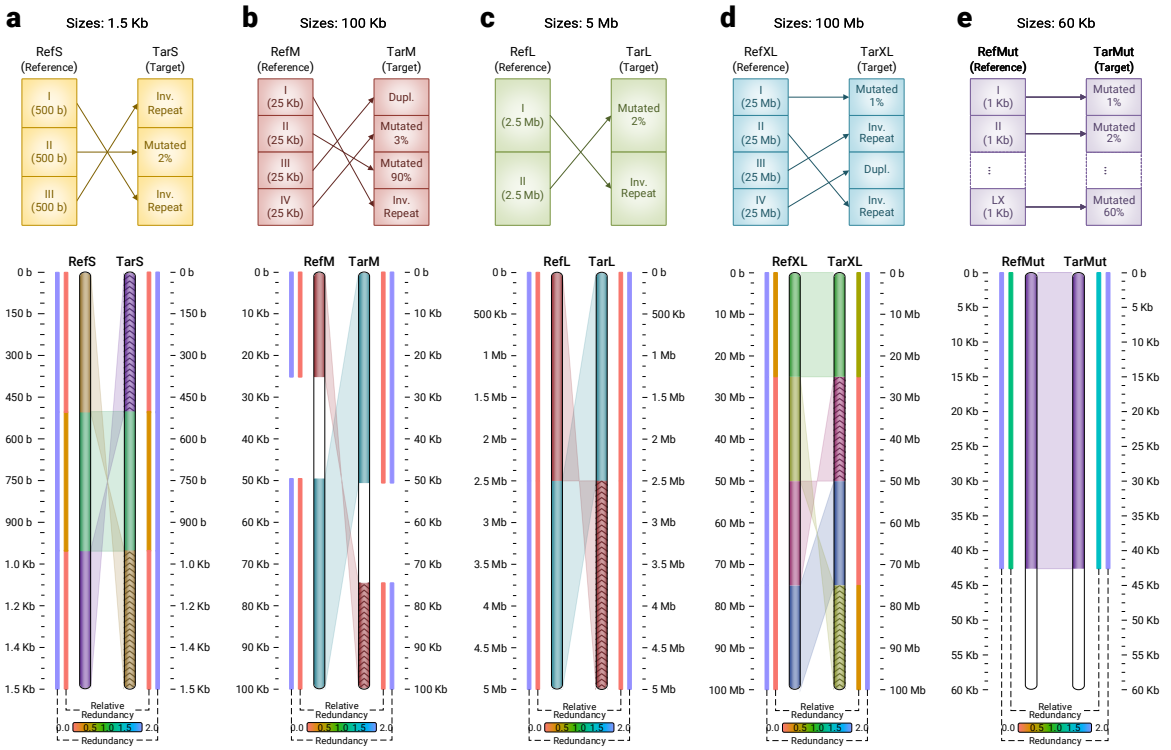

Figure 6.6: Similarities between synthetic sequences with different sizes, detected by Smash++. The parameters used are k-mer size = 14 and number of substitutions in STMM = 5, which are the default parameters used by Smash++. For the threshold, the default value of 1.5 and 1.97 are used for panels a-d and e, respectively. (a) 1.5 kb sequences; (b) 100 kb sequences. No similarity is detected for part II of the reference, since it is mutated 90%. Parts III and IV of the reference and I and II of the target are joined, since there is no space between consecutive regions; (c) 5 Mb sequences; (d) 100 Mb sequences; (e) 60 kb sequences. Roughly 43% of mutation is detected.

and commercial importance, are estimated to have diverged at 37.2 Mya [296]. Figures 6.7a and 6.7b demonstrate that Smash++ was able to find the inversions confirmed by FISH analysis, reported at [133], [297].

Fig. 6.7c demonstrates similarities between chromosomes 12 of *Homo sapiens* and *Pan troglodytes*, that are estimated to have diverged at 6.7 Mya. Fig. 6.7d illustrates similarities between *Xanthomonas oryzae* pv. *oryzae* PXO99A and *Xanthomonas oryzae* pv. *oryzae* MAFF 311018, two strains of *Xanthomonas oryzae* pv. *oryzae* (Xoo) pathogen, which causes the disease of bacterial blight of rice (*Oryzae sativa* L.). It is the most serious bacterial disease of rice that can reduce yields by as much as 50% [294]. Note that to have a clearer picture, we have not plotted the shades connecting similar regions; this can be achieved by "-l 6" option while calling the Smash++ visualizer.

Figures 6.8 and 6.9 show the result of applying Smash++ and other methods on GGA 18 / MGA 20 and GGA 14 / MGA 16 chromosomes, respectively. The methods included in these figures are as follows: (a) Smash++; (b) progressiveMauve [280], which uses an alignment objective score to detect rearrangement breakpoints when genomes have unequal gene content. It also applies a probabilistic alignment filtering method in order for removing erroneous alignments of unrelated sequences;

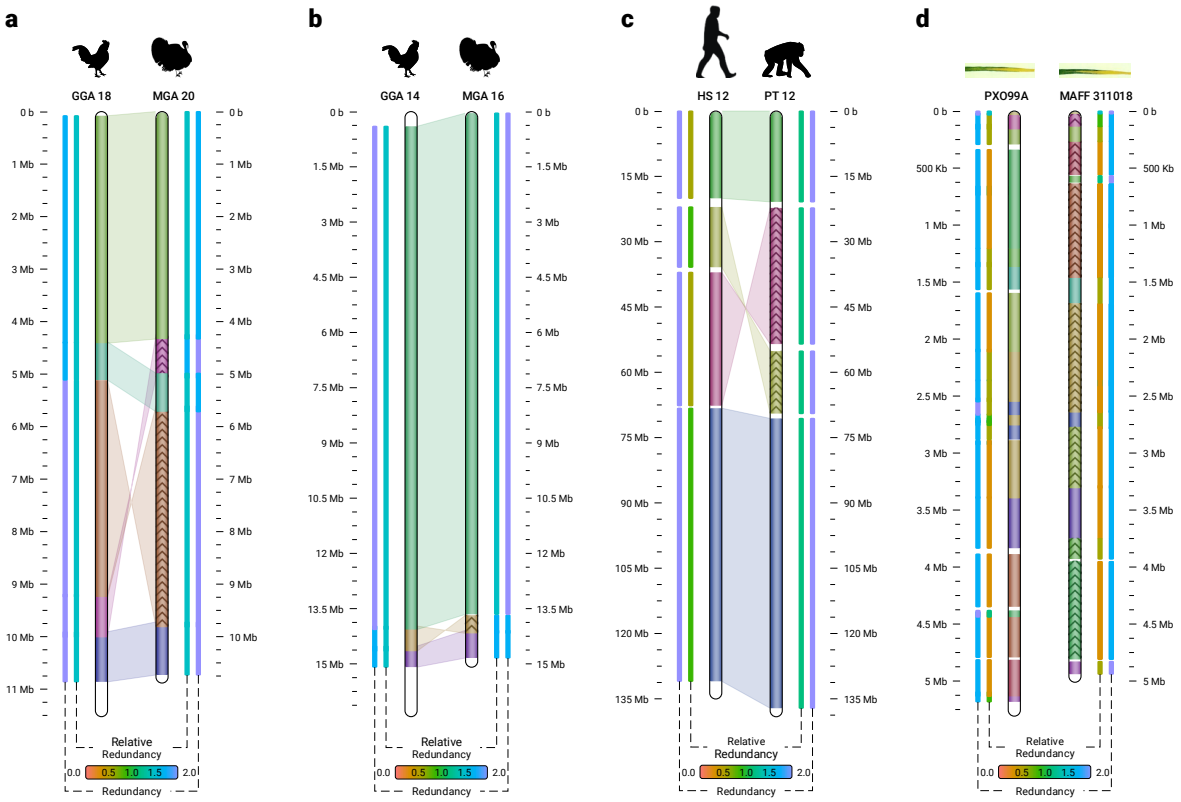

Figure 6.7: Similarities in a real dataset, detected by Smash++. (a) *G. gallus* chr. 18 and *M. gallopavo* chr. 20. The parameters were k-mer size = 14, No. substitutions in STMM = 5, threshold = 1.9 and min block size (m) = 500,000, i.e., the regions smaller than 500,000 bases were not considered for further processing; (b) *G. gallus* chr. 14 and *M. gallopavo* chr. 16. The result is obtained by setting k = 14, No. substitutions = 5, threshold = 1.95 and m = 400,000; (c) *H. sapiens* chr. 12 and *P. troglodytes* chr. 12. The parameters were k = 14, without using STMM, threshold = 1.9 and m = 100,000; (d) *X. oryzae* pv. *oryzae* PXO99A and *X. oryzae* pv. *oryzae* MAFF 311018 (two rice pathogens). The result obtained by setting k = 13, threshold = 1.55 and m = 10,000.

(c) the method proposed in [297], that takes a bacterial artificial chromosome (BAC)-based approach along with FISH analysis to develop an integrated physical, genetic and comparative map of chicken and turkey; (d) SynBrowser [284], which constructs synteny blocks using prebuilt alignments in the UCSC genome browser database; and (e) FISH analysis [133].

A comparison of Smash++ and other methods applied on chromosomes 12 of *H. sapiens* and *P. troglodytes* is provided in Fig. 6.10. The methods include (a) Smash++; (b) progressiveMauve; (c) Cinteny [285], that performs sensitivity analysis for synteny block detection and for the subsequent computation of reversal distances, by means of an extended version of ternary search trees (TST). Embedded in this extension are "walks" through the leaves of the tree, that correspond to walks on the genome markers in their linear order; (d) SynBrowser; and (e) D-GENIES [298], that works based on alignment of genomes by minimap2 software package [299].

Fig. 6.11 provides the comparison of our tool with progressiveMauve and the study [294], which uses an alignment method to find genome rearrangements in Xoo. As seen, the result provided by

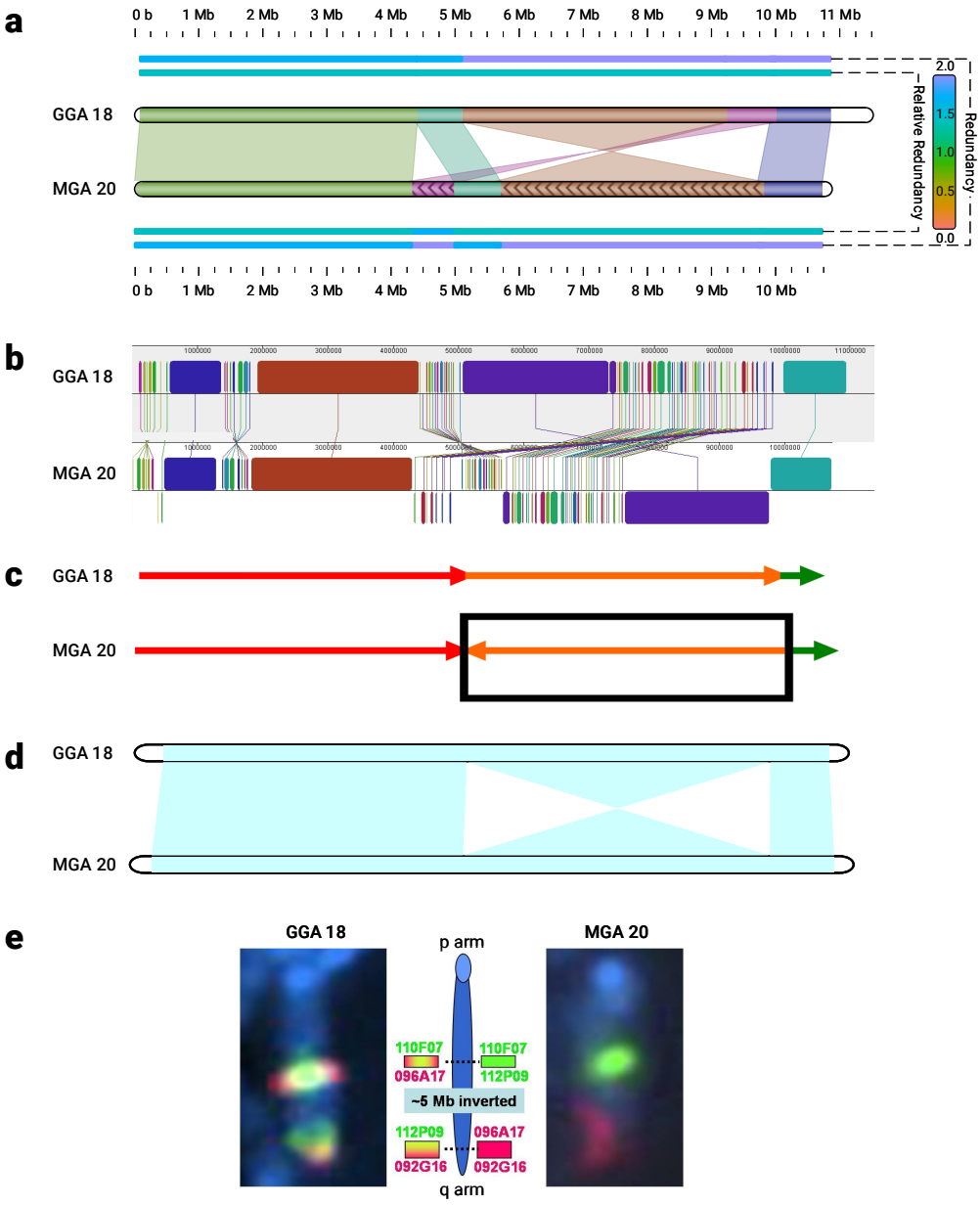

Figure 6.8: Pair-wise comparison of *G. gallus* chr. 18 and *M. gallopavo* chr. 20. (a) Smash++, with $k = 14$ and 5 used by an FCM and an STMM, respectively. The blocks smaller than 500 kb are discarded; (b) progressiveMauve [280], with LCB (locally collinear block) weight of 18,692. Reverse complements are shown in lower level; (c) adopted from [297], which is confirmed by FISH analysis. The box shows a local rearrangement; (d) SynBrowser [284], with the resolution of 150 kb (minimum size of a reference block); (e) adopted from [133], which confirms an inversion rearrangement of size ~5 Mb by FISH analysis.

Smash++ conforms to the one presented in the study [294], without performing an alignment.

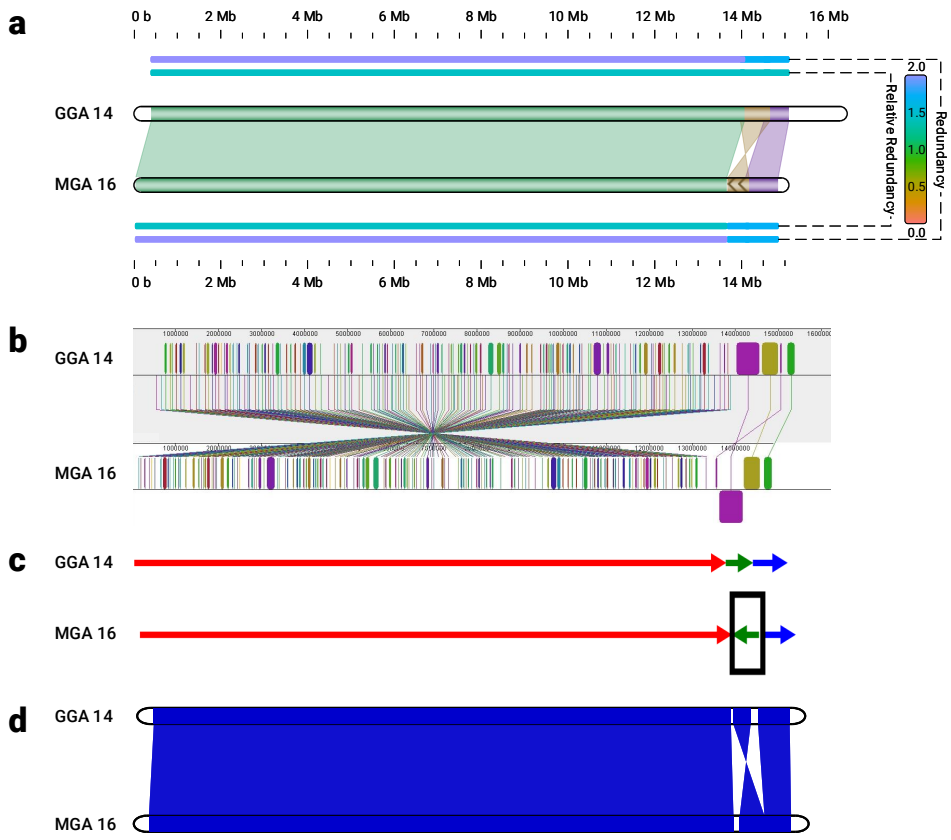

Figure 6.9: *G. gallus* chr. 14 compared to *M. gallopavo* chr. 16. (a) Smash++, employing an FCM and an STMM with $k = 14$ and 5, respectively. The blocks smaller than 400 kb are discarded; (b) progressiveMauve, with LCB weight of 27,424; (c) adopted from [297]. The box shows an inversion rearrangement; (d) SynBrowser, with the resolution of 150 kb.

6.3.4 Comparison to Smash

To have a better understanding of the improvement made over the first version (Smash), we compare the two tools on a synthetic and a real dataset (see Figures 6.12 and 6.13). In Table 6.1, the procedure of making the synthetic data (RefComp and TarComp) is described. Fig. 6.12 shows the comparison of Smash and Smash++ running on the synthetic dataset. Smash used an FCM with k-mer size of 14 and Smash++ used a cooperation of an FCM with k-mer size of 14 and an STMM with number of substitutions of 5. As the information profiles show, Smash++ is able to model better the data, since it uses less information (lower information contents) to describe the target based on the reference; this is possible because of employing a cooperation of the FCM and the STMM instead of using solely an FCM. We expect the output to have the following format: parts I, II, III and IV of the reference and the target are similar (including rearrangements), there is also inverted repeats between parts V, VI and VII of the sequences, and finally, there are rearrangements between parts VIII, IX and X of the sequences. When there is no space between consecutive regions, Smash++ joins them; therefore, we expect Smash++ to detect three similar regions: the one including parts I, II, III and

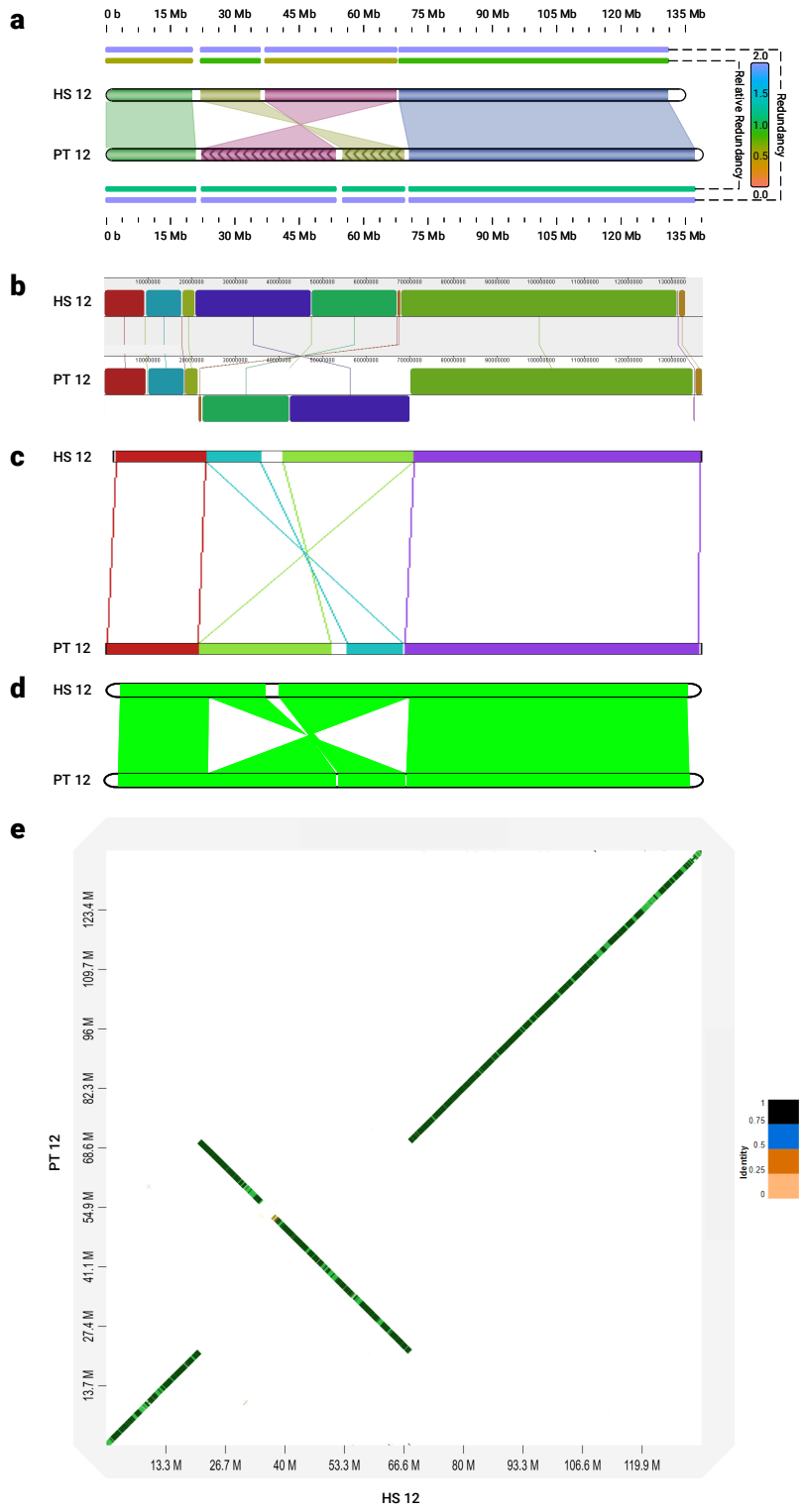

Figure 6.10: Comparison of *H. sapiens* chr. 12 and *P. troglodytes* chr. 12. (a) Smash++, with $k = 14$ used by an FCM. The blocks smaller than 100 kb are discarded; (b) progressiveMauve, with LCB weight of 55,186; (c) Cinteny [285], with minimum length of synteny block = 1 kb, maximum gap between adjacent markers = 5 Mb and minimum number of markers = 1; (d) SynBrowser, with the resolution of 150 kb; (e) D-GENIES [298], in "strong precision" mode.

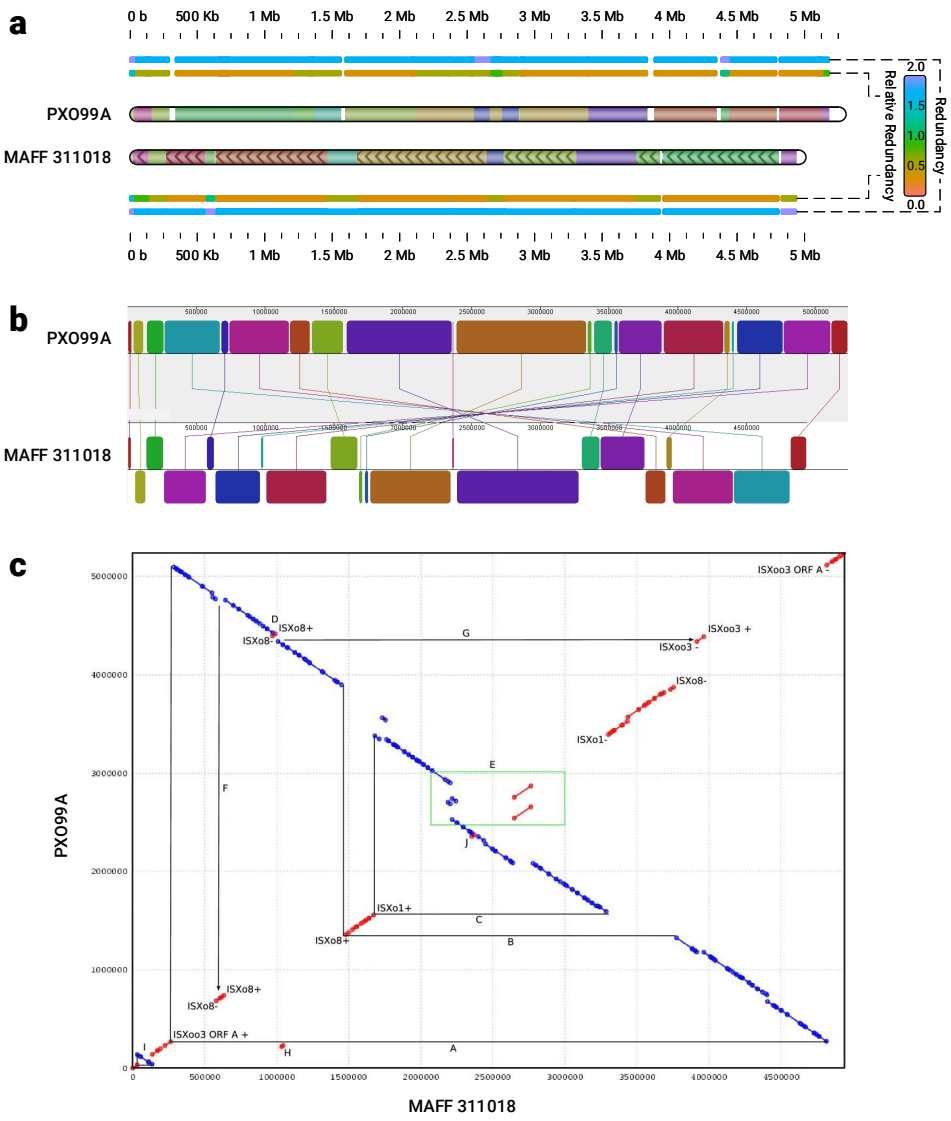

Figure 6.11: Pair-wise comparison of PXO99A and MAFF 311018. (a) Smash++, with $k = 13$ used by an FCM. The blocks smaller than 10 kb are discarded. In order to make the figure clearer, the shaded paths for connecting corresponding regions are not drawn; (b) progressiveMauve, with LCB weight of 3926; (c) adopted from [294], which employs an alignment-based method to obtain this dot plot. The blue and red colors shows regions of PXO99A that align to the same or opposite strand of MAFF 311018, respectively.

IV, the one with parts V, VI and VII, and the one including parts VIII, IX and X. The rearrangements map shows that Smash++ fulfills our expectation. On the other side, Smash was not able to detect all rearrangements, showing that to model such dataset we need more than a single FCM.

The result of running Smash and Smash++ on a real dataset, *Saccharomyces cerevisiae* chromosome VII and *Saccharomyces paradoxus* chromosome VII, is demonstrated in Fig. 6.13. *S. cerevisiae* is a species of yeast that plays a key role in winemaking, baking and brewing. It has been a eukaryotic model organism that gives insights into molecular functioning of human cells [300]. *S. paradoxus*

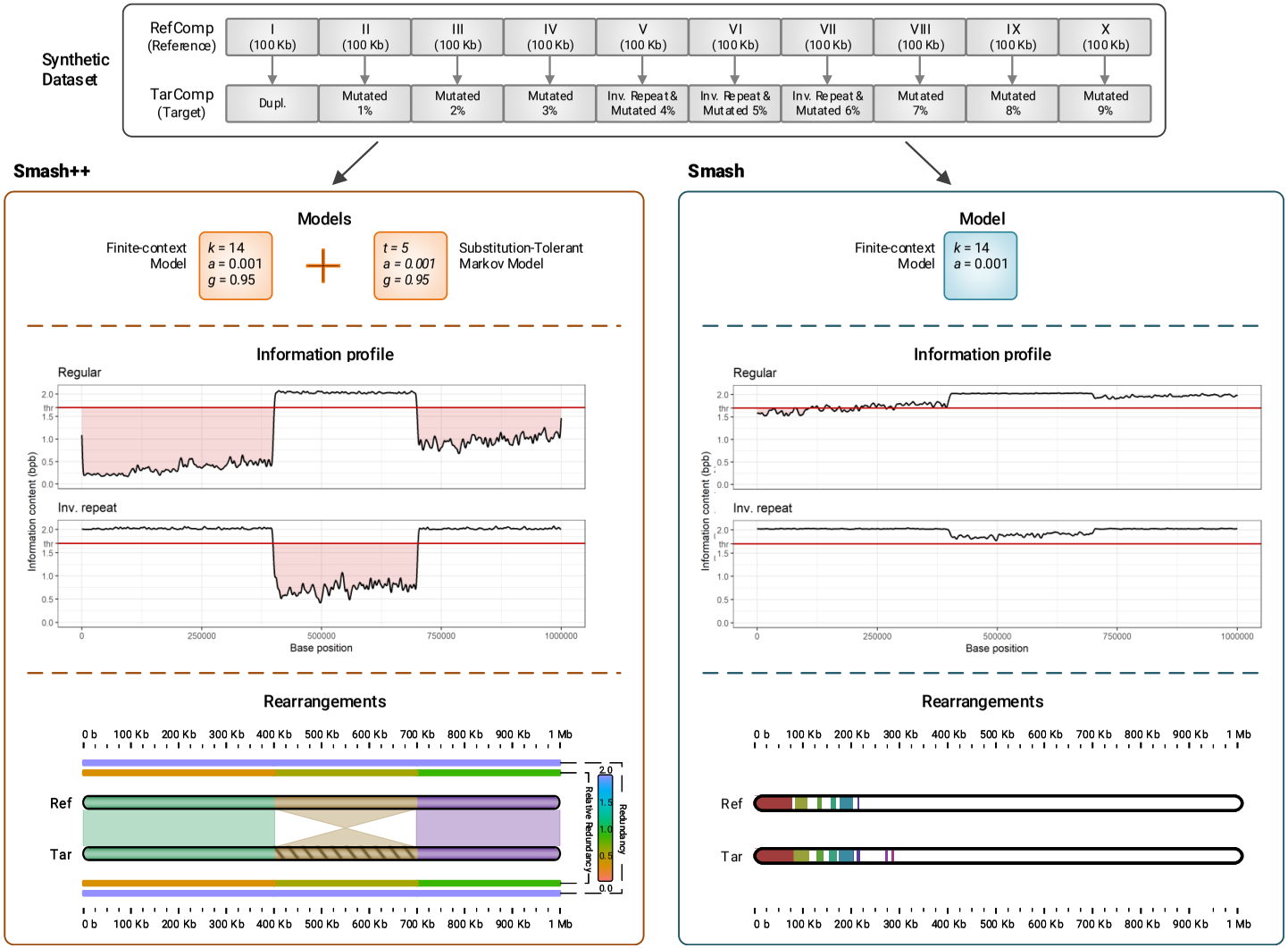

Figure 6.12: Comparison of Smash++ and Smash on the synthetic dataset. Using cooperation of an FCM and an STMM (in Smash++) produces more accurate results rather than using a single FCM (in Smash).

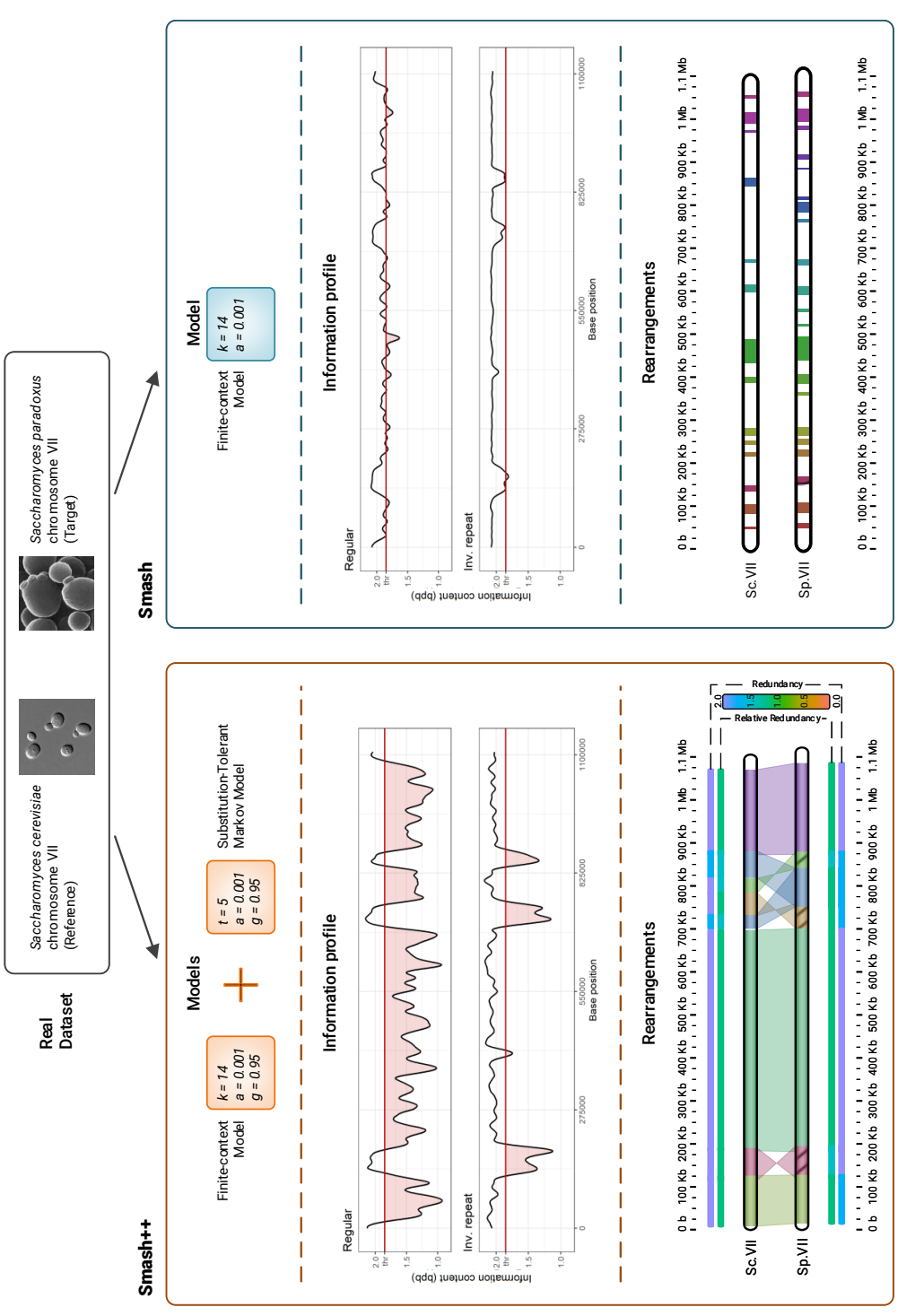

Figure 6.13: Comparison of Smash++ and Smash on the real dataset, including *S. cerevisiae* chr. VII and *S. paradoxus* chr. VII. The rearrangements maps clearly show the improvement made over Smash, using an FCM along with an STMM.

is closest known species to the *S. cerevisiae*, that has proved its importance on different fields of the life sciences, including evolution, ecology and biotechnology [301]. For the experiment, we ran Smash using an FCM with k-mer size of 14, and Smash++ using an FCM with k-mer size of 14 co-operated with an STMM with number of substitutions of 5. As can be seen, using an FCM along with an STMM could drastically improve modeling the data, which led to find rearrangements more accurately. The rearrangements map of Smash++ conforms to the previous study [300].

6.3.5 Robustness against fragmented data

Inherited from Smash, Smash++ is capable of finding similarities between a fragmented reference and a target sequence. Fig. 6.14 shows robustness of the proposed tool against fragmented data, for different randomly permuted block sizes. As can be seen, the same three target regions are detected even when the reference is fragmented to 100,000 blocks of 30 bases. This capability might be of interest in case of non-assembled sequences or in presence of assembly errors.

6.3.6 Benchmarking

Fig. 6.15 illustrates the performance of the proposed tool in terms of memory and time usage for all datasets (see Table 6.2 for more details). Size of the datasets and number of detected similar regions between each pair of sequences ("# Rearr") are shown at the bottom of the figure. The pair dataset "Perm30" and the pair "Perm1000" are, as outliers, not shown. Fig. 6.15a shows the peak memory in gigabytes used by Smash++ on all synthetic and real datasets. As can be seen, it is ~1 GiB for all datasets. The maximum peak memory, ~1.08 GiB, was used when the proposed tool was run on human and chimpanzee chromosomes 12. It should be mentioned that the memory usage of Smash++ is related to the k-mer size that is used to model the data because different data structures are used for different k-mer sizes (see Section 6.2.2). Sizes 13 and 14 were used to perform the experiments. The maximum memory usage of ~1 GiB enables Smash++ to run on any present-day standard computer.

Fig. 6.15b demonstrates elapsed (wall clock) times, in minutes. The elapsed times rely on the file sizes along with the number of detected similar regions, meaning that the greater the number of regions and/or the greater the dataset size, the more time will be taken. Note that it is not a linear relation. As an example, the pair dataset "Large" and the pair "PXO99A_MAFF311018" have approximately the same total size of 10 Mb; but in the former case that two similarities is detected, Smash++ takes ~16 seconds, and in the latter case with 23 similarities (~12 times more than the former case), the proposed tool takes ~48 seconds (3 times more) to run. As another example, carrying out Smash++ on the pair "Large" with 50 times larger size than the pair "Medium" leads to detection of the same number of rearrangements, i.e., 2, while it takes only ~3 times more time. Regarding the pair "Perm30" with 11,565 similarities detected (Table 6.2), we note that it has a mas-

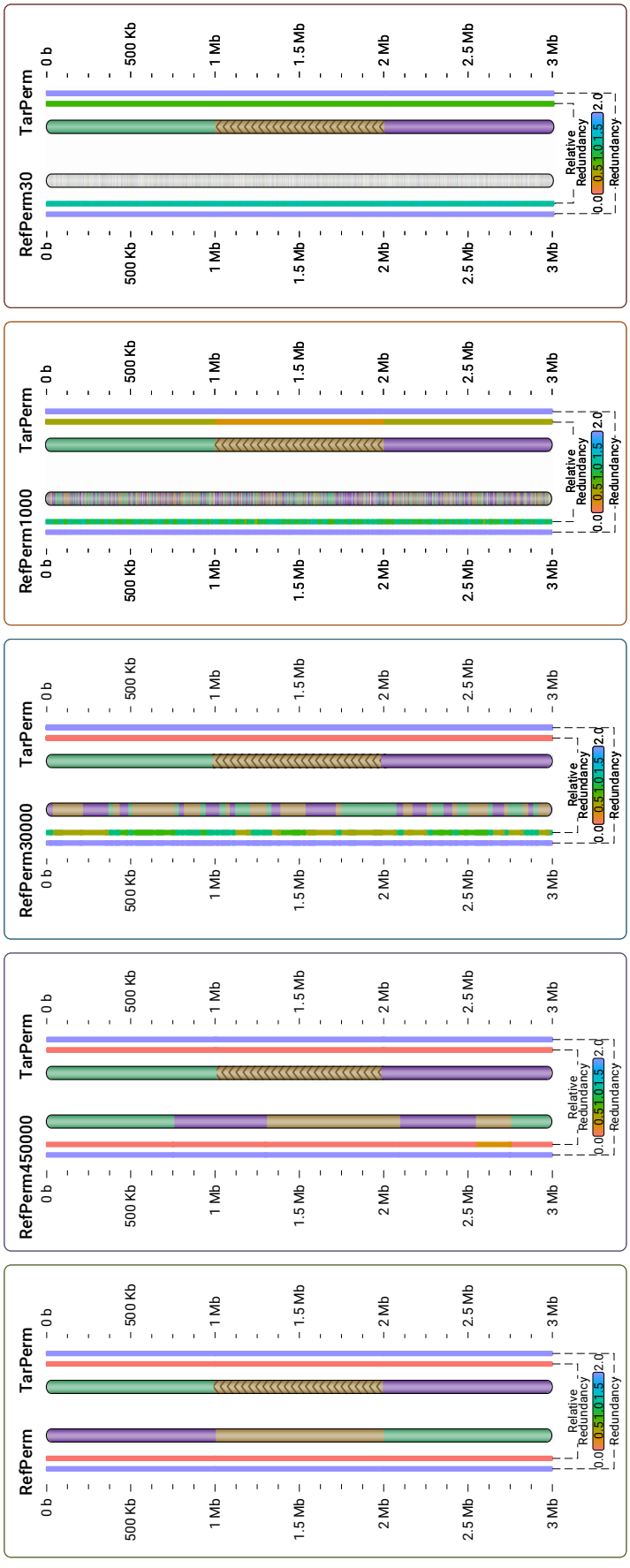

Figure 6.14: Similarity of a target sequence to a reference sequence, when the reference is permuted by blocks of different sizes of 450 kb, 30 kb, 1 kb and 30 b. To run Smash++, an FCM with k-mer size of 14 and the threshold of 1.5 was used. It shows the proposed method is resistant to fragmentation of sequence; for example, in the case of 30 b, although the reference is highly fragmented, Smash++ could detect the same three similar regions in the target as the original case.

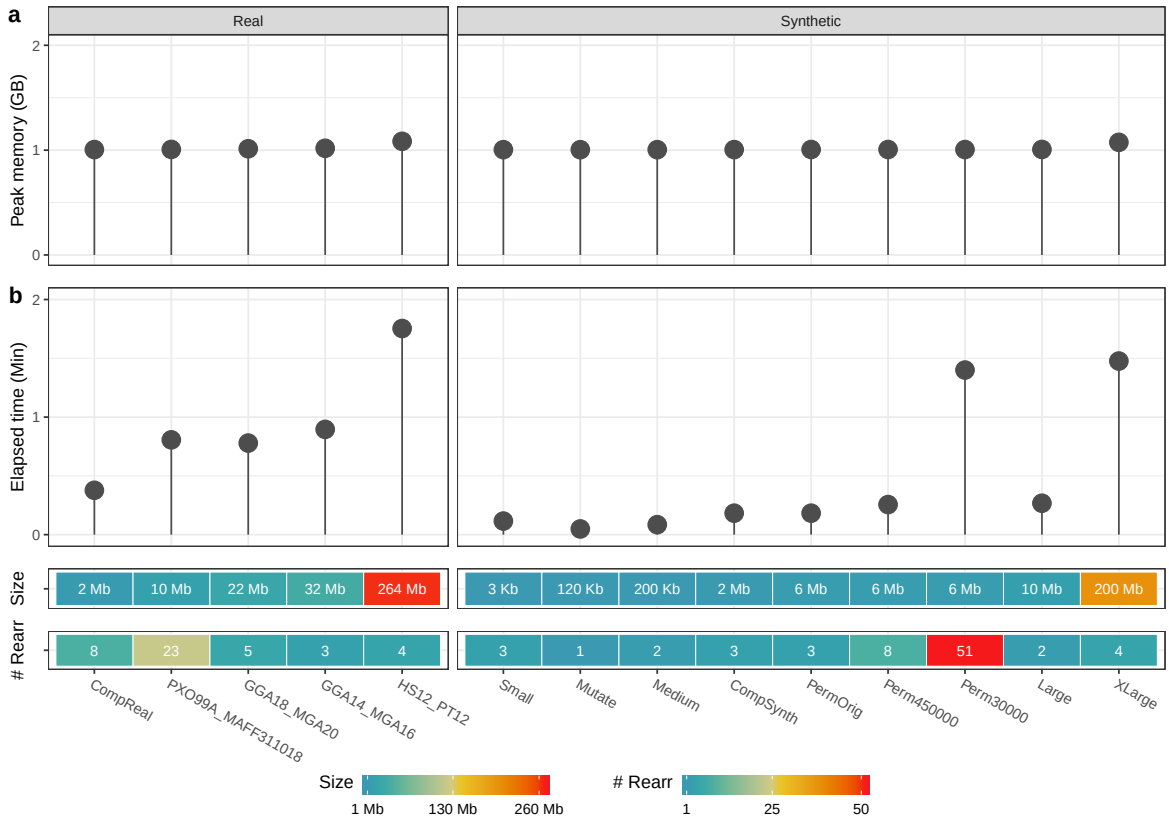

Figure 6.15: (a) The peak memory consumption, in gigabytes; and (b) the elapsed (wall clock) time usage, in minutes, of Smash++ by running on all synthetic and real datasets described in Table 6.1.

sively fragmented reference sequence with 10,000 fragments of 30 b; therefore it is by far the most time-consuming dataset. Note that the difference between the values of 10,000 (number of reference fragments) and 11,565 (number of similar regions) arises from the fact that a number of the reference chunks are similar to more than one target region and vice versa.

A major advantage of Smash++ over Smash is using a combination of FCMs and STMMs to better model the data; however, to have an idea about how the performance of these two tools can be compared, we let Smash++ run with only one FCM on the dataset described in Table 6.1. We also did not compute self-complexity, similarly to Smash. Fig. 6.16 and Table 6.3 show the results. In Fig. 6.16a, the range of peak memory usages (from minimum to maximum usage) are compared, while running Smash and Smash++ on different real and synthetic datasets. The diamond symbol shows the mean value. The results show that the maximum peak memory usage by Smash++ is 1.9 times less than that by Smash. In Fig. 6.16b, the range of wall clock times is compared for these two tools. As mentioned in the figure, Smash++ runs 5.4 times faster than Smash on the tested datasets. It is worth mentioning that due to the absence of another tool that provides relative compression in addition to detecting rearrangements, we cannot have a fair quantitative comparison to other tools, in terms of time and memory usage; therefore, we have only included the results obtained by Smash++ and Smash.

Table 6.2: Performance of Smash++, in terms of memory and time usage, running on all synthetic and real datasets described in Table 6.1.

Dataset (Reference + Target)	Size (b)	# Rearr	Memory (KB)	Elapsed time (s)	User time (s)	System time (s)
Synthetic						
Small	3000	3	1,053,528	6.95	1.76	5.22
Medium	200,000	2	1,053,328	5.11	1.45	3.68
Large	10,000,000	2	1,055,860	16.06	12.20	3.73
XLarge	200,000,000	4	1,126,892	88.61	78.67	7.95
Mutate	120,000	1	1,053,308	2.93	0.87	2.06
PermOrig	6,000,000	3	1,055,304	10.98	4.43	6.50
Perm450000	6,000,000	8	1,055,128	15.37	5.74	9.55
Perm30000	6,000,000	51	1,054,900	84.02	24.58	59.14
Perm1000	6,000,000	702	1,054,660	652.58	176.82	474.16
Perm30	6,000,000	11,565	1,057,804	24,958.90	16,542.13	7629.04
CompSynth	2,000,000	3	1,054,236	10.98	5.58	5.26
Real						
GGA14_MGA16	31,542,564	3	1,067,380	53.74	44.80	8.57
GGA18_MGA20	22,419,393	5	1,062,700	46.75	37.77	8.73
HS12_PT12	264,271,225	4	1,137,216	105.21	95.15	7.36
PXO99A_MAFF311018	10,324,186	23	1,056,048	48.39	16.24	31.86
CompReal	2,228,294	8	1,054,532	22.66	9.21	13.35

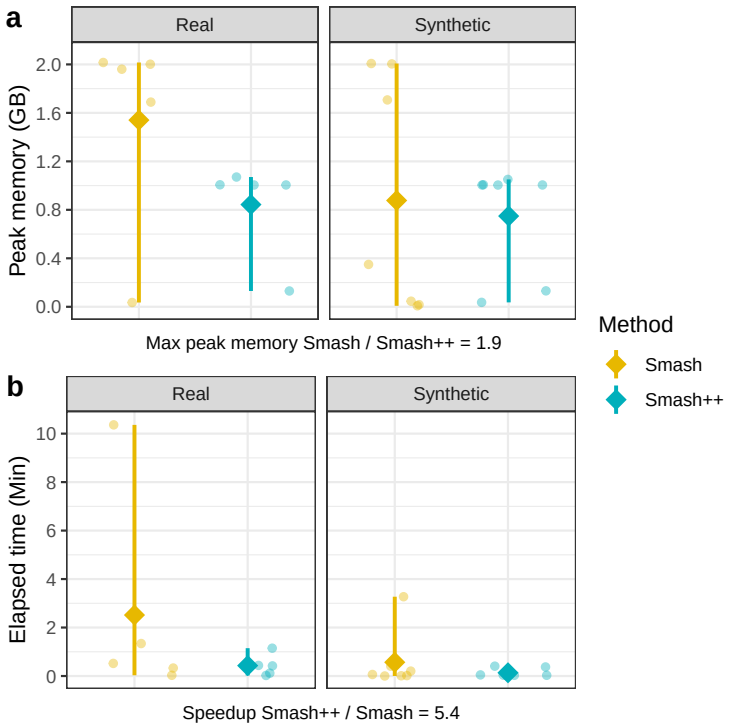

Figure 6.16: Comparison of Smash++ and Smash, in terms of (a) memory usage; and (b) time usage, running on real and synthetic data described in Table 6.1. To have a fair comparison, only one model (FCM) is used by Smash++, and also self-complexity is not computed. Diamonds indicate the mean, and bars, the ranges from minimum to maximum values.

Table 6.3: The memory and time usage of Smash++ and Smash, running on synthetic and real dataset (Table 6.1). To have a fair comparison, Smash++ uses only one model (FCM), as Smash does.

Dataset (Reference + Target)	Size (b)	Memory (KB)		Elapsed time (s)	
		Smash	Smash++	Smash	Smash++
Synthetic					
Small	3000	8440	1,053,416	0.03	2.14
Medium	200,000	366,624	1,053,560	0.95	1.77
Large	10,000,000	2,104,976	1,055,852	12.15	2.75
XLarge	200,000,000	2,101,496	1,102,036	196.29	24.36
Mutate	120,000	18,268	38,136	0.61	1.78
PermOrig	6,000,000	48,252	137,784	3.76	0.97
CompSynth	2,000,000	1,789,796	1,053,960	24.33	22.39
Real					
GGA14_MGA16	31,542,564	2,113,716	1,053,672	19.74	7.01
GGA18_MGA20	22,419,393	2,055,804	1,054,124	31.20	25.06
HS12_PT12	264,271,225	2,098,868	1,122,652	621.70	68.87
PXO99A_MAFF311018	10,324,186	1,771,048	1,055,612	80.38	26.23
CompReal	2,228,294	36,992	136,588	1.79	1.36

Note: the pair datasets "Perm450000", "Perm30000", "Perm1000" and "Perm30" were excluded from this experiment, since they were basically used to show the potential of Smash++ to handle fragmented data.

6.4 Conclusions

Finding genomic rearrangements is crucial, since they play an important role in genetic disorders, cancer and chromosomal evolution. We presented Smash++, an alignment-free tool that accurately finds small- and large-scale genomic rearrangements between pairs of DNA sequences, by employing a data compression approach. This memory-efficient tool was successfully tested on several synthetic and real data from bacteria, fungi, Aves and mammalia. The presented results showed that the detected rearrangements complied with previous studies which used alignment-based methods or performed FISH analysis. Smash++ consumed a maximum of ~1 GiB of memory, among all experiments, which showed that it can be run on any computer, nowadays. The proposed tool has the potential to improve accuracy of diagnostic and genetic counselling, and also to guide future investigations into development of personalized therapeutic.

Chapter 7

Conclusions

The present thesis is motivated by the increasing growth of the production of omics data that has been led by advancements in high-throughput sequencing technologies along with accessing new samples, e.g. those coming from archaeological remains. These data need to be stored, analyzed and transmitted. In this thesis, we have investigated methods that are able to provide efficient representations of omics data by means of compression or encryption, and then, employed them for the purpose of analysis.

In order to quantify the amount of information in and between omics sequences, we have described a number of measures, among which the normalized relative compression is the most lightweight comparative measure in terms of time and memory consumption. We have also presented a number of models, such as finite-context models and substitution-tolerant Markov models, that can be employed for compression and analysis of omics data. As an application to the mentioned measures and models, we investigated the role of inverted repeats on similarities of pairs of genomic sequences, and found that applying IRs will make our models better fit the genomic data.

To facilitate the storage and analysis of genomic and proteomic data, we proposed three lossless compression methods: GeCo2 and Jarvis for DNA and AC for protein sequences. GeCo2 and AC work based on a combination of FCMs and STMMs and Jarvis exploits repeat models and a competitive prediction model in addition to FCMs and STMMs. Applying them on various synthetic and real genomic and proteomic sequences showed that they can outperform previously proposed algorithms in terms of compression ratio. Moreover, we applied AC on 10,677 sequences from viruses, eukaryota, archaea and bacteria domains for the purpose of analyzing their complexity. The results showed that viruses and eukaryota are the most and the least complex sequences to compress.

The omics data used in the fields such as personalized medicine is naturally sensitive; therefore, its security needs to be preserved. We have designed a tool, Cryfa, that can securely encrypt genomic data and at the same time, is able to compact FASTA and FASTQ sequences by a factor of three. It

employs AES encryption and a shuffling mechanism to enhance the security of data. Applied on numerous synthetic and real data showed that the proposed method is faster than general-purpose and special-purpose algorithms. Also, it has only a few megabytes RAM usage.

Aside from efficient representation of omics data in compressed or encrypted manners, we can exploit the compression methods and compression-based measures for the purpose of analysis. We have investigated the detection of unique regions in a species with respect to close species to have an insight into evolutionary traits. As a result, we proposed two alignment-free tools, CHESTER and FRUIT, that are capable of finding and visualization of distinct regions between a pair of collections of genomic or proteomic sequences. These tools employ Bloom filters for detection and localization of distinct regions. By applying the proposed methods on genomic and proteomic sequences of modern human and Neanderthals, as one of the closest hominins to humans, we identified a number of regions (present in modern humans and absent in Neanderthals) which may express new functions linked with evolution of modern human.

Finally, as another application of data compression in omics data analysis, we investigated the identification of genomic rearrangements, which play an important role in chromosomal evolution, genetic disorders and cancer. We have proposed an alignment-free tool, Smash++, that is able to identify and visualize small- and large-scale rearrangements between pairs of genomic sequences. Tested on various synthetic and real data from bacteria, fungi, Aves and mammalia showed that the proposed method can find accurately the rearrangements that were partially reported by other methods such as FISH analysis, which is a wet laboratory approach.

In the following we mention some research directions for future work. Concerned with omics data compression, it is worthwhile to provide random access capability, which allows one to access only a part of genome or proteome and removes the need to decompress the entire compressed file. Moreover, it may be desired to perform operations directly on compressed data, which can lead to a faster analysis. These ideas can also be extended to the context of security preservation (encryption/decryption).

Another idea worth exploring is to reconstruct phylogenetic trees by means of the compression models that are developed in the current thesis, since they are able to measure the distance of information. It would be an application of data compression in omics data analysis, in which evolutionary relationships between a group of organisms is represented.

Bibliography

[1] L. Pray, "Discovery of DNA structure and function: Watson and Crick," *Nature Education*, vol. 1, no. 1, p. 100, 2008.

[2] B. Alberts, A. Johnson, J. Lewis, *et al.*, *Molecular Biology of the Cell*, 6$^{\text{th}}$ edition. Garland Science, 2017.

[3] M. E. Hudson, "Sequencing breakthroughs for genomic ecology and evolutionary biology," *Molecular ecology resources*, vol. 8, no. 1, pp. 3–17, 2008.

[4] A. E. Shearer, A. P. DeLuca, M. S. Hildebrand, *et al.*, "Comprehensive genetic testing for hereditary hearing loss using massively parallel sequencing," *Proceedings of the National Academy of Sciences*, vol. 107, no. 49, pp. 21 104–21 109, 2010.

[5] P. Bradley, N. C. Gordon, T. M. Walker, *et al.*, "Rapid antibiotic-resistance predictions from genome sequence data for *Staphylococcus aureus* and *Mycobacterium tuberculosis*," *Nature communications*, vol. 6, no. 1, pp. 1–15, 2015.

[6] D. Marco, *Metagenomics: current innovations and future trends*. Horizon Scientific Press, 2011.

[7] S. D. J. Pena and R. Chakraborty, "Paternity testing in the DNA era," *Trends in Genetics*, vol. 10, no. 6, pp. 204–209, 1994.

[8] N. Fierer, C. L. Lauber, N. Zhou, *et al.*, "Forensic identification using skin bacterial communities," *Proceedings of the National Academy of Sciences*, vol. 107, no. 14, pp. 6477–6481, 2010.

[9] T. Lan and C. Lindqvist, "Paleogenomics: Genome-scale analysis of ancient DNA and population and evolutionary genomic inferences," in *Population Genomics*, Springer, 2018, pp. 323–360.

[10] A. T. Duggan, M. F. Perdomo, D. Piombino-Mascali, *et al.*, "17$^{\text{th}}$ century variola virus reveals the recent history of smallpox," *Current Biology*, vol. 26, no. 24, pp. 3407–3412, 2016.

[11] R. Wu, "Nucleotide sequence analysis of DNA: I. Partial sequence of the cohesive ends of bacteriophage λ and 186 DNA," *Journal of molecular biology*, vol. 51, no. 3, pp. 501–521, 1970.

[12] F. Sanger, S. Nicklen, and A. R. Coulson, "DNA sequencing with chain-terminating inhibitors," *Proceedings of the national academy of sciences*, vol. 74, no. 12, pp. 5463–5467, 1977.

[13] A. M. Maxam and W. Gilbert, "A new method for sequencing DNA," *Proceedings of the National Academy of Sciences*, vol. 74, no. 2, pp. 560–564, 1977.

[14] S. Beck and F. M. Pohl, "DNA sequencing with direct blotting electrophoresis," *The EMBO journal*, vol. 3, no. 12, pp. 2905–2909, 1984.

[15] L. M. Smith, J. Z. Sanders, R. J. Kaiser, *et al.*, "Fluorescence detection in automated DNA sequence analysis," *Nature*, vol. 321, no. 6071, pp. 674–679, 1986.

[16] J. M. Prober, G. L. Trainor, R. J. Dam, *et al.*, "A system for rapid DNA sequencing with fluorescent chain-terminating dideoxynucleotides," *Science*, vol. 238, no. 4825, pp. 336–341, 1987.

[17] R. Y. Tsien, P. Ross, M. Fahnestock, *et al.*, "DNA sequencing," pat. WO1991006678A1, Oct. 1990.

[18] M. Ronaghi, S. Karamohamed, B. Pettersson, *et al.*, "Real-time DNA sequencing using detection of pyrophosphate release," *Analytical biochemistry*, vol. 242, no. 1, pp. 84–89, 1996.

[19] E. Kawashima, L. Farinelli, and P. Mayer, "Method of nucleic acid amplification," pat. WO1998044151A1, Apr. 1998.

[20] S. Brenner, M. Johnson, J. Bridgham, *et al.*, "Gene expression analysis by massively parallel signature sequencing (MPSS) on microbead arrays," *Nature biotechnology*, vol. 18, no. 6, pp. 630–634, 2000.

[21] A. Grada and K. Weinbrecht, "Next-generation sequencing: Methodology and application," *The Journal of investigative dermatology*, vol. 133, no. 8, e11, 2013.

[22] J. Straiton, T. Free, A. Sawyer, *et al.*, "From Sanger sequencing to genome databases and beyond," *BioTechniques*, vol. 66, no. 2, pp. 60–63, 2019.

[23] F. S. Collins, M. Morgan, and A. Patrinos, "The Human Genome Project: Lessons from large-scale biology," *Science*, vol. 300, no. 5617, pp. 286–290, 2003.

[24] S. Behjati and P. S. Tarpey, "What is next generation sequencing?" *Archives of Disease in Childhood-Education and Practice*, vol. 98, no. 6, pp. 236–238, 2013.

[25] C.-S. Chin, D. H. Alexander, P. Marks, *et al.*, "Nonhybrid, finished microbial genome assemblies from long-read SMRT sequencing data," *Nature methods*, vol. 10, no. 6, pp. 563–569, 2013.

[26] D. A. Rasko, D. R. Webster, J. W. Sahl, *et al.*, "Origins of the *E. coli* strain causing an outbreak of hemolytic–uremic syndrome in Germany," *New England Journal of Medicine*, vol. 365, no. 8, pp. 709–717, 2011.

[27] B. Tran, A. M. Brown, P. L. Bedard, *et al.*, "Feasibility of real time next generation sequencing of cancer genes linked to drug response: Results from a clinical trial," *International journal of cancer*, vol. 132, no. 7, pp. 1547–1555, 2013.

[28] A. H. Van Vliet, "Next generation sequencing of microbial transcriptomes: Challenges and opportunities," *FEMS microbiology letters*, vol. 302, no. 1, pp. 1–7, 2010.

[29] A. Payne, N. Holmes, V. Rakyan, *et al.*, "BulkVis: A graphical viewer for Oxford nanopore bulk FAST5 files," *Bioinformatics*, vol. 35, no. 13, pp. 2193–2198, 2018.

[30] P. J. A. Cock, C. J. Fields, N. Goto, *et al.*, "The Sanger FASTQ file format for sequences with quality scores, and the Solexa/Illumina FASTQ variants," *Nucleic Acids Research*, vol. 38, pp. 1767–1771, 2009.

[31] D. J. Lipman and W. R. Pearson, "Rapid and sensitive protein similarity searches," *Briefings in Bioinformatics*, vol. 227, no. 4693, pp. 1435–1441, 1985.

[32] H. Li, B. Handsaker, A. Wysoker, *et al.*, "The sequence alignment/map format and SAMtools," *Bioinformatics*, vol. 25, no. 16, pp. 2078–2079, 2009.

[33] C. Alberti, M. Mattavelli, A. Hernandez, *et al.*, "Investigation on genomic information compression and storage," *ISO/IEC JTC 1/SC 29/WG 11 N15346*, pp. 1–28, 2015.

[34] T. S. F. S. W. Group, "The sequence alignment/map format specification," pp. 1–16, 2015.

[35] K. Sayood, *Introduction to data compression*, 5th edition. Morgan Kaufmann, 2017.

[36] J. K. Bonfield and M. V. Mahoney, "Compression of FASTQ and SAM format sequencing data," *PLoS ONE*, vol. 8, no. 3, e59190, 2013.

[37] Z. Zhu, Y. Zhang, Z. Ji, *et al.*, "High-throughput DNA sequence data compression," *Briefings in Bioinformatics*, vol. 16, no. 1, pp. 1–15, 2013.

[38] D. Huffman, "A method for the construction of minimum redundancy codes," in *Proceedings of the IRE*, vol. 40, 1952, pp. 1098–1101.

[39] S. W. Golomb, "Run-length encodings," *IEEE Transactions on Information Theory*, vol. 12, no. 3, pp. 399–401, 1966.

[40] C. E. Shannon, "A mathematical theory of communication," *Bell system technical journal*, vol. 27, no. 3, pp. 379–423, 1948.

[41] J. Rissanen and G. G. Langdon, "Arithmetic coding," *IBM Journal of research and development*, vol. 23, no. 2, pp. 149–162, 1979.

[42] G. P. Patrinos, *Applied Genomics and Public Health*. Elsevier, 2020.

[43] K. H. Chuong, D. R. Mack, A. Stintzi, *et al.*, "Human microbiome and learning healthcare systems: Integrating research and precision medicine for inflammatory bowel disease," *OMICS: A Journal of Integrative Biology*, vol. 22, no. 2, pp. 119–126, 2018.

[44] A. J. Titus, A. Flower, P. Hagerty, *et al.*, "SIG-DB: Leveraging homomorphic encryption to securely interrogate privately held genomic databases," *PLOS Computational Biology*, vol. 14, no. 9, e1006454, 2018.

[45] ISO/IEC 19772:2009, "Information technology – security techniques – authenticated encryption," 2009.

[46] K. Beckers, *Pattern and Security Requirements: Engineering-Based Establishment of Security Standards*. Springer International Publishing, 2015.

[47] J. E. Boritz, "IS practitioners' views on core concepts of information integrity," *International Journal of Accounting Information Systems*, vol. 6, no. 4, pp. 260–279, 2005.

[48] J. Andress, *The basics of information security: understanding the fundamentals of InfoSec in theory and practice*. Syngress, 2014.

[49] L. Y. Zhang, Y. Liu, C. Wang, *et al.*, "Improved known-plaintext attack to permutation-only multimedia ciphers," *Information Sciences*, vol. 430, pp. 228–239, 2018.

[50] B. A. Black, R. R. Neely, and M. Manga, "Campanian Ignimbrite volcanism, climate, and the final decline of the Neanderthals," *Geology*, vol. 43, no. 5, pp. 411–414, 2015.

[51] M. Bradtmöller, A. Pastoors, B. Weninger, *et al.*, "The repeated replacement model–rapid climate change and population dynamics in Late Pleistocene Europe," *Quaternary International*, vol. 247, pp. 38–49, 2012.

[52] D. Wolf, T. Kolb, M. Alcaraz-Castaño, *et al.*, "Climate deteriorations and Neanderthal demise in interior Iberia," *Scientific reports*, vol. 8, no. 1, pp. 1–10, 2018.

[53] C. Finlayson and J. S. Carrion, "Rapid ecological turnover and its impact on Neanderthal and other human populations," *Trends in Ecology & Evolution*, vol. 22, no. 4, pp. 213–222, 2007.

[54] W. E. Banks, F. d'Errico, A. T. Peterson, *et al.*, "Neanderthal extinction by competitive exclusion," *PLoS ONE*, vol. 3, no. 12, e3972, 2008.

[55] S. Underdown, "A potential role for transmissible spongiform encephalopathies in Neanderthal extinction," *Medical hypotheses*, vol. 71, no. 1, pp. 4–7, 2008.

[56] A. P. Sullivan, M. de Manuel, T. Marques-Bonet, *et al.*, "An evolutionary medicine perspective on Neandertal extinction," *Journal of human evolution*, vol. 108, pp. 62–71, 2017.

[57] D. Reich, R. E. Green, M. Kircher, *et al.*, "Genetic history of an archaic hominin group from Denisova Cave in Siberia," *Nature*, vol. 468, no. 7327, p. 1053, 2010.

[58] K. Prüfer, F. Racimo, N. Patterson, *et al.*, "The complete genome sequence of a Neanderthal from the Altai Mountains," *Nature*, vol. 505, no. 7481, p. 43, 2014.

[59] J. R. Lupski, "Genomic rearrangements and sporadic disease," *Nature genetics*, vol. 39, no. 7, S43–S47, 2007.

[60] P. Stankiewicz and J. R. Lupski, "Genome architecture, rearrangements and genomic disorders," *Trends in Genetics*, vol. 18, no. 2, pp. 74–82, 2002.

[61] J. M. Levsky and R. H. Singer, "Fluorescence in situ hybridization: Past, present and future," *Journal of cell science*, vol. 116, no. 14, pp. 2833–2838, 2003.

[62] A. Zielezinski, H. Z. Girgis, G. Bernard, *et al.*, "Benchmarking of alignment-free sequence comparison methods," *Genome Biology*, vol. 20, no. 1, p. 144, 2019.

[63] J. Xiong, *Essential bioinformatics*. Cambridge University Press, 2006.

[64] A. Zielezinski, S. Vinga, J. Almeida, *et al.*, "Alignment-free sequence comparison: Benefits, applications, and tools," *Genome biology*, vol. 18, no. 1, p. 186, 2017.

[65] S. Vinga, "Information theory applications for biological sequence analysis," *Briefings in bioinformatics*, vol. 15, no. 3, pp. 376–389, 2013.

[66] A. J. Pinho, S. P. Garcia, D. Pratas, *et al.*, "DNA sequences at a glance," *PLoS ONE*, vol. 8, no. 11, e79922, 2013.

[67] Y. Gao and L. Luo, "Genome-based phylogeny of dsDNA viruses by a novel alignment-free method," *Gene*, vol. 492, no. 1, pp. 309–314, 2012.

[68] Z. Liu, J. Meng, and X. Sun, "A novel feature-based method for whole genome phylogenetic analysis without alignment: Application to HEV genotyping and subtyping," *Biochemical and biophysical research communications*, vol. 368, no. 2, pp. 223–230, 2008.

[69] G. E. Sims and S.-H. Kim, "Whole-genome phylogeny of *Escherichia coli*/*Shigella* group by feature frequency profiles (FFPs)," *Proceedings of the National Academy of Sciences*, vol. 108, no. 20, pp. 8329–8334, 2011.

[70] H. Wang, Z. Xu, L. Gao, *et al.*, "A fungal phylogeny based on 82 complete genomes using the composition vector method," *BMC evolutionary biology*, vol. 9, no. 1, p. 195, 2009.

[71] P. Kolekar, M. Kale, and U. Kulkarni-Kale, "Alignment-free distance measure based on return time distribution for sequence analysis: Applications to clustering, molecular phylogeny and subtyping," *Molecular phylogenetics and evolution*, vol. 65, no. 2, pp. 510–522, 2012.

[72] K. Hatje and M. Kollmar, "A phylogenetic analysis of the brassicales clade based on an alignment-free sequence comparison method," *Frontiers in plant science*, vol. 3, p. 192, 2012.

[73] C.-A. Leimeister, M. Boden, S. Horwege, *et al.*, "Fast alignment-free sequence comparison using spaced-word frequencies," *Bioinformatics*, vol. 30, no. 14, pp. 1991–1999, 2014.

[74] G. Reinert, D. Chew, F. Sun, *et al.*, "Alignment-free sequence comparison (I): Statistics and power," *Journal of Computational Biology*, vol. 16, no. 12, pp. 1615–1634, 2009.

[75] B. D. Ondov, T. J. Treangen, P. Melsted, *et al.*, "Mash: Fast genome and metagenome distance estimation using MinHash," *Genome biology*, vol. 17, no. 1, p. 132, 2016.

[76] R. Bromberg, N. V. Grishin, and Z. Otwinowski, "Phylogeny reconstruction with alignment-free method that corrects for horizontal gene transfer," *PLOS Computational Biology*, vol. 12, no. 6, e1004985, 2016.

[77] S. Sarmashghi, K. Bohmann, M. T. P. Gilbert, *et al.*, "Skmer: Assembly-free and alignment-free sample identification using genome skims," *Genome biology*, vol. 20, no. 1, p. 34, 2019.

[78] S. Röhling, A. Linne, J. Schellhorn, *et al.*, "The number of k-mer matches between two DNA sequences as a function of k and applications to estimate phylogenetic distances," *PLoS ONE*, vol. 15, no. 2, e0228070, 2020.

[79] I. Ulitsky, D. Burstein, T. Tuller, *et al.*, "The average common substring approach to phylogenomic reconstruction," *Journal of Computational Biology*, vol. 13, no. 2, pp. 336–350, 2006.

[80] B. Haubold, N. Pierstorff, F. Möller, *et al.*, "Genome comparison without alignment using shortest unique substrings," *BMC bioinformatics*, vol. 6, no. 1, p. 123, 2005.

[81] A. J. Pinho, P. J. S. G. Ferreira, S. P. Garcia, *et al.*, "On finding minimal absent words," *BMC bioinformatics*, vol. 10, no. 1, p. 137, 2009.

[82] L. Yang, X. Zhang, T. Wang, *et al.*, "Large local analysis of the unaligned genome and its application," *Journal of Computational Biology*, vol. 20, no. 1, pp. 19–29, 2013.

[83] J. Wen and Y. Zhang, "A 2D graphical representation of protein sequence and its numerical characterization," *Chemical Physics Letters*, vol. 476, no. 4-6, pp. 281–286, 2009.

[84] J. S. Almeida, "Sequence analysis by iterated maps, a review," *Briefings in bioinformatics*, vol. 15, no. 3, pp. 369–375, 2013.

[85] M. Hosseini, D. Pratas, B. Morgenstern, *et al.*, "Smash++: An alignment-free and memory-efficient tool to find genomic rearrangements," *GigaScience*, vol. 9, no. 5, giaa048, 2020.

[86] M. Hosseini, D. Pratas, and A. J. Pinho, "AC: A compression tool for amino acid sequences," *Interdisciplinary Sciences: Computational Life Sciences*, vol. 11, no. 1, pp. 68–76, 2019.

[87] D. Pratas, M. Hosseini, J. M. Silva, *et al.*, "A reference-free lossless compression algorithm for DNA sequences using a competitive prediction of two classes of weighted models," *Entropy*, vol. 21, no. 11, p. 1074, 2019.

[88] M. Hosseini, D. Pratas, and A. J. Pinho, "A probabilistic method to find and visualize distinct regions in protein sequences," in *The 27th European Signal Processing Conference* (*EUSIPCO*), IEEE, 2019, pp. 1–5.

[89] D. Pratas, M. Hosseini, and A. J. Pinho, "Visualization of similar primer and adapter sequences in assembled archaeal genomes," in *The 13th International Conference on Practical Applications of Computational Biology & Bioinformatics* (*PACBB*), Springer, 2019, pp. 129–136.

[90] ——, "GeCo2: An optimized tool for lossless compression and analysis of DNA sequences," in *The 13th International Conference on Practical Applications of Computational Biology & Bioinformatics* (*PACBB*), Springer, 2019, pp. 137–145.

[91] M. Hosseini, D. Pratas, and A. J. Pinho, "Clustering DNA sequences by relative compression," in *The 25th Portuguese Conference on Pattern Recognition* (*RECPAD*), Oct. 2019.

[92] M. Hosseini, D. Pratas, A. Amorim, *et al.*, "Improving the detection of mtDNA rearrangements using a fast and accurate algorithm," in *XV Encontro Nacional de Biologia Evolutiva*, Nov. 2019.

[93] M. Hosseini, D. Pratas, and A. J. Pinho, "Cryfa: A secure encryption tool for genomic data," *Bioinformatics*, vol. 35, no. 1, pp. 146–148, 2018.

[94] D. Pratas, M. Hosseini, G. Grilo, *et al.*, "Metagenomic composition analysis of an ancient sequenced polar bear jawbone from Svalbard," *genes*, vol. 9, no. 9, p. 445, 2018.

[95] D. Pratas, M. Hosseini, and A. J. Pinho, "Compression of amino acid sequences," in *The 12th International Conference on Practical Applications of Computational Biology & Bioinformatics* (*PACBB*), Springer, 2018, pp. 105–113.

[96] M. Hosseini, D. Pratas, and A. J. Pinho, "On the role of inverted repeats in DNA sequence similarity," in *The 11th International Conference on Practical Applications of Computational Biology & Bioinformatics* (*PACBB*), Springer, 2017, pp. 228–236.

[97] D. Pratas, M. Hosseini, and A. J. Pinho, "Substitutional tolerant Markov models for relative compression of DNA sequences," in *The 11th International Conference on Practical Applications of Computational Biology & Bioinformatics* (*PACBB*), Springer, 2017, pp. 265–272.

[98] ——, "Cryfa: A tool to compact and encrypt FASTA files," in *The 11th International Conference on Practical Applications of Computational Biology & Bioinformatics* (*PACBB*), Springer, 2017, pp. 305–312.

[99] D. Pratas, M. Hosseini, R. M. Silva, *et al.*, "Visualization of distinct DNA regions of the modern human relatively to a Neanderthal genome," in *The 8th Iberian Conference on Pattern Recognition and Image Analysis* (*IbPRIA*), Springer, 2017, pp. 235–242.

[100] M. Hosseini, D. Pratas, and A. J. Pinho, "A survey on data compression methods for biological sequences," *information*, vol. 7, no. 4, p. 56, 2016.

[101] G. J. Chaitin, "On the length of programs for computing finite binary sequences," *Journal of the ACM*, vol. 13, no. 4, pp. 547–569, 1966.

[102] A. Kolmogorov, "Three approaches to the quantitative definition of information," *Problems of Information Transmission*, vol. 1, no. 1, pp. 1–7, 1965.

[103] A. Turing, "On computable numbers, with an application to the Entscheidungsproblem," *Proceedings of the London Mathematical Society*, vol. 42, no. 2, pp. 230–265, 1936.

[104] M. Li and P. M. B. Vitányi, *An introduction to Kolmogorov complexity and its applications*, 4th edition. Springer, 2019.

[105] A. K. Zvonkin and L. A. Levin, "The complexity of finite objects and the development of the concepts of information and randomness by means of the theory of algorithms," *Russian Mathematical Surveys*, vol. 25, no. 6, pp. 83–124, 1970.

[106] D. Hammer, A. Romashchenko, A. Shen, *et al.*, "Inequalities for Shannon entropy and Kolmogorov complexity," *Journal of Computer and System Sciences*, vol. 60, no. 2, pp. 442–464, 2000.

[107] C. H. Bennett, P. Gács, P. M. B. Vitányi, *et al.*, "Information distance," *IEEE Transactions on Information Theory*, vol. 44, no. 4, pp. 1407–1423, 1998.

[108] M. Li, X. Chen, X. Li, *et al.*, "The similarity metric," *IEEE Transactions on Information Theory*, vol. 50, no. 12, pp. 3250–3264, 2004.

[109] R. Cilibrasi and P. M. B. Vitányi, "Clustering by compression," *IEEE Transactions on Information Theory*, vol. 51, no. 4, pp. 1523–1545, 2005.

[110] M. Cebrián, M. Alfonseca, and A. Ortega, "Common pitfalls using the normalized compression distance: What to watch out for in a compressor," *Communications in Information and Systems*, vol. 5, no. 4, pp. 367–384, 2005.

[111] D. Pratas, "Compression and analysis of genomic data," Ph.D. dissertation, Universidade de Aveiro (Portugal), 2016.

[112] N. Nikvand and Z. Wang, "Generic image similarity based on Kolmogorov complexity," in *IEEE International Conference on Image Processing*, 2010, pp. 309–312.

[113] S. Vickers, "Localic completion of generalized metric spaces i," *Theory and Applications of Categories*, vol. 14, no. 15, pp. 328–356, 2005.

[114] Q. Xia, "The geodesic problem in quasimetric spaces," *Journal of Geometric Analysis*, vol. 19, no. 2, pp. 452–479, 2009.

[115] D. Pratas, R. Silva, and A. J. Pinho, "Comparison of compression-based measures with application to the evolution of primate genomes," *Entropy*, vol. 20, no. 6, p. 393, 2018.

[116] A. J. Pinho and D. Pratas, "MFCompress: a compression tool for FASTA and multi-FASTA data," *Bioinformatics*, vol. 30, no. 1, pp. 117–118, 2013.

[117] A. J. Pinho, P. J. S. G. Ferreira, A. J. R. Neves, *et al.*, "On the representability of complete genomes by multiple competing finite-context (Markov) models," *PLoS ONE*, vol. 6, no. 6, e21588, 2011.

[118] E. D. Jarvis, S. Mirarab, A. J. Aberer, *et al.*, "Whole-genome analyses resolve early branches in the tree of life of modern birds," *Science*, vol. 346, no. 6215, pp. 1320–1331, 2014.

[119] M. Wink, P. Heidrich, and C. Fentzloff, "A mtDNA phylogeny of sea eagles (genus *haliaeetus*) based on nucleotide sequences of the cytochrome *b*-gene," *Biochemical Systematics and Ecology*, vol. 24, no. 7-8, pp. 783–791, 1996.

[120] J. Prado-Martinez, P. H. Sudmant, J. M. Kidd, *et al.*, "Great ape genetic diversity and population history," *Nature*, vol. 499, no. 7459, p. 471, 2013.

[121] T. M. G. Sequencing, K. C. Worley, W. C. Warren, *et al.*, "The common marmoset genome provides insight into primate biology and evolution," *Nature genetics*, vol. 46, no. 8, p. 850, 2014.

[122] A. Lesk, *Introduction to bioinformatics*. Oxford University Press, 2013.

[123] J. Lee, K. Han, T. J. Meyer, *et al.*, "Chromosomal inversions between human and chimpanzee lineages caused by retrotransposons," *PLoS ONE*, vol. 3, no. 12, e4047, 2008.

[124] A. J. Pinho, A. J. R. Neves, and P. J. S. G. Ferreira, "Inverted-repeats-aware finite-context models for DNA coding," in *The 16th European Signal Processing Conference*, 2008, pp. 1–5.

[125] J. W. Ijdo, A. Baldini, D. C. Ward, *et al.*, "Origin of human chromosome 2: An ancestral telomere-telomere fusion," *Proceedings of the National Academy of Sciences*, vol. 88, no. 20, pp. 9051–9055, 1991.

[126] J. F. Hughes, H. Skaletsky, T. Pyntikova, *et al.*, "Chimpanzee and human Y chromosomes are remarkably divergent in structure and gene content," *Nature*, vol. 463, no. 7280, pp. 536–539, 2010.

[127] H. Kehrer-Sawatzki, C. Sandig, N. Chuzhanova, *et al.*, "Breakpoint analysis of the pericentric inversion distinguishing human chromosome 4 from the homologous chromosome in the chimpanzee (*Pan troglodytes*)," *Human Mutation*, vol. 25, no. 1, pp. 45–55, 2005.

[128] T. Mikkelsen, L. Hillier, E. Eichler, *et al.*, "Initial sequence of the chimpanzee genome and comparison with the human genome," *Nature*, vol. 437, no. 7055, 2005.

[129] D. Bachtrog, "Y-chromosome evolution: Emerging insights into processes of Y-chromosome degeneration," *Nature Reviews Genetics*, vol. 14, no. 2, pp. 113–124, 2013.

[130] D. Pratas and A. J. Pinho, "A conditional compression distance that unveils insights of the genomic evolution," in *Data Compression Conference* (*DCC*), 2014, p. 421.

[131] D. Pratas, R. M. Silva, A. J. Pinho, *et al.*, "An alignment-free method to find and visualise rearrangements between pairs of DNA sequences," *Scientific reports*, vol. 5, no. 1, pp. 1–9, 2015.

[132] R. V. Samonte and E. E. Eichler, "Segmental duplications and the evolution of the primate genome," *Nature Reviews Genetics*, vol. 3, no. 1, pp. 65–72, 2002.

[133] R. A. Dalloul, J. A. Long, A. V. Zimin, *et al.*, "Multi-platform next-generation sequencing of the domestic turkey (*Meleagris gallopavo*): Genome assembly and analysis," *PLoS Biology*, vol. 8, no. 9, e1000475, 2010.

[134] P. Muir, S. Li, S. Lou, *et al.*, "The real cost of sequencing: Scaling computation to keep pace with data generation," *Genome Biology*, vol. 17, no. 1, pp. 1–9, 2016.

[135] S. D. Kahn, "On the future of genomic data," *Science*, vol. 331, no. 6018, pp. 728–729, 2011.

[136] R. Giancarlo, S. E. Rombo, and F. Utro, "Compressive biological sequence analysis and archival in the era of high-throughput sequencing technologies," *Briefings in Bioinformatics*, vol. 15, no. 3, pp. 390–406, 2014.

[137] E. R. Dougherty, *Genomic signal processing and statistics*. Hindawi Publishing Corporation, 2005, vol. 2.

[138] S. Grumbach and F. Tahi, "Compression of DNA sequences," in *Data Compression Conference* (*DCC*), 1993, pp. 340–350.

[139] E. Rivals, J. P. Delahaye, M. Dauchet, *et al.*, "A guaranteed compression scheme for repetitive dna sequences," in *Data Compression Conference*(*DCC*), 1996, p. 453.

[140] T. Matsumoto, K. Sadakane, and H. Imai, "Biological sequence compression algorithms," *Genome Informatics*, vol. 11, pp. 43–52, 2000.

[141] X. Chen, S. Kwong, M. Li, *et al.*, "A compression algorithm for DNA sequences and its applications in genome comparison," in *The 4th Annual International Conference of Research in Computational Molecular Biology* (*RECOMB*), 2000, pp. 107–117.

[142] X. Chen, M. Li, and B. Ma, "DNACompress: Fast and effective DNA sequence," *Bioinformatics*, vol. 18, no. 12, pp. 1696–1698, 2002.

[143] I. Tabus, G. Korodi, and J. Rissanen, "DNA sequence compression using the normalized maximum likelihood model for discrete regression," in *Data Compression Conference* (*DCC*), 2003, pp. 253–262.

[144] G. Korodi and I. Tabus, "An efficient normalized maximum likelihood algorithm for DNA sequence compression," *ACM Transactions on Information Systems*, vol. 23, no. 1, pp. 3–34, 2005.

[145] M. D. Cao, T. I. Dix, L. Allison, *et al.*, "A simple statistical algorithm for biological sequence compression," in *Data Compression Conference* (*DCC*), 2007, pp. 43–52.

[146] A. Gupta and S. Agarwal, "A novel approach for compressing DNA sequences using semi-statistical compressor," *International Journal of Computers and Applications*, vol. 33, no. 3, pp. 245–251, 2011.

[147] Z. Zhu, J. Zhou, Z. Ji, *et al.*, "DNA sequence compression using adaptive particle swarm optimization-based memetic algorithm," *IEEE Transactions on Evolutionary Computation*, vol. 15, no. 5, pp. 643–658, 2011.

[148] P. Li, S. Wang, J. Kim, *et al.*, "DNA-COMPACT: DNA compression based on a pattern-aware contextual modeling technique," *PLoS ONE*, vol. 8, no. 11, e80377, 2013.

[149] H. Guo, M. Chen, X. Liu, *et al.*, "Genome compression based on Hilbert space filling curve," in *The 3rd International Conference on Management, Education, Information and Control* (*MEICI*), 2015, pp. 1685–1689.

[150] X. Xie, S. Zhou, and J. Guan, "CoGI: Towards compressing genomes as an image," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 12, no. 6, pp. 1275–1285, 2015.

[151] M. Chen, J. Chen, Y. Zhang, *et al.*, "Optimized context weighting based on the least square algorithm," *Lecture Notes in Electrical Engineering Springer*, vol. 348, pp. 1037–1045, 2016.

[152] D. Pratas, A. J. Pinho, and P. J. S. G. Ferreira, "Efficient compression of genomic sequences," in *Data Compression Conference* (*DCC*), 2016, pp. 231–240.

[153] A. A. Ginart, J. Hui, K. Zhu, *et al.*, "Optimal compressed representation of high throughput sequence data via light assembly," *Nature communications*, vol. 9, no. 1, p. 566, 2018.

[154] N. Ramezanipoor and M. Yaghoobi, "A new approach in DNA sequence compression: Fast DNA sequence compression using parallel chaos game representation," *Expert Systems with Applications*, vol. 116, pp. 487–493, 2019.

[155] S. Chandak, K. Tatwawadi, I. Ochoa, *et al.*, "SPRING: A next-generation compressor for FASTQ data," *Bioinformatics*, vol. 35, no. 15, pp. 2674–2676, 2019.

[156] S. Deorowicz, "FQSqueezer: *k*-mer-based compression of sequencing data," *Scientific Reports*, vol. 10, no. 1, p. 578, 2020.

[157] S. Christley, Y. Lu, C. Li, *et al.*, "Human genomes as email attachments," *Bioinformatics*, vol. 25, no. 2, pp. 274–275, 2009.

[158] S. Kuruppu, S. J. Puglisi, and J. Zobel, "Relative Lempel-Ziv compression of genomes for large-scale storage and retrieval," in *International Symposium on String Processing and Information Retrieval*, Springer, 2010, pp. 201–206.

[159] ——, "Optimized relative Lempel-Ziv compression of genomes," in *The 34th Australasian Computer Science Conference*, 2011, pp. 91–98.

[160] C. Wang and D. Zhang, "A novel compression tool for efficient storage of genome resequencing data," *Nucleic Acids Research*, vol. 39, no. 7, e45–e45, 2011.

[161] S. Deorowicz and S. Grabowski, "Robust relative compression of genomes with random access," *Bioinformatics*, vol. 27, no. 21, pp. 2979–2986, 2011.

[162] S. Wandelt and U. Leser, "Adaptive efficient compression of genomes," *Algorithms for Molecular Biology*, vol. 7, no. 1, p. 30, 2012.

[163] A. J. Pinho, D. Pratas, and S. P. Garcia, "GReEn: A tool for efficient compression of genome resequencing data," *Nucleic Acids Research*, vol. 40, no. 4, e27–e27, 2012.

[164] S. Kuruppu, B. Beresford-Smith, T. Conway, *et al.*, "Iterative dictionary construction for compression of large DNA data sets," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 9, no. 1, pp. 137–149, 2012.

[165] W. Dai, H. Xiong, X. Jiang, *et al.*, "An adaptive difference distribution-based coding with hierarchical tree structure for DNA sequence compression," in *Data Compression Conference* (*DCC*), 2013, pp. 371–380.

[166] S. Wandelt and U. Leser, "FRESCO: Referential compression of highly-similar sequences," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 10, no. 5, pp. 1275–1288, 2013.

[167] S. Jung and I. Sohn, "Streamlined genome sequence compression using distributed source coding," *Cancer Informatics*, vol. 13, pp. 123–131, 2014.

[168] S. Deorowicz, A. Danek, and M. Niemiec, "GDC 2: Compression of large collections of genomes," *Scientific reports*, vol. 5, p. 11 565, 2015.

[169] S. Saha and S. Rajasekaran, "ERGC: An efficient referential genome compression algorithm," *Bioinformatics*, vol. 31, no. 21, pp. 3468–3475, 2015.

[170] I. Ochoa, M. Hernaez, and T. Weissman, "iDoComp: A compression scheme for assembled genomes," *Bioinformatics*, vol. 31, no. 5, pp. 626–633, 2015.

[171] Z.-A. Huang, Z. Wen, Q. Deng, *et al.*, "LW-FQZip 2: A parallelized reference-based compression of FASTQ files," *BMC bioinformatics*, vol. 18, no. 1, p. 179, 2017.

[172] Y. Liu, H. Peng, L. Wong, *et al.*, "High-speed and high-ratio referential genome compression," *Bioinformatics*, vol. 33, no. 21, pp. 3364–3372, 2017.

[173] N. Jammula and S. Aluru, "ParRefCom: Parallel reference-based compression of paired-end genomics read datasets," in *The 10th ACM International Conference on Bioinformatics, Computational Biology and Health Informatics* (*ACM BCB*), 2019, pp. 447–456.

[174] Ł. Roguski, I. Ochoa, M. Hernaez, *et al.*, "FaStore: A space-saving solution for raw sequencing data," *Bioinformatics*, vol. 34, no. 16, pp. 2748–2756, 2018.

[175] R. Giancarlo, D. Scaturro, and F. Utro, "Textual data compression in computational biology: Algorithmic techniques," *Computer Science Review*, vol. 6, no. 1, pp. 1–25, 2012.

[176] J. Ziv and A. Lempel, "A universal algorithm for sequential data compression," *IEEE Transactions on Information Theory*, vol. 23, no. 3, pp. 337–343, 1977.

[177] J. J. Liang, A. K. Qin, P. N. Suganthan, *et al.*, "Comprehensive learning particle swarm optimizer for global optimization of multimodal functions," *IEEE Transactions on Evolutionary Computation*, vol. 10, no. 3, pp. 281–295, 2006.

[178] S. A. Mohamed and M. Fahmy, "Binary image compression using efficient partitioning into rectangular regions," *IEEE Transactions on Communications*, vol. 43, no. 5, pp. 1888–1893, 1995.

[179] S. Chandak, K. Tatwawadi, and T. Weissman, "Compression of genomic sequencing reads via hash-based reordering: Algorithm and analysis," *Bioinformatics*, vol. 34, no. 4, pp. 558–567, 2017.

[180] G. Malysa, M. Hernaez, I. Ochoa, *et al.*, "QVZ: Lossy compression of quality values," *Bioinformatics*, vol. 31, no. 19, pp. 3122–3129, 2015.

[181] R. Grossi and J. S. Vitter, "Compressed suffix arrays and suffix trees with applications to text indexing and string matching," in *32nd ACM Symposium on Theory of Computing*, 2000, pp. 397–406.

[182] D. Pratas and A. J. Pinho, "A DNA sequence corpus for compression benchmark," in *International Conference on Practical Applications of Computational Biology & Bioinformatics* (*PACBB*), Springer, 2018, pp. 208–215.

[183] A. Moffat, R. M. Neal, and I. H. Witten, "Arithmetic coding revisited," *ACM Transactions on Information Systems*, vol. 16, no. 3, pp. 256–294, 1998.

[184] S. M. Rafizul Haque, T. Mallick, and I. S. Kabir, "A new approach of protein sequence compression using repeat reduction and ASCII replacement," *IOSR Journal of Computer Engineering*, vol. 10, no. 5, pp. 46–51, 2013.

[185] M. Ward, *Virtual organisms: The startling world of artificial life*. Macmillan, 2014.

[186] M. S. Baker, S. B. Ahn, A. Mohamedali, *et al.*, "Accelerating the search for the missing proteins in the human proteome," *Nature communications*, vol. 8, no. 1, pp. 1–13, 2017.

[187] U. Eckhard, G. Marino, G. S. Butler, *et al.*, "Positional proteomics in the era of the human proteome project on the doorstep of precision medicine," *Biochimie*, vol. 122, pp. 110–118, 2016.

[188] P. Legrain, R. Aebersold, A. Archakov, *et al.*, "The human proteome project: Current state and future direction," *Molecular & cellular proteomics*, vol. 10, no. 7, 2011.

[189] Y.-K. Paik, S.-K. Jeong, G. S. Omenn, *et al.*, "The chromosome-centric human proteome project for cataloging proteins encoded in the genome," *Nature biotechnology*, vol. 30, no. 3, p. 221, 2012.

[190] U. Consortium, "UniProt: The universal protein knowledgebase," *Nucleic acids research*, vol. 46, no. 5, p. 2699, 2018.

[191] I.-I. Comm, "A one-letter notation for amino acid sequences. tentative rules," *Biochemistry*, vol. 7, no. 8, pp. 2703–2705, 1968.

[192] J. C. Wootton, "Non-globular domains in protein sequences: Automated segmentation using complexity measures," *Computers & Chemistry*, vol. 18, no. 3, pp. 269–285, 1994.

[193] D. Benedetto, E. Caglioti, and C. Chica, "Compressing proteomes: The relevance of medium range correlations," *EURASIP Journal on Bioinformatics and Systems Biology*, vol. 2007, no. 1, p. 60 723, 2007.

[194] J.-F. Yu, Z. Cao, Y. Yang, *et al.*, "Natural protein sequences are more intrinsically disordered than random sequences," *Cellular and Molecular Life Sciences*, vol. 73, no. 15, pp. 2949–2957, 2016.

[195] Ö. U. Nalbantoglu, D. J. Russell, and K. Sayood, "Data compression concepts and algorithms and their applications to bioinformatics," *Entropy*, vol. 12, no. 1, pp. 34–52, 2009.

[196] C. G. Nevill-Manning and I. H. Witten, "Protein is incompressible," in *Data Compression Conference* (*DCC*), 1999, pp. 257–266.

[197] A. Hategan and I. Tabus, "Protein is compressible," in *The 6th Nordic Signal Processing Symposium* (*NORSIG*), 2004, pp. 192–195.

[198] A. Hategan and I. Tabus, "Jointly encoding protein sequences and their secondary structure," in *IEEE International Workshop on Genomic Signal Processing and Statistics* (*GENSIPS*), 2007, pp. 1–4.

[199] N. M. Daniels, A. Gallant, J. Peng, *et al.*, "Compressive genomics for protein databases," *Bioinformatics*, vol. 29, no. 13, pp. i283–i290, 2013.

[200] M. Hayashida, P. Ruan, and T. Akutsu, "Proteome compression via protein domain compositions," *Methods*, vol. 67, no. 3, pp. 380–385, 2014.

[201] F. M. J. Willems, Y. M. Shtarkov, and T. J. Tjalkens, "The context tree weighting method: Basic properties," *IEEE Transactions on Information Theory*, vol. 41, no. 3, pp. 653–664, 1995.

[202] W. Kabsch and C. Sander, "Dictionary of protein secondary structure: Pattern recognition of hydrogen-bonded and geometrical features," *Biopolymers*, vol. 22, no. 12, pp. 2577–2637, 1983.

[203] B. Korber, R. M. Farber, D. H. Wolpert, *et al.*, "Covariation of mutations in the V3 loop of human immunodeficiency virus type 1 envelope protein: An information theoretic analysis," *Proceedings of the National Academy of Sciences*, vol. 90, no. 15, pp. 7176–7180, 1993.

[204] F. Pereira, S. Duarte-Pereira, R. M. Silva, *et al.*, "Evolution of the NET (NocA, Nlz, Elbow, TLP-1) protein family in metazoans: Insights from expression data and phylogenetic analysis," *Scientific reports*, vol. 6, p. 38 383, 2016.

[205] D. A. Pelta, J. R. Gonzalez, and N. Krasnogor, "Protein structure comparison through fuzzy contact maps and the universal similarity metric," in *The 4th Conference of the European Society for Fuzzy Logic and Technology* (*EUSFLAT*), 2005, pp. 1124–1129.

[206] J. Rocha, F. Rosselló, and J. Segura, "Compression ratios based on the universal similarity metric still yield protein distances far from CATH distances," *arXiv preprint q-bio/0603007*, 2006.

[207] R. P. Bywater, "Prediction of protein structural features from sequence data based on Shannon entropy and Kolmogorov complexity," *PLoS ONE*, vol. 10, no. 4, e0119306, 2015.

[208] D. Pratas and A. J. Pinho, "On the approximation of the Kolmogorov complexity for DNA sequences," in *Iberian Conference on Pattern Recognition and Image Analysis* (*IbPRIA*), 2017, pp. 259–266.

[209] C. Kumar-Sinha and A. M. Chinnaiyan, "Precision oncology in the age of integrative genomics," *Nature biotechnology*, vol. 36, no. 1, p. 46, 2018.

[210] T. M. Porter and M. Hajibabaei, "Scaling up: A guide to high-throughput genomic approaches for biodiversity analysis," *Molecular ecology*, vol. 27, no. 2, pp. 313–338, 2018.

[211] K. A. Jagadeesh, D. J. Wu, J. A. Birgmeier, *et al.*, "Deriving genomic diagnoses without revealing patient genomes," *Science*, vol. 357, no. 6352, pp. 692–695, 2017.

[212] T. Bradley, X. Ding, and G. Tsudik, "Genomic security (lest we forget)," *IEEE Security and Privacy Magazine*, vol. 15, no. 5, pp. 38–46, 2017.

[213] C. Bouillaguet, P. Derbez, O. Dunkelman, *et al.*, "Low-data complexity attacks on AES," *IEEE Transactions on Information Theory*, vol. 58, no. 11, pp. 7002–7017, 2012.

[214] J. Daemen and V. Rijmen, "The design of Rijndael: AES — the Advanced Encryption Standard," in *Information Security and Cryptography*, Springer, 2002.

[215] J. Chen, Y. Feng, and I. Dillig, "Precise detection of side-channel vulnerabilities using quantitative cartesian hoare logic," in *ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 875–890.

[216] J. Xie, K. Qian, J. Si, *et al.*, "Conserved noncoding sequences conserve biological networks and influence genome evolution," *Heredity*, vol. 120, no. 5, pp. 437–451, 2018.

[217] C. Bouillaguet, P. Derbez, O. Dunkelman, *et al.*, "Low-data complexity attacks on AES," *IEEE Transactions on Information Theory*, vol. 58, no. 11, pp. 7002–7017, 2012.

[218] S. Ahmadi, Z. Ahmadian, J. Mohajeri, *et al.*, "Low-data complexity biclique cryptanalysis of block ciphers with application to Piccolo and HIGHT," *IEEE Transactions on Information Forensics and Security*, vol. 9, no. 10, pp. 1641–1652, 2014.

[219] P. Morawiecki, "Practical attacks on the round-reduced PRINCE," *IET Information Security*, vol. 11, no. 3, pp. 146–151, 2016.

[220] Y. Zhang, X. Liu, and M. Sun, "DNA based random key generation and management for OTP encryption," *Biosystems*, vol. 159, pp. 51–63, 2017.

[221] L. Y. Zhang, Y. Liu, C. Wang, *et al.*, "Improved known-plaintext attack to permutation-only multimedia ciphers," *Information Sciences*, vol. 430, pp. 228–239, 2018.

[222] Z. Shan, K. Ren, M. Blanton, *et al.*, "Practical secure computation outsourcing: A survey," *ACM Computing Surveys*, vol. 51, no. 2, pp. 1–40, 2018.

[223] D. McGrew and J. Viega, "The Galois/counter mode of operation (GCM)," *submission to NIST Modes of Operation Process*, vol. 20, p. 10, 2004.

[224] S. Lemsitzer, J. Wolkerstorfer, N. Felber, *et al.*, "Multi-gigabit GCM-AES architecture optimized for FPGAs," in *International Workshop on Cryptographic Hardware and Embedded Systems*, 2007, pp. 227–238.

[225] K.-T. Huang, J.-H. Chiu, and S.-S. Shen, "A novel structure with dynamic operation mode for symmetric-key block ciphers," *International Journal of Network Security & Its Applications*, vol. 5, no. 1, p. 17, 2013.

[226] D. Pratas, A. J. Pinho, and J. M. O. S. Rodrigues, "XS: A FASTQ read simulator," *BMC Research Notes*, vol. 7, no. 1, p. 40, 2014.

[227] M. H. Mohammed, A. Dutta, T. Bose, *et al.*, "DELIMINATE – a fast and efficient method for loss-less compression of genomic sequences: Sequence analysis," *Bioinformatics*, vol. 28, no. 19, pp. 2527–2529, 2012.

[228] D. C. Jones, W. L. Ruzzo, X. Peng, *et al.*, "Compression of next-generation sequencing reads aided by highly efficient de novo assembly," *Nucleic Acids Research*, vol. 40, no. 22, e171–e171, 2012.

[229] Ł. Roguski and S. Deorowicz, "DSRC 2–industry-oriented compression of FASTQ files," *Bioinformatics*, vol. 30, no. 15, pp. 2213–2215, 2014.

[230] A. Dutta, M. M. Haque, T. Bose, *et al.*, "FQC: A novel approach for efficient compression, archival, and dissemination of FASTQ datasets," *Journal of bioinformatics and computational biology*, vol. 13, no. 03, p. 1 541 003, 2015.

[231] F. Soler-Toscano, H. Zenil, J.-P. Delahaye, *et al.*, "Calculating Kolmogorov complexity from the output frequency distributions of small Turing machines," *PLoS ONE*, vol. 9, no. 5, e96223, 2014.

[232] R. M. Silva, D. Pratas, L. Castro, *et al.*, "Three minimal sequences found in Ebola virus genomes and absent from human DNA," *Bioinformatics*, vol. 31, no. 15, pp. 2421–2425, 2015.

[233] M. Crochemore, A. Héliou, G. Kucherov, *et al.*, "Absent words in a sliding window with applications," *Information and Computation*, vol. 270, p. 104 461, 2020.

[234] P. Charalampopoulos, M. Crochemore, G. Fici, *et al.*, "Alignment-free sequence comparison using absent words," *Information and Computation*, vol. 262, pp. 57–68, 2018.

[235] M. Höss, P. Jaruga, T. H. Zastawny, *et al.*, "DNA damage and DNA sequence retrieval from ancient tissues," *Nucleic acids research*, vol. 24, no. 7, pp. 1304–1307, 1996.

[236] M. Hofreiter, D. Serre, H. N. Poinar, *et al.*, "Ancient DNA," *Nature Reviews Genetics*, vol. 2, no. 5, p. 353, 2001.

[237] M. Krings, A. Stone, R. W. Schmitz, *et al.*, "Neandertal DNA sequences and the origin of modern humans," *Cell*, vol. 90, no. 1, pp. 19–30, 1997.

[238] R. E. Green, J. Krause, S. E. Ptak, *et al.*, "Analysis of one million base pairs of Neanderthal DNA," *Nature*, vol. 444, no. 7117, p. 330, 2006.

[239] J. P. Noonan, G. Coop, S. Kudaravalli, *et al.*, "Sequencing and analysis of Neanderthal genomic DNA," *Science*, vol. 314, no. 5802, pp. 1113–1118, 2006.

[240] R. E. Green, A.-S. Malaspinas, J. Krause, *et al.*, "A complete Neandertal mitochondrial genome sequence determined by high-throughput sequencing," *Cell*, vol. 134, no. 3, pp. 416–426, 2008.

[241] R. E. Green, J. Krause, A. W. Briggs, *et al.*, "A draft sequence of the Neandertal genome," *Science*, vol. 328, no. 5979, pp. 710–722, 2010.

[242] Q. Fu, M. Hajdinjak, O. T. Moldovan, *et al.*, "An early modern human from Romania with a recent Neanderthal ancestor," *Nature*, vol. 524, no. 7564, p. 216, 2015.

[243] F. J. Ayala and C. J. C. Conde, *Processes in Human Evolution: The journey from early hominins to Neanderthals and modern humans*. Oxford University Press, 2017.

[244] J. Hawks, "Significance of Neandertal and Denisovan genomes in human evolution," *Annual Review of Anthropology*, vol. 42, pp. 433–449, 2013.

[245] P. Skoglund, B. H. Northoff, M. V. Shunkov, *et al.*, "Separating endogenous ancient DNA from modern day contamination in a Siberian Neandertal," *Proceedings of the National Academy of Sciences*, vol. 111, no. 6, pp. 2229–2234, 2014.

[246] M. Hofreiter, V. Jaenicke, D. Serre, *et al.*, "DNA sequences from multiple amplifications reveal artifacts induced by cytosine deamination in ancient DNA," *Nucleic acids research*, vol. 29, no. 23, pp. 4793–4799, 2001.

[247] A. W. Briggs, U. Stenzel, P. L. Johnson, *et al.*, "Patterns of damage in genomic DNA sequences from a Neandertal," *Proceedings of the National Academy of Sciences*, vol. 104, no. 37, pp. 14 616–14 621, 2007.

[248] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of the ACM*, vol. 13, no. 7, pp. 422–426, 1970.

[249] L. Luo, D. Guo, R. T. Ma, *et al.*, "Optimizing Bloom filter: Challenges, solutions, and comparisons," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 2, pp. 1912–1949, 2018.

[250] A. Goel and P. Gupta, "Small subset queries and Bloom filters using ternary associative memories, with applications," *ACM SIGMETRICS Performance Evaluation Review*, vol. 38, no. 1, pp. 143–154, 2010.

[251] Y.-L. Lin, P. Pavlidis, E. Karakoc, *et al.*, "The evolution and functional impact of human deletion variants shared with archaic hominin genomes," *Molecular biology and evolution*, vol. 32, no. 4, pp. 1008–1019, 2015.

[252] R. Qu, Q. Sang, Y. Xu, *et al.*, "Identification of a novel homozygous mutation in *myo3a* in a Chinese family with DFNB30 non-syndromic hearing impairment," *International journal of pediatric otorhinolaryngology*, vol. 84, pp. 43–47, 2016.

[253] I. M. Silva, J. Rosenfeld, S. A. Antoniuk, *et al.*, "A 1.5 Mb terminal deletion of 12p associated with autism spectrum disorder," *Gene*, vol. 542, no. 1, pp. 83–86, 2014.

[254] K. Baker, S. L. Gordon, D. Grozeva, *et al.*, "Identification of a human synaptotagmin-1 mutation that perturbs synaptic vesicle cycling," *The Journal of clinical investigation*, vol. 125, no. 4, pp. 1670–1678, 2015.

[255] M. Meyer, M. Kircher, M.-T. Gansauge, *et al.*, "A high-coverage genome sequence from an archaic Denisovan individual," *Science*, vol. 338, no. 6104, pp. 222–226, 2012.

[256] J. Hendy, F. Welker, B. Demarchi, *et al.*, "A guide to ancient protein studies," *Nature ecology & evolution*, vol. 2, no. 5, pp. 791–799, 2018.

[257] E. Cappellini, M. J. Collins, and M. T. P. Gilbert, "Unlocking ancient protein palimpsests," *Science*, vol. 343, no. 6177, pp. 1320–1322, 2014.

[258] M. G. Giuffrida, R. Mazzoli, and E. Pessione, "Back to the past: Deciphering cultural heritage secrets by protein identification," *Applied microbiology and biotechnology*, vol. 102, no. 13, pp. 5445–5455, 2018.

[259] M. Sikora, V. V. Pitulko, V. C. Sousa, *et al.*, "The population history of northeastern Siberia since the Pleistocene," *Nature*, vol. 570, no. 7760, pp. 182–188, 2019.

[260] R. Sawafuji, E. Cappellini, T. Nagaoka, *et al.*, "Proteomic profiling of archaeological human bone," *Royal Society open science*, vol. 4, no. 6, p. 161 004, 2017.

[261] M. Buckley, "Paleoproteomics: An introduction to the analysis of ancient proteins by soft ionisation mass spectrometry," in *Paleogenomics*, Springer, 2018, pp. 31–52.

[262] Y.-C. Lee, C.-C. Chiang, P.-Y. Huang, *et al.*, "Evidence of preserved collagen in an Early Jurassic sauropodomorph dinosaur revealed by synchrotron FTIR microspectroscopy," *Nature communications*, vol. 8, no. 1, pp. 1–8, 2017.

[263] E. Willerslev, A. J. Hansen, R. Rønn, *et al.*, "Long-term persistence of bacterial DNA," *Current Biology*, vol. 14, no. 1, R9–R10, 2004.

[264] S. Castellano, G. Parra, F. A. Sánchez-Quinto, *et al.*, "Patterns of coding variation in the complete exomes of three Neandertals," *Proceedings of the National Academy of Sciences*, vol. 111, no. 18, pp. 6666–6671, 2014.

[265] M. Dietzfelbinger, "Universal hashing and k-wise independent random variables via integer arithmetic without primes," in *STACS*, Springer, 1996, pp. 567–580.

[266] A. Sajantila, "Editors' pick: Contamination has always been the issue!" *Investigative Genetics*, vol. 5, p. 17, 2014.

[267] J. Dabney, M. Meyer, and S. Pääbo, "Ancient DNA damage," *Cold Spring Harbor perspectives in biology*, vol. 5, no. 7, a012567, 2013.

[268] E. W. Sayers, R. Agarwala, E. E. Bolton, *et al.*, "Database resources of the National Center for Biotechnology Information," *Nucleic acids research*, vol. 47, no. Database issue, p. D23, 2019.

[269] G. M. Boratyn, A. A. Schäffer, R. Agarwala, *et al.*, "Domain enhanced lookup time accelerated BLAST," *Biology direct*, vol. 7, no. 1, p. 12, 2012.

[270] J. Reuter, D. V. Spacek, and M. Snyder, "High-throughput sequencing technologies," *Molecular Cell*, vol. 58, no. 4, pp. 586–597, 2015.

[271] D. E. V. Villamor, T. Ho, M. Al Rwahnih, *et al.*, "High throughput sequencing for plant virus detection and discovery," *Phytopathology*, vol. 109, no. 5, pp. 716–725, 2019.

[272] S. M. Rego and M. P. Snyder, "High throughput sequencing and assessing disease risk," *Cold Spring Harbor perspectives in medicine*, vol. 9, no. 1, a026849, 2019.

[273] T. Hartmann, M. Middendorf, and M. Bernt, "Genome rearrangement analysis: Cut and join genome rearrangements and gene cluster preserving approaches," in *Comparative Genomics*, Springer, 2018, pp. 261–289.

[274] R. J. M. Gardner and D. J. Amor, *Gardner and Sutherland's Chromosome Abnormalities and Genetic Counseling*, 5th edition. Oxford University Press, 2018.

[275] A. Theisen and L. G. Shaffer, "Disorders caused by chromosome abnormalities," *The application of clinical genetics*, vol. 3, pp. 159–174, 2010.

[276] J. Damas, D. C. Samuels, J. Carneiro, *et al.*, "Mitochondrial DNA rearrangements in health and disease–a comprehensive study," *Human mutation*, vol. 35, no. 1, pp. 1–14, 2014.

[277] A. Dufke, J. Seidel, M. Schöning, *et al.*, "Microdeletion 4p16.3 in three unrelated patients with Wolf-Hirschhorn syndrome," *Cytogenetic and Genome Research*, vol. 91, no. 1-4, pp. 81–84, 2000.

[278] V. Timmerman, E. Nelis, W. Van Hul, *et al.*, "The peripheral myelin protein gene *pmp-22* is contained within the Charcot-Marie-Tooth disease type 1A duplication," *Nature genetics*, vol. 1, no. 3, p. 171, 1992.

[279] L. Huang, L. V. Abruzzo, J. R. Valbuena, *et al.*, "Acute myeloid leukemia associated with variant t(8;21) detected by conventional cytogenetic and molecular studies: A report of four cases and review of the literature," *American Journal of Clinical Pathology*, vol. 125, no. 2, pp. 267–272, 2006.

[280] A. E. Darling, B. Mau, and N. T. Perna, "progressiveMauve: Multiple genome alignment with gene gain, loss and rearrangement," *PLoS ONE*, vol. 5, no. 6, e11147, 2010.

[281] M. Brudno, S. Malde, A. Poliakov, *et al.*, "Glocal alignment: Finding rearrangements during alignment," *Bioinformatics*, vol. 19, no. suppl_1, pp. i54–i62, 2003.

[282] S. K. Pham and P. Pevzner, "DRIMM-Synteny: Decomposing genomes into evolutionary conserved segments," *Bioinformatics*, vol. 26, no. 20, pp. 2509–2516, 2010.

[283] P. Pevzner and G. Tesler, "Genome rearrangements in mammalian evolution: Lessons from human and mouse genomes," *Genome research*, vol. 13, no. 1, pp. 37–45, 2003.

[284] J. Lee, W.-y. Hong, M. Cho, *et al.*, "Synteny Portal: A web-based application portal for synteny block analysis," *Nucleic Acids Research*, vol. 44, no. W1, W35–W40, 2016.

[285] A. U. Sinha and J. Meller, "Cinteny: Flexible analysis and visualization of synteny and genome rearrangements in multiple organisms," *BMC Bioinformatics*, vol. 8, no. 1, p. 82, 2007.

[286] M. C. Frith and S. Khan, "A survey of localized sequence rearrangements in human DNA," *Nucleic acids research*, vol. 46, no. 4, pp. 1661–1673, 2017.

[287] C. B. Nielsen, M. Cantor, I. Dubchak, *et al.*, "Visualizing genomes: Techniques and challenges," *Nature methods*, vol. 7, no. 3, S5–S15, 2010.

[288] R. Morris, "Counting large numbers of events in small registers," *Communications of the ACM*, vol. 21, no. 10, pp. 840–842, 1978.

[289] G. Pitel and G. Fouquier, "Count-min-log sketch: Approximately counting with approximate counters," in *International Symposium on Web AlGorithms*, Jun. 2015.

[290] R. B. Blackman and J. W. Tukey, "Particular pairs of windows," *The measurement of power spectra, from the point of view of communications engineering*, pp. 95–101, 1959.

[291] H. Zenil, F. Soler-Toscano, J.-P. Delahaye, *et al.*, "Two-dimensional Kolmogorov complexity and an empirical validation of the Coding theorem method by compressibility," *PeerJ Computer Science*, vol. 1, e23, 2015.

[292] R. Antão, A. Mota, and J. A. T. Machado, "Kolmogorov complexity as a data similarity metric: Application in mitochondrial DNA," *Nonlinear Dynamics*, vol. 93, no. 3, pp. 1059–1071, 2018.

[293] C. Faloutsos and V. Megalooikonomou, "On data mining, compression, and Kolmogorov complexity," *Data Mining and Knowledge Discovery*, vol. 15, no. 1, pp. 3–20, 2007.

[294] S. L. Salzberg, D. D. Sommer, M. C. Schatz, *et al.*, "Genome sequence and rapid evolution of the rice pathogen *Xanthomonas oryzae* pv. *oryzae* PXO99A," *BMC Genomics*, vol. 9, no. 1, pp. 1–16, 2008.

[295] H. Ochiai, Y. Inoue, M. Takeya, *et al.*, "Genome sequence of *Xanthomonas oryzae* pv. *oryzae* suggests contribution of large numbers of effector genes and insertion sequences to its race diversity," *Japan Agricultural Research Quarterly: JARQ*, vol. 39, no. 4, pp. 275–287, 2005.

[296] S. Kumar, G. Stecher, M. Suleski, *et al.*, "TimeTree: A resource for timelines, timetrees, and divergence times," *Molecular Biology and Evolution*, vol. 34, no. 7, pp. 1812–1819, 2017.

[297] Y. Zhang, X. Zhang, T. H. O'Hare, *et al.*, "A comparative physical map reveals the pattern of chromosomal evolution between the turkey (*Meleagris gallopavo*) and chicken (*Gallus gallus*) genomes," *BMC Genomics*, vol. 12, no. 1, p. 447, 2011.

[298] F. Cabanettes and C. Klopp, "D-GENIES: Dot plot large genomes in an interactive, efficient and simple way," *PeerJ*, vol. 6, e4958, 2018.

[299] H. Li, "Minimap2: Pairwise alignment for nucleotide sequences," *Bioinformatics*, vol. 34, no. 18, pp. 3094–3100, 2018.

[300] G. Fischer, E. P. Rocha, F. Brunet, *et al.*, "Highly variable rates of genome rearrangements between hemiascomycetous yeast lineages," *PLoS Genetics*, vol. 2, no. 3, e32, 2006.

[301] G. Charron, J.-B. Leducq, C. Bertin, *et al.*, "Exploring the northern limit of the distribution of *Saccharomyces cerevisiae* and *Saccharomyces paradoxus* in North America," *FEMS yeast research*, vol. 14, no. 2, pp. 281–288, 2014.