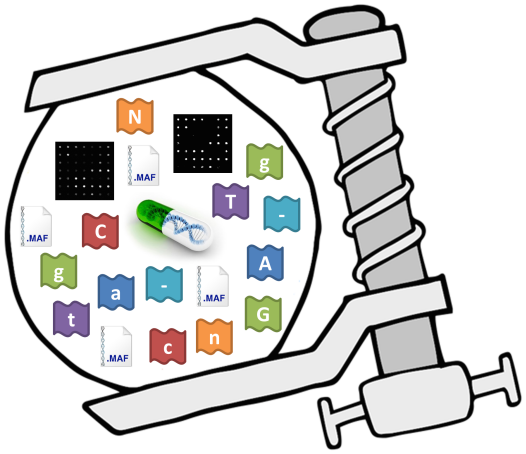
Universidade de Aveiro
2015

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

Luís Miguel de
Oliveira Matos

Algoritmos de compressão sem perdas para imagens de microarrays e alinhamento de genomas completos

Lossless compression algorithms for microarray images and whole genome alignments



Universidade de Aveiro
2015

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

Luís Miguel de Oliveira Matos

Algoritmos de compressão sem perdas para imagens de microarrays e alinhamento de genomas completos

Lossless compression algorithms for microarray images and whole genome alignments

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 António José Ribeiro Neves, Professor Auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro, e do Doutor Armando José Formoso de Pinho, Professor Associado com Agregação do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro.

Trabalho financiado pelas seguintes entidades:

o júri / the jury

presidente / president

Doutor João Carlos Matias Celestino Gomes da Rocha
Professor Catedrático da Universidade de Aveiro

vogais / examiners committee

Doutor Paulo Jorge dos Santos Gonçalves Ferreira
Professor Catedrático da Universidade de Aveiro

Doutor João Miguel Raposo Sanches
Professor Auxiliar com Agregação da Universidade de Lisboa

Doutor Luís Filipe Barbosa Almeida Alexandre
Professor Associado com Agregação da Universidade da Beira Interior

Doutor Luís Manuel Dias Coelho Soares Barbosa
Professor Associado da Escola de Engenharia da Universidade do Minho

Doutor António José Ribeiro Neves (Orientador)
Professor Auxiliar da Universidade de Aveiro

agradecimentos

Primeiro que tudo, gostava de agradecer aos meus orientadores, Professor António Neves e Professor Armando Pinho pela oportunidade, orientação e todo o apoio dado durante o meu doutoramento. Foi graças à experiência, recomendações e paciência deles que foi possível concluir este trabalho de investigação. Também gostava de agradecer ao Professor José Moreira pela oportunidade e apoio dado num trabalho de investigação paralelo, no primeiro ano do meu doutoramento. A todos os meus colegas do IEETA, principalmente ao Diogo Pratas, Mário Rodrigues, David Campos, Luís Ribeiro e Marco Pereira, por todo o apoio e conselhos dados nos últimos anos. Quero também agradecer à Universidade de Aveiro, especialmente ao IEETA e ao DETI, por me providenciar as condições necessárias para executar este trabalho. Agradeço também todo o apoio financeiro prestado nas mais diversas ocasiões. Por último, um agradecimento especial à minha família, aos meus pais Armindo e Margarida e ao meu irmão Gabriel, pelo seu apoio genuíno e ilimitado durante este período.

acknowledgements

First, I would like to thank my supervisors, Professor António Neves and Professor Armando Pinho for the opportunity, guidance and all the support given through the Ph.D. It was thanks to their experience, recommendations and patience, that it was possible to conclude this research work. I also would like to thank Professor José Moreira for the opportunity and the support given in a parallel research work, in the first year of my Ph.D. To all my IEETA colleagues, mainly Diogo Pratas, Mário Rodrigues, David Campos, Luís Ribeiro and Marco Pereira, for all the support and advice given through the last years. I also want to thank University of Aveiro, specially IEETA and DETI, for providing the necessary conditions to execute this work. I also thank to all financial support provided in several occasions. Finally, a special thanks to my family, my parents Armindo and Margarida and my brother Gabriel for their genuine and unlimited support during this period.

Palavras-chave

Compressão de imagem sem perdas, imagens de microarrays, modelos de contexto-finito, decomposição em arvore binária, decomposição em planos binários, redução de planos binários, alinhamento de genomas completos, formato de multi-alinhamento.

Resumo

Hoje em dia, no século XXI, a expansão interminável de informação é uma grande preocupação mundial. O ritmo ao qual os recursos de armazenamento e comunicação estão a evoluir não é suficientemente rápido para compensar esta tendência. De forma a ultrapassar esta situação, são necessárias ferramentas de compressão sofisticadas e eficientes. A compressão consiste em representar informação utilizando a menor quantidade de bits possível. Existem dois tipos de compressão, com e sem perdas. Na compressão sem perdas, a perda de informação não é tolerada, por isso a informação descodificada é exatamente a mesma que a informação que foi codificada. Por outro lado, na compressão com perdas alguma perda é aceitável. Neste trabalho, focámo-nos apenas em métodos de compressão sem perdas. O objetivo desta tese consistiu na criação de ferramentas de compressão sem perdas para dois tipos de dados. O primeiro tipo de dados é conhecido na literatura como imagens de microarrays. Estas imagens têm 16 bits por píxel e uma resolução espacial elevada. O outro tipo de dados é geralmente denominado como alinhamento de genomas completos, particularmente aplicado a ficheiros MAF. Relativamente às imagens de microarrays, melhorámos alguns métodos de compressão específicos utilizando algumas técnicas de pré-processamento (segmentação e redução de planos binários). Além disso, desenvolvemos também um método de compressão baseado em estimação dos valores dos píxeis e em misturas de modelos de contexto-finito. Foi também considerada, uma abordagem baseada em decomposição em árvore binária. Foram desenvolvidas duas ferramentas de compressão para ficheiros MAF. A primeira ferramenta, é baseada numa mistura de modelos de contexto-finito e codificação aritmética, onde apenas as bases de ADN e os símbolos de alinhamento foram considerados. A segunda, designada como MAFCO, é uma ferramenta de compressão completa que consegue lidar com todo o tipo de informação que pode ser encontrada nos ficheiros MAF. MAFCO baseia-se em vários modelos de contexto-finito e permite compressão/descompressão paralela de ficheiros MAF.

Keywords Lossless image compression, microarray images, finite-context models, binary-tree decomposition, bitplane decomposition, bitplane reduction, whole genome alignment, multiple alignment format.

Abstract Nowadays, in the 21st century, the never-ending expansion of information is a major global concern. The pace at which storage and communication resources are evolving is not fast enough to compensate this tendency. In order to overcome this issue, sophisticated and efficient compression tools are required. The goal of compression is to represent information with as few bits as possible. There are two kinds of compression, lossy and lossless. In lossless compression, information loss is not tolerated so the decoded information is exactly the same as the encoded one. On the other hand, in lossy compression some loss is acceptable. In this work we focused on lossless methods. The goal of this thesis was to create lossless compression tools that can be used in two types of data. The first type is known in the literature as microarray images. These images have 16 bits per pixel and a high spatial resolution. The other data type is commonly called Whole Genome Alignments (WGA), in particularly applied to MAF files. Regarding the microarray images, we improved existing microarray-specific methods by using some pre-processing techniques (segmentation and bitplane reduction). Moreover, we also developed a compression method based on pixel values estimates and a mixture of finite-context models. Furthermore, an approach based on binary-tree decomposition was also considered. Two compression tools were developed to compress MAF files. The first one based on a mixture of finite-context models and arithmetic coding, where only the DNA bases and alignment gaps were considered. The second tool, designated as MAFCO, is a complete compression tool that can handle all the information that can be found in MAF files. MAFCO relies on several finite-context models and allows parallel compression/decompression of MAF files.

Contents

Contents i

Acronyms v

1 Introduction 1
- 1.1 Motivation . . . 2
- 1.2 Research Goals/Main Contributions . . . 4
- 1.3 Publications and tools . . . 5
  - 1.3.1 Book chapter . . . 5
  - 1.3.2 Articles in peer-reviewed journals . . . 5
  - 1.3.3 International peer-reviewed conferences/proceedings . . . 5
  - 1.3.4 National peer-reviewed conferences/proceedings . . . 6
  - 1.3.5 Compression tools . . . 6
- 1.4 Thesis outline . . . 7

2 Lossless image compression 9
- 2.1 Lossless image coding . . . 9
  - 2.1.1 Lossless image compression standards . . . 9
    - 2.1.1.1 JBIG . . . 9
    - 2.1.1.2 PNG . . . 11
    - 2.1.1.3 JPEG-LS . . . 11
    - 2.1.1.4 JPEG2000 . . . 12
  - 2.1.2 Other compression tools . . . 14
    - 2.1.2.1 Intra-mode of H.264/Advance Video Coding (AVC) standard 14
    - 2.1.2.2 Intra-mode of the High Efficiency Video Coding (HEVC) standard . . . 15
  - 2.1.3 Image decomposition . . . 16
    - 2.1.3.1 Bitplane decomposition . . . 17
    - 2.1.3.2 Binary-tree decomposition . . . 18
  - 2.1.4 Finite-context models . . . 18
    - 2.1.4.1 Arithmetic coding . . . 20
- 2.2 Microarray-specific compression techniques . . . 21
  - 2.2.1 Microarray images . . . 21
  - 2.2.2 State of the art in DNA microarray image compression . . . 22
    - 2.2.2.1 Segmented LOCO (SLOCO) . . . 22
    - 2.2.2.2 Hua's method . . . 23
    - 2.2.2.3 Faramarzpour's method . . . 24
    - 2.2.2.4 MicroZip . . . 26

2.2.2.5 Zhang's method . . . 27
2.2.2.6 Neekabadi's method . . . 28
2.2.2.7 Battiato's method . . . 30
2.2.2.8 Neves' method . . . 32
2.2.2.9 Other methods . . . 32
2.3 Summary . . . 35

3 Lossless compression of microarray images 37
3.1 Microarray image data sets . . . 38
3.2 The use of standard image compression methods . . . 40
3.3 Microarray-specific compression methods . . . 40
3.4 Bitplane decomposition approaches . . . 41
3.4.1 Segmentation . . . 43
3.4.1.1 Experimental results . . . 47
3.4.1.2 Complexity . . . 49
3.4.2 Bitplane reduction . . . 49
3.4.2.1 Experimental results . . . 51
3.4.2.2 Complexity . . . 53
3.5 Simple bitplane coding using pixel value estimates . . . 53
3.5.1 The proposed approach inspired on Kikuchi's method . . . 54
3.5.1.1 Experimental results . . . 55
3.5.2 Mixture of finite-context models . . . 57
3.5.2.1 Experimental results . . . 59
3.5.3 Complexity . . . 61
3.6 Proposed method based on binary tree decomposition . . . 62
3.6.1 Hierarchical organization of the intensity levels . . . 62
3.6.2 Encoding pixel locations . . . 63
3.6.3 Experimental results . . . 64
3.6.4 Complexity . . . 65
3.7 Rate-distortion study . . . 67
3.8 Summary . . . 70

4 Compression of whole genome alignments 73
4.1 Whole genome alignments . . . 73
4.1.1 Multiple Alignment Format (MAF) . . . 75
4.1.2 Genomic data sets . . . 76
4.2 Specialized compression methods for MAF files . . . 77
4.2.1 Hanus' method . . . 78
4.2.1.1 Nucleotides compression . . . 78
4.2.1.2 Gaps compression . . . 79
4.2.2 MAF-BGZIP . . . 79
4.3 Proposed method for the MSABs based on a mixture of finite-context models 80
4.3.1 Method description . . . 80
4.3.2 Proposed models . . . 80
4.3.2.1 Typical image templates . . . 80
4.3.2.2 Ancestral Context Model (ACM) . . . 81
4.3.2.3 Static Column Model (SCM) . . . 82

4.3.2.4 Column Model 5 (CM5) . . . 83
4.3.3 Experimental results . . . 84
4.3.4 Complexity . . . 89
4.4 MAFCO: a compression tool for MAF files . . . 92
4.4.1 Compression of the 's' lines . . . 93
4.4.2 Compression of the 'q' lines . . . 96
4.4.3 Compression of the 'i' lines . . . 96
4.4.4 Compression of the 'e' lines . . . 96
4.4.5 Parallel processing and partial decoding . . . 97
4.4.6 Experimental results . . . 97
4.4.7 Complexity . . . 101
4.5 Summary . . . 102

5 Conclusion and future work **105**
5.1 Conclusions . . . 105
5.2 Future work . . . 107
5.3 Acknowledgments . . . 107

A Microarray images data sets **109**
A.1 ApoA1 data set . . . 109
A.2 Arizona data set . . . 109
A.3 IBB data set . . . 110
A.4 ISREC data set . . . 110
A.5 Omnibus data set . . . 110
A.6 Stanford data set . . . 111
A.7 Yeast data set . . . 111
A.8 YuLou data set . . . 113

B Global microarray image compression results **121**

C Multiple Alignment Format (MAF) **123**
C.1 The header lines . . . 123
C.2 The 'a' lines . . . 124
C.3 The 's' lines . . . 124
C.4 The 'q' lines . . . 125
C.5 The 'i' lines . . . 126
C.6 The 'e' lines . . . 126
C.7 MAF file examples . . . 127

D Multiple alignments data sets **129**
D.1 Statistics regarding the average number of columns and rows of each MSAB . 129
D.2 Statistics regarding the symbols of 's', 'q', 'i', and 'e' line types . . . 130

E Detailed results of the proposed compression methods for MAF files **135**
E.1 Results of the compression algorithm for the MSABs . . . 136
E.2 Results for the MAFCO tool . . . 140

Bibliography **147**

Acronyms

ACM	Ancestral Context Model
AP	Adaptive Pixel
AQV	Actual Quality Value
AVC	Advance Video Coding
BAC	Binary Arithmetic Coder
BAM	Binary Alignment/Map format
BASICA	Background Adjustment, Segmentation, Image Compression and Analysis of microarray images
BFS	Background/Foreground Separation
BGZF	Blocked GZip Format
BTD	Binary Tree Decomposition
BWT	Burrows-Wheeler Transform
C2S	Circle To Square
CABAC	Context-Adaptive Binary Arithmetic Coding
CALIC	Context-Based, Adaptive, Lossless Image Coder
CAVLC	Context-Adaptive Variable Length Coding
CCSDS	Consultative Committee for Space Data System
CM5	Column Model 5
CNN	Cellular Neural Network
CPU	Central Processing Unit
CREW	Compression with Reversible Embedded Wavelets
DCT	Discrete Cosine Transform
DNA	DeoxyriboNucleic Acid
DFT	Direct Fourier Transform

DST	Discrete Sine Transform
DWT	Discrete Wavelet Transform
EBCOT	Embedded Block Coding with Optimized Truncation
EIDAC	Embedded Image-Domain Adaptive Compression
FCM	Finite-Context Model
FELICS	Fast Efficient & Lossless Image Compression System
GEO	Gene Expression Omnibus
GI	Gini Index
GIF	Graphics Interchange Format
GOBs	Group Of Blocks
Gzip	GNU zip
HC	Histogram Compaction
HD	High Definition
HDR	High Dynamic Range
HEVC	High Efficiency Video Coding
HIV	Human Immunodeficiency Virus
IBB	Institut de Biotecnologia i Biomedicina
IEC	International Electronic Commission
ISO	International Organization for Standardization
ITU-T	International Telecommunication Union
JBIG	Joint Bi-Level Image Experts Group
JBIG2	Joint Bi-Level Image Experts Group 2
JPEG	Joint Photographic Experts Group
JPEG-LS	Joint Photographic Experts Group-Lossless Standard
JPEG2000	Joint Photographic Experts Group 2000
LOCO	LOw COmplexity LOssless COmpression
LOCO-I	LOw COmplexity LOssless COmpression of Images
LSBP	Least Significant BitPlane
LZ77	Lempel-Ziv 1977

LZW Lempel-Ziv-Welch

MAE Maximum Absolute Error

MAF Multiple Alignment Format

MAFCO MAF COmpressor

MAFQV MAF Quality Value

ME Microarray Error

MENT Microarray comprEssioN Tools

MMR Modified-Modified READ (Relative Element Address Designate)

MP3 Moving Picture Experts Group Layer-3 Audio

MDM Microarray Distortion Metric

MPEG Moving Pictures Experts Group

MSA Multiple Sequence Alignment

MSAB Multiple Sequence Alignment Block

MSBP Most Significant BitPlane

NCBI National Center for Biotechnology Information

PMc Puncturing Matrix compressor

PNG Portable Network Graphics

PPAM Prediction by Partial Approximate Matching

PPM Prediction by Partial Matching

RGB Red Green Blue

RLE Run Length Encoding

RMSE Root Mean Square Error

RNA RiboNucleic Acid

ROI Region Of Interest

SACO Sequence Alignment COmpressor

SAM Sequence Alignment/Map format

SBAC Syntax-Based context-adaptive binary Arithmetic Coding

SBC Simple Bitplane Coding

SBR Scalable Bitplane Reduction

SCM	Static Column Model
SIB	Swiss Institute for Bioinformatics
SLOCO	Segmented LOw COmplexity LOssless COmpression
SNR	Signal-to-Noise Ratio
SPITH	Set Partitioning in Hierarchical Trees
SPMG	Signal Processing and Multimedia Group
TSS	Telecommunication Standardization Sector
UAB	Universitat Autònoma de Barcelona
USCSC	University of California Santa Cruz
UQ	Uniform Quantizer
URL	Uniform Resource Locator
VCEG	Video Coding Experts Group
WGA	Whole Genome Alignments
WWW	World Wide Web

"Knowledge is power. Information is liberating. Education is the premise of progress, in every society, in every family."

Kofi Annan

1 Introduction

Nowadays, at the information age, data compression is one of the most active and intense research topics. For the most of us, data compression is sometimes invisible but its presence is ubiquitous. Without it, the information revolution would probably not occur as fast as it was and some technological advances could not be done. We can see data compression in almost all information technology that we have in this era, from MP3 players, to smartphones, digital television, movies, Internet, etc [1].

Data compression is the art or science of representing information in a compact form. This compact representation is achieved by removing or reducing the redundancy that can be found in the data. A compression technique always has two algorithms associated with it. An encoding algorithm that will be responsible to transform a given input data in its compressed representation and a decoding method that will perform the reverse operation. These two algorithms together are usually denominated as a compression algorithm or method [1].

Usually, compression techniques are classified according to their capability for recovering the original data. Lossy methods waive away that capability in exchange for increased compression, whereas lossless techniques are stick to the principle of exact recovering of the original data, even if that implies modest compression rates. Lossy methods are typically used in consumer products, such as photographic cameras. Lossless methods are generally required in applications where cost, legal issues or value play a decisive role, such as, for example, in remote and medical imaging or in image archiving [2].

During this research work, we studied and developed compression tools for two specific data types. The first one is a specific image type known as microarrays. The other type is a particularly voluminous dataset in molecular genomics, called Whole Genome Alignments (WGAs). DNA microarray imaging is an important tool and a powerful technology for large-scale gene sequence and gene expression analysis, allowing the study of gene function, regulation and interaction across a large number of genes, and even across entire genomes [3, 4]. DNA microarrays are currently used, for example, for genome-wide monitoring in areas such as cancer [5] and HIV research [6]. The result of a microarray experiment consists on a pair of 16 bits per pixel grayscale images with a high spatial resolution.

Regarding the WGAs, they gained a considerable importance over the last years. These

WGAs provide an opportunity to study and analyze the evolutionary process of several species. The Multiple Alignment Format MAF is commonly used to store this kind of data. Because these files store alignments of several species and chromosomes, the output files after the alignment can be very large, reaching several gigabytes per file in raw format. The WGAs are also been used to help locating certain kinds of functional non-coding regions [7] and more recently for finding protein-coding genes [8, 9] and non-coding RNA genes [10]. Moreover, it is possible to observe the similarities and differences between the DNA sequences of humans and other species that share a common ancestral, providing critical data for finding the course of evolution.

In this research work we were only interested on lossless methods, i.e., no loss of information was acceptable. For the case of microarray images, some of the most well-known general purpose compression methods are usually used. Furthermore, there are also several specific compression methods that can be found in the literature and that consider three approaches: pure lossless, lossy, and lossy-to-lossless coding. Regarding the WGAs, usually gzip or other general purpose method is used to compress this kind of genomic data. However, a more specialized method for this kind of data is essential, because genomic data requires several gigabytes to be stored and transmitted.

This research work will be focused in improving and developing lossless compression algorithms for microarray images and WGAs, with the goal of outperforming other general/specific compression methods available. The performance of all compression methods developed will be compared with other compression tools in terms of speed, compression performance and, for the case of microarrays, rate-distortion.

1.1 Motivation

Regardless of the data that represents a microarray image or a MAF file, usually some redundancy exists. The amount of redundancy is dependent of the type of data that we are dealing with. Different types of images might be redundant in different ways. The same is also true for genomic data, where depending on the specie, chromosome or even the sequence method used to obtain the DNA data, we will have different kinds of redundancy.

The specificity of contents presented in some types of image (e.g. in microarray images) and MAF files presents a challenge to the general purpose techniques, which have been designed to deal with more general data. The most well known image coding standards have been used to compress images, but because they were designed with the aim of compressing natural images, their performance on some specific image types (e.g. microarrays) could be degraded. The same is also true for genomic data. Usually, gzip or other general purpose method is used to compress this kind of data, but because of the generic design of these general purpose tools, their performance is limited.

In Figure 1.1 we illustrate an example of the eight Most Significant Bitplanes (MSBPs) of a microarray image (on the left) and its histogram on the right. We also provide in Figure 1.2 the eight Least Significant Bitplanes (LSBPs) of the same microarray image (on the left) and its histogram on the right. For the eight MSBPs, we notice that this kind of image have a higher percentage of pixels with low intensities, implying a highly asymmetrical histogram. On the contrary, the histogram for the eight LSBPs have a very different distribution. The distribution along the intensities is more spread when compared to the eight MSBPs. This behavior is due to the amount of noise that is particular in these kind of images. These

characteristics are not considered by the general purpose compression methods such as JBIG, PNG, JPEG-LS, and JPEG2000. This causes degradation in compression performance when using such general purposed methods on microarray images.

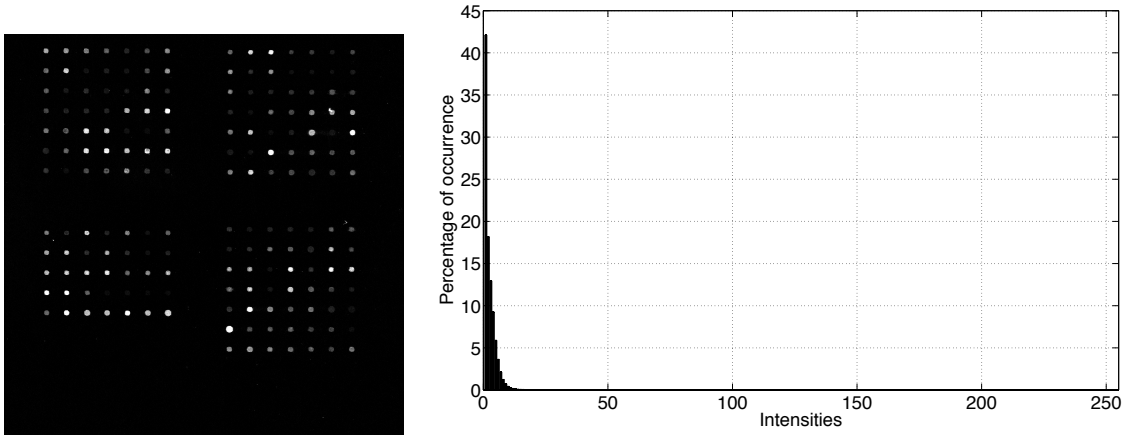


Figure 1.1: The 8 MSBPs of the "Def661Cy3" microarray image from the *ISREC* data set on the left. On the right, its the corresponding histogram.

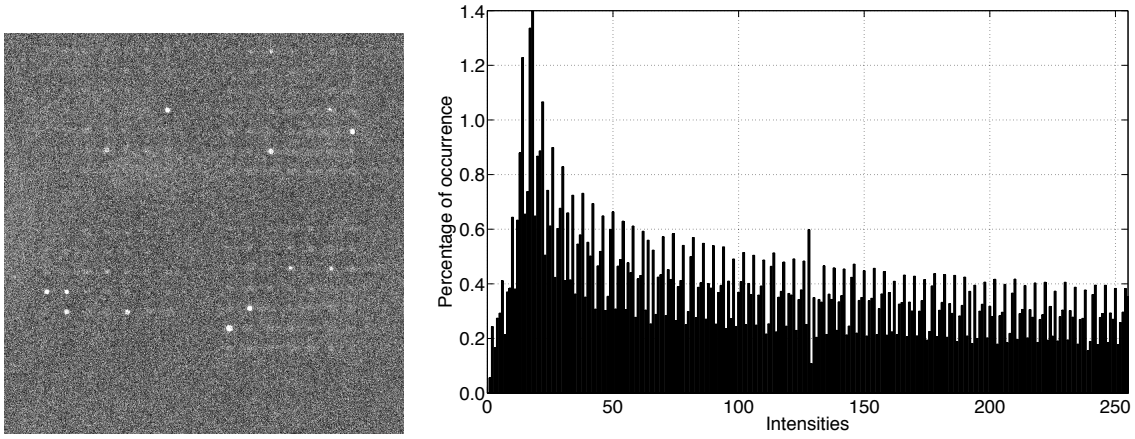


Figure 1.2: The 8 LSBPs of the "Def661Cy3" microarray image from the *ISREC* data set on the left. On the right, its the corresponding histogram.

In Figure 1.3 we can see a portion of the MAF file "chrM.maf" from the *multiz46way* data set. Some direct redundancies are identified with a rectangular shape in the figure. The source names on the left and the alignment gaps ('-') of the 'q' lines are two of the most common examples of redundancy that can be found in MAF files. Furthermore, there are other types of redundancies that are not identified in Figure 1.3, but that were considered in the development of the compression tools presented in this research work.

```
##maf version=1 scoring=autoMZ.v1

    .                                   .                                   .
    .                                   .                                   .
    .                                   .                                   .

a score=4190.000000
s hg19.chrM                          187 50 + 16571 AACATACCTACTAAA-GTGTGTTAATTAATTAATGCTTGTAGGACATAATA
s xenTro2.scaffold_19023            2405 51 +  3053 AACAGATGTATTAGGCATGTGTACATTCATTAATGCATGACTGACATAAAA
i xenTro2.scaffold_19023           N 0 C 0
s gorGor1.Supercontig_0439211       205 31 +   616 -------------------GTGTTAATTAATTCATGCTTGTTGGACATAA--
q gorGor1.Supercontig_0439211                      -------------------99999999999999999999999999999999--
i gorGor1.Supercontig_0439211      I 21 I 99
s galGal3.chrM                      851 44 + 16775 ------CCCGGTAAA-TGGTGCTATTTAGTGAATGCTTGTCGGACATATTT
q galGal3.chrM                                     ------FFFFFFFFF-FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
i galGal3.chrM                     N 0 C 0

a score=-5291.000000
s hg19.chrM                            237 40 + 16571 -----ATAACAATTGAATGTCTGCACAGCCGC------TTTCCACACAGAC
s turTru1.scaffold_98585             25144 31 - 72340 -----ATTGTAATTATAAACTTGCACA----------------CATATTATC
q turTru1.scaffold_98585                           -----999999999999999999999-----------------999999999
i turTru1.scaffold_98585           N 0 C 0
s galGal3.chrM                         895 35 + 16775 ------------TTATCAATTTTCACTTCCTC---TATTTTCTTCACAAA-
q galGal3.chrM                                     ------------FFFFFFFFFFFFFFFFFFFF---FFFFFFFFFFFFFFF-
i galGal3.chrM                     C 0 C 0
s xenTro2.scaffold_19023            2456 51 +  3053 ATTTAACTACCATAATGAATTCTCAGCTTTTTACCTATTTTCCACCCGGGG
i xenTro2.scaffold_19023           C 0 C 0
e gorGor1.Supercontig_0439211        236 99 +   616 I

    .                                   .                                   .
    .                                   .                                   .
    .                                   .                                   .

##eof maf
```

Figure 1.3: Small portion of the MAF file "chrM.maf" from the *multiz46way* data set. Some direct examples of redundancy are identified with a rectangular shape.

1.2 Research Goals/Main Contributions

The main goal of this doctorate was to study the problem of lossless compression of two specific data types. The first one microarrays. The other data type addressed is a particular voluminous data set in molecular genomics known as Whole Genome Alignments (WGA).

The most known compression image coding standards (e.g. PNG, JPEG-LS, JPEG2000) that allow lossless compression, were developed to efficiently compress images representing natural content. Because microarray images have specific characteristics, they usually lead to poor performance when using the most well known image coding standards. Despite the alternative nature of WGAs, some of its content can be treated as a special kind of symbolic image. Taking into consideration the previous statement, similar image compression models can be used to both microarrays images and WGAs. In the case of WGAs, generally the most popular general-purpose compression tool, gzip is used. Also in this case, these general-purpose compression tools were not specially designed to compress this kind of data, and often fall short when the intention is to reduce the data size as much as possible.

The main objectives of this research work are enumerated next:

- Review the most well known image coding standards and other methods that have been used in the literature to lossless image compression.

- Review and analyze specific compression tools developed for lossless compression of microarray images.

- Develop compression tools for lossless compression of microarray images and compare the obtained results with other compression methods (image coding standards and specific compression tools).

- Study and analyze the Multiple Alignment Format (MAF) which is commonly used to store WGAs.

- Study the specific compression tools developed for WGAs that can be found in the literature.

- Develop an alternative compression tool for lossless compression of WGAs. Compare the obtained results with other general methods (e.g. gzip, bzip2) and also with specific compression tools.

1.3 Publications and tools

The main contributions of this thesis are methods to compress two different data types, as mention earlier. Some publications resulted of the developed tools. The publications with a $\diamondsuit$ are associated to compression of microarray images, whereas the ones marked with a $\star$ are related to the compression of MAF files. In what follows, we summarize the publications that were produced during this research work.

1.3.1 Book chapter

$\diamondsuit$ **L. M. O. Matos**, A. J. R. Neves, and A. J. Pinho, "Lossy-to-lossless compression of biomedical images based on image decomposition", in *Applications of Digital Signal Processing through Practical Approach*, Ed. S. Radhakrishnan, InTech, pp. 125–158, October 2015.

1.3.2 Articles in peer-reviewed journals

$\star$ **L. M. O. Matos**, A. J. R. Neves, D. Pratas, and A. J. Pinho, "MAFCO: a compression tool for MAF files", *PLoS ONE*, vol. 10, no. 3, pp. e0116082, March 2015.

$\star$ **L. M. O. Matos**, D. Pratas, and A. J. Pinho, "A compression model for DNA Multiple Sequence Alignment Blocks", *IEEE Transactions on Information Theory*, vol. 59, no. 5, pp. 3189–3198, May 2013.

1.3.3 International peer-reviewed conferences/proceedings

$\diamondsuit$ **L. M. O. Matos**, A. J. R. Neves, and A. J. Pinho, "Compression of microarray images using a binary tree decomposition", in *Proceedings of the 22nd European Signal Processing Conference, EUSIPCO-2014*, pp. 531–535, Lisbon, Portugal, September 2014.

⋆ **L. M. O. Matos**, D. Pratas, and A. J. Pinho, "Compression of whole genome alignments using a mixture of finite-context models", in *Proceedings of International Conference on Image Analysis and Recognition, ICIAR 2012*, ser. LNCS, Eds. A. Campilho and M. Kamel, pub. Springer, vol. 7324, pp. 359–366, Aveiro, Portugal, June 2012.

1.3.4 National peer-reviewed conferences/proceedings

♢ **L. M. O. Matos**, A. J. R. Neves, and A. J. Pinho, "A rate-distortion study on microarray image compression", in *Proceedings of the 20th Portuguese Conference on Pattern Recognition, RecPad 2014*, Covilhã, Portugal, October 2014.

♢ **L. M. O. Matos**, A. J. R. Neves, and A. J. Pinho, "Compression of DNA microarrays using a mixture of finite-context models", in *Proceedings of the 18th Portuguese Conference on Pattern Recognition, RecPad 2012*, Coimbra, Portugal, October 2012.

♢ **L. M. O. Matos**, A. J. R. Neves, and A. J. Pinho, "Lossy-to-lossless compression of microarray images using expectation pixel values", in *Proceedings of the 17th Portuguese Conference on Pattern Recognition, RecPad 2011*, Porto, Portugal, October 2011.

♢ **L. M. O. Matos**, A. J. R. Neves, and A. J. Pinho, "Lossless compression of microarray images based on background/foreground separation", in *Proceedings of the 16th Portuguese Conference on Pattern Recognition, RecPad 2010*, Vila Real, Portugal, October 2010.

1.3.5 Compression tools

The compression tools created during this work, were developed in ANSI C. The gcc version used was the 4.8.2 on both Linux and OS X. All the experiments were performed on a Linux Server running Ubuntu with 16 Intel(R) Xeon(R) CPUs (E7320 2.13GHz) and 256 gigabytes of memory. The compression tools developed during this research work are summarized next:

♢ MENT - Microarray comprEssioN Tools

- `http://bioinformatics.ua.pt/software/ment`
- `https://github.com/lumiratos/ment`

⋆ SACO - Sequence Alignment COmpressor

- `http://bioinformatics.ua.pt/software/saco`
- `https://github.com/lumiratos/saco`

⋆ MAFCO - MAF COmpressor

- `http://bioinformatics.ua.pt/software/mafco`
- `https://github.com/lumiratos/mafco`

1.4 Thesis outline

This thesis is divided into five chapters and five appendixes:

- Chapter 2 provides a review of the most important lossless image compression standards, including the intra-modes of H.264/AVC and HEVC. It also describes some image decomposition approaches used by image compression algorithms. Moreover, we provide with some detail the core of the compression algorithms proposed in this research work that relies on finite-context models and arithmetic coding. In the last part of this chapter, we present an overview of the specific compression methods for microarray images that can be found in the literature.

- Chapter 3 presents a study regarding the compression of microarray images using several image coding standards including JBIG, PNG, JPEG-LS, and JPEG2000. We also propose compression methods based on bitplane decomposition, using some pre-processing techniques (segmentation, histogram reduction, etc.). We describe a compression tool that we developed, based on simple bitplane coding using pixel values estimates. Furthermore, we introduce a compression method based on binary tree decomposition for microarray images. At the end of the chapter, we provide a rate-distortion evaluation of two specific compression methods and two image coding standards, JBIG and JPEG2000.

- Chapter 4 deals with the compression of WGAs. We provide an overview of MAF and the compression methods that can be found in the literature for this kind of data. A specialized method for compressing the DNA bases and alignments gaps based on a mixture of finite-context models is described in this chapter. Moreover, a complete compression tool for MAF files is also provided with parallel compression/decompression capabilities.

- Chapter 5 summarizes the main contributions of this work and presents some possible future work.

- Appendix A provides the microarray images data sets used to evaluate the performance of the compression methods presented in this thesis.

- Appendix B contains a global view of the compression results obtained by the proposed methods for microarray images.

- Appendix C describes in detail the Multiple Alignment Format (MAF).

- Appendix D presents the MAF data sets used to evaluate the performance of several compression algorithms including those proposed in this research work.

- Appendix E contains compression results for the MAF data sets in more detail.

"If you cannot explain it simply, you do not understand it well enough."

Albert Einstein

2 Lossless image compression

The aim of lossless image coding is to represent an arbitrary image with the smallest possible number of bits, without losing any information from the original image. In other words, it is always possible to recover the original image from the compressed one. In this chapter, we present the most important state-of-the-art image coding standards, namely JBIG, PNG, JPEG-LS, and JPEG2000. We also describe some image decomposition approaches used in image compression and explain how finite-context models work. Finally, we present the state-of-the-art methods for microarray image compression.

2.1 Lossless image coding

2.1.1 Lossless image compression standards

In this section, we will briefly describe the most relevant state-of-the-art image coding standards, namely JBIG, PNG, JPEG-LS, and JPEG2000. Each standard was developed with different goals in mind. For example, JBIG was created specially to provide progressive lossless compression of binary and low-precision gray-level images. PNG was developed for lossless compression of computer graphics images, however supporting also the grayscale and true-color images. JPEG-LS was designed for lossless compression of continuous-tone images. Finally, the JPEG2000 standard was created with the aim of providing a wide range of functionalities, as we will discuss next.

2.1.1.1 JBIG

The JBIG [20, 23] (Joint Bi-Level Image Experts Group) standard was issued in 1993 by the International Organization for Standardization / International Electronic Commission (ISO/IEC) and by the Telecommunication Standardization Sector (TSS) of the International Telecommunication Union (ITU-T). This standard was specially created to provide progressive lossless compression of binary images.

One of the main positive points of JBIG when compared with other standards is its capability of progressive encoding and its compression efficiency which is usually superior (e.g. for MMR) [21–23]. This progressive encoding characteristic allows to save the image in several "layers" of the compressed stream. In the decoding phase, the image can be progressively decoded and the first image that the viewer sees is an imprecise version of the original image (that correspond to the first layer). After that, the remaining layers are decoded, improving the image version to its original form (higher layers). Initially JBIG was designed for bi-level images, however it is possible to apply it to grayscale images by separating the input image into several bitplanes. After the bitplane separation, JBIG can be applied to each individual bitplane, as if it was a bi-level image. In this scenario, it is possible to improve the compression efficiency by using a Gray Code, instead of the standard binary code [24].

In Figure 2.1 we present the block diagram of a JBIG encoder. According to it, we can see that the JBIG core consists of an adaptive finite-context model followed by arithmetic coding. More information about finite-context models can be found in Section 2.1.4.

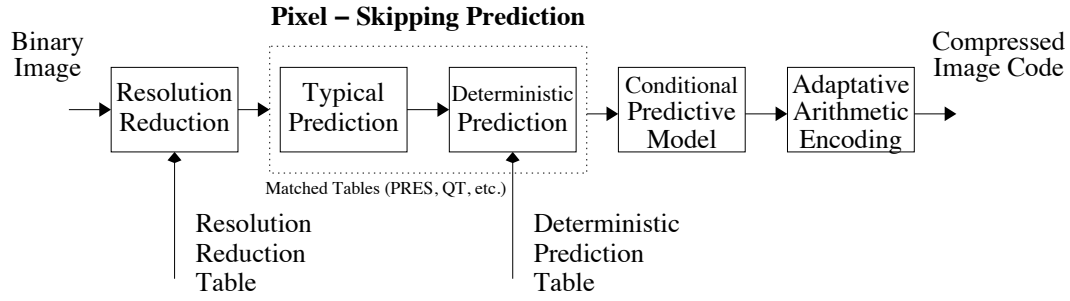


Figure 2.1: Block diagram of JBIG compression encoder [25]

JBIG uses several different context templates in its encoding procedure. Figure 2.2 depicts two templates used for the low resolution layer. In the encoding procedure, JBIG decides whether to use the three-line (left template of Figure 2.2) or the two-line template (right template of Figure 2.2). According to [23], it seems that the two-line template is faster but the three-line template produces slightly better compression results. The template pixel labeled as "A" in Figure 2.2 is known as Adaptive Pixel AP. The encoder is allowed to use as AP a pixel outside of the template in use. Two parameters T_x and T_y are used in each layer to indicate to the decoder the location of the AP.

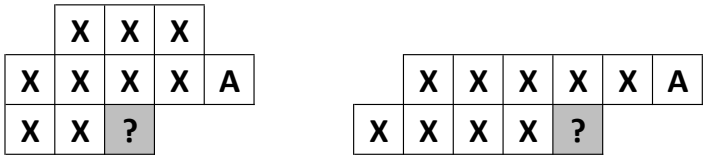


Figure 2.2: JBIG templates used for lowest-resolution layer: on the left a three-line template on the right a two-line template. Pixel "A" is called Adaptive Pixel AP.

The lossy mode of JBIG produces very poor images in terms of visual quality when compared to the original ones. In order to overcome this problem, JBIG2 [26] was proposed. In this new version, some additional functionalities were introduced such as multipage document compression, two modes of progressive compression, lossy compression and differentiated

compression methods for different regions of the image (e.g. text, halftone and generic) [23]. Splitting the image into several different regions allows JBIG2 to encode each one using an appropriate method. On one hand, the compression results can be improved using this approach. On the other hand, this decomposition in several regions is very important for interactive video applications [27].

2.1.1.2 PNG

The Portable Network Graphics (PNG) [28, 29] file format was developed in 1996 by a group headed by Thomas Boutell. This file format was initially created to replace the GIF file format, due to legal issues. PNG is a very robust format that supports several image types such as color-indexed, grayscale, and true-color, with optional transparency (alpha channel). Several features were included in this standard namely support for images with 1, 2, 4, 8, and 16 bitplanes, and sophisticated color matching, lossless compression by means of a *deflate* algorithm [30]. PNG was designed to online viewing applications, such as the World Wide Web (WWW). In this kind of applications, PNG supports a progressive display option using a 2-D interlacing algorithm. This algorithm, known as Adam7, uses seven passes and operates in both dimensions. Compared to GIF, that only uses four passes in the vertical dimension, Adam7 allows an approximation of the entire image more quickly in the early passes. Moreover, the PNG standard provides both full file integrity checking and simple detection of common transmission errors, which is a very important feature when using online applications.

One of the most important parts of the PNG standard is the compression scheme core. Its core is a descendant of the LZ77 algorithm [31] known as the deflate algorithm [32]. Both deflate and the Lempel-Ziv-Welch (LZW) algorithms have similar encoding and decoding speeds however, deflate generally attains better compression results. The deflate algorithm uses a sliding windows of up to 32 kilobytes, with a Huffman encoder [23] in the back end. The encoding procedure consists on finding the longest matching string (or at least a long string) in the 32 kilobytes windows immediately prior to the current position. After the search is complete, a pointer (distance backwards) and a length must be stored. Then, the current position and the window is advanced, accordingly.

The limits match-lengths of deflate is between 3 and 258 bytes. An alternative mechanism is used if the encoded sequence is for instance less than tree bytes (particularly single bytes). In order to be able to encode single bytes, the encoder must be able to encode plain characters, or "literals". This means that the deflate algorithm needs to handle three kinds of symbols: lengths, distances, and literals. These three alphabets are the input for the Huffman stage of the deflate engine. In reality, deflate merges the length and literal codes into a single alphabet of 286 symbols. A similar approach is used for the distance alphabet. Both alphabets (length/literal and distances) are fed to the Huffman encoder and compressed using either fixed or dynamic Huffman codes.

2.1.1.3 JPEG-LS

The JPEG-LS [33–35] is the state-of-the-art International Standard for lossless and near-lossless coding of continuous tone still images. This standard was developed by the Joint Photographic Experts Group (JPEG) with the goal of providing a low complexity lossless image standard that could be able to offer better compression efficiency than lossless JPEG.

The first part was finalized in 1999. Several compression algorithms were proposed to be part of the JPEG-LS standard. After a long phase of tests, the committee decided to choose LOCO-I (LOssless COmpression for Images) [35] from the Hewlett-Packard Laboratories to be integrated in the JPEG-LS standard. LOCO-I relies on prediction, residual modeling and context-based coding of the residuals. Other alternatives have been proposed to be integrated in JPEG-LS such as CALIC [36], FELICS [37] and CREW [38] from Ricoh. One of the reasons that led the committee to select the LOCO-I algorithm to be incorporated in JPEG-LS is its simplicity and also because is relatively fast and easy to implement. There are some applications that usually need to run in lower performance machines. Taking into consideration this scenario, having a less complex compression method is crucial, despite sacrificing the compression rate that is attained by other more complex and sophisticated methods. Most of the low complexity of LOCO-I comes from the assumptions that prediction residuals follow a two-sided geometric probability distribution and from the use of Golomb codes, which are known to be optimal for this kind of distributions. Apart from the lossless mode, JPEG-LS also has a lossy mode where the maximum absolute error can be controlled be the encoder. This mode is also known as near-lossless compression or L_∞-constrained compression. The basic block diagram of JPEG-LS is given in Figure 2.3.

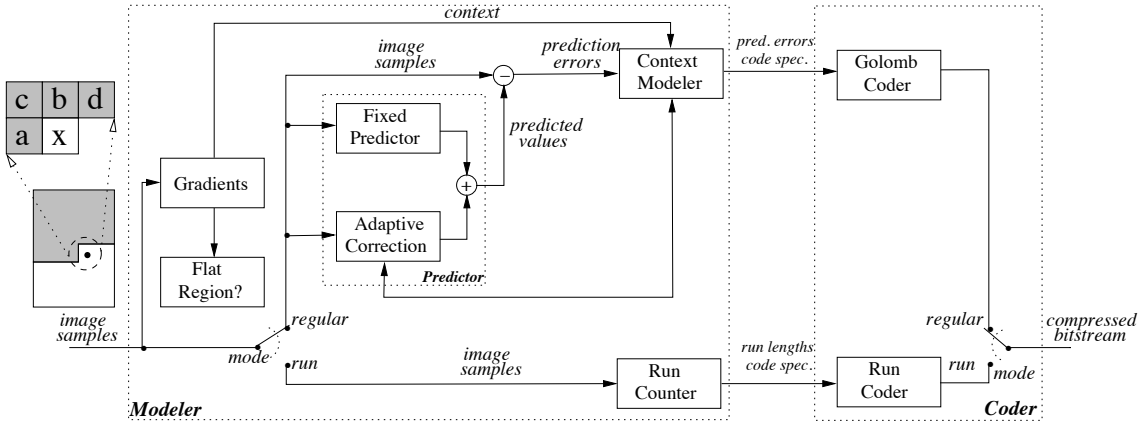


Figure 2.3: Block diagram of a JPEG-LS encoder [35].

2.1.1.4 JPEG2000

JPEG2000 (ISO/IEC 15444-1) [2, 39–41] is the most recent international standard for still image compression. Part 1 was published as an International Standard in the year 2000 [42]. The seven parts of JPEG2000 are:

- Part 1 - JPEG2000 image coding system (core)
- Part 2 - Extension (added some resources and core improvement)
- Part 3 - Motion JPEG2000
- Part 4 - Conformance
- Part 5 - Reference software

- Part 6 - Compound image file format

- Part 7 - Technical report

This standard was designed to provide improved image quality at the expenses of an increased computation power that we have nowadays [43]. JPEG2000 is based on wavelet technology (see Figure 2.4). The resulting wavelet coefficients are embedded block encoded using the Embedded Block Coding with Optimized Truncation (EBCOT) [39, 44] algorithm. This technology seems to provide very good compression performance for a wide range of bit rates, including lossless coding. Furthermore, JPEG2000 is very versatile because it allows the generation of embedded code-streams that will provide means of extracting lower bit rate instances, without the need for re-encoding, from a higher bit rate stream. This compression system allows great flexibility in the compression of images and also for the access into the compressed stream. Several mechanisms are provide by the codestream for locating and extracting data for the purpose of re-transmition, storage, display or editing. This sophisticated access is very useful for storage and retrieval of data appropriate for a given application, without decoding.

The block diagrams of both the encoder and decoder of JPEG2000 are illustrated in Figure 2.5. The Discrete Wavelet Transform (DWT) is the first operation applied in the source image. Then, the obtained transform coefficients are quantized and entropy coded. The output codestream (bitstream) is then obtained after the entropy encoding procedure. The entropy coder uses an adaptive arithmetic encoding strategy with no more than nine different models. This context models are restarted in the beginning of each block to be coded and the arithmetic encoder is always ended in the end of each block. This strategy allows a much more reliable error control [40]. The decoder works in the reverse way of the encoder. First, the codestream is first entropy decoded. Then, the obtained quantized coefficients are dequantized (reverse of the quantization operation of the encoder). In the end, the inverse discrete transform is applied to obtain the reconstructed image data. More details can be found in [40, 42].

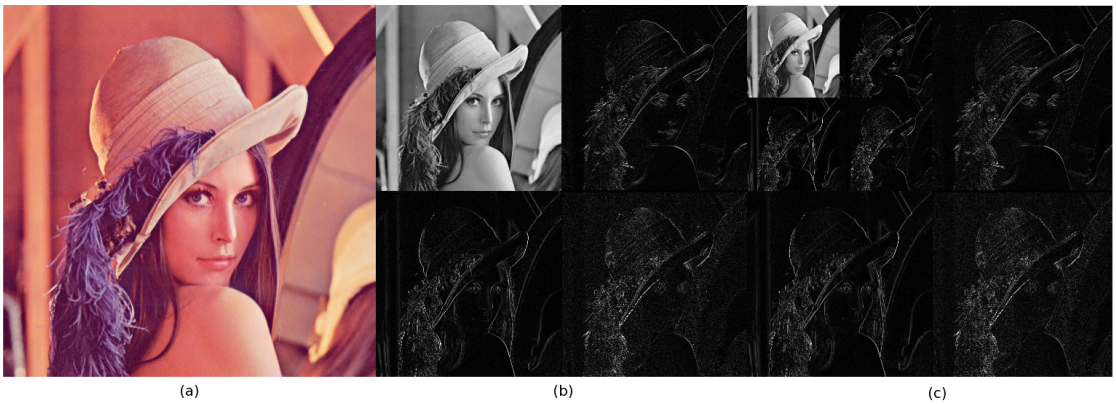


Figure 2.4: (a) original image; (b) wavelet first level decomposition; (c) wavelet second level decomposition (adapted from [45]).

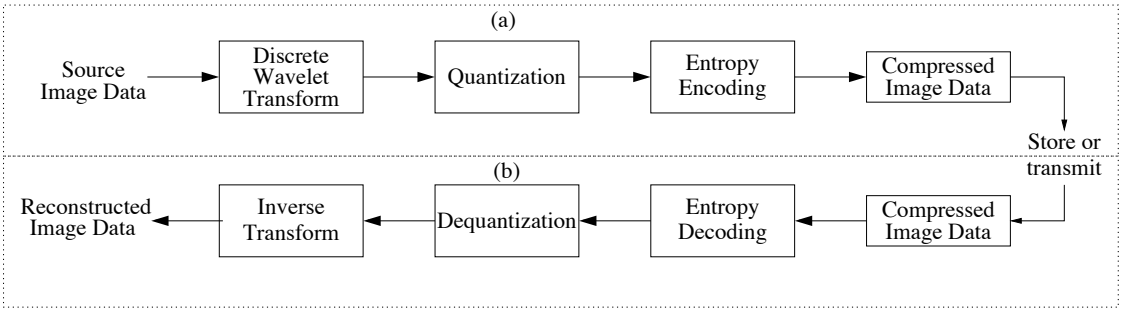


Figure 2.5: Block diagram of JPEG2000 (a) encoder (b) decoder [40].

2.1.2 Other compression tools

H.264/AVC and HEVC are the most recent and sophisticated video standards algorithms. Both are hybrid video coding schemes consisting of block-based spatial or temporal prediction, combined with transform coding of the prediction error. These two standards are designed for video compression applications so they achieve high coding gains in particular by the use of temporal prediction. Despite their nature being video coding, these two standards also have an intra-coding mode that can be applied to still images as well. As a matter of fact, some recent studies in the last years were conducted in order to evaluate the performance of both H.264/AVC and HEVC in still images [46–49]. According to Nguyen and Marpe [47], H.264/AVC and HEVC can achieve an average bit-rate saving of about 32% and 44% relative to JPEG, respectively. Their study was performed using only the intra-mode of H.264/AVC and HEVC. Moreover, the image test set used contain grayscale and color images. Both H.264/AVC and HEVC standards are very sophisticated and efficient even when applied to still image compression. Despite this, they both lack support for specific image types, such as images with high bit depth (e.g., some medical images, microarrays). Nevertheless, we will provide in the following two sections a short description of the intra-coding mode of both H.264/AVC and HEVC.

2.1.2.1 Intra-mode of H.264/Advance Video Coding (AVC) standard

H.264/AVC (Advance Video Coding) is a video coding standard created by the ITU-T Video Coding Experts Group (VCEG) and the ISO/IEC Moving Pictures Experts Group (MPEG) [50–52].

As mentioned earlier, we only will be focusing our attention on the intra-mode of H.264/AVC that can be used for still images. In Figure 2.6, we can see the block diagram of the H.264/AVC intra coding process. The intra coding mode has in its core an intra prediction procedure that reduces the spatial redundancy between spatially adjacent blocks. This procedure uses pixels from the already coded neighboring blocks to predict the current block. For the luminance signal, H.264/AVC allows a total of 9 intra prediction modes for 4×4 and 8×8 blocks (see Figure 2.7), and a total of 4 intra prediction modes for 16×16 blocks. On the other hand, for the chrominance signal, a total of 4 intra modes are available to predict 8×8 chroma blocks.

After having the prediction block, the residual block can be easily obtained by subtracting the prediction block and the original block. Then, a DCT-like integer transform is applied to the residuals. This transformation process is more complex for the 16×16 blocks, where an

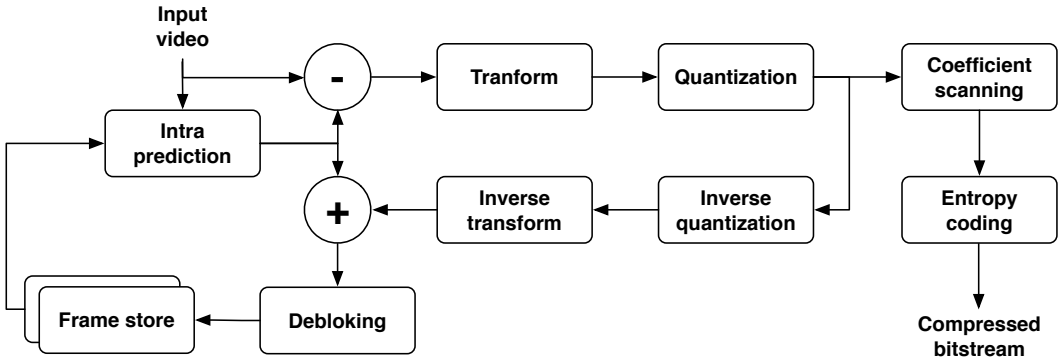


Figure 2.6: Structure of the H.264 intra coding mode [53].

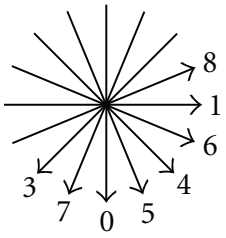


Figure 2.7: The nine intra prediction modes for a 4×4 and 8×8 block based luminance spatial prediction [54].

additional Hadamard transform is required.

The transformed coefficients are then quantized using a non-uniform scalar quantization model. A set of 53 quantization parameters can be used to each block individually. Quantization is a lossy process, so it will not be used if a lossless compression is required. In the end, the quantized transform coefficients are entropy coded using Context-Adaptive Variable Length Coding (CAVLC) or Context-Adaptive Binary Arithmetic Coding (CABAC).

A more detailed description of the H.264/AVC can be found in [51, 52, 55].

2.1.2.2 Intra-mode of the High Efficiency Video Coding (HEVC) standard

High Efficiency Video Coding (HEVC) is the most recent video compression standard jointly developed by ITU-T VCEG and ISO/IEC MPEG [56, 57]. This video compression standard is the successor of H.264/AVC and was defined as a standard on April, 2013. According to [58], HEVC can achieve a bit-rate reduction near 50% when compared to is predecessor, H.264/AVC, at the same level of video quality. Furthermore, it also supports the most recent video formats that go beyond HD format (e.g. 4k×2k or 8×4k).

The intra coding mode of HEVC is similar to the one illustrated in Figure 2.6 for H.264/AVC. Both H.264/AVC and HEVC intra coding modes are based on spatial prediction followed by transform coding. The main difference is that HEVC have much more intra prediction modes when compared to H.264/AVC. There are mainly three types of intra prediction modes for the luminance component: intra-angular, intra-DC and intra-planar. HEVC provides a total of 33 different prediction modes in the intra-angular model (see Figure 2.8). The intra-angular model of HEVC is much more finer than the prediction method

used by H.264/AVC, due to the higher number of prediction modes. Intra-DC is the most simple intra prediction mode of HEVC, where the mean value of the surrounding pixels is used to predict the current region. Finally, the intra-planar mode is designed to predict smooth images regions in a visually friendly way. It provides maximal continuity of the image plane at the block border and follows the gradual changes of the pixel values.

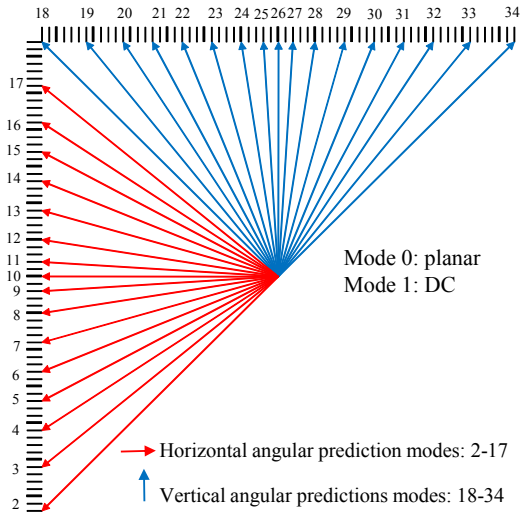


Figure 2.8: Intra prediction modes available in HEVC. Angular modes are classified into two groups: horizontal (red arrows) vertical and horizontal (blue arrows) [59].

After the prediction process, the obtained residuals are subjected to a DFT and quantization. HEVC specifies two-dimensional transforms of various sizes from 4×4 to 32×32 that are finite precision approximations of the DCT. Furthermore, HEVC also provides an alternate 4×4 integer transform based on the Discrete Sine Transform (DST) for use with 4×4 luma intra prediction residual blocks. The quantization process of HEVC is similar to the one implemented in H.264/AVC, where a quantization parameter in the range of 0-51 is used. However, some modifications to the quantization procedure were introduced due to additional transform sizes.

Contrarily to H.264/AVC that has two entropy coding methods, HEVC has only a single entropy coding method designated as Syntax-Based context-adaptive binary Arithmetic Coding (SBAC). The main differences between SBAC and CABAC (implemented in H.264/AVC) are related to the provision of parallel coding and decoding possibility. SBAC is an adaptive binary arithmetic encoding method, which employs context models (up to 27) and provides high coding efficiency of various syntax elements with different statistical properties. Each context is adaptively changed based on the data already processed.

A more detailed description of the HEVC can be found in [56, 57, 60].

2.1.3 Image decomposition

One of the key aspect in the image compression field is the way that an image is "fed" to the compression method. In this research work we used two image decomposition approaches: bitplane and binary-tree decomposition. These two techniques are detailed in the following sections.

2.1.3.1 Bitplane decomposition

The technique to separate an image into different planes (bitplanes), known as bitplane decomposition, plays an important role in image compression. Usually, each pixel of a grayscale image is represented by 8 bits. Imagine that the image has $N \times M$ pixels and each one is composed of eight bitplanes, ranging from bitplane 0 for the least significant bit (LSBP) to bitplane 7 for the most significant bit (MSBP). In fact, the plane 0 contains all the lowest order bits in the bytes comprising the pixels in the image as well as plane 7 holds the most significant bits [61]. Figure 2.9 illustrates these ideas and Fig. 2.10 shows the various bitplanes for the image presented on the left. As we can see, the MSBPs (especially 7-4) contain the majority of the visually significant data. On the other hand, the lower planes (namely planes 0-3) contribute to more subtle details in the image.

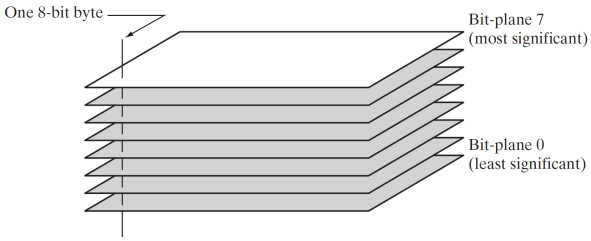


Figure 2.9: Bitplane representation of an eight-bit image [61].

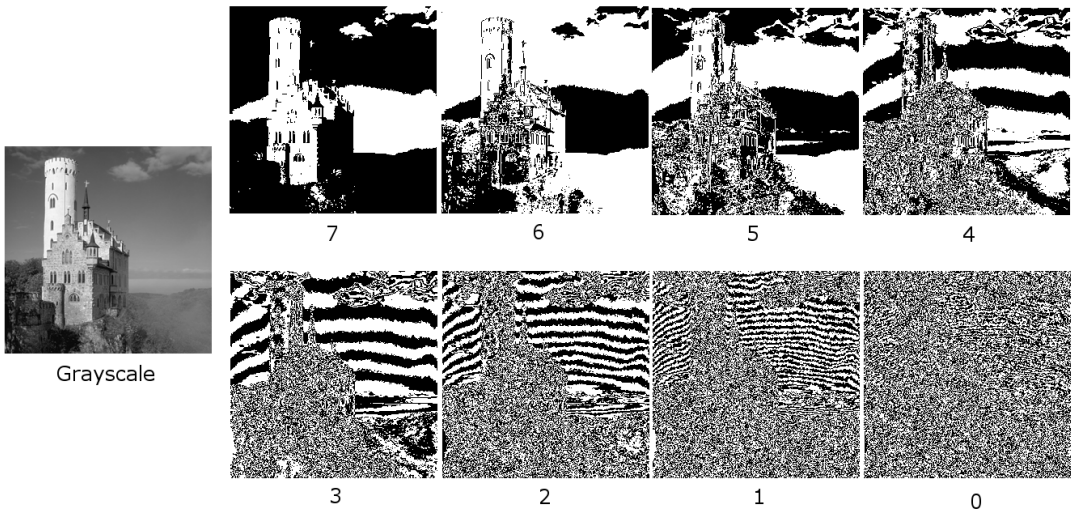


Figure 2.10: An 8-bit grayscale image and its eight bitplanes. The numbers at the bottom of each image identify the bitplane, where 0 denotes the less significant plane and 7 the most significant plane (adapted from [62]).

The bitplane decomposition technique is very useful in the image compression field. On one hand, it allows some bi-level compression methods, such as JBIG, to be applied to typical grayscale images. The compression method is applied to each bitplane after the decomposition. On the other hand, it is possible to create sophisticated models that take advantage of

this decomposition. For instance, it is possible to use information of the previous bitplanes (usually the MSBPs) to improve the compression performance of the LSBPs.

In 1999, Yoo *et al.* [63] presented an Embedded Image-Domain Adaptive Compression (EIDAC) scheme where this approach was used with success. EIDAC was developed specifically to compress simple images (with a reduced number of active intensity values), using a context-adaptive bitplane coder, where each bitplane is encoded using a binary arithmetic coder.

2.1.3.2 Binary-tree decomposition

Binary trees are also an important data structure that can be used in several algorithms. In the case of image compression, we can associate each leaf node of the binary tree to an image intensity. The binary tree can be viewed as a simple non-linear generalization of lists; instead of having one way to continue to another element, there are two alternatives that lead to two different elements [64]. Every node (or vertex) in an arbitrary tree has at least two children (see Figure 2.11). Each child is designated by left child or right child, according to the position in relation to the tree root.

One of the first methods where binary trees were used for image compression was proposed by Chen *et al.* [65] regarding the compression of color-quantized images. Chen's method uses a binary-tree structure of color indexes instead of a linear list structure. Using this binary-tree structure, Chen's method can progressively recover an image from two colors to all of the colors contained in the original image. Inspired by the work done by Chen *et al.*, a few years later Pinho and Neves [66–68] developed a lossy-to-lossless method based on binary-tree decomposition and context-based arithmetic coding. In the last approach, the authors studied the performance of their method in several kinds of grayscale images, including medical images. As can been seen, this decomposition approach is very versatile because it can be applied in color-quantized images and also in grayscale images.

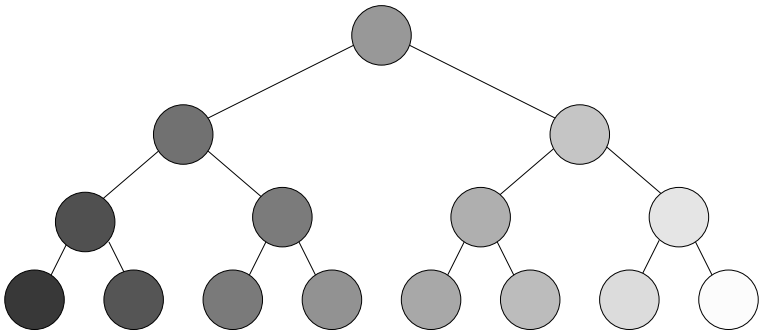


Figure 2.11: An example of a binary-tree with eight intensities or gray-levels. Each node represents an representative intensity.

2.1.4 Finite-context models

A finite-context (Markov) model of order k yields the probability distribution of the next symbol in a sequence of symbols, given a recent past up to depth k (a finite past). Markov modeling is widely used in several fields, including image [16, 66–69] and DNA [70–72] compression. Lets consider an information source that generates symbols, s, from an alphabet

$\mathcal{A}$. At instant t, the sequence of outcomes generated by the source is $x^t = x_1 x_2 \ldots x_t$. A finite-context model of an information source (see Figure 2.12) assigns probability estimates to the symbols of the alphabet ($\mathcal{A}$ in this case), according to a conditioning context computed over a finite number, k, of past outcomes (order-k finite-context model) [23, 73, 74]. At a given instant t, these conditioning outcomes are represented by $c^t = x_{t-k+1}, \ldots, x_{t-1}, x_t$. The number of conditioning states of the model is $|A|^k$, dictating the model complexity or cost. For the example illustrated in Figure 2.12, $\mathcal{A} = \{0, 1\}$ and then $|\mathcal{A}| = 2$, an order-k model implies having 2^k conditioning states. In this case $k = 5$ so the number of conditioning states is $2^5 = 32$.

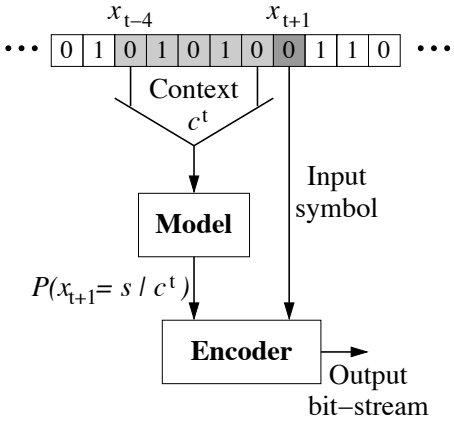


Figure 2.12: Finite-context model: the probability of the next outcome x_{t+1}, is conditioned by the k last outcomes. In this example, $k = 5$.

In practice, the probability that the next outcome, x_{t+1}, is s, where $s \in \mathcal{A} = \{0, 1\}$, is obtained using the Lidstone estimator [75],

$$P(x_{t+1} = s|c^t) = \frac{n_s^t + \delta}{\sum_{a \in \mathcal{A}} n_a^t + |A|\delta}, \tag{2.1}$$

where n_s^t represents the number of times that, in the past, the information source generated symbol s having c^t as the conditioning context and $|A| = 2$ for a binary alphabet. The parameter δ controls how much probability is assigned to unseen (but possible) events, and plays a key role in the case of high-order models. Furthermore, it is important to say that the Lidstone's estimator can be reduced to the Laplace's estimator for $\delta = 1$ [76] and to the frequently used Jeffreys [77] and Krichevsky [78] estimators when $\delta = 1/2$.

Initially, all counters are set to zero, so for the case of a binary alphabet the symbols have probability of 1/2, i.e., they are assumed equally probable. The counters are updated each time a symbol is encoded. However, it is possible to update the counters according to a specific rule. Since the context templates are causal, the decoder is able to reproduce the same probability estimates without needing additional side information.

Table 2.1 presents a simple example of how a finite-context is typically implemented. In this example, we are dealing with an order-5 finite-context model which means that the context uses the last five encoded symbols to assign the symbol probabilities. Each row of Table 2.1 represents a probability model that is used to encode the current symbol, using the

last five encoded ones. For example, if the last symbols were "11000", i.e., $c^t = 11000$, then the model sends the following probabilities to the arithmetic encoder (denoted as "Encoder" in Figure 2.12): $P(0|11000) = 28/69$ and $P(1|11000) = 41/69$.

Table 2.1: Simple example illustrating how finite-context models are implemented. The rows of the table correspond to probability models at a given instant t. In this example, the particular model that is chosen for encoding a symbol depends on the last five encoded symbols (order-5 context).

Context, c^t	n_0^t	n_1^t	$\sum_{a \in \mathcal{A}} n_a^t$
00000	23	41	64
⋮	⋮	⋮	⋮
00110	14	34	48
⋮	⋮	⋮	⋮
01100	25	12	37
⋮	⋮	⋮	⋮
11000	28	41	69
⋮	⋮	⋮	⋮
11111	8	2	10

One important aspect that must be considered is which size should the context have? If k represents the size of the context and $|\mathcal{A}| = 2$, the table size is 2^k and, therefore, the table grows exponentially with k. Using a deeper context, we might achieve higher performance, but this requires more memory (deeper context tables).

Usually, a compression algorithm can be divided into two parts, modeling and coding. The Markov models are responsible to provide a statistical model as reliable as possible to be used later in the coding stage. The coding stage is where the statistic model provided earlier is used to compress the data. Arithmetic coding is one of the many coding techniques that can be used. In the following section, we will address the arithmetic coding algorithm.

2.1.4.1 Arithmetic coding

Arithmetic coding is a compression technique developed by Jorma J. Rissanen [79] in the late 70's. This method is a good alternative to Huffman [80] coding, because usually generates better compression results. In order to obtain better results, an appropriate context must be used in the arithmetic encoder (described above). This method represents a set of symbols using a single number in the interval $[0, 1)$. As the number of symbols of the message grows, the initial interval $[0, 1)$ will decrease and the number of bits necessary to represent the codeword (the number) will increase. When we are processing the pixels of an image in a raster-scan order, the probabilities of the intensities of the pixels are conditioned by the context determined by a combination of the already encoded neighboring pixels. The encoder and the decoder estimate this context model dynamically adapting it to the input data, during the encoding/decoding process. According to [23, 73, 74], this arithmetic encoding

method generates output bit-streams with average bitrates almost identical to the entropy of the source model. The theoretical code-length average produced by the finite-context model, after encoding N symbols, is given by

$$H_N = -\frac{1}{N}\sum_{t=0}^{N-1} \log_2\Big(P(x_{t+1} = s|c^t)\Big) \text{ (bps)}, \tag{2.2}$$

where "bps" stands for "bits per symbol". Usually, when we are dealing with images, we use "bpp", which stands for "bits per pixel".

2.2 Microarray-specific compression techniques

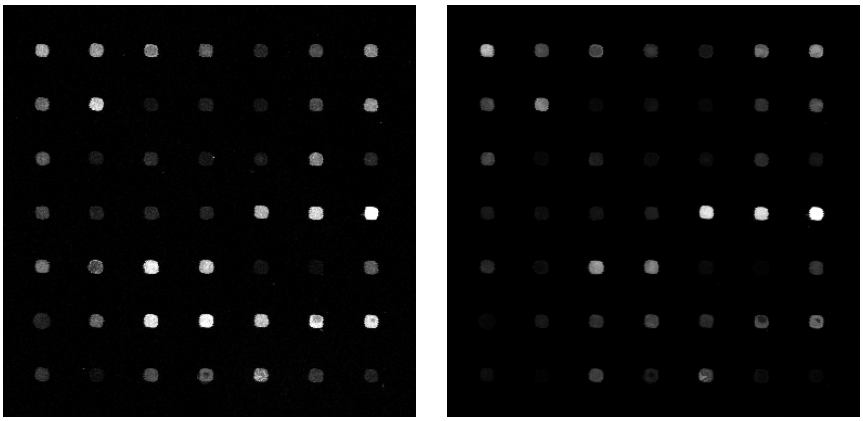
2.2.1 Microarray images

DNA microarray imaging is an important tool and a powerful technology for large-scale gene sequence and gene expression analysis, allowing the study of gene function, regulation and interaction across a large number of genes, and even across entire genomes [3, 4]. DNA microarrays are currently used, for example, for genome-wide monitoring in areas such as cancer [5] and HIV research [6].

According to [81], this imaging technique evolved from the convergence of two very different fields, namely, molecular genetics and computer electronics. In 1975, Edwin M. Southern introduced and proved the concept that DNA attached to a solid support could be used to attract complementary DNA strands. This process was known as Southern blotting [82]. In 1991, a team from Affymax company, leaded by Stephen P. A. Fodor, reported the fabrication of DNA microarrays on the surface of glass chips, by combining the photo-lithographic method used to produce semiconductors with combinatorial synthesis of oligonucleotides [83]. Later, in 1993, he co-founded Affymetrix, in order to develop microarrays with hundreds of thousands of different oligonucleotides. On year later, Affymetrix started to manufacture and selling its first DNA microarrays, GeneChip®, and the DNA microarray market was born. Since then, there has been a continuous development of methods and technologies for making microarrays more effective and precise.

The output data obtained in a microarray experiment is a pair of 16 bits per pixel grayscale images, one from the so-called green channel and the other from the red channel (see Figure 2.13). Gene expression can vary in a very wide range, justifying the need for image pixel intensities having a depth of 16 bits. Usually, these images also have a high spatial resolution, from 1000×1000 to 13000×4000 or even more, due to the microscopic size of the spots. Hence, these images may require several tens of megabytes in order to be stored or transmitted.

Although the final goal is to extract from the microarray images information related to expression levels, it is usually desirable to keep both the genetic information extracted and the original microarray experiments. The main reasons are, on one hand, the fact that the analysis techniques are still evolving and, on the other hand, because repeating the microarray experiment is expensive and sometimes even impossible. However, due to the need of running many experiments under different conditions, huge amounts of data is currently produced in laboratories all over the world. For these reasons, the need for efficient long-term storage, sharing and transmission of microarray images, is an important challenge.



(a) Green channel (b) Red channel

Figure 2.13: An example of a microarray experiment (crop portion of "Def661" image from ISREC dataset). The output consists of a pair of 16 bits per pixel grayscale images.

2.2.2 State of the art in DNA microarray image compression

2.2.2.1 Segmented LOCO (SLOCO)

According to the literature, the first work in compression of microarray images was presented in 2000 by Jörnsten *et al*. [84]. This first compression method attained good results for lossless compression but it had also a lossy mode. Even in lossless mode, the compression method only encoded a *conservative* version of the ROIs, which are composed by the spots and only a small portion of the background around them. At the time, these ROIs were considered enough to compute the gene expression level, thus the concepts of lossless and lossy were only applied to what was considered a ROI.

Jörnsten *et al*. introduced later a more structured method [85–89], know as Segmented LOCO (SLOCO). This method, as the name itself says, is based on LOCO, but it uses an extra segmentation step that is done in order to extract the ROIs that are to be encoded. In order to perform the segmentation of the spots, the method starts by estimating the spots grid, to compute their approximate center. After that, the method uses a seeded region growing algorithm for the initial spot segmentation, followed by a two-component Gaussian mixture model. The last one is used to further refine the boundaries of the spots.

SLOCO also provides a progressive transmission scheme that allows to fully reconstruct the microarray image without any loss. In this scheme, there is a step for extracting genetic information. Given an initial lossy reconstructed image, the method can refine it spot by spot, background region by background region, or even pixel by pixel, to any bitrate above the minimum bitrate necessary to decode the specified region. Initially, the method need to supply the header information which allow the user to choose which are the interesting image subsets that are required. After that, only the intended subset information is transferred. It is possible to extract the genetic information using the differential expression level, given as the log ratio of the spot intensities, spot variances and shapes, and product intensities.

In the first step of the encoding procedure, the segmentation and genetic information is transmitted first. The spot and background means are encoded using adaptive Lempel-Ziv. On the other hand, the segmentation map, obtained after the segmentation step, is efficiently

encoded using a chain code.

SLOCO encodes each image using a modified version of the LOCO-I algorithm. LOCO-I is the algorithm behind the JPEG-LS standard. The compression algorithm differs from LOCO in mainly three aspects. First, the spots and background are encoded separately. Second, a UQ-adjust (Uniform Quantizer) quantizer is used instead of a UQ quantizer. Finally, the method allows for varying the maximum pixel-wise error bounds δ. It uses a runlength code that takes the segmentation map and δ into account.

The SNR of each spot is mainly computed using the genetic information transmitted in the first step of the encoding procedure. In order to be able to have a subset reconstruction capability, the algorithm divides the image into sub-blocks. For each image sub-block, a modified LOCO is applied to the spots and background separately, with a locally determined error bound according to the SNR in question. These schemes allows the user to have a residual image from the previous compression step, that can be refined until the method reconstructs the original image (lossy-to-lossess behavior).

The lossy-to-lossless scheme uses the residual image as a reference to progressively decode the image bitplane by bitplane, from the MSBP to the LSBP. The previous strategy, progressively refines the lossy version of the lossy microarray image, as well as the residual one. The quality of the microarray image lossy reconstruction is controlled/defined by how much is coded from the residual image. In the limit of the lossy reconstruction, the method builds the full lossless version of the image. In [85], the authors used a vector quantized residual image instead of a scalar one. However, this approach does not allow to have an error bound in the final reconstruction.

2.2.2.2 Hua's method

In 2002, Hua *et al.* proposed a fast transform based microarray compression method that uses a modified Mann-Whitney test-based algorithm, to perform the segmentation [90]. Later in 2003, the same method with an extra segmentation preprocessing step was used in a microarray compression and analysis framework named BASICA [91, 92].

As mentioned before, the segmentation used by Hua *et al.* is done using a modified Mann-Whitney algorithm that is also described in [90]. The Mann-Whitney algorithm is an iterative threshold algorithm, which is used to rank various intensity distributions in order to perform the segmentation of the microarray image. According to the authors, this modified version has increased speed over the previous version. The main reason to this high performance in terms of speed is due to the iterative process that converges much faster. The algorithm begins with a carefully chosen predefined threshold mask and iteratively adjusts the threshold while the Mann-Whitney test holds.

It is common to have undesirable shape irregularities after the segmentation process. Those irregularities are usually seen in the spot edges and can severely reduce the compression efficiency. In order to overcome this problem, a post-processing step is applied before the segmentation in [91, 92]. In order to remove the irregularities, BASICA uses an operation similar to the standard morphological pruning [93]. For isolated noisy pixels or tiny regions, BASICA can detect and remove them directly from the foreground information. The segmentation information is coded separately using a chain code.

After the segmentation, the compression method uses a modified version of EBCOT (Embedded Block Coding with Optimized Truncation) [2] (also used in [90]). The original version of EBCOT was first used in the JPEG2000 standard. The modified EBCOT algorithm is an

object-based wavelet transform coding scheme. A shape-adaptive integer wavelet transform is applied first both to the background and foreground, independently, in order to prevent any interference between the coefficients from adjacent areas. After applying the shape-adaptive integer wavelet transform, they used a modified EBCOT context modeling for arbitrarily shaped regions (object-based coding support). This alternative EBCOT method allows to code spots and background separately, a characteristic that is not available in the original version.

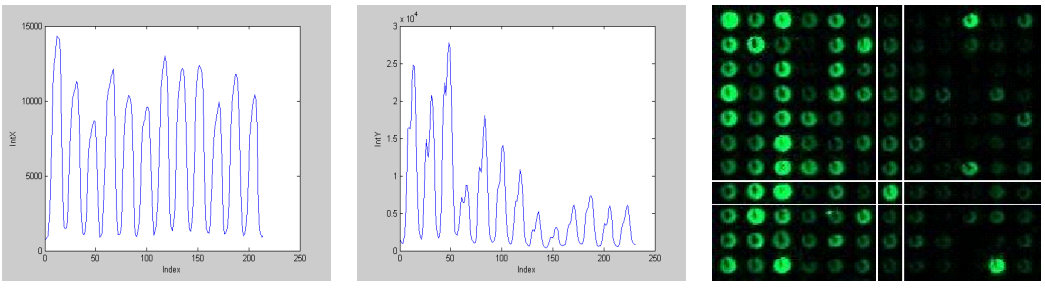
Moreover, it is possible to perform lossless coding of the foreground and lossy-to-lossless coding of the background separately, thanks to the modified EBCOT algorithm that supports object-based coding. A lossy-to-lossless coding of the spots can also be used if desired.

2.2.2.3 Faramarzpour's method

Faramarzpour and Shirani presented in [94] a compression method for microarray images that uses a two step segmentation procedure. In the first step, the spot regions are located using the period of the signal obtained from summing the intensities by rows and by columns. The minimum sum values are used to define the spot limits, where each one of them will be isolated into individual ROIs. Considering a $M \times N$ input image I, where $I(i,j)$ is the pixel intensity at the position (i,j), two signals are computed according to

$$Int_x(i) = \sum_{j=1}^{N} I(i,j), \;\; Int_y(j) = \sum_{i=1}^{N} I(i,j), \tag{2.3}$$

where each element of $Int_x(i)$ and $Int_y(j)$ represents the average of a row and a column respectively. In Figure 2.14, we can see the corresponding integrals in the two left charts and the microarray image used (c). A Discrete Fourier Transform (DFT) is then applied to those two signals (Int_x and Int_y) in order to estimate their periods. Using the local minima of Int_x and Int_y, it is possible to estimate the period and form two vectors which entries are the rectangular regions where the spots are located. A typical spot sub-image is presented in Figure 2.14 (c). After having all the spots extracted, the authors used a new



(a) (b) (c)

Figure 2.14: (a) Int_x and (b) Int_y for the microarray image I show in (c). The white lines in (c) show how spot sub-images are extracted [95].

scanning method, appropriated for images with circular or central behavior. The goal of this circular scanning method, known as "spiral path", is to transform the 2-dimensional ROIs (the extracted microarray spots) into 1-dimensional data with minimal number of transition.

The "spiral path" is initially centered in the spot "mass center" (usually associated with the pixel with highest intensity), and optimized to minimize the first order entropy. The purpose of this optimization is to reduce the transitions that occur when the path reaches the spot border (see Figure 2.15). A typical raster scan approach creates a path that enters and exits the spots several times during the process. Since the spots have a circular shape, the "spiral path" scanning method reduces the number of transitions during the process. Sometimes, the shape of the spots deviates considerably from the circular shape, which can affect the compression performance of the method. Irregularities in the shape of the spots causes more transitions. Moreover, if the center of the "spiral path" is not chosen carefully, an edge effect occurs, more transitions will also occur near the spot edge, causing a significant reduction of the coding efficiency.

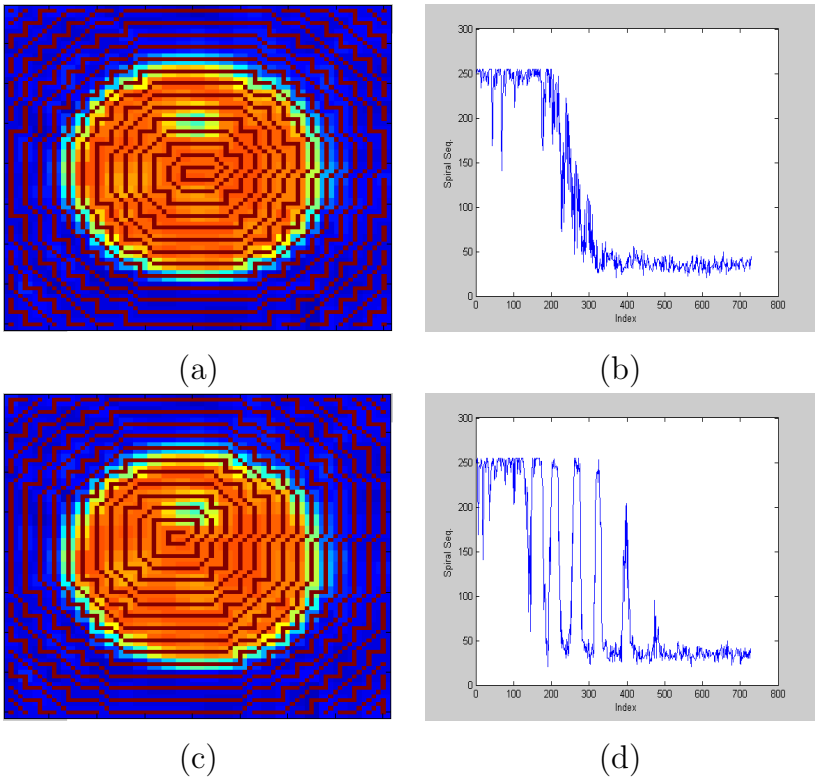


Figure 2.15: (a) Spiral path superimposed on a spot, (b) spiral sequence, (c) mismatched spiral, and (d) edge effect along the path [95].

Then, a special form of liner is applied prediction to the pixel intensities while moving along the spiral path previously determined. The prediction is done using the already coded pixels spatially close. Unlike typical prediction coding schemes, the number and arrangement of the neighbors used in the prediction of the pixel value is dynamic and changes depending on the position of the pixel in the spiral path. The prediction can be built using pixel neighbors that have the same distance from the center as the pixel that is being coded. This means that the pixels used in the prediction are not necessarily from the immediately previously pixels of the spiral path.

The residual sequence obtained from the previous step is entropy encoded using Adaptive

Huffman codes. The authors showed that the residual sequence has different statistical properties between the spots and the background regions. In order to improve the compression results of their method, the algorithm divides the residual sequence into spots and background, and code them separately. The split criterion is defined by the different statistical behaviors, thus reducing the residuals entropy.

The method proposed in [95] has also a lossy version. Contrarily to other methods where the lossy mode is an adaptation of the lossless method, in this case it has an algorithm of its own, rather than a lossy-to-lossless version of the lossless algorithm. Despite this, the lossy algorithm has the same basic ideas and even share some of the steps of the lossless version.

Similar to the lossless version, the lossy version starts by the spot extraction procedure. Then, instead of assigning a spiral path, the method assigns a circle to each spot, with a circle center and radius being optimized to best fit the spot. After assigning the circles to each spot, a Circle to Square (C2S) transform is applied to the previously determined circles. The square images obtained are put together and divided into 8×8 blocks. Finally, the blocks are lossy compressed be means of DCT transform, quantization, and entropy coding. The information loss occurs in two phases of the algorithm. The first loss happens when the circles are assigned to each spot where they simply remove the background. The second loss occurs in the quantization of the DCT coefficients.

2.2.2.4 MicroZip

Lonardi and Luo [96] proposed in 2004 a lossless and lossy compression algorithm for microarray images, known as "MicroZip". Their method uses an automatic procedure to determine the grid and the spots that are present in the microarray image. This procedure does not require any kind of input from the user like the grid geometry, number of rows/columns, etc. The only thing required to obtain the grid and the spots is the microarray image itself. Initially, the grid is obtained in two steps. In the first step, the griding algorithm computes the average intensities row-by-row and column-by-column on the whole image (similar to $Int_x(i)$ and $Int_x(j)$ presented earlier in Faramarzpour's method). Due to the noise presence that is typical in microarray images, the authors applied a low-pass filter to the obtained signals in order to remove the noise. The smoothing process uses a rectangular windows with 25 pixels for the subgrids search and 4 pixels when searching for the spots. Typically, there is a considerable background space between subgrids so the smoothed signal should have minima in those regions. This minimum values are used to create the first estimate of the subgrid structure. Inside each subgrid, a similar procedure (but with a 4 sample filter) is used to isolate each spot. Using the same signals computed earlier (the minimum values), a value B is calculated as the average of the auxiliary signals minima. B is then used as a reference background threshold to classify the pixels as foreground or background. It is important to say that this griding algorithm only works for images where the grids are perfectly aligned with the image border. The algorithm does not work on a different spot layout which is found in the *Arizona* and *Omnibus* data set, for example (see Appendix A). The final stage of the segmentation procedure is to identify the segmented spots. The authors used an adaptive shape segmentation method, such as seeded region growing. A circle center is chosen taking into account the average of the foreground pixel coordinates for the spot. The limit of the circle (the radius) is progressively computed until the average of the pixels inside the circle is slightly bellow B.

After the segmentation procedure, the pixels are assigned to the foreground (spots) or

background channel accordingly. The previous channels are then divided into two subchannels: the eight most significant bits (MSByte) and the eight least significant bits (LSByte). The previous four channels are then entropy coded using the Burrows-Wheeler Transform (BWT) [97]. In addition to the four channels (see Figure 2.16), MicroZip needs to code some header information that will contain griding and segmentation information. In the end, the four channels and header information are combined together in a final compressed file.

This format, where the image information is split into several channels is very useful, namely if a lossy compression scheme is desired. The MicroZip creators only used a lossy compression approach in the LSByte of the background. The authors tried several lossy methods to encoded the LSByte of the background channel, and in the end they used a method called SPITH [98] (Set Partitioning in Hierarchical Trees). SPITH is a very effective image compression method based on a partitioning of the hierarchical trees of the wavelets decomposition. In order to effectively apply the wavelets decomposition, the authors converted the one dimensional data stream into a two-dimensional image. Because the wavelet based methods work best in perfectly square images, the dimensions x and y of the obtained image are chosen such that $|x - y|$ is close to zero as possible.

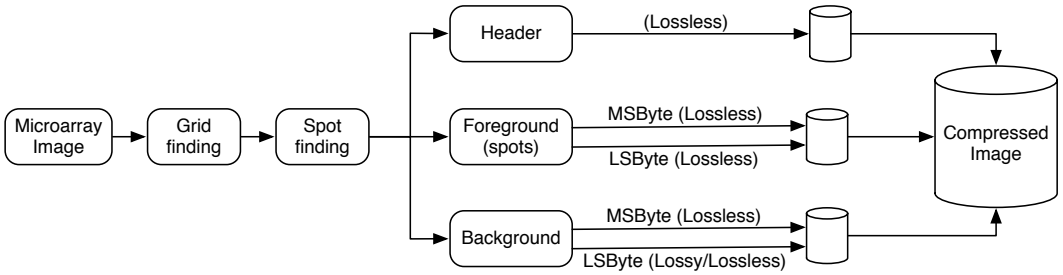


Figure 2.16: Flow chart of MicroZip [96].

2.2.2.5 Zhang's method

In 2005, Zhang *et al*. [99] proposed a compression method based on PPAM (Prediction by Partial Approximate Matching). PPAM [100] is a method for compression and context modeling for images which is an extension of the PPM text compression algorithm, considering the special characteristics of natural images. Unlike the traditional PPM modeling method, where the exact contexts are used, PPAM introduces the notion of approximate contexts.

Initially, the input microarray image is split into foreground (spots) and background. After that, for each component (channel) the pixel representation is separate into its most significant and least significant parts (see Figure 2.17). Then, the most significant parts of both components are first processed by an error prediction scheme and the obtained residual values are encoded using the PPAM context model and encoder. On the other hand, the least significant parts of both components are feed directly to the PPAM context model and encoder. The reason to not pass the least significant parts through an error prediction scheme is because of its random nature. The error prediction scheme would probably increase the entropy of the image part. The header information, which contains the segmentation information, is saved without compression.

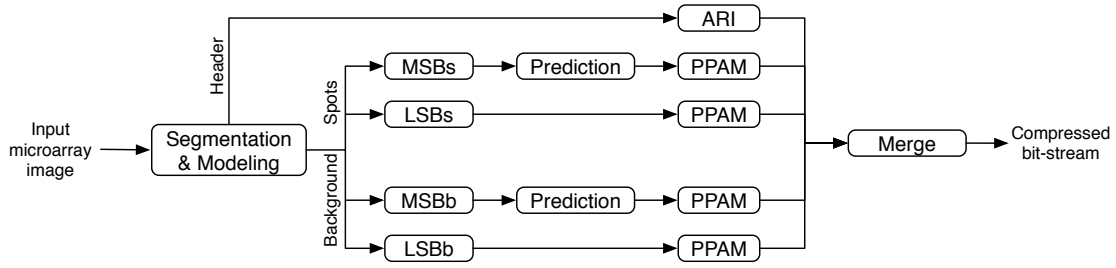


Figure 2.17: Microarray image compression scheme presented in [99]. ARI stands for arithmetic coding.

2.2.2.6 Neekabadi's method

In 2007, Neekabadi *et al.* [101] introduced a lossless image compression method which segments the pixels of a microarray image into three categories: background, foreground and spot edges. According to the authors, the third region (the spot edges) causes large errors in prediction based methods, thus they decided to use a different predictor for each region.

In Figure 2.18, we can see the block diagram structure of Neekabadi's method. The input image first enters in the Segmentation Unit where it is split into foreground and background. After that, the foreground is also divided into edges and spot regions. Hence, the output of the Segmentation Unit has three masks. The Compression Units presented in Figure 2.18 are responsible to perform a two-dimensional prediction and a statistical coding routine (Huffman coding). The last block, named in Figure 2.18 as "Mask Compressor" is a dedicated compressor for the segmentation map.

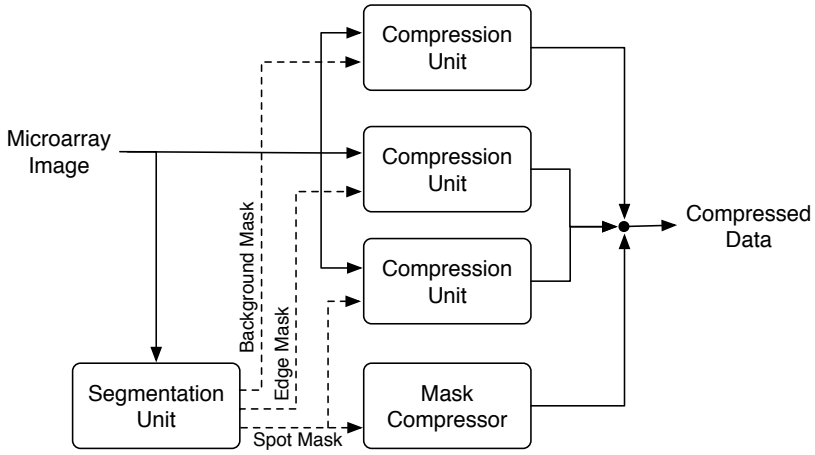


Figure 2.18: Main functional units of Neekabadi's method [101].

The first step of Neekabadi's method is the segmentation of the image into three distinct regions (described in Figure 2.19). They perform the segmentation using a dynamic thresholding scheme. By applying a threshold value, the pixels of the image can be split in two sets (background and foreground). For each threshold value, it is possible to obtain the number of pixels and the standard deviation of the intensities of these pixels in each set. The de-

sired threshold is calculated according to (2.4). Using (2.4), it is always guaranteed that the weighted sum of the standard deviation of both the background and foreground is minimal,

$$\mathcal{T} = \text{argmin}\{f(T)\},\ T = \{t \in \mathbb{N} \mid 0 \leq t \leq 2^{16} - 1\}, \tag{2.4}$$

where $f(T) = \text{stdev}(B_T) \times \text{size}(B_T) + \text{stdev}(F_T) \times \text{size}(F_T)$, $B_T = \{p \in \text{Image} \mid p < T\}$ represents the set of pixels in the background section, $F_T = \{p \in \text{Image} \mid p \geq T\}$ represents the set of pixels in the foreground section, $\text{stdev}(x)$ is the standard deviation of x and $\text{size}(y)$ is the number of pixels of set y. Instead of testing all possible threshold values to find the minimum value of $f(T)$, the authors used a recursive search algorithm which accelerates the search routine. It is possible to use this recursive search algorithm because $f(T)$ plunges down at a certain threshold value, which is chosen as the final threshold value. They also mention that the $f(T)$ plots are similar between different microarray images.

After the threshold search is completed, a binary map is created where the foreground pixels will be set to 1 and the background pixels to 0. After having the binary mask, an erosion operation is applied to it in order to remove isolated points (pixels) in the mask. This process is depicted in Figure 2.19. To obtain the spot's edges, a morphological dilation process is performed on the output of the erosion step and the result is subtracted from the outcome of the erosion step. The outcome of that subtraction gives the edge mask. After having the three masks, it is possible to segment the whole image into the three regions: foreground, background, and spots boundaries. The obtained masks are used in the compression units depicted on Figure 2.18.

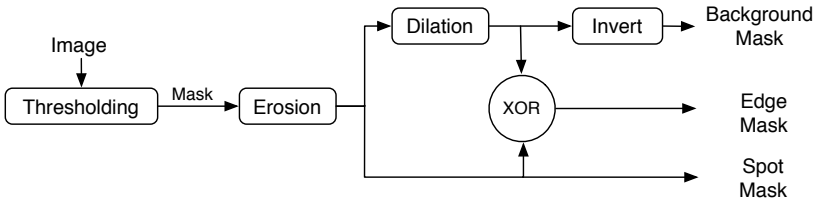


Figure 2.19: Block diagram of the Segmentation Unit presented in Figure 2.18.

In Figure 2.20, we can see an example of a segmentation performed on a part of a microarray image (depicted in (a) of Figure 2.20). The background, foreground, and spots masks are illustrated in parts (b), (c), and (d) of Figure 2.20, respectively. After analyzing Figure 2.20, it is easy to conclude that the edge and background masks can be created from the spot masks. Thus, the spot mask is the only mask that needs to be coded. The spot mask is compressed using first RLE and then Huffman coding.

The pixels of each region are independently compressed using different predictors. The function of the predictor is defined by the region where the pixel belongs and also by the location of the pixel inside the region. The function of the predictor used by the author is

$$\hat{x} = \lfloor k \times m \rfloor, \tag{2.5}$$

where $\hat{x}$ denotes the predicted value of a given pixel x, m corresponds to the mean of the neighbors of x, and k represents a robust linear regression coefficient [102]. The utilization of this coefficient will guarantee that the sum of the squared error is minimized as far as possible. For each region, a k value is pre-computed in a first scan and then used in the second scan of the image. Then, the CCSDS (Consultative Committee for Space Data System) [103]

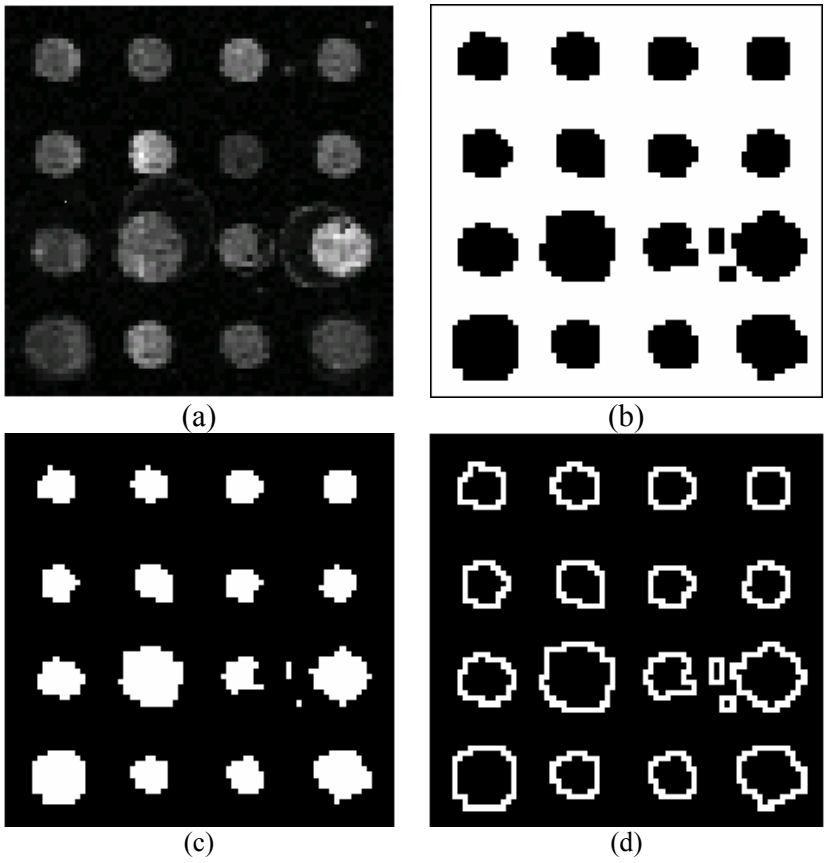


Figure 2.20: Segmentation results obtained by the Neekabadi's method [101] for a part of a microarray image. (a) Original Image, (b) Background mask, (c) Spot mask, and (d) Edge mask.

algorithm is applied to the produced prediction errors. This algorithm ensures that the prediction error values are all turned into positive numbers. In the end, Huffman coding is applied to the output of the CCSDS algorithm.

2.2.2.7 Battiato's method

In 2009, Battiato and Rundo [104] proposed an approach based on Cellular Neural Networks (CNNs). In the first stage of their method, the microarray image is efficiently split into foreground (image spots) and background. The segmentation process is not an easy task and can sometimes introduce some undesired effects. Those undesired effects are mainly caused by the presence of background noise, irregular spots, low spot intensity, which causes often a confusion between foreground and background [91]. In order to overcome the previous issues, their segmentation method uses CNNs to perform an adaptive non-invasive segmentation by making use of some stability properties of the CNNs. This segmentation pipeline is the same for both channels of the microarray image. For each microarray image, they define two layers, each one with as many cells as the image pixels. The input and the state of the first layer is defined by the pixels of the original image. Its output (from the first layer) will be used as input of the second layer. The whole CNNs dynamics is driven by the defined cloning

templates, which tends the second layer toward its saturation equilibrium state. The result output tends to become a "local binary image" where the spot pixels tend to white while the background pixels tend to black (see Figure 2.21).

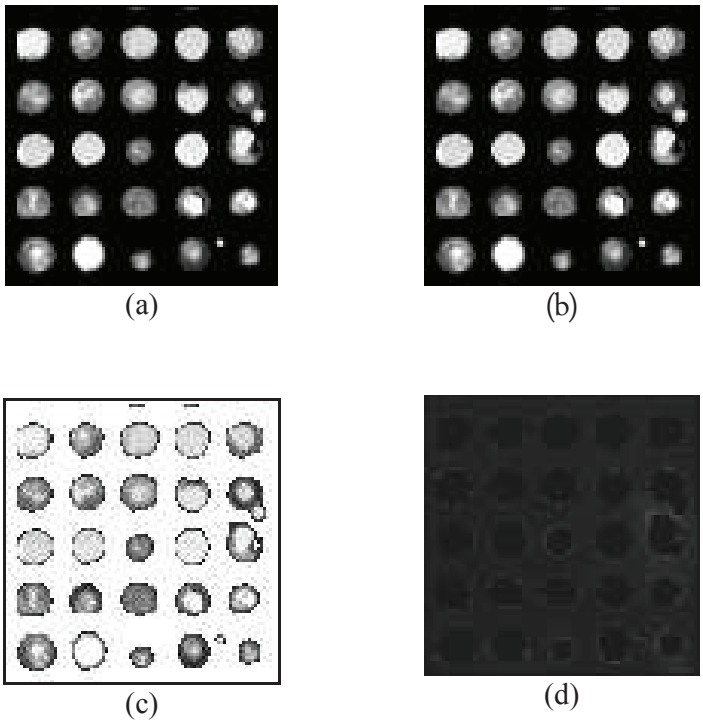


Figure 2.21: DNA microarray image portion segmented by CNN: (a) source (b) DNA microarray processed by CNN (background is all zero) (c) foreground and (d) background extracted by the proposed CNN pipeline presented in [104].

After the segmentation process, the obtained foreground sub-images are lossless compressed by means of general purpose codecs. The authors used the standard PNG codec based on LZ77 algorithm. Regarding the background sub-image, Battiato and Rundo used an innovative method based on the theory of palette re-indexing [105]. Re-indexing strategies cannot be directly applied to microarray images, because they are generally composed of two 16-bits images. In order to overcome this problem, the authors used an alternative approach in order to be able to use re-indexing. First, the background sub-images of both channels are properly scaled by a constant k_b (they used $k_b = 72$ for their simulations) in order to obtain two 8-bit sub-images. After that, a 24-bits RGB image is created taking the two 8-bits sub-images obtained earlier as the two first layers (red and green) and the third layer (blue) set to zero. Then, the re-indexing pipeline re-maps the 24-bits image with the goal of reducing the "color" palette size associated to this image. The authors proposed a swap-based algorithm in order to be able to reduce the overall number of "colors" (palette length). After having all the indexed images, their method applies one of the existing re-indexing algorithms [105] to reduce the overall residual entropy. In the end, the obtained set of re-indexing images is also compressed by means of lossless codec engine and once again the authors use the PNG codec.

2.2.2.8 Neves' method

In 2006, Neves and Pinho [106] presented a compression method based on a bitplane decomposition approach that uses a 3D finite-context model to drive the arithmetic coder. Their method was inspired by EIDAC [63], which is a compression method that has been used with success for compressing images with a reduce number of active intensities.

The method processes the microarray image on a bitplane basis, starting from the most significant bitplane (MSBP) and stopping at the least significant bitplane (LSBP). If the encoder detects that the current bitplane will require more than one bit per pixel for encoding, the encoder will swap the encoding approach where the rest of the bitplanes are sent uncoded (no compression). For the first bitplane, the compressor uses a causal context model to drive the arithmetic encoder created from the pixels values of the current bitplane (see Figure 2.22). For the other bitplanes, the causal context model uses pixels both from the bitplane currently being encoded (C_{intra}) and from the previous bitplanes already encoded (C_{inter}). In order to avoid the degradation in compression due to, in general, the LSBPs of microarrays images being close to random, the authors proposed a different context model for the last 8 LSBPs. For those bitplanes, the compression method uses a context model only formed by pixels from the upper bitplanes, as can be seen in Figure 2.23 (e).

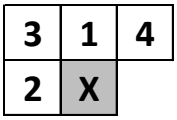


Figure 2.22: Context configuration used in method [106] for the intra-plane context (C_{intra}). This context is not used for the eight LSBPs.

In 2009, they improved their method using image-dependent context models that are built with the goal of finding the "best" context configuration to encode each bitplane, based on the 3D context templates of Figure 2.23 [107]. A brute-force approach where all possible context configurations are tested, is virtually impossible, due to the huge number of possibilities and the amount of time required to encode the image. In order to overcome this problem, the authors used a greedy approach to find which context configuration maximizes the compression ratio.

The greedy approach consists on testing several context configurations before encoding each bitplane. Before encoding each bitplane, the algorithm constructs the most appropriate context configuration using an iterative process where a bit context is added according to Figure 2.24 (a) and Figure 2.24 (b) if, and only if it improves the compression performance for the current bitplane. The configuration of the context bits for a particular bitplane is sent to the decoder in order to be possible to use the exactly same context configuration in the decoder. Being a greedy approach, it is not guaranteed that the "best" context is found. However, it is a very reliable alternative when compared to the brute-force approach where all context configurations are tested.

2.2.2.9 Other methods

Along the literature we can find several specialized methods for microarray image compression. Some author considered the lossy approach as a reasonable possibility as can be seen in [84, 108–114]. Other authors, proposed methods that have both possibilities i.e., they

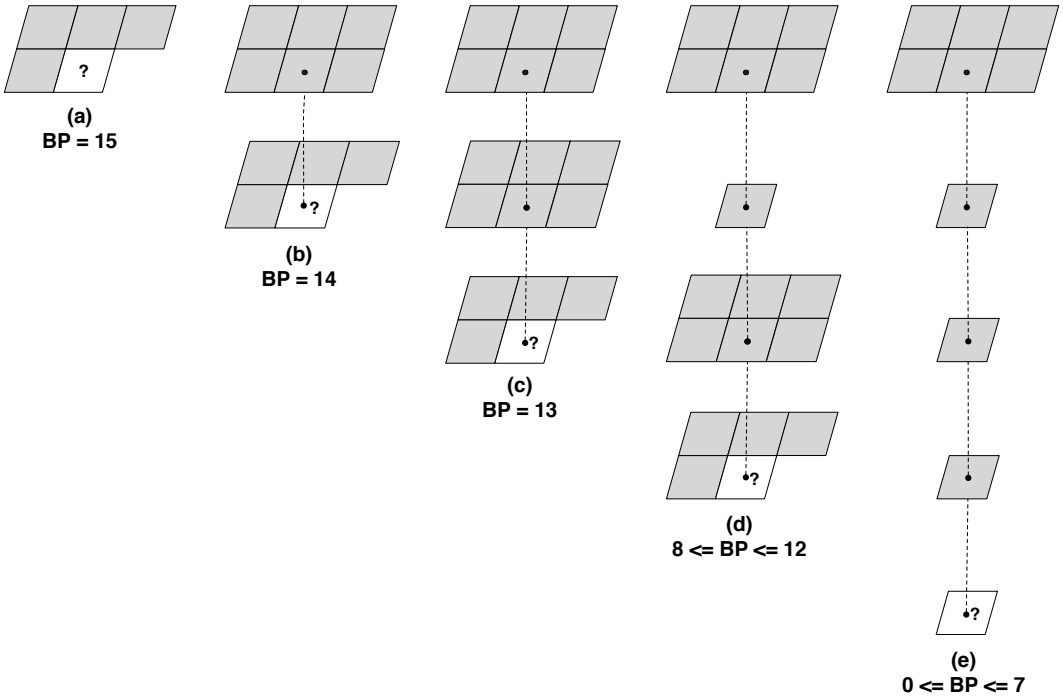


Figure 2.23: Context configurations used by the Neves' method [107] at five different compression stages.

	10	6	9	
8	4	2	3	7
5	1	X		

(a) Intra-context

6	3	7
2	1	4
9	5	8

(b) Inter-context

Figure 2.24: (a) Template used at the level of the bitplane currently being encoded; (b) Template used for growing the context corresponding to the bitplanes already encoded.

support lossless and lossy compression [85, 86, 89, 91, 92, 95, 96, 115]. In this case, the authors can in one hand perform a lossless compression of the ROI's (the microarray spots) and a lossy compression of the background. On the other hand, the authors implemented two separate modes of their compression method. One where the compression is performed without any loss and the other where some loss is tolerated. However, there are other authors that defend that information loss in the case of microarray images can affect the analytic methods used to extract information, so they only focused their methods on reversible techniques [94, 99, 101, 104, 106, 107, 116, 117]. Moreover, other approaches have benn proposed more recently [118–120] for microarray image compression. In what follows, we will provide a brief description of these recent methods.

In [118], Hernández-Cabronero *et al.* analyze the relationship between the histograms of DNA microarray images and the performance of JPEG2000. They tested the performance impact of using different number of quality layers and DWT decomposition levels on JPEG2000.

The authors concluded that some improvements are attained when using one quality layer and five DWT decomposition levels. Hernández-Cabronero *et al.* also implemented a reversible transform based on histogram swapping, which transforms the images in a version close to JPEG2000 assumptions for context modeling. With this modification, the performance of JPEG2000 was improved from 1.97% to 15.53%.

Hernández-Cabronero *et al.* had also studied the correlation present between the pairs of DNA microarray images in [119]. The correlation among pairs of images was analyzed using the Pearson's r as a metric. According to the authors, a certain amount of correlation is found, specially for the red/green channel image pairs (average of Pearson's r values over 0.75 were attained for all data sets used by the authors). Taking into account the correlation values found, they used the multi-component compression feature of JPEG2000, considering different spectral and spatial transformations to improve the compression performance of JPEG2000. Improvements of up to 0.6 bpp were obtained, depending on the transform considered. The obtained compression performance is consistent with the correlation values observed.

The most recent work in microarray image compression was proposed by Koc *et al.* [120]. They showed that the application of the Inversion Coder after the Burrows-Wheeler Transformation (BWT) along with the Run-Length Encoding (RLE) and the adaptive context-modeled binary arithmetic coder yields a compression gain of 5% and 25.5% over the entropy coders Bzip2 and BAC, respectively. When compared to generic image compressors (such as CALIC, JBIG, and JPEG-LS), the average improvement is about 6.5%.

In [121, 122] Hernández-Cabronero *et al.* reviewed the state-of-the-art in DNA microarray image compression. They described the most relevant approaches published in the literature and they also classified them according to the stage of the typical image compression process.

The DNA microarray images are an intermediate product of a DNA microarray experiment. In order to obtain the information about the genetic expression intensities, image analysis is required. The image techniques used are continuously being developed and are not fully mature or universally accepted [123–126]. Due to this, the image techniques used for extracting genetic information can change in the future which means that it will be desirable to have both the genetic data and the microarray image experiment in order to evaluate new methods. Other alternative would be to repeat the whole experiment, but in this case it is not a viable option due to the fact that biological samples are usually not available. The most reasonable option here is to always store the DNA microarray images (without any loss) along with the extracted genetic data. Despite all this, Hernández-Cabronero *et al.* [127] proposed a novel microarray-specific distortion metric to assess the loss of relevant information. This distortion metric, known as Microarray Distortion Metric (MDM), takes into account the basic image features employed by most DNA microarray analysis techniques.

The metric is computed taking into account three main features: the mean intensity ratio of the spots, the average intensity of the local background and the global image intensity. This novel distortion metric is, in our opinion, interesting because it takes into consideration the key image aspects used in the extraction of the genetic information. However, it is not flawless because if we create a copy of the original image where we would change the pixel intensities in a way that the three key aspects mention earlier were the same, we would obtain a "false" distortion value. In our opinion, while the image analysis techniques used to extract the genetic information of the DNA microarray images are not fully mature, lossy compression is not an option. For those reasons, we focused this work in only lossless compression methods.

2.3 Summary

In this chapter, we described the state-of-the-art standards that allow coding of digital images, namely JBIG, PNG, JPEG-LS, and JPEG2000. Then, we provided a brief description of the intra-mode coding of two of the most recent and sophisticated video coding standards, H.264/AVC and HEVC. Despite H.264/AVC and HEVC are primarily used for video compression, their intra-mode can be used to compress still images. Furthermore, we also described two image decomposition approaches commonly used in the image compression field. We also described in some detail the finite-context models that are the core of the proposed methods of this research work. Finally, in the last part of this chapter, the main specific-microarray compression methods that can be found in the literature were presented.

"Humility is the only solid foundation of all the virtues"

Confúcio

3

Lossless compression of microarray images

This chapter is based on:

- **L. M. O. Matos**, A. J. R. Neves, and A. J. Pinho, "Lossy-to-lossless compression of biomedical images based on image decomposition", in *Applications of Digital Signal Processing through Practical Approach*, Ed. S. Radhakrishnan, InTech, pp. 125–158, October 2015.
- **L. M. O. Matos**, A. J. R. Neves, and A. J. Pinho, "A rate-distortion study on microarray image compression", in *Proceedings of the 20th Portuguese Conference on Pattern Recognition, RecPad 2014*, Covilhã, Portugal, October 2014.
- **L. M. O. Matos**, A. J. R. Neves, and A. J. Pinho, "Compression of microarray images using a binary tree decomposition", in *Proceedings of the 22nd European Signal Processing Conference, EUSIPCO-2014*, pp. 531–535, Lisbon, Portugal, September 2014.
- **L. M. O. Matos**, A. J. R. Neves, and A. J. Pinho, "Compression of DNA microarrays using a mixture of finite-context models", in *Proceedings of the 18th Portuguese Conference on Pattern Recognition, RecPad 2012*, Coimbra, Portugal, October 2012.
- **L. M. O. Matos**, A. J. R. Neves, and A. J. Pinho, "Lossy-to-lossless compression of microarray images using expectation pixel values", in *Proceedings of the 17th Portuguese Conference on Pattern Recognition, RecPad 2011*, Porto, Portugal, October 2011.
- **L. M. O. Matos**, A. J. R. Neves, and A. J. Pinho, "Lossless compression of microarray images based on background/foreground separation", in *Proceedings of the 16th Portuguese Conference on Pattern Recognition, RecPad 2010*, Vila Real, Portugal, October 2010.

In this chapter, we address the problem of microarray image compression. We start with a study regarding the compression of microarray images using the most common image coding standards such as JBIG, PNG, JPEG-LS, and JPEG2000. Then, several microarray-specific compression algorithms are also introduced. These algorithms are based on the two decomposition approaches described earlier on Section 2.1.3 of the previous chapter. We also used some pre-processing techniques, such as segmentation and histogram reduction, in order to improve the compression results. All the methods introduced in this chapter are lossless with progressive decoding capabilities, in other words, they are lossy-to-lossless methods. Due to this particular feature, in the end of the chapter, we provide a rate-distortion evaluation of some of the proposed methods and also two image coding standards, JBIG and JPEG2000. The compression tools presented in this chapter are available online at `http://bioinformatics.ua.pt/software/ment` or `https://github.com/lumiratos/ment` for testing.

3.1 Microarray image data sets

Along the last years, several microarray image data sets have been used to report compression results. One of the hurdles that often the researcher had to face is the lack of a reasonably consensual data sets with which the performance of the algorithms can be measured.

In the literature several data sets have been used to evaluate the performance of microarray image compression methods. Pinho *et al.* [128] used a benchmarking data set composed of 49 publicly available microarray images from three different data sets, to have been used in most of the works published. However, it is natural that new images, obtained using more recent technologies, need to be added to the benchmarking data set. Recently, Hernández-Cabronero *et al.* [121, 122] extended Pinho *et al.* work to other data sets that were not considered. In this research work, we used seven microarray image data sets as can be seen in Table 3.1. In Appendix A we show several representative images of each data set.

Regarding the *Omnibus* data set, we decided to split it into two sub-data sets: Low Mode (LM) images and High Mode (HM) images. The LM and HM images are associated with the same experiment but they were scanned using two different modes: High or Low. The scanning mode affects the properties of the obtained images mainly in terms of entropy and percentage of active intensities (see Table 3.1 for more details). In Table 3.1 we show the number of images, the approximate size of the images, the first order entropy, and also the average percentage of active pixel intensities of each data set. All measures were obtained taking into account the different sizes of the images, i.e., they correspond to the total number of bits/intensities divided by the total number of image pixels.

It is easy to conclude that the *ApoA1* and *ISREC* data sets are the ones with higher entropy, suggesting that they may contain more background noise than the other data sets. On the contrary, the *Omnibus (LM)*, *Stanford*, and *Yeast* data sets have the lowest average entropy, probably because they are less noisy. Moreover, taking into account the entropy values, it seems probable that the *Yeast* data set will attain the lowest compressed bitrate among the data sets.

The percentage of intensity usage, i.e., the intensities that occur indeed in the images, is also presented in Table 3.1. It is clear that the percentage of active intensities is lower than 40% for all data sets except *Arizona* and the two *Omnibus* data sets. Along the data sets, we can see that there are some of them where almost all the intensities available are used and in other ones where a small percentage is used (approximate 5% in the case of the *Yeast* data

set). These differences will cause different compression performance between the data sets.

Another interesting measure, which usually is not referred when presenting image data sets, is sparsity. Hurley *et al.* [129] compared several commonly used sparsity measures, based on intuitive attributes. They tested several sparsity measures and verified if whether or not they satisfy the six proposed propositions. According to them, only two of those measures satisfy all six prepositions: the *pq*-mean, with $p \leq 1, q > 1$, and the Gini Index (GI). Later, Zonoobi *et al.* [130] explored the use of GI as a measure of sparsity using synthetic and real signals/images. According to them, GI is a more reliable and robust alternative to the popular ℓ_p (pseudo-) norm-based (for $0 < p \leq 1$) sparsity measure. Hence, we decided to use the GI as a sparsity measure in this research work.

Table 3.1: Microarray image data sets used in this work. The number of images represents the total number of images that each data set contains (each image corresponds to one channel). More information regarding the data set illustrated in this table can be found in Appendix A.

Data sets	**Year**	**Images**	**Approximate size (cols × rows)**	**Average entropy (bpp)**	**Average intensity usage (percentage)**	**Gini Index [0 − 1]**
ApoA1	2001	32	$> 1044 \times 1041$	11.038	39.507%	0.494
Arizona	2011	6	$= 13800 \times 4400$	9.306	82.821%	0.774
IBB	2013	44	$= 2019 \times 6235$	8.503	54.072%	0.806
ISREC	2001	14	$= 1000 \times 1000$	10.435	33.345%	0.710
Omnibus (LM)	2006	25	$= 12200 \times 4320$	5.713	50.130%	0.726
Omnibus (HM)	2006	25	$= 12200 \times 4320$	7.906	98.076%	0.892
Stanford	2001	40	$> 1900 \times 2000$	8.306	27.515%	0.615
Yeast	1998	109	$= 1024 \times 1024$	6.614	5.391%	0.518
YuLou	2004	3	$> 1800 \times 1900$	9.422	36.906%	0.556
Overall	–	298	–	7.415	67.003%	0.782

Given a vector $\vec{v} = [v_1\ v_2\ v_3\ \ldots\ v_N]$, with N ascending values, the GI is given by

$$GI(\vec{v}) = 1 - 2\sum_{k=1}^{N} \frac{v_k}{||\vec{v}||_1}\left(\frac{N-k+1/2}{N}\right), \tag{3.1}$$

where

$$||\vec{v}||_p = \left(\sum_{k=1}^{N} v_k^p\right)^{1/p},\ 0 \leq p \leq 1. \tag{3.2}$$

Because GI is a normalized measure, it assumes values between 0 and 1 for any vector. If the obtained GI is close to 0, it means that the signal/image has a lower sparsity. On the other hand, when the GI is close to 1, it means that the signal/image is very sparse. Using (3.1), the GI values for the microarray data sets are displayed in Table 3.1. After analyzing the obtained GI values, we conclude that all data sets, except the *ApoA1*, have an overall GI value higher than 0.5.

3.2 The use of standard image compression methods

In this section, we present compression results that have been obtained for providing some reference point regarding the performance of standard image coding techniques. The standards used are JBIG [20, 23], PNG [28, 29], JPEG-LS [33–35] and JPEG2000 [39, 41]. The experiments were conducted using the lossless version of each standard.

The compression results presented in Table 3.2 were obtained using the microarray images described earlier in Table 3.1. The default parameters were used in all compression tools, which means that we did not try to adjust some parameters in order to improve the compression results. The only thing mandatory used in these tools is the lossless mode, due to the fact that this work is focused on this type of compression. JBIG results were obtained using version 2.0 of the JBIG Kit package[1]. The results for the JPEG-LS standard were obtained using version 2.2 of the SPMG JPEG-LS codec[2]. JPEG2000 lossless compression was obtained using version 5.1 of JJ2000 codec with default parameters for lossless compression[3]. For additional reference, we also provided compression results using the gzip, bzip2, ppmd, and lzma general purpose compression tools.

It can be seen that the obtained results depicted in Table 3.2 can vary slightly when compared to the results presented in [128] and [122]. First, the researchers sometimes do not take into account the size of each image (number of pixels) to compute the average compression results for each data set. Instead of performing a weighted mean they used sometimes the arithmetic mean which is not the most appropriate approach. Second, the architecture or the operating system used to obtained the results can also induce minor differences in the compression results. Third, the codec used is also a key aspect that can affect somehow the compression results (there are several implementations of JBIG, JPEG-LS, and JPEG2000). If two users use a different implementation in the same data set, it is possible to attained slightly different compression results. If we now look at the compression results themselves, we can conclude that the JPEG-LS is the compression standard with the overall best compression performance. Nevertheless, ppmd outperform JPEG-LS in the *IBB*, *ISREC*, and *Yeast* data sets. JPEG2000 attained the worst compression results for all data sets. This seems to reinforce the idea that wavelet-based methods (as JPEG2000) are not very effective in compressing microarray images. However, despite the weak performance of JPEG2000 in microarrays images when using default parameters, Hernández-Cabronero *et al.* [122] have shown that, if the number of wavelet decomposition levels is increased, in general the compression performance improves by approximately 0.5 bits per pixel.

3.3 Microarray-specific compression methods

As described in Section 2.2.2, in the literature there are several microarray-specific compression methods. Most of these methods rely on lossless coding and are specially designed to exploit the properties of microarray images.

In Table 3.3 we can find results for some of the microarray-specific algorithms. The results are expressed in bits per pixel (bpp), so lower values are better. The dashes mean that the

[1] `http://www.cl.cam.ac.uk/~mgk25/jbigkit`.

[2] The original web-site of this codec, `http://spmg.ece.ubc.ca`, is currently unavailable. However, it can be obtained from `http://sweet.ua.pt/luismatos/codecs/jpeg_ls_v2.2.tar.gz`.

[3] The original web-site of this codec, `http://jj2000.epfl.ch`, is currently unavailable. Nevertheless, this codec can be obtained from `http://sweet.ua.pt/luismatos/codecs/jj2000_5.1-src.zip`.

Table 3.2: Lossless compression results, in bits per pixel (bpp), using gzip, bzip2, ppmd, lzma, PNG, JBIG, JPEG-LS, and JPEG2000. Default compression parameters have been used for all algorithms. The best results are highlighted in bold.

	Compression methods							
Data sets	**Gzip**	**Bzip2**	**PPMd**	**LZMA**	**PNG**	**JBIG**	**JPEG-LS**	**JPEG2000**
ApoA1	12.711	11.068	10.984	11.374	12.568	10.851	**10.608**	11.063
Arizona	11.263	9.040	8.980	9.402	11.017	8.896	**8.676**	9.107
IBB	10.453	9.081	**8.495**	8.985	10.090	9.344	9.904	10.516
ISREC	12.464	10.922	**10.730**	11.126	12.476	10.925	11.145	11.366
Omnibus (LM)	7.124	5.346	4.977	5.527	6.781	5.130	**4.936**	5.340
Omnibus (HM)	9.558	7.523	7.219	7.787	9.160	7.198	**6.952**	7.587
Stanford	9.972	7.961	7.809	8.273	9.776	7.906	**7.684**	8.060
Yeast	7.672	6.075	**5.794**	6.389	8.303	6.888	8.580	9.079
YuLou	11.434	9.394	9.285	9.708	11.428	9.298	**8.974**	9.515
Average	9.044	7.189	**6.859**	7.388	8.729	7.051	6.996	7.511

results are not provided by the authors for a particular image data set. In most cases, there is not available implementation of these methods. Therefore, results cannot be generated for other more representative data set. However, we were able to provide results for all data set for methods [66] and [68], introduced by Neves and Pinho.

According to the available results, it seems that Battiato's method [104] is the one with the best compression performance, for three out of nine data sets used in this work. Despite all this, we need to be aware that the authors may present their results in different ways. For instance, in Table 3.3 we have results that were computed without taking into consideration the image sizes among each data set. In addition, some of the results were obtained using a set of representative images of a given data set. This means that some of the results depicted in the table lack precision, thus, they are not reliable to perform a proper comparison with other methods.

3.4 Bitplane decomposition approaches

In Section 3.2, we presented compression results for several image coding standards: JBIG, PNG, JPEG-LS, and JPEG2000. According to the results of Table 3.2, it seems that technology behind JBIG is the most promising for compressing microarray images. Taking into consideration the results of JBIG, Neves and Pinho presented in [106, 107] a compression method that is based on JBIG. In this section, we will present some modifications to their work with the aim of improving the compression results.

The technique to separate an image into different planes, known as bitplane decomposition, plays an important role in image compression. In the case of typical grayscale images, where each pixel is represented by eight bits, we can split the original image into eight bitplanes, ranging from bitplane 0, the least significant bitplane (LSBP), to biplane 7, the most significant bitplane (MSBP). We can find several compression algorithms that use this decomposition approach. In [107] Neves an Pinho used this approach for the compression of microarray images. Their method is based on the same technology as JBIG. However, unlike

Table 3.3: Lossless compression results for some microarray-specific schemes. The results are expressed in bits per pixel. All results have been adopted from the information found in the references specified in the table. The best results are in bold.

	Microarray-specific compression methods						
	SLOCO [86]	**MicroZip [96]**	**PPAM [99]**	**Neves [106]**	**Neekabadi [101]**	**Batiato [104]**	**Neves [107]**
ApoA1	(a) 8.556	—	—	10.280	10.250	(d) **9.520**	10.194
Arizona	—	—	—	8.394	—	—	**8.242**
IBB	—	—	—	8.063	—	—	**7.974**
ISREC	—	—	—	10.217	10.202	**9.490**	10.159
Omnibus (LM)	—	—	—	5.309	—	—	**4.567**
Omnibus (HM)	—	—	—	7.047	—	—	**6.471**
Stanford	—	—	—	7.664	—	—	**7.379**
Yeast	—	—	(b,c) 6.501	5.610	—	—	**5.453**
YuLou	—	9.534	(c) 9.243	8.840	8.856	(d) **8.369**	8.619
Average	—	—	—	6.772	—	—	**6.284**

(a) Computed taking into account eight replicate image pairs from the *ApoA1* data set with 1044×1041 pixels, and a compression ratio of 1.87:1.
(b) Computed using three images from the *Yeast* data set.
(c) The results do not include the header information that on average requires about 0.04 bpp, according to the authors.
(d) The value provided by the authors does not take into account the different image sizes of the data set.

JBIG, it exploits inter-bitplane dependencies, providing coding gains in relation to JBIG. Their first method was proposed in 2006 [106] and was inspired by EIDAC [131]. This first method introduced an image-independent context based model that drives the arithmetic encoder. The casual finite-context model is built using pixels from the bitplane being encoded and from the previous bitplanes already encoded. The 3D context configuration is static for every microarray image which means that there is space for some improvement here. Later, Neves and Pinho proposed a more sophisticated method in [107], where the 3D context configuration is image dependent, i.e., for each microarray image a different template configuration is used to encode the image.

The methods [106, 107] were described in Section 2.2.2.8. In the following sections we introduce the compression tools based on method [107].

3.4.1 Segmentation

As presented previsouly in Section 2.2.2, there are several specialized microarray methods that use segmentation in their compression pipeline. The goal of segmentation is to separate the input image into two sub-images in order to explore the characteristic of each component (background and foreground). By doing this, it is possible to encode each component more efficiently. Furthermore, image gridding and segmentation is also a key aspect used in the analysis of DNA microarray images to locate each spot. In the gridding process each spot is identified and confined individually into a rectangular area. This location process can be done automatically or using geometrical information provided by the DNA microarray manufacturer. After the gridding is done, the next step consists on determining which pixels belong to the spot region (foreground), and which ones are background. This is known as segmentation and as mentioned before this is one of the most active research topics on the analysis of DNA microarray images [126]. Several approaches for this purpose have been presented, namely clustering-based [132–136], threshold-based [101, 137], graph-based [138] and even wavelet-based [139] methods.

The Neekabadi *et al.* [101] segmentation procedure, described earlier in Section 2.2.2.6, is simple and fast so we decided to use it in our approach. In this section, we present a modification to method [107], by adding a segmentation step before the encoding procedure itself. In Figure 3.1 we can find the flow chart of the proposed encoding procedure. Initially, the input image is segmented into foreground and background, according to a pre-computed threshold value. In order to obtain the threshold value, a dynamic thresholding scheme is used according to 2.4, previously presented in Section 2.2.2.6. As mentioned before, the foreground and background are obtained using a threshold value that guarantees that the weighted sum of the standard deviation of both the foreground and background is minimal. In order to accelerate the threshold search routine (designated as "Compute threshold value" in Figure 3.1), a recursive search approach is used to find the threshold value that minimizes $f(T)$. Using this approach, we can avoid testing all possible threshold values which would take some extra computational time. Furthermore, this approach is acceptable because $f(T)$ plunges down at a certain threshold value, which is chosen as the final threshold value (see Figure 3.2). The $f(T)$ plot of Figure 3.2 was obtained for image "Deff661Cy3" from *ISREC* data set, $f(T)$ has a similar behavior for other microarray images. In this particular case, the threshold value selected/used is 17874. The output of this segmentation algorithm is composed of three images: foreground, background and the binary mask (see Figure 3.3 (b)). In the mask image, the pixels that are classified as foreground are set to 1 (white) and the remaining ones (background pixels) are 0 (black). This mask image is then compressed using JBIG.

After having both the foreground and background images, the encoding procedure starts by compressing each one separately. Both images are compressed on a bitplane basis, starting from the most significant bitplane (MSBP) and stopping at the least significant bitplane (LSBP) or whenever a bitplane requires more than one bit per pixel for encoding (in the case of the background image). The number of bitplanes of each component (foreground and background) must be previously computed in order to avoid encoding extra bitplanes that do not have relevant information (bitplanes with only zeros). This computation is always required because the segmentation can introduce some bitplanes with only zeros in both the foreground and background images. Because the foreground image has a highly compressibility, the encoder never requires more than one bit per pixel for each bitplane. Due to that, the

encoding procedure for the foreground image always ends when it reaches the last bitplane. Regarding the background image, it is possible a given bitplane to require more than one bit per pixel to encode it. If that happens, the remaining bitplanes are saved uncompressed.

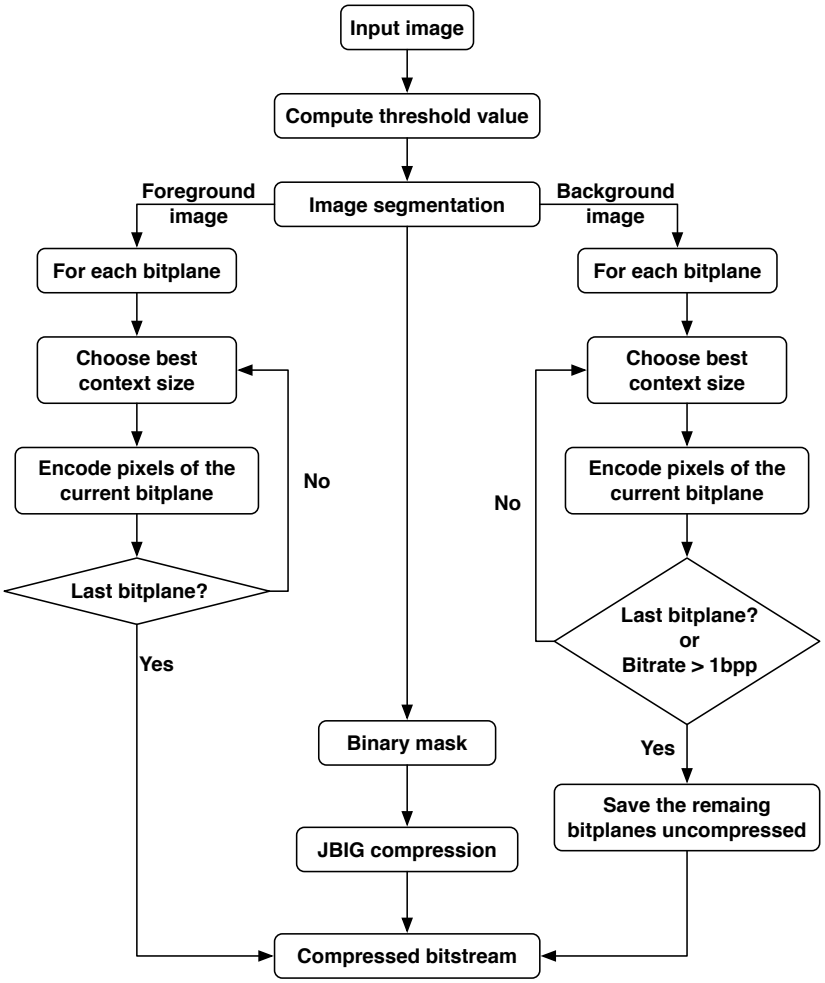


Figure 3.1: Encoding procedure of the proposed method. The context creation procedure denoted as "Choose best context size" is explained in more detail in Figure 3.4.

The procedure used for creating the 3D context configuration, represented as "Choose best context size" in Figure 3.1, is detailed in Figure 3.4. The context configuration is created based on the templates presented in Figure 2.24 (see Section 2.2.2.8). A brute-force approach, where all possible template configurations are tested, is a hard task and virtually impossible, due to the huge number of possibilities. In order to overcome this drawback, Neves and Pinho [107] developed a greedy approach to obtain the optimal 3D context configuration. We used the term optimal because the procedure does not exhaustively test all possible context configurations. Due to that, we cannot characterize the context configuration obtained by this procedure as the "best" configuration. Before encoding each bitplane, the algorithm constructs an appropriate context configuration through an iterative process. In each iteration, an additional context bit is tested (Figure 2.24). Context bits are added according to Figure 2.24 (a) if the current bitplane is the one being encoded. On

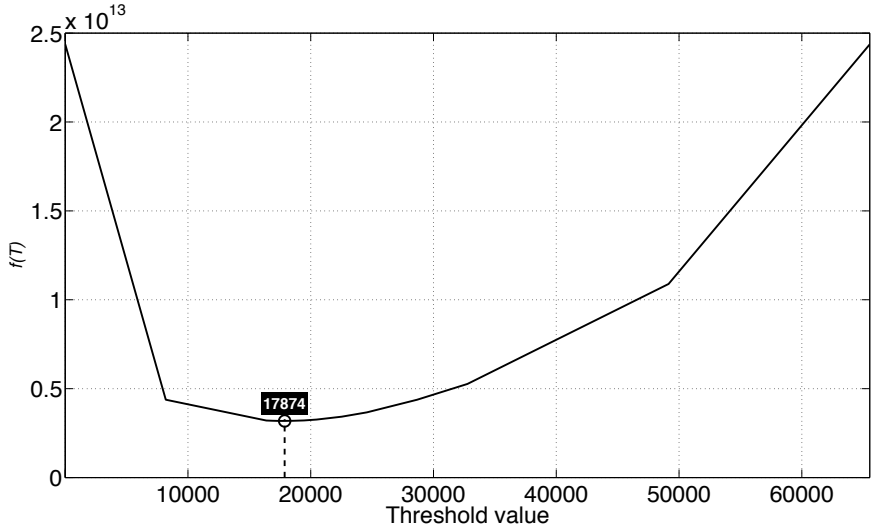
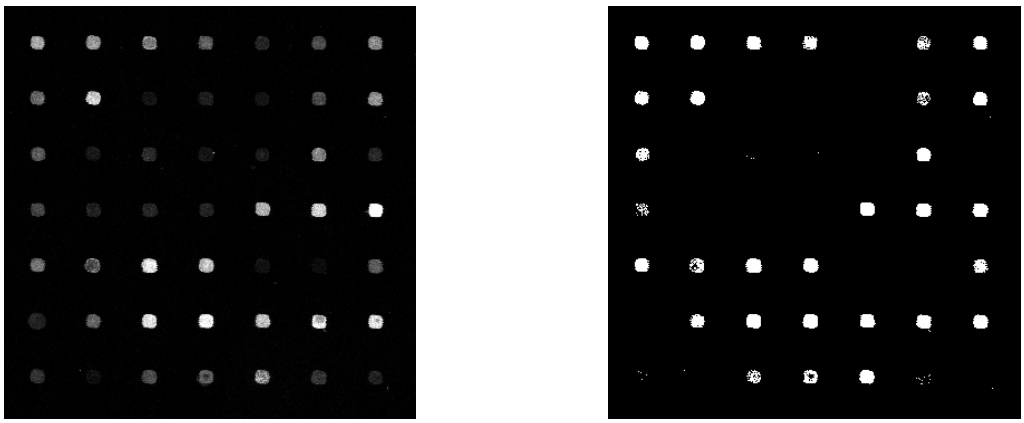


Figure 3.2: Plot of $f(T)$ for a range of possible threshold values for image "Deff661Cy3" from *ISREC* data set. In this case the selected threshold values that minimizes $f(T)$ is 17874 (marked in a circle).



(a) Original image (b) Binary mask

Figure 3.3: The eight MSBPs cropped portion of image "Deff661Cy3" from *ISREC* data set on the left. On the right, the binary mask obtained after applying the threshold scheme described in this section.

the other hand, if the current bitplane was already processed, the context bits are added according to Figure 2.24 (b). Each context bit that is added is tested and if it improves the compression result it will be added to the final context configuration. This search routine ends when the context size is ≥ 20 or when adding context bits does not improve the compression performance.

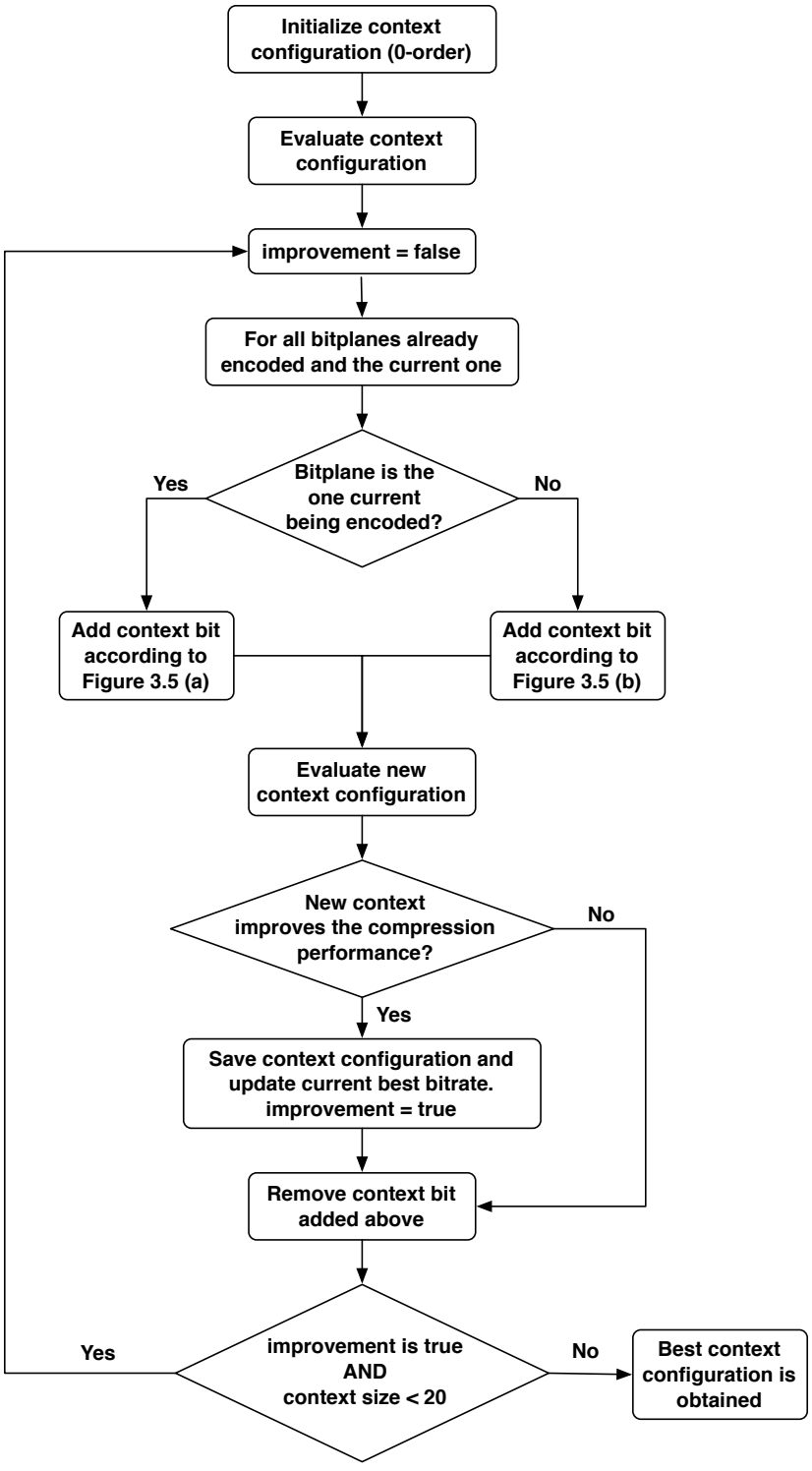


Figure 3.4: Flow chart of the 3D context creation procedure presented in [107].

3.4.1.1 Experimental results

In this section, we present the simulation results of the proposed method. We compared the proposed method with methods [106], [107] and the best standard image coding technique for microarray images JPEG-LS.

In Table 3.4, we present the compression results for several methods, including JPEG-LS, methods [106], [107], and the proposed method. We present the results for each data set and the overall results in order to be easier to make some conclusions. Two modes were used for the context evaluation process. In the first mode, indicated in the Table as "256×256", the search area has 256×256 pixels. For now on, we will designate this mode as *SA-256*. The second mode uses the entire image as a search area, in order to evaluate the context configuration. This mode will be designated from now on as *Full*. This last mode is much slower as can be seen in the results in Table 3.4. In the last four rows, we present the encoding and decoding time ("CTime" and "DTime") in hours and speed ("CSpeed" and "DSpeed") in kilobytes per second, globally for all data sets. Regarding the performance of each method in terms of bits per pixels, the proposed method attained about 1% better results when compared with method [107], for the first mode (*SA-256*). This small improvement can be observed specially in data sets *Omnibus (HM)* and *Yeast*. In the *Full* mode, the proposed method is slightly worse globally, however there is a small improvement in the *Stanford* and *Yeast* data sets. In terms of coding speed, the proposed method is not as fast as the other methods. When compared to method [107], the encoding time of the proposed method is slightly superior, probably due to the threshold value search procedure. The decoding time is very similar between the proposed method and method [107]. The JPEG-LS standard is the fastest method and, according to Table 3.4, it is between 30 to 50 times faster to encode the data sets and 19 to 23 times faster to decode the same data sets, when compared to methods [106], [107] and the proposed method, using the first mode (*SA-256*).

In Table 3.5, we present some information regarding the segmentation method used in this work. The first column contains the average threshold value for each data set used to split the each microarray image into foreground and background. In the second and third column, we can find the average percentage of pixels of each component (foreground and background). In the last two columns, we present the average percentage of active intensities for each component. The percentage of pixels of each component was computed taking into account the number of pixels classified as foreground/background in all data sets.

It is clear that the percentage of foreground pixels is very small when compared to the percentage of background ones. It is important to emphasize that the presented values do not correspond to the amount of pixels that are spots (classified as foreground) and non-spot (background region). The segmentation algorithm used is not very sophisticated so it is possible that some pixels of a given spot are classified as background (see Figure 3.3). As mentioned earlier in Section 2.2.1, the microarray images have a bit depth of 16 bits per pixel which allow 65536 different intensities. Given the nature of the segmentation method, the pixels are classified as background if their values are lower than the selected threshold and as foreground otherwise. According to the obtained threshold values, we can conclude that the intensity range for the background region corresponds to 10% to 30% of the total of the intensities available. This means that almost 99% of the pixels of a given microarray image only used up to 30% of the total intensities. The foreground region, which is a very small component when compared to the background region, uses up to 90% of the available intensities of the original image. The last two columns of Table 3.5 depict the average percentage

Table 3.4: Compression results in bits per pixel (bpp), using JPEG-LS, methods [106, 107] and the proposed improvement based on a segmentation approach (denoted as "Proposed" in the table). Results for two modes are presented in the table. One where the search area used to evaluate each context configuration have 256×256 pixels. The other one denoted as "Full" in the table, corresponds to a search area that covers the entire image. The values with a "•" represent improvements between method [107] and the proposed method (local improvement). The best results are in bold. In the last four rows, we present the encoding/decoding time ("CTime" and "DTime") in hours and speed ("CSpeed" and "DSpeed") in kilobytes per second, globally for all the data sets.

	Compression methods					
Data sets	**JPEG-LS**	**Neves [106]**	**Neves [107] (256×256)**	**Proposed (256×256)**	**Neves [107] (Full)**	**Proposed (Full)**
ApoA1	10.608	10.280	10.225	10.265	**10.194**	10.234
Arizona	8.676	8.394	8.293	8.291 •	**8.242**	8.245
IBB	9.903	8.063	8.039	8.041	7.974	7.982
ISREC	11.145	10.217	10.199	10.215	**10.159**	10.193
Omnibus (LM)	4.936	5.309	5.679	5.637 •	**4.567**	4.570
Omnibus (HM)	6.952	7.047	7.744	7.616 •	**6.471**	6.479
Stanford	7.684	7.664	7.468	7.415 •	7.379	**7.349** •
Yeast	8.580	5.610	5.511	5.430 •	5.453	**5.395** •
YuLou	8.974	8.840	8.667	8.675	**8.619**	8.641
Average	6.996	6.772	7.101	7.041 •	**6.284**	6.288
CTime (hours)	0.18	3.97	6.24	8.33	643.19	964.29
DTime (hours)	0.19	4.30	3.67	3.55	6.00	5.63
CSpeed (KB/s)	11904	539	343	257	3	2
DSpeed (KB/s)	11246	498	583	602	357	380

of active intensities of each component after the segmentation procedure. This measure was obtained taking into account the new number of intensities available after segmentation and not the number of intensities available in the original image. For example in the *Stanford* data set, the average threshold value used was 15617, which means that there are 15618 intensities available to the background image and $65536 - 15618 = 49918$ for the foreground image. Taking into consideration the previous explanation, we can conclude that, in the foreground region, the average percentage of active intensities is higher than 65% for almost all data sets. This is no surprise for us because the genetic information of the DNA microarray images is usually extracted from the spots which correspond to the foreground region. On the other hand, in the background, region the average percentage of active intensities is on average $\approx 27.6\%$. Despite the low percentage of active intensities for the background component, there are some data sets where the average percentage is very high when compared to the overall average (e.g. *Arizona* and *Omnibus (HM)*). We believe that these differences are probably caused by the different modes used to scan the microarray images. Different modes can introduce different amounts of noise in the microarray image. Moreover, the threshold value selected in the segmentation procedure is also a key aspect that can affect the results of Table 3.5.

Table 3.5: Segmentation information for all data sets: average threshold used, percentage of pixels of each component (foreground/background) and percentage of active intensities in the foreground and background images.

Data sets	Measures				
	Average threshold	% of pixels		% active intensities	
		Foreground	Background	Foreground	Background
ApoA1	11164	2.19	97.81	95.84	28.33
Arizona	10917	4.05	95.95	99.73	80.96
IBB	16598	0.45	99.55	97.56	39.46
ISREC	17801	1.47	98.53	70.02	19.92
Omnibus (LM)	6837	0.30	99.70	99.87	46.73
Omnibus (HM)	19047	1.05	98.95	100.00	97.79
Stanford	15617	1.18	98.82	67.63	15.53
Yeast	8338	1.48	98.52	12.19	4.41
YuLou	6332	1.76	98.24	99.47	31.12
Overall	12087	1.03	98.97	61.30	27.57

3.4.1.2 Complexity

The proposed modification to method [107] is based on finite-context models that, depending on the bitplane that is being encoded and the image itself, we have a different number of pixels used to build up the finite-context. The number of pixels in this implementation can go up to a maximum of 20. Considering that we are dealing with a binary alphabet, the maximum number of counters for the model used to encoded each image is $2 \times 2^{20} = 2,097,152$. Taking into account that each counter is stored in 2 bytes, the total amount of computer memory required to store the counters is approximately 4 megabytes.

According to Table 3.4, we can see that the JPEG-LS standard is in fact the fastest method and the proposed method and method [107] in the *Full* mode are the slowest. Regarding the first mode (*SA-256*), the proposed improvement takes approximately 2 hours more to encode the data sets, when compared to method [107]. In the decoding phase, the time required is very similar between the proposed method and method [107].

3.4.2 Bitplane reduction

Bitplane reduction is an interesting method that can further improve compression efficiency by eliminating redundancy in the pixel precision for simple images. Simple images are images where the number of different intensities that occur is very small, compared to the total number of possible intensities. For example, if we have only 24 different intensities out of 256 for an 8 bits image, we only need five bits to represent each pixel intensity. This means that, when we are encoding the image, we only need to encode five bitplanes instead of the original eight bitplanes.

In 1998, Yoo *et al.* [131] have shown that it is possible to obtain compression gains using the simplest form of bitplane reduction, know as Histogram Compaction (HC). Later in 1999 [63], they presented a more robust bitplane reduction method called Scalable Bitplane

Reduction (SBR). This approach finds the reduced bitplane codeword by growing a binary tree. The method splits each node of the binary tree into two nodes using a simple MINMAX metric to measure the distortion.

In order to better understand the SBR algorithm, we present a small example in Figure 3.5. Initially, we associate all the active pixel values to the root node. In this small example, the image only has four active pixel values. Starting in the root node, the algorithm splits all the children sub-nodes using a MINMAX criterion. The split process in the root node starts by computing the value 32767 from $\lfloor (0 + 65535)/2 \rfloor$. The computed value is then used to split the node. All the intensities that are lower or equal than 32767 are inserted in the left sub-node. On the other hand, the remaining intensities (> 32767) are associated with the right sub-node. After splitting the root node, the SBR algorithm adds a zero to the left node codeword and a one to the right node codeword. This spliting process is repeated until all sub-nodes have only one intensity associated with them. In this specific example, "0" is a complete codeword for the original pixel value zero, due to the fact that the first left node or partition does not have more sub-nodes. As a result of the variable-length codewords, the average codeword length can be less than three bits. This is useful because, during the encoding process, it is possible to skip some bitplanes of the pixel codewords with a lower length.

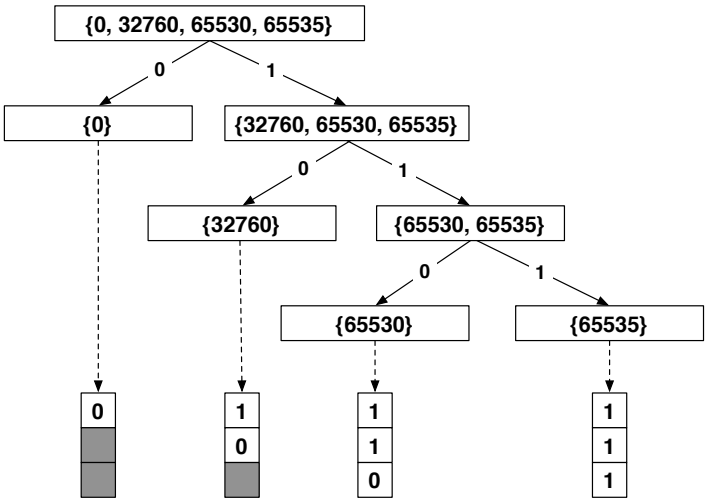


Figure 3.5: Binary tree to obtain the codewords to represent the active pixels values $\{0, 32760, 65530, 65535\}$ in the reduced bitplane space after applying the SBR algorithm. Each of the tree nodes is associated with a symbol (an intensity) set to be partitioned, except the end nodes. Each branch defines the bit value of a specific symbols at the corresponding bitplane in the reduced bitplane domain. For the intermediate nodes, $\{32760, 65530, 65535\}$ and $\{65530, 65535\}$, the split process is done using the MINMAX criterion.

Taking as example the intensities $\{0, 32760, 65530, 65535\}$ presented in Figure 3.5, we present in Table 3.6 the codewords for the two histogram reduction methods. As can be seen, the codewords obtained using the HC algorithm are dependent on the number of active intensities. The codeword size can be computed according to

$$S = \lceil \log_2 (N) \rceil, \tag{3.3}$$

where N denotes the number of active intensities. In the example presented in Figure 3.5, we have four different intensities, so the codeword size is $\lceil \log_2(4) \rceil = 2$. The codeword size is constant for all intensities for the HC method. On the contrary, the resulting codewords obtained using the SBR algorithm have different sizes (see Table 3.6).

Table 3.6: A small example showing the differences between the two bitplane reduction methods HC and SBR. The codeword in each column represents the new value that will be assigned to the pixels values of the first column. The codeword size represents the size of the codeword. For the HC the codeword size is constant. On the other hand, for the SBR the codeword size is variable.

Intensity Value	**HC**		**SBR**	
	Codeword	**Codeword size**	**Codeword**	**Codeword size**
0	00	2	0	1
32760	01	2	10	2
65530	10	2	110	3
65535	11	2	111	3

The two bitplane reduction methods described earlier are quite interesting to be used in compression of microarray images. According to Table 3.1 of Section 2.2.1, the majority of the data sets have less than 50% of active intensities. The only two exceptions are the *Arizona* and *Omnibus (HM)* data sets. We decided to analyze the effect of these two bitplane reduction algorithms when applied in method [107]. In order to incorporate these two bitplane reduction algorithms in method [107], we only need to add an extra pre-processing step in the encoding pipeline, where the input image I is processed and transformed into a second image J, with the modified pixel values. Moreover, it is necessary to send a 65536-bit indicator that will identify which intensities actually occur in the original image. After having that 65536-bit indicator in the decoder and the decoding process is completed, it is possible to perform the inversion operation in order to obtain the original image.

In the next sub-section, we will present the compression results obtained when adding the two bitplane reduction features described earlier.

3.4.2.1 Experimental results

In this section, we present a set of experiments that have been performed in order to show the performance of the proposed algorithm. Similar to the previous section, we provide results for the proposed algorithm and compare them with methods [106], [107], and the best standard image coding technique for microarray images, JPEG-LS.

As mentioned in the previous section, we added two bitplane reduction features to method [107]. In Table 3.7 we present the compression results for the Histogram Compaction (HC) improvement. The results in bold are the best ones for each data set. The values with a "•" correspond to a local improvement between the original method [107] and the proposed HC improvement. This local improvement observation was done inside each mode, *SA-256* and *Full*. For example, a local improvement can be found in the last column for the *Stanford* data set, because the attained value for the proposed method in *Full* mode (7.350 bpp), is lower than the one obtained by method [107], in the same mode (method [107] for the *Stan-*

ford data set attained on average 7.379 bpp). After analyzing the results of Table 3.7, we can notice that for mode *SA-256* there is a small improvement in three data sets. For the *Full* mode, the compression results between method [107] and the proposed HC improvement are quite similar. Despite this, we can notice a small improvement in the *Omnibus (LM)* and *Stanford* data sets.

Regarding the coding time, we can observe that the encoding and decoding time are very similar between the proposed improvement and method [107].

Table 3.7: Compression results in bits per pixel (bpp), using JPEG-LS, methods [106, 107] and the proposed improvement based on Histogram Compaction (HC). Results for two modes are presented in the table. One where the search area used to evaluate each context configuration have 256×256 pixels. The other one denoted as "Full" in the table, corresponds to a search area that covers the entire image. The values with a "•" represent improvements between method [107] and the proposed method (local improvement). The best results are in bold. In the last four rows we present the encoding/decoding time ("CTime" and "DTime") in hours and speed ("CSpeed" and "DSpeed") in kilobytes per second, globally for all the data sets.

	Compression methods					
Data sets	**JPEG-LS**	**Neves [106]**	**Neves [107] (256×256)**	**Proposed HC (256×256)**	**Neves [107] (Full)**	**Proposed HC (Full)**
ApoA1	10.608	10.280	10.225	10.259	**10.194**	10.231
Arizona	8.676	8.394	8.293	8.300	**8.242**	8.244
IBB	9.903	8.063	8.039	8.041	**7.974**	7.978
ISREC	11.145	10.217	10.199	10.239	**10.159**	10.195
Omnibus (LM)	4.936	5.309	5.679	5.652 •	4.567	**4.561 •**
Omnibus (HM)	6.952	7.047	7.744	7.743 •	**6.471**	6.473
Stanford	7.684	7.664	7.468	7.433 •	7.379	**7.350 •**
Yeast	8.580	5.610	5.511	5.601	**5.453**	5.527
YuLou	8.974	8.840	8.667	8.667	**8.619**	8.626
Average	6.996	6.772	7.101	6.092 •	**6.284**	6.286
CTime (hours)	0.18	3.67	6.24	6.32	643.19	676.07
DTime (hours)	0.19	4.30	3.67	4.65	6.00	7.44
CSpeed (KB/s)	11904	539	343	338	3	3
DSpeed (KB/s)	11246	498	583	460	357	289

We also tested the bitplane reduction feature that was explained earlier, known as Scalable Bitplane Reduction (SBR). In Table 3.8, we present the results for the SBR improvement and compared the results with other methods, similar to Table 3.7. The results are very similar between the two bitplane reduction approaches. Once again we have a small local improvement in the *SA-256* mode for three data sets: *Omnibus (LM)*, *Omnibus (HM)*, and *Stanford*. Regarding the *Full* mode, we also obtained a small improvement in the *Omnibus (LM)* and *Stanford* data sets. Globally, the results for this mode are very similar between the SBR improvement proposed and method [107]. In the last four rows of Table 3.8, we can find the encoding/decoding time ("CTime" and "DTime") in hours and speed ("CSpeed" and "DSpeed") in kilobytes per second for each method for the data sets used in this work. According to the obtained values in terms of coding time, we can see that the encoding and decoding time between the SBR improvement proposed and method [107] are quite similar in each mode (*SA-256* and *Full*).

Table 3.8: Compression results in bits per pixel (bpp), using JPEG-LS, methods [106, 107] and the proposed improvement based on Scalable Bitplane Reduction (SBR). Results for two modes are presented in the table. One where the search area used to evaluate each context configuration have 256×256 pixels. The other one denoted as "Full" in the table, corresponds to a search area that covers the entire image. The values with a "•" represent improvements between method [107] and the proposed method (local improvement). The best results are in bold. In the last four rows we present the encoding/decoding time ("CTime" and "DTime") in hours and speed ("CSpeed" and "DSpeed") in kilobytes per second, globally for all the data sets.

	Compression methods					
Data sets	**JPEG-LS**	**Neves [106]**	**Neves [107] (256×256)**	**Proposed SBR (256×256)**	**Neves [107] (Full)**	**Proposed SBR (Full)**
ApoA1	10.608	10.280	10.225	10.263	**10.194**	10.232
Arizona	8.676	8.394	8.293	8.297	**8.242**	8.243
IBB	9.903	8.063	8.039	8.041	**7.974**	7.978
ISREC	11.145	10.217	10.199	10.235	**10.159**	10.199
Omnibus (LM)	4.936	5.309	5.679	5.661 •	4.567	**4.565** •
Omnibus (HM)	6.952	7.047	7.744	7.738 •	**6.471**	6.472
Stanford	7.684	7.664	7.468	7.436 •	7.379	**7.349** •
Yeast	8.580	5.610	5.511	5.506 •	**5.453**	5.466
YuLou	8.974	8.840	8.667	8.668	**8.619**	8.626
Average	6.996	6.772	7.101	7.094 •	**6.284**	6.285
CTime (hours)	0.18	3.97	6.24	6.99	643.19	727.33
DTime (hours)	0.19	4.30	3.67	4.04	6.00	6.71
CSpeed (KB/s)	11904	539	343	306	3	3
DSpeed (KB/s)	11246	498	583	529	357	319

3.4.2.2 Complexity

Similar to the previous method, this approach is also based on finite-context models that depending on the bitplane being coded and the image itself, a different number of pixels is used to build up the finite-context. As mentioned before (Section 3.4.1.2), the maximum number of pixels used to build the context can go up to 20 so, the amount of computer memory required to store the counters is approximately 4 megabytes. Taking into account the rows of "CTime" and "DTime" of Tables 3.7 and 3.8, it is easy to verify once again that the *SA-256* mode is considerably more fast (approximately 100 times) when compared to the *Full* mode. We can see that in the *SA-256* mode the encoding time is $\approx$ 6.3 hours for the HC approach and $\approx$ 7 hours for the SBR alternative. On the other hand, the decoding time is between 4 and 4.7 hours.

3.5 Simple bitplane coding using pixel value estimates

In 2009, Kikuchi *et al.* [140] introduced the concept of bit modeling by the pixel value estimates. In their approach, instead of using the true bit values of each bitplane, they used the expectation values of the pixels to build up the contexts. This approach is known as *bit modeling by pixel values estimate*. They extended their work more recently to be applied to

various types of images (color, grayscale, color-quantized, bi-level, and halftone) [141] and for HDR (High Dynamic Range) images [142].

In this section, we will describe a compression method that was inspired on Kikuchi's method. The goal is to evaluate this approach when applied to microarray images.

3.5.1 The proposed approach inspired on Kikuchi's method

Lets consider that a given pixel value at location (i, j) in a given image to be encoded is written as $x(i, j)$. Its decoded value is written by $y(i, j)$. In order to facilitate the explanation, the location indexes (i, j) are omitted for now on. A typical raster scanning order is used to process each pixel of a given image. Similar to Kikuchi's method, as the process of the bitplane coding proceeds to lower bitplanes, the decoded value, y, of the target pixel approaches the true pixel value, $x = (x_{16}\, x_{15} \cdots x_1)$ where x_n denotes the n^{th} bit of x in the case of a 16-bit grayscale image.

Contrarily to Kikuchi's method, our approach uses only one type of context, denoted as *neighborhood context* in Kikuchi's work. The contexts are built by the estimates of partially-decoded pixels based on the template depicted in Figure 3.6. The pixel location of x is labeled by "X" on the 15-pixel template of Figure 3.6, and

$$c_k = \begin{cases} 1, & \text{for } y(k) > y \\ 0, & \text{otherwise} \end{cases}, \tag{3.4}$$

where $k \in \{1, 2, \ldots, 15\}$ denotes the spacial location on the template illustrated in Figure 3.6. $y(i)$ and y represent the most recent estimates of the neighboring pixels and the target pixel, respectively.

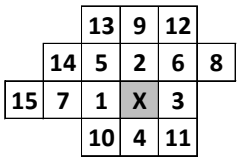


Figure 3.6: Fifteen-pixel template for building up the context. The target pixel to be encoded is labeled by a "X".

In Kikuchi's method, the inter-bit correlation on a bitplane is not used. Instead, for modeling a target bit, the authors used the pixel value estimates of which more significant bits have been already available at the decoder. Their method is referred to as *bit modeling by pixel values*, where the pixel value estimates are used rather than of the unknown true values at the decoder. Contrarily to Kikuchi's method, that uses a 9-pixel template, our approach uses a variable-size 15-pixel template (see Figure 3.6). In our case, a greedy search routine is performed in each bitplane in order to obtain the context size that attains the best compression performance (lower bits per pixel as possible). According to Kikuchi *et al.*, the decoded pixel values are spatially correlated to each other as significantly high as the true pixel values which will make sense to use a larger template. Furthermore, since the alphabet size is only two, the probability of having context dilution is low. Similar to the Kikuchi's method, we are considering some non-causal locations with the respect to the scanning order of the pixels (locations $y(3), y(4), y(10)$, and $y(11)$ in Figure 3.6). The usage of non-causal

pixels is only possible because the context bits are defined by using the estimates of pixel values, which are available in the decoder.

Lets consider that we are coding a N-bit depth image, where $N \leq 16$. Suppose that the n^{th} bitplane is being encoded at present, where $n \in \{1 \dots N\}$. For every pixel, the higher bits from the $(n+1)^{\text{th}}$ until N^{th} bitplane are known at the decoder. The other lower n bits are unknown. The value of the unknown part can be distributed over the interval of $[0, 2^n - 1]$. Similar to Kikuchi's method, the values zero and one occur with equal probability in the unknown less significant n bits. Under this assumption, the pixel value estimate of the target pixel is expressed by

$$y^{(n)} = \left\lfloor \frac{y}{2^n} \right\rfloor + 2^{n-1} - 1 \tag{3.5}$$

at the n^{th} bitplane encoding/decoding, where y is the latest decoded value and $\lfloor . \rfloor$ denotes truncation. In Figure 3.7 we illustrate an example of a binary representation of the pixel value estimate in the case $n = 8$.

Initially, all the pixel estimate values are set to $y = 2^{N-1} - 1$. The encoding procedure starts at the MSBP and stops at the LSBP. Assuming a target pixel x_n of bitplane n is the one being encoded, under the context of $\{c_k\}$, the pixel estimate, y of the target pixel is immediately updated by a simple bit operation as

$$y \leftarrow y + x_n 2^{n-1} - \left\lfloor 2^{n-2} \right\rfloor. \tag{3.6}$$

The pixel value estimate is used in a coming chance of reference and will be the decoded pixel value, when the decoding is stopped (after all the bitplanes are processed). The most recent estimate of a given pixel is always made up of two parts: its significant bits are those already encoded/decoded true bits and the other less significant bits are 0 (zero) followed by a successive 1's (ones). The value of the less significant bits is equal to the expectation value of the unknown lower bits, if binary symbols of 0 and 1 are assumed to occur in those bits with equal probability.

$y^{(8)} =$

x_{16}	x_{15}	x_{14}	x_{13}	x_{12}	x_{11}	x_{10}	x_9	0	1	1	1	1	1	1	1

Figure 3.7: Binary representation of a pixel value estimate in the case of $n = 8$.

3.5.1.1 Experimental results

In this section, we present the experimental results using the compression model described in previous Section. We also compare its results with methods [106, 107] and JPEG-LS. In Table 3.9 we provide results for several methods including the ones indicated earlier and the proposed method. The column indicated as "Greedy" corresponds to the results using a greedy context size procedure that starts with size = 15 (see Figure 3.6) and stops when the obtained compression results are worse than the previous best. The last column corresponds to an exhaustive context size search routine where the best context size is always found. This means that all context sizes from 1 to 15 are evaluated and the one that attained the best compression performance is selected to be used. The context size requires an extra 4-bit flag that needs to be sent for each bitplane, as side information. If we look to the results of the

last two columns, we can see that they are very similar (they are in fact equal for 3 decimal places). In terms of encoding time, we can see that the "Greedy" version is almost 3 times faster when compared to the "Best" version. Compared to the other methods, the proposed approach attained a gain of $\approx 8\%$ when compared to JPEG-LS and $\approx 5\%$ when compared to method [106]. When compared to the "Full" version of method [107], the proposed method attained worse results, but it is much faster in the encoding phase. On the other hand, a compression gain of $\approx 9\%$ is observed when compared to the "SA-256" mode of method [107].

Table 3.9: Compression results in bits per pixel (bpp), using JPEG-LS, methods [106, 107] and the proposed method based on *bit modeling by pixel value estimates*. Results for two modes are presented in the table. One where the search area used to evaluate each context configuration have 256×256 pixels. The other one denoted as "Full" in the table, corresponds to a search area that covers the entire image. Regarding the proposed approach, two model for context size search are also presented. The best results are in bold. In the last four rows we present the encoding/decoding time ("CTime" and "DTime") in hours and speed ("CSpeed" and "DSpeed") in kilobytes per second, globally for all the data sets.

	Compression methods					
Data sets	**JPEG-LS**	**Neves [106]**	**Neves [107]**		**Proposed**	
			(256×256)	**(Full)**	**(Greedy)**	**(Best)**
ApoA1	10.608	10.280	10.225	**10.194**	10.205	10.205
Arizona	8.676	8.394	8.293	**8.242**	8.308	8.308
IBB	9.903	8.063	8.039	**7.974**	8.537	8.537
ISREC	11.145	10.217	10.199	**10.159**	10.260	10.260
Omnibus (LM)	4.936	5.309	5.679	**4.567**	4.645	4.645
Omnibus (HM)	6.952	7.047	7.744	**6.471**	6.581	6.581
Stanford	7.684	7.664	7.468	**7.379**	7.403	7.403
Yeast	8.580	5.610	5.511	**5.453**	5.492	5.492
YuLou	8.974	8.840	8.667	**8.619**	8.669	8.669
Average	6.996	6.772	7.101	**6.284**	6.437	6.437
CTime (hours)	0.18	3.97	6.24	643.19	21.40	60.95
DTime (hours)	0.19	4.30	3.67	6.00	4.91	4.90
CSpeed (KB/s)	11904	539	343	3	100	35
DSpeed (KB/s)	11246	498	583	357	434	437

As mentioned earlier, the proposed approach computes, for each bitplane, the context sizes that maximize the compression ratio. We decided to compare the two context size search routines. The one designated as "Greedy" tests several context sizes, starting with size = 15 and stopping when a worse result occurs compared with the previous best. The "Best" search routine test all possibles context sizes from 1 to 15 and always get the best context size. In Figure 3.8 we present the average context size for each bitplane using the two approaches described before. All data sets used in this work were considered to plot the results of Figure 3.8. As can be seen, the average context size between both modes ("Greedy" and "Best") are very similar along all bitplanes. The only exception is for bitplanes 14 and 15 where a more relevant difference is observed. This is due to the lack of statistical information

on the first bitplanes, where very few bits of information are available to build up the context. Nevertheless, the "Greedy" routine for finding the context size is recommended, due to the similar results in terms of context size and in terms of compression performance (encoding time and average bits per pixel). The previous conclusion was made taking into account also the results in Table 3.9.

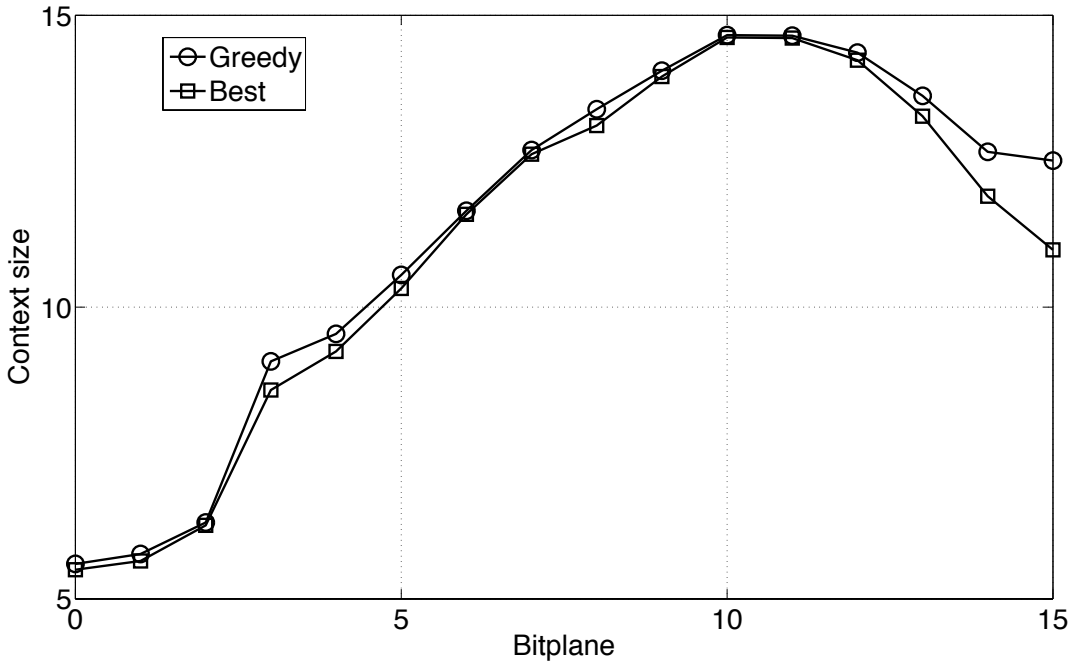


Figure 3.8: Context sizes for both "Greedy" and "Best" approach of the method proposed in this section, without mixture. The bitplane 0 corresponds to the LSBP, whereas the bitplane 15 corresponds to the MSBP. The curves correspond to the average context size used for each bitplane for all the data sets used in this work.

3.5.2 Mixture of finite-context models

Our initial approach was inspired on Kikuchi's method [140]. We decided to implement a second approach based on a mixture of models where two models are used to encode the input image. In this case we decided to use two methods, the method previously described in Section 3.5.1 and Neves's method [107]. Taking into consideration that we are dealing with a binary alphabet ($\mathcal{A} = \{0, 1\}$), for each model we assign probability estimates for each symbol, regarding the next outcome, according to a conditioning context computed over a finite and fixed number $k > 0$ of past outcomes $x_{n-k+1..n} = x_{n-k+1} \dots x_n$ (order-k finite-context model [73] with $|\mathcal{A}|^{\mathrm{k}}$ states, for the case of a binary alphabet). The probability estimates $P(x_{n+1}|x_{n-k+1..n})$ are calculated using symbol counts that are accumulated while processing each pixel of the input image, making them dependent not only on the past k symbols, but also on n. The estimator used is

$$P(s|x_{n-k+1..n}) = \frac{C(s|x_{n-k+1..n}) + \alpha}{C(x_{n-k+1..n}) + |\mathcal{A}|\alpha}, \tag{3.7}$$

where $C(s|x_{n-k+1..n})$ represents the number of times that, in the past, symbol s was found having $x_{n-k+1..n}$ as the conditioning context and where

$$C(x_{n-k+1..n}) = \sum_{a \in \mathcal{A}} C(a|x_{n-k+1..n}) \tag{3.8}$$

is the total number of events that has occurred so far in association with context $x_{n-k+1..n}$. Parameter α allows balancing between the maximum likelihood estimator and an uniform distribution (when the total number of events, n, is large, it behaves as a maximum likelihood estimator). For $\alpha = 1$, (3.7) is the well-known Laplace estimator.

The per symbol information content average provided by the finite-context model of order-k, after having processed n symbols, is given by

$$H_{k,n} = -\frac{1}{n} \sum_{i=0}^{n-1} \log_2 \Big(P(x_{i+1}|x_{i-k+1..i}) \Big) \tag{3.9}$$

bits per symbol. When using several models simultaneously, the $H_{k,n}$ can be viewed as measures of the performance of those models until that instant. Therefore, the probability estimate can be given by a weighted average of the probabilities provided by each model, according to

$$P(x_{n+1}) = \sum_{k} P(x_{n+1}|x_{n-k+1..n})\, w_{k,n}, \tag{3.10}$$

where $w_{k,n}$ denotes the weight assigned to model k and

$$\sum_{k} w_{k,n} = 1. \tag{3.11}$$

In order to compute the probability estimate for a certain symbol, it is necessary to combine the probability estimates given by (3.7) using (3.10). The weight assigned to model k can be computed according to

$$w_{k,n} = P(k|x_{1..n}), \tag{3.12}$$

i.e., by considering the probability that model k has been generated the sequence until that point. In that case, we would get

$$w_{k,n} = P(k|x_{1..n}) \propto P(x_{1..n}|k)P(k), \tag{3.13}$$

where $P(x_{1..n}|k)$ denotes the likelihood of sequence $x_{1..n}$ being generated by model k and $P(k)$ denotes the prior probability of model k. Assuming

$$P(k) = \frac{1}{K}, \tag{3.14}$$

where K denotes the number of models, we also obtain

$$w_{k,n} \propto P(x_{1..n}|k). \tag{3.15}$$

Calculating the logarithm we get

$$\log_2\Big(P(x_{1..n}|k)\Big) = \log_2 \prod_{i=1}^{n} P(x_i|k, x_{1..i-1}) = \tag{3.16a}$$

$$= \sum_{i=1}^{n} \log_2\Big(P(x_i|k, x_{1..i-1})\Big), \tag{3.16b}$$

which is related to the code length that would be required by model k for representing the sequence $x_{1..n}$. It is, therefore, the accumulated measure of the performance of model k until instant n. In order to obtain a good performance in each model, we decided to use a mechanism of progressive forgetting of past performances. This mechanism allows each model to progressively forget the past and, consequently, to give more importance to the most recent past. Therefore, we rewrite (3.16b) as

$$\sum_{i=1}^{n} \log_2\Big(P(x_i|k, x_{1..i-1})\Big) = \tag{3.17a}$$

$$= \gamma \sum_{i=1}^{n-1} \log_2\Big(P(x_i|k, x_{1..i-1})\Big) + \log_2\Big(P(x_n|k, x_{1..n-1})\Big), \tag{3.17b}$$

where $\gamma \in [0, 1)$ dictates the forgetting factor to be used. Defining

$$p_{k,n} = \prod_{i=1}^{n} P(x_i|k, x_{1..i-1}) \tag{3.18}$$

and removing the logarithms, we can rewrite (3.16) as

$$p_{k,n} = p_{k,n-1}^{\gamma} P(x_n|k, x_{1..n-1}) \tag{3.19}$$

and, finally, set the weights to

$$w_{k,n} = \frac{p_{k,n}}{\sum_k p_{k,n}}. \tag{3.20}$$

3.5.2.1 Experimental results

In this section, we present experimental results using the compression model based on a mixture of finite-context models, described in the previous section. We compared the obtained results with methods [106, 107] and JPEG-LS. The algorithm introduced in the previous section only uses two different models in its mixture core. One of the models is based on method [107] and the other one is the one presented in Section 3.5.1. In this case, we decided to use the greedy version of the method introduced in Section 3.5.1, because as mentioned earlier, it is much faster than the "Best" version (see Table 3.9). In Table 3.10, we present some experimental results of the proposed approach based on a mixture of model and also for methods [106, 107] and for the JPEG-LS standard. If we look at the attained results, we can notice a gain of $\approx 11\%$ of the proposed method when compared to method [107] for the "SA-256" mode. Regarding the other mode, the improvement is $\approx 0.4\%$, which is lower than the one attained by the "SA-256" mode.

Table 3.10: Compression results in bits per pixel (bpp), using JPEG-LS, methods [106, 107] and the model based on a mixture between method [107] and the method described in Section 3.5.1. Results for two modes are presented in the table. One where the search area used to evaluate each context configuration have 256×256 pixels. The other one denoted as "Full" in the table, corresponds to a search area that covers the entire image. The values with a "•" represent improvements between method [107] and the proposed method (local improvement). The best results are in bold. In the last four rows we present the encoding/decoding time ("CTime" and "DTime") in hours and speed ("CSpeed" and "DSpeed") in kilobytes per second, globally for all the data sets.

	Compression methods					
Data sets	**JPEG-LS**	**Neves [106]**	**Neves [107] (256×256)**	**Proposed (256×256)**	**Neves [107] (Full)**	**Proposed (Full)**
ApoA1	10.608	10.280	10.225	10.149 •	10.194	**10.142** •
Arizona	8.676	8.394	8.293	8.238 •	8.242	**8.219** •
IBB	9.903	8.063	8.039	8.001 •	7.974	**7.966** •
ISREC	11.145	10.217	10.199	10.169 •	10.159	**10.148** •
Omnibus (LM)	4.936	5.309	5.679	4.646 •	4.567	**4.545** •
Omnibus (HM)	6.952	7.047	7.744	6.571 •	6.471	**6.443** •
Stanford	7.684	7.664	7.468	7.331 •	7.379	**7.305** •
Yeast	8.580	5.610	5.511	5.354 •	5.453	**5.326** •
YuLou	8.974	8.840	8.667	8.609 •	8.619	**8.591** •
Average	6.996	6.772	7.101	6.343 •	6.284	**6.257** •
CTime (hours)	0.18	3.97	6.24	40.09	643.19	686.63
DTime (hours)	0.19	4.30	3.67	18.83	6.00	23.64
CSpeed (KB/s)	11904	539	343	53	3	3
DSpeed (KB/s)	11246	498	583	114	357	91

In Figure 3.9, we provide an overview of the models usage for the proposed method, that is based on a mixture between method [107] (denoted as "Neves" on the chart) and the method described in Section 3.5.1 (denoted as "SBC"). The average model usage is computed taking into account the number of times, that during compression, each model was the best one. The best model in the mixture is the one that, if considered alone and for each symbol, could generate the best compression results among the others (in this case is just one). Results are presented for individual bitplanes as well for the entire image in the last pair of bars. The bar chart presents the LSBPs on the left and the MSBPs on the right. We can divide the presented results into three groups. The first group, that corresponds to the three LSBPs, where method [107] had an average percentage usage of $\approx 67\%$, whereas the proposed method only had about 33% of percentage usage. The second group, corresponds to bitplanes 3-5, where both models seem to have on average $\approx 50\%$ usage for all data sets. The last group corresponds to the bitplanes 6-15, where the proposed method attains an average usage between $\approx 53\%$ and $\approx 77\%$. Globally, for the entire image, the model based on method [107] has a similar performance as the proposed method. Method [107] has an average usage percentage of $\approx 46\%$, whereas the proposed method is slightly best with an average usage percentage of $\approx 54\%$.

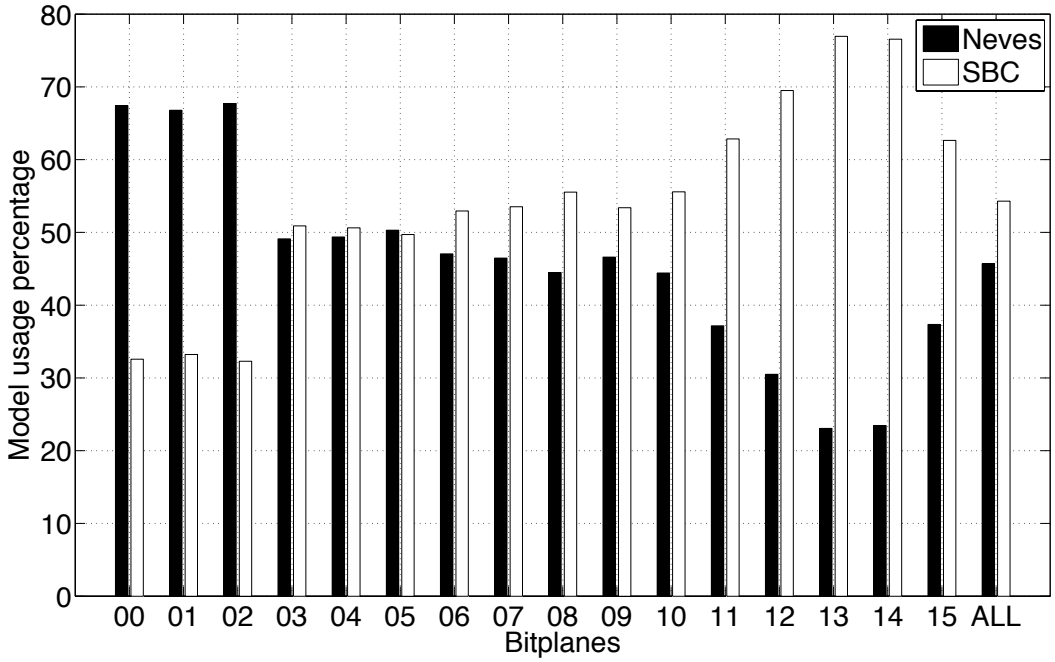


Figure 3.9: Average models usage of the proposed mixture method, for each bitplane and the overall for the entire image. The bars identified as "Neves" correspond to the model of method [107], whereas the other ones are related to the method described in Section 3.5.1. The LSBP is denoted as "00", the MSBP is identified as "15", and the results for all bitplanes are in the last pair of bars. The results were obtained using all the data sets.

3.5.3 Complexity

Similar to method [107], the method proposed in this section uses finite-context models that depend on the bitplane that is being encoded and the image itself. The context size can go up to 15 pixels, as can be seen in Figure 3.6, for each bitplane. Taking into consideration that we are using a binary alphabet, the maximum number of counts for the model used to encode each image is $2 \times 2^{15} = 65,536$. The counter used is stored in 2 bytes and the total amount of computer memory required is approximately 128 kilobytes. For the second approach where a mixture of models is used, the amount of memory is higher. Method [107] is one of the methods used in the mixture that requires about 4 megabytes of computer memory for the $2,097,152$ counters. This means that 5 megabytes are enough for storing all the counters, in the case of the approach based on a mixture of models.

In terms of speed, the first version of the proposed method where no mixture was applied, the encoding procedure is slower compared to all the methods illustrated in Table 3.9. The only exception is the "Full" mode of method [107]. In terms of decoding, the proposed method attains similar results, when compare to methods [106, 107]. For the second version of the proposed method where a mixture of models was used, the encoding/decoding time is always worse when compared to all the other methods. This lower coding time performance is due to the nature of the mixture model. For each symbol (each bit of each bitplane), it is necessary to compute the model mixture before encoding and update the model weights after encoding. This requires extra computing time that considerably increases the encoding/decoding time.

3.6 Proposed method based on binary tree decomposition

In Section 3.4, we described several improvements that were applied to a compression method based on a bitplane decomposition approach. Bitplane decomposition is a technique where the input image is split in several planes, each one corresponding to one bit of the pixel value. In the literature we can find other alternatives to process an image that are not based on a bitplane decomposition approach. One of those alternatives is known as Binary Tree Decomposition BTD. Chen *et al.* [65] was one of the first authors that used this kind of decomposition. In their work, they introduced a compression scheme for color-quantized images based on progressive coding of color information where instead of sorting the color indexes into a linear list structure, they used a binary tree structure of color indexes. Using that binary tree structure, their algorithm can progressively recover an image from two colors to all the colors contained in the original image (lossy-to-lossless capability). Inspired by the work done by Chen *et al.* [65], a few years later Neves and Pinho [66–68] developed a lossy-to-lossless method based on binary tree decomposition and context-based arithmetic coding. They studied the performance of their approach in several kinds of images from 8 up to 16-bit images, including medical images.

This binary tree decomposition approach was intended to be used in images with a small number of intensities, usually with 8 or less bits per pixel, due to a tight relation between the processing time and the number of different intensities of the image. In this work, we further extended this approach to be able to handle images with a large number of intensities, as is the case of microarray images. The main goal here was to evaluate the performance of this approach using microarray images. According to Table 3.1 of Section 3.1, 5 out of 9 of the data sets described have an average of intensity usage lower than 50%. This particular characteristic led us to believe that this approach is worth to be studied. Furthermore, this approach also has progressive decoding capability, which means that the decoding process can be stopped at any moment according to a specific distortion metric, obtaining a image with some loss. Moreover, it is possible to obtain the original image without any loss if the full decoding process is performed. In this section we describe in more detail a compression algorithm for microarray images using a binary-tree decomposition and context-based arithmetic coding.

3.6.1 Hierarchical organization of the intensity levels

This method is based on a hierarchical organization of the intensities levels of the image. This organization of the intensity levels is attained by means of a binary tree. Each node of the binary tree, n, represents a certain subset, $\mathcal{S}^n$, of the intensities of the image. The root node contains all active pixel values of the image $\mathcal{I} = \{I_1, I_2, \ldots, I_N\}$, where N represents the number of different intensities that occur in the image. Therefore, $\mathcal{S}^n \subset \mathcal{I}$ and $\mathcal{S}^1 \equiv \mathcal{I}$. Each node possesses a representative intensity, I^n, given by

$$I^n = \left\lfloor \frac{I^n_m + I^n_M}{2} \right\rfloor, \tag{3.21}$$

where I^n_m and I^n_M are, respectively, the smallest and largest pixel value in $\mathcal{S}^n$, and where $\lfloor x \rfloor$ denotes the largest integer less than or equal to x. Computing the value of I^n according to (3.21) leads to the smallest possible L_∞ reconstruction error when the intensities associated

to node n (those in $\mathcal{S}^n$) are all substituted by I^n. The error is given by

$$\epsilon_\infty^n = I_M^n - I^n. \tag{3.22}$$

In order to better understand the construction of the binary tree, we present in Figure 3.10 a small example for an image with only five active pixel values $\{32, 50, 250, 33768, 65530\}$. The construction of this tree begins with the association to the root node of the set of intensities that occur in the original image. After this association, it is necessary to compute I^1 according to (3.21). In the example depicted in Figure 3.10, $I^1 = \lfloor (32+65530)/2 \rfloor = 32781$ and $\epsilon_\infty^1 = 65530 - 32781 = 32749$, for the root node. The next step consists in splitting the root node into two sub-nodes and, therefore, splitting $\mathcal{S}^1$ into two subsets. In order to split $\mathcal{S}^1$, we need only to compare the intensity $I \in \mathcal{S}^1$ with I^1. The intensities lower than I^1 are associated with the left node, and the other ones with the right one. This procedure is repeated until expanding all nodes, i.e., until having a tree with N leaves (N is the number of active intensities presented in the original image). The next node to expand is chosen taking into consideration the smallest possible L_∞ reconstruction error. In case of a tie, one is arbitrarily chosen, although it is necessary that the decoder picks the same one. In order to the decoder be able to build the same tree, it is necessary to send the information of the active pixels values to the decoder using a 65536-bit indicator. In order to encode this indicator the encoder uses the following strategy. First, the maximum intensity value I_N, is sent. After that, a string of I_n bits is transmitted, such that if the n^{th} bit of the string is one it means that the intensity $n-1$ is present in the image and zero otherwise. The previous 65536-bit indicator is enough for the decoder construct the exactly same binary tree.

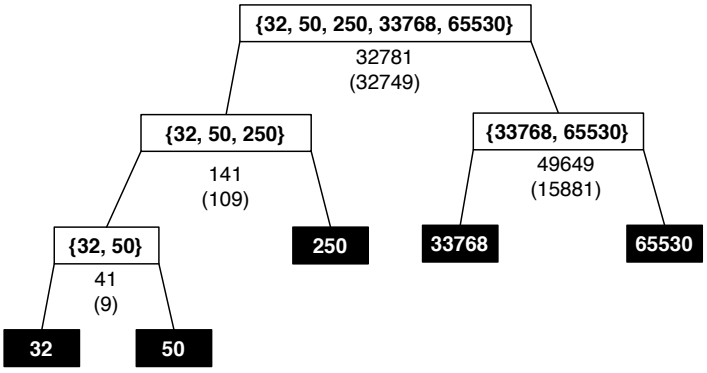


Figure 3.10: Example of a small binary tree that illustrates the hierarchical organization of the intensity values for an image with five active pixel values $\{32, 50, 250, 33768, 65530\}$.

3.6.2 Encoding pixel locations

After each node is expanded, two new nodes are created each one with a representative intensity (I_l^n for the left node and I_r^n for the right node). This step can be seen as a region of arbitrary shape, containing zeros (relative to the left node) and ones (regarding the right node), that needs to be communicated to the decoder. The position where the pixels in the image are associated with the parent node that was expanded are known by the decoder. However, it is necessary to communicate to the decoder the zeros and ones that correspond to the pixels that after the expand procedure will be associated to the left and right nodes

respectively. Since the decoder has access to the pixels associated to the parent node that was expanded, it is enough to encode a binary mask, where a zero indicates that the pixel needs to change its intensity to I_l^n and a one indicates a change to I_r^n. This binary mask is encoded using arithmetic coding based on variable size finite-context models [23, 73, 74].

The performance of the compression method is directly dependent on the encoding of these binary masks. The encoding efficiency of these binary masks can be controlled by a carefully chosen context modeling that will then drive the binary arithmetic encoder. The context are constructed based on the template depicted in Figure 3.11. The number of context pixels can go up to sixteen at most and they are numbered according to their distance to the encoding pixel (represented in gray in Figure 3.11). A particular context is represented using a sequence of bits,

$$b_1 b_2 \dots b_k \tag{3.23}$$

where

$$b_i = \begin{cases} 0, & \text{if} \quad |I(i) - I_l^n| \leq |I(i) - I_r^n| \\ 1, & \text{otherwise} \end{cases},$$

and where $I(i)$ denotes the intensity of the pixel in the current reconstructed image corresponding to position i of the context template.

The value k defines the model order used. In this case, the k value varies as the encoding proceeds. This variation is necessary in order to improve the compression performance. Furthermore, it is expected to have larger mask regions initially in the first nodes that are expanded, and smaller regions when $n \approx N$. This variation is also useful to avoid the problem of context dilution. In this research work we present two modes of context creation. One denoted as "Greedy" where the context size is first chosen using a k value according to [143]. After that, the method test incrementally several context sizes bigger and smaller than k and stops when reaches one context than produces worse results than the previous best. In the end, the algorithm has two context sizes. One attained when applying an increment to k and the other one when applying a decrement to k. The best context size is then chosen to encode the binary mask. The other mode is slower because it tests all possible context sizes. This second mode, denoted as "Best", always attains the best context size that minimizes the bitrate. For both cases, the context size needs to be sent to the decoder, for each node that is expanded. There is also an alternative way to encode the binary mask. If the number of bits required to encode the mask and the context size is bigger than the total number of pixels associated with the node to be expanded, the encoder sent the binary mask as a binary string, without compression. In order to the decoder differentiate between these two modes, a binary stream is needed to be encoded for each node that is expanded.

3.6.3 Experimental results

In what follows, we present the compression results attained by the proposed method based on a binary tree decomposition. We compared the obtained results with methods [106], [107] and the best image coding standard for microarray images JPEG-LS. In Table 3.11 we can find the obtained results for the different data sets used in this research work. Similar to the results presented in the previous section, we provide results for two modes of method [107]. One where the search area used to evaluate the context configuration was 256×256 pixels (denoted in Table 3.11 as "256×256"). The other mode presented, designated as "Full", corresponds to a search area comprising the complete image. This last mode is much slower due to the large

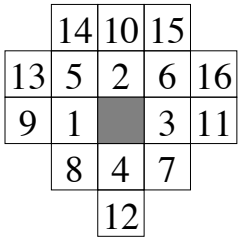


Figure 3.11: Context template used in this work. The use of non-causal pixels is possible, because context information can be obtained from the previous version of the reconstructed image.

sizes that are typical in microarray images (see Table 3.1 for more informations). Regarding the proposed method, there are also two modes that we used during our experiments. One of them denoted as "Greedy" in Table 3.11, where a locally optimal solution is computed, not necessarily the globally best solution. In this case, instead of testing all possible context sizes, we progressively test several sizes until obtaining a bitrate worse than the previous "best". In the other mode, designated as "Best", all possible sizes from 1 to 16 (see template presented in Figure 3.11) are tested. This solution is slower than the previous mode, but it is guaranteed that the best solution is always found. According to the results presented in Table 3.11, we can conclude that the results, on average, are quite similar among both approaches ("Greedy" and "Best") in the proposed method. Furthermore, the "Best" version of the proposed method attained $\approx 9\%$ better results when compared to the best compression standard (JPEG-LS). On the other hand, the "Full" version of method [107] attained $\approx 8\%$ better results when compared to the best compression standard (JPEG-LS).

3.6.4 Complexity

Taking into account the template depicted in Figure 3.11, we have at most of 16 context pixels that can be used. As mentioned earlier, the key aspect of this method is the encoding of the binary masks. Given the alphabet size and the maximum allowed finite-context that can be used, the maximum number of counter used by the method to encode each image is $2 \times 2^{16} = 131,072$. As in the previous approaches described, each counter can be stored in two bytes, so the total amount of computer memory required to store the counters is 256 kilobytes.

Taking into account rows "CTime" and "DTime" of Table 3.11, we can conclude that in the encoding phase, the *Full* mode of method [107] is much slower when compared to method proposed in this section (for both modes "Greedy" and "Best"). For the *SA-256* mode, method [107] is faster than the proposed method. In the decoding phase, the proposed method took about 15 hours to decode all data sets. On the other hand, method [107] only took ≈ 3.7 hours for the *SA-256* mode and ≈ 6 hours for the *Full* mode to decode all data sets. The decoding phase for the proposed approach seems to be slower than method [107].

In Tables 3.12 and 3.13 we can observe in more detail the encoding and decoding time in minutes for the JPEG-LS standard, for methods [106],[107], and for the proposed approach

Table 3.11: Compression results in bits per pixel (bpp), using JPEG-LS, methods [106, 107] and the proposed improvement based on a Binary Tree Decomposition (BTD). Regarding method [107], two modes are presented. One where the search area used to evaluate each context configuration have 256×256 pixels. The other one denoted as "Full" in the table, corresponds to a search area that covers the entire image. Regarding the proposed method, we also present two modes for context creation. The "Greedy" mode where several context sizes are tested until a size that produces worse results than the previous best. On the other hand, the "Best" mode tests all possible context sizes but is much more slower than the previous mode. The best results are in bold. In the last four rows we present the encoding/decoding time ("CTime" and "DTime") in hours and speed ("CSpeed" and "DSpeed") in kilobytes per second, globally for all the data sets.

	Compression methods					
Data sets	**JPEG-LS**	**Neves [106]**	**Neves [107]**		**Proposed**	
			(256 × 256)	**(Full)**	**(Greedy)**	**(Best)**
ApoA1	10.608	10.280	10.225	**10.194**	10.199	**10.194**
Arizona	8.676	8.394	8.293	8.242	**8.186**	**8.186**
IBB	9.903	8.063	8.039	7.974	**7.943**	**7.943**
ISREC	11.145	10.217	10.199	**10.159**	10.200	10.198
Omnibus (LM)	4.936	5.309	5.679	4.567	4.540	**4.539**
Omnibus (HM)	6.952	7.047	7.744	6.471	6.401	**6.400**
Stanford	7.684	7.664	7.468	7.379	7.306	**7.303**
Yeast	8.580	5.610	5.511	5.453	5.323	**5.318**
YuLou	8.974	8.840	8.667	8.619	8.593	**8.592**
Average	6.996	6.772	7.101	6.284	6.236	**6.235**
CTime (hours)	0.18	3.97	6.24	643.19	63.10	168.33
DTime (hours)	0.19	4.30	3.67	6.00	14.42	15.70
CSpeed (KB/s)	11904	539	343	3	34	13
DSpeed (KB/s)	11246	498	583	357	148	136

for each data set. In the encoding phase, we can notice that the proposed approach in the "Best" mode is faster than method [107] in *Full* mode. The only exceptions are the *ApoA1* and *ISREC* data sets. This two data sets have a percentage of active intensities lower than 40%, but they are also the ones with the highest entropy values (see Table 3.1). In the decoding phase, the proposed approach seems to be globally slower when compared to the other methods depicted in Table 3.11. The only exception is the *Yeast* data set, where the proposed approach is faster when compared to methods [106, 107]. We believe that the performance of the decoding phase of this approach in terms of time is dependent on the percentage of active intensities. Moreover, it is easy to conclude that the performance of the decoding phase only outperforms methods [106, 107] for data sets with very low percentage of active intensities (near 5%).

Table 3.12: Encoding time in minutes for JPEG-LS, methods [106, 107] and the proposed method based on BTD.

Data sets	Compression methods					
	JPEG-LS	Neves [106]	Neves [107]		Proposed	
			(256 × 256)	(Full)	(Greedy)	(Best)
ApoA1	0.11	5.19	13.89	167.66	97.17	273.03
Arizona	1.07	19.93	32.40	3,005.30	389.07	1,226.48
IBB	1.71	35.54	64.63	4,675.74	605.66	1,802.10
ISREC	0.05	2.78	5.84	79.06	31.51	96.35
Omnibus (LM)	3.38	64.62	86.42	14,608.98	947.32	2,161.82
Omnibus (HM)	3.69	69.24	85.26	13,405.58	1,402.42	3,703.99
Stanford	0.59	16.85	33.41	1,765.24	216.54	582.15
Yeast	0.33	22.38	59.82	719.38	65.23	183.33
YuLou	0.07	1.60	2.89	163.95	25.22	70.73
Total	10.99	238.12	374.10	38,591.30	3,780.13	10,100.00

Table 3.13: Decoding time in minutes for JPEG-LS, methods [106, 107] and the proposed method based on BTD.

Data sets	Compression methods					
	JPEG-LS	Neves [106]	Neves [107]		Proposed	
			(256 × 256)	(Full)	(Greedy)	(Best)
ApoA1	0.13	5.37	2.36	2.66	34.98	39.05
Arizona	1.15	21.83	26.18	31.79	70.98	81.83
IBB	1.86	38.12	38.45	48.12	140.74	161.07
ISREC	0.05	2.78	0.93	1.08	12.49	11.81
Omnibus (LM)	3.49	71.47	61.68	121.82	217.25	227.94
Omnibus (HM)	3.91	75.46	67.82	126.10	332.13	358.61
Stanford	0.65	18.12	13.31	17.83	44.33	46.48
Yeast	0.37	22.87	7.99	8.96	7.55	7.71
YuLou	0.07	1.71	1.46	1.82	4.65	4.86
Total	11.67	257.71	220.18	360.18	865.10	942.07

3.7 Rate-distortion study

There are several measures that can be used to evaluate compression tools. The most common one is related to the amount of bits that is required to store a given data. If the amount of bits required by method A is lower than method B, it is clear that method A is better. On the other hand, there are other measures that can be used. For example, the rate-distortion measure is associated with the distance between the decoded data and the original one [144]. Why is this measure important? Sometimes the available resources in terms of bandwidth are limited. In this scenario, we are interested in methods that can provide a

more precise (with less error as possible when compared to the original image) image using a particular rate. In addition, there are also other situations where we only intend to decode a partial image (e.g., if the user is using a portable device with a low resolution screen) where it is important that the decoded image is as precise as possible. In this context, methods that provide the best precise image using a particular rate are essential.

In the literature, we can find several rate-distortion metrics that are usually used to measure the performance of compression algorithms. In this section we present a rate-distortion evaluation of two image coding standards, JBIG and JPEG2000 with methods [107] and the method described in Section 3.6. We did not include results for other standards, because only JBIG and JPEG2000 have support for lossy-to-lossless compression. After plotting the rate-distortion curves of several microarray images, we verified that the attained results are very similar, regardless of the image/data set used. In our experiments, we decided to use the "1230c1G" from the *ApoA1* data set and "array1" from the *YuLou* data set.

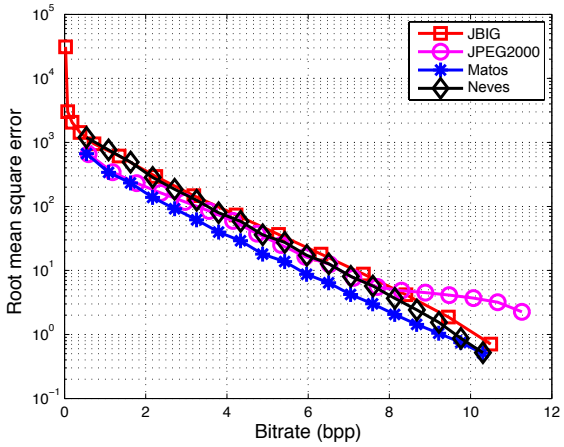
In Figure 3.12 we can find the rate-distortion curves of the two previously cited microarray images in terms of Root Mean Square Error (RMSE). After analyzing the rate-distortion curves, we can conclude that JPEG2000 provides similar results for lower bitrates (lower than 8 bpp), when compared to JBIG, the method [107] and the method detailed in Section 3.6. On the other hand, the JPEG2000 curves have a sudden deviation for higher bitrates. We believe that this deviation can be explained by the default parameters used in JPEG2000. These default parameter values could not be suited for this kind of images with 16 bits per pixel and their particular characteristics in terms of noise, histogram sparseness, etc.

Regarding the L_∞-norm, we provide the rate-distortion curves as well in Figure 3.13. In this case, we can notice that JPEG2000 has in fact the worst rate-distortion results for the two images used in our experiments. These conclusions are also true for other images. Similar to the previous distortion metric, we can see that once again the JPEG2000 curves suffer a sudden deviation for higher bitrates, which is probably related to the same problem pointed out earlier. If we now compare the obtained results of methods [107] and the method introduced in Section 3.6, we can see that our method provides better rate-distortion results when compared to method [107]. The main reason to the previous statement is due to the nature of method proposed in Section 3.6. According to what we mentioned in that section, the proposed method has in its core a mechanism that minimizes the L_∞ error along the encoding/decoding process. This allows the method to obtain better rate-distortion results. Furthermore, method [107] was not designed with an error minimization goal in mind. It processes each bitplane of the microarray image without looking to any kind of error metric. The error after processing each bitplane depends on the remaning information that was not yet processed (on the lower bitplanes).

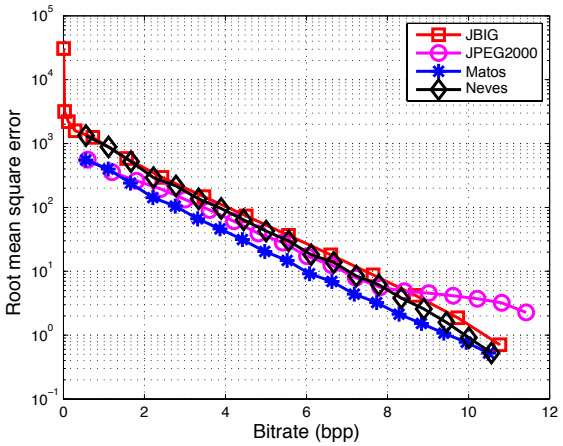
In 2013, Hernández-Cabronero *et al.* [127] proposed a novel microarray-specific distortion metric to assess the loss of relevant information. This distortion metric, known as Microarray Distortion Metric (MDM), takes into account the basic image features employed by most DNA microarray analysis techniques. The metric is computed taking into account three main features: the mean intensity ratio of the spots, the average intensity of the local background and the global image intensity. According to [127], the MDM is defined as

$$\mathrm{MDM} = 10 \log_{10} \left(\frac{\mathrm{max\_val}^2}{\mathrm{ME}} \right), \tag{3.24}$$

where "max_val" is the maximum intensity of the image and the microarray error (ME) is a noise measure that is sensitive to relevant changes in any of the three main features mentioned

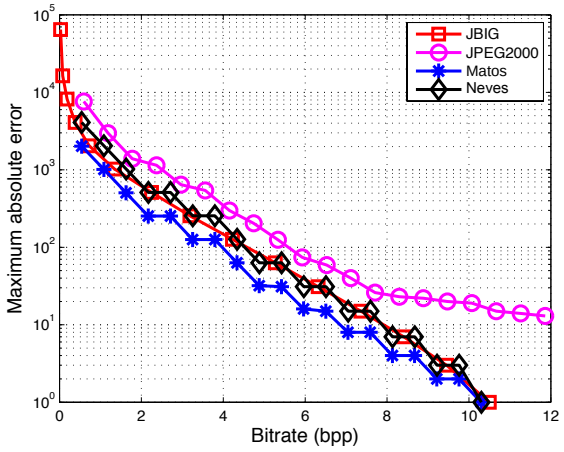


"1230c1G" (*ApoA1* data set)

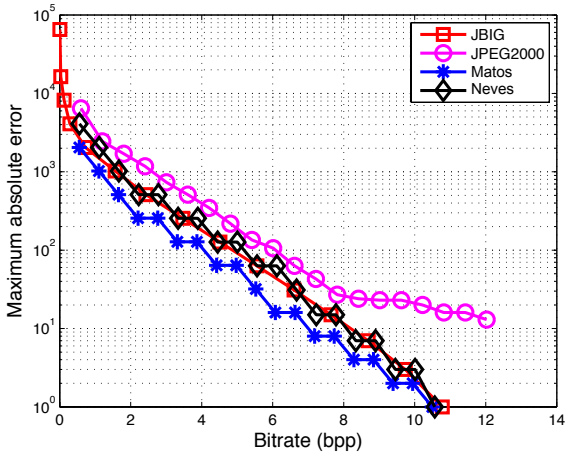


"array1" (*YuLou* data set)

Figure 3.12: Rate-distortion curves for methods [107], the proposed method described in Section 3.6, JBIG, and JPEG2000, regarding images "1230c1G" from the *ApoA1* data set (on the left side) and "array1" from the *YuLou* data set (on the right side). Results are given in terms of L2-norm (Root Mean Squared Error or RMSE). The curves indicated as "Matos" correspond to the method introduced in Section 3.6, whereas the curves indicated as "Neves" correspond to method [107].



"1230c1G" (*ApoA1* data set)



"array1" (*YuLou* data set)

Figure 3.13: Rate-distortion curves for methods [107], the proposed method described in Section 3.6, JBIG, and JPEG2000, regarding images "1230c1G" from the *ApoA1* data set (on the left side) and "array1" from the *YuLou* data set (on the right side). Results are given in terms of L∞-norm (Maximum Absolute Error or MAE). The curves indicated as "Matos" correspond to the method introduced in Section 3.6, whereas the curves indicated as "Neves" correspond to method [107].

earlier. For this metric, the authors employed the following expressions to calculate the

distortion of the three key microarray image features:

$$r_{\text{spot}} = \max\left(\text{max\_sr}, \frac{1}{\text{min\_sr}}\right), \tag{3.25}$$

$$r_{\text{localBG}} = \max\left(\text{max\_lBGr}, \frac{1}{\text{min\_lBGr}}\right), \tag{3.26}$$

$$r_{\text{global}} = \max\left(\text{global\_ir}, \frac{1}{\text{global\_ir}}\right), \tag{3.27}$$

where "max_sr", "min_sr", "max_lBGr", "min_lBGr" and "global_ir" stand for maximum spot ratio, minimum spot ratio, maximum local background ratio, minimum local background ratio and global intensity ratio respectively. When such relevant distortions are introduced in any of the three key image features, the MDM should decrease towards 0. In order to achieve this, the definition of ME is based on "max_val" raised to p, a logistic function of r_{spot}, r_{localBG} and r_{global}

$$p = \frac{2}{1 + \exp(-\alpha(r_{\text{spot}} + r_{\text{localBG}} + r_{\text{global}} - 3))}, \tag{3.28}$$

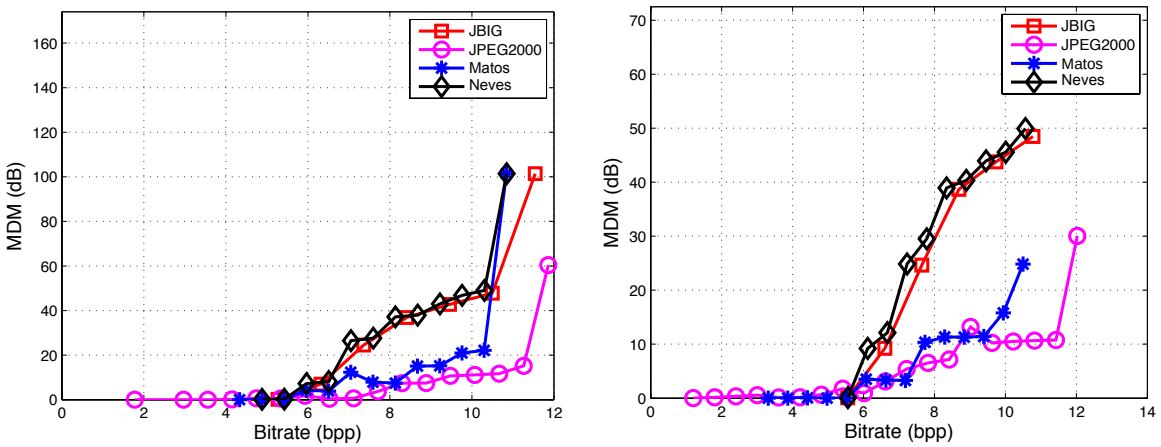
$$\text{ME} = (\text{max\_val})^p - \text{max\_val} + \min(\text{max\_val}, \text{MSE}_{\text{image}}). \tag{3.29}$$

The sensitivity of the MDM to changes in the three key images features can be adjusted through the α parameter. The authors found out that $\alpha = 3$ is a balanced choice for this distortion metric. Using their distortion metric, we were able to create some charts showing this MDM for JBIG, JPEG2000, method [107], and our method described in Section 3.6. Figure 3.14 shows the MDM for the two images used in this study for the four compression methods mentioned earlier. After analyzing the MDM results, we can observe that the curves for JBIG and method [107] are very similar, because both rely on a bitplane decomposition approach. Apparently, methods based on bitplane decomposition attain better results in terms of MDM, when compared to other approaches such as the ones used in JPEG2000 and method [16]. Some irregularities can be found in some images that are probably caused by the nature of how the MDM is computed. It is important to clarify that none of the methods evaluated in this section take into account the MDM. However, method [107] and our method described in Section 3.6 can be modified in order to use the MDM in their core.

The MDM introduced by Hernández-Cabronero *et al.* requires a segmentation step that identifies the spots of each images. The authors used a Matlab implementation of the circular Hough transform [145]. The results obtained using the circular Hough transform are not quite perfect, so they need to be further refined due to some false positive spots and other unidentified ones. This refinement procedure requires a specialized user that is familiar with the microarray technology. Furthermore, taking into account the typical size of the microarray images and the number of spots that each one has, the amount of time required to detect/refine the spots is tremendous.

3.8 Summary

This chapter presented several methods for microarray image compression. Several improvements were added to a bitplane decomposition method [107]. Furthermore, an alternative approach based on Binary-Tree Decomposition (BTD) was presented.



"1230c1G" (*ApoA1* data set) "array1" (*YuLou* data set)

Figure 3.14: Rate-distortion curves for methods [16, 107], JBIG, and JPEG2000, regarding images "1230c1G" from the *ApoA1* data set (on the left side) and "array1" from the *YuLou* data set (on the right side). Results are given in terms of the Microarray Distortion Metric (MDM) introduced by Hernández-Cabronero *et al.* [127]. The curves indicated as "Matos" correspond to method [16], whereas the curves indicated as "Neves" correspond to method [107].

At the beginning of the chapter, we presented the microarray image data sets used in this research work. We also tested the performance of several standard image coding methods, namely JBIG, PNG, JPEG-LS, and JPEG2000. According to the obtained results, we concluded that globally the JPEG-LS standard is the one with the best compression performance. These results were then used as a reference point in order to evaluate the specialized methods described in this chapter.

Segmentation was the first improvement added to method [107]. The goal was to try to understand the effects of image segmentation in the performance of method [107]. According to the obtained results, we can observe a minor improvement of $\approx 1\%$ for mode *SA-256* of the proposed approach. On the other hand, in the second mode (*Full*), the proposed approach based on segmentation produced similar results when compared to the same mode of method [107]. Despite this, there are two data sets where minor improvement are observed in *Full* mode (data sets *Stanford* and *Yeast*). In terms of encoding time, method [107] is ≈ 1.3 and ≈ 1.5 times faster when compared to the proposed approach, for modes *SA-256* and *Full*, respectively. This low performance of the encoding time for the proposed approach is due to the threshold search procedure used to perform the segmentation step. The decoding time is very similar between method [107] and the proposed approach.

The second improvement added to method [107] is known as bitplane reduction. We used two forms of bitplane reduction: Histogram Compaction (HC) and Scalable Bitplane Reduction (SBR). Globally, the results of both bitplane reduction methods are very similar to method [107]. Even so, there are some data sets where a small improvement is observed. In terms of encoding/decoding time, we attained very similar results between the presented bitplane reduction approach and method [107].

Section 3.5 introduced a compression method inspired on method [140]. The method

uses pixel value estimates to build up the context model to encode each pixel. The obtained results outperform JPEG-LS, method [106], and method [107] (for the *SA-256* mode). A second compression method was also presented in Section 3.5, based on a mixture of models. The attained results are better when compared to both modes of method [107].

In Section 3.6, we described a compression method based on a hierarchical organization of the intensity levels of the image. This organization is attained by means of a binary tree. According to the obtained results, the proposed approach attained $\approx 9\%$ better results when compared to JPEG-LS. On the other hand, the best results that method [107] attained were $\approx 8\%$ better when compared to JPEG-LS. In terms of encoding time, we concluded that this approach is faster that the *Full* mode of method [107]. However it is slower when compared to the *SA-256* mode. Regarding the decoding phase, the proposed method is always slower when compared to the other methods. We believe that this lower performance in the decoding phase is due to the size of the binary tree (number of leafs). Due to the nature of this method, that is more appropriated to simple images with a lower percentage of active intensities, we concluded that the decoding phase is much more effective (in term of coding time) in microarray images with a percentage of active intensities close to 5% (example the *Yeast* data set).

In Appendix B we provide a global view of the results obtained by all the methods proposed in this chapter. According to the results presented in Table B.1, we can conclude that the method based on a hierarchical organization of the intensity levels of the image, described in Section 3.6, is the one that provides better results.

In the last part of this chapter, we presented a rate-distortion study in order to evaluate the method presented in Section 3.6 and method [107]. We also included two image coding standards, JBIG and JPEG2000 in our study. In terms of RMSE and MAE, it seems that the method [16] (described in Section 3.6) is the one that attained better ratio-distortion results, when compared to the other three methods evaluated. We also included results for a recent distortion metric specially designed for microarray images. According to the obtained results, we concluded that methods based on a bitplane decomposition, such as JBIG and method [107], attain better results in terms of MDM when compared to JPEG2000 and method [16].

We believe that the microarray image compression methods are near their limits. There are some minor improvements that can be attained, using some approaches, but it is not usually attained for all data sets. The reason for this is probably caused by the way that the microarray images of each data set are obtained. Different modes and methods are used, which causes data sets to differ greatly in terms of entropy, spot configuration, noise, number of spot regions, etc. Moreover, this type of images typically have a considerably amount of noise, particularly in the lower bitplanes. This noise is the bottleneck of the lossless compression methods. No matter how sophisticated a given model is, it will not be effective in regions with a considerable level of noise.

"The true sign of intelligence is not knowledge but imagination."

Albert Einstein

4

Compression of whole genome alignments

This chapter is based on:

- **L. M. O. Matos**, A. J. R. Neves, D. Pratas, and A. J. Pinho, "MAFCO: a compression tool for MAF files", *PLoS ONE*, vol. 10, no. 3, pp. e0116082, March 2015.
- **L. M. O. Matos**, D. Pratas, and A. J. Pinho, "A compression model for DNA Multiple Sequence Alignment Blocks", *IEEE Transactions on Information Theory*, vol. 59, no. 5, pp. 3189–3198, May 2013.
- **L. M. O. Matos**, D. Pratas, and A. J. Pinho, "Compression of whole genome alignments using a mixture of finite-context models", in *Proceedings of International Conference on Image Analysis and Recognition, ICIAR 2012*, ser. LNCS, Eds. A. Campilho and M. Kamel, pub. Springer, vol. 7324, pp. 359–366, Aveiro, Portugal, June 2012.

4.1 Whole genome alignments

Computational genome annotations and evolutionary genomics are two molecular biology research areas that use multiple genome alignment data. The alignment of DNA sequences has been used to help locating certain kinds of functional non-coding regions [7] and more recently for finding protein-coding genes [8, 9] and non-coding RNA genes [10]. Moreover, it is possible to observe the similarities and differences between the DNA sequences of humans and other species that share a common ancestral, providing critical data for finding the course of evolution. Furthermore, we can also perform a computational reconstruction of ancestral genome sequences that explains certain characteristics of species [146]. DNA sequences that have evolved from the same ancestral sequence are called homologous. In the case of genes,

they are likely to encode similar functions and each function that is experimentally verified in one species can be mapped to a homologous gene in other species.

The detection of homologous sequences in different genomes is a computationally non-trivial task, because different genomes of different species can greatly differ due to mutations that occurred during the species evolution [147]. There are two kinds of mutations that can be found in the genome. The first one, known as large scale mutations, affect large regions, leading to reorganizations of the whole genome. Duplications, deletions, insertions, inversions and transactions are examples of large scale mutations that can occur through genetic recombination of the DNA [148]. The second one, the small scale mutations, affect the DNA sequence only locally, changing a single or several neighboring nucleotides. There are three subtypes of small scale mutations that affect DNA: substitutions, insertions, and deletions. In a substitution, a DNA nucleotide is changed by another one. Insertion and deletion mutations (*InDels*) remove/insert a single or multiple nucleotides from/into the DNA sequence [149].

Due to large scale mutations, different genomes can greatly differ in size, which is a problem if we want to compare different genomes. The process of sequence alignment is used to describe parts of the genome that have evolved from a common ancestor. The algorithm is usually divided in two main steps. In the first step, homologous regions in different species are identified. Homologous regions that have diverged considerably cannot be identified properly, since they cannot be distinguished from the other non-homologous ones. In the second step, the identified homologous sequences from different species are properly aligned into several MSABs (Multiple Sequence Alignment Blocks), accounting for substitutions and small scale *InDels*. The missing entries caused by the *InDels* are filled with a gap symbol '-'. First, the evolutionary closest sequences are aligned. Then, the process is progressively repeated, until all identified homologous sequences have been included into a single MSAB [150].

In Figure 4.1, we can find a small MSAB example of homologous sequences from the human, platypus, chicken, lizard, and zebrafish genomes. The gaps that were inserted in positions 9-15 were likely caused by a deletion in the lizard genome. On the other hand, the gaps in positions 21-28 were probably caused by an insertion in the chicken genome. A part of the phylogenetic tree describing the evolutionary relationship of the 28 vertebrates is depicted in Figure 4.2.

		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41
human	1	C	C	G	C	T	G	C	T	A	A	C	C	C	C	A	T	A	C	C	C	-	-	-	-	-	-	-	-	C	G	A	A	-	-	-	C	C	A	A	C	C
platypus	2	C	T	T	T	T	C	T	T	G	G	T	C	T	C	A	T	T	C	C	C	-	-	-	-	-	-	-	-	A	G	G	A	A	G	T	C	T	A	A	A	C
chicken	3	C	A	A	T	C	G	T	T	A	T	T	T	A	T	A	T	T	G	T	T	A	A	T	T	A	G	C	A	A	A	C	A	-	-	-	C	A	-	-	-	-
lizard	4	C	A	T	T	A	G	T	T	-	-	-	-	-	-	-	C	T	G	T	T	-	-	-	-	-	-	-	-	A	G	A	A	-	-	-	C	A	-	-	-	-
zebrafish	5	C	C	G	T	T	T	T	T	A	G	C	C	T	A	A	A	A	A	C	C	-	-	-	-	-	-	-	-	C	C	A	A	-	-	-	C	T	A	A	-	-
estimated ancestor		C	A	G	T	T	G	T	T	A	G	C	C	T	C	A	T	T	C	C	C	A	A	T	T	A	G	C	A	A	G	A	A	A	G	T	C	T	A	A	C	C

Figure 4.1: Example of an alignment of five homologous sequences and the estimated maximum *a posteriori* common ancestor nucleotide for each column.

In the literature, we can find several MSA algorithms [151–157]. The most recent algorithms rely on heuristic optimization strategies and require huge computational resources. Despite the computational requirements, we can find alignments of whole genomes in large databases, such as those of USCSC [158] and Ensembl [159].

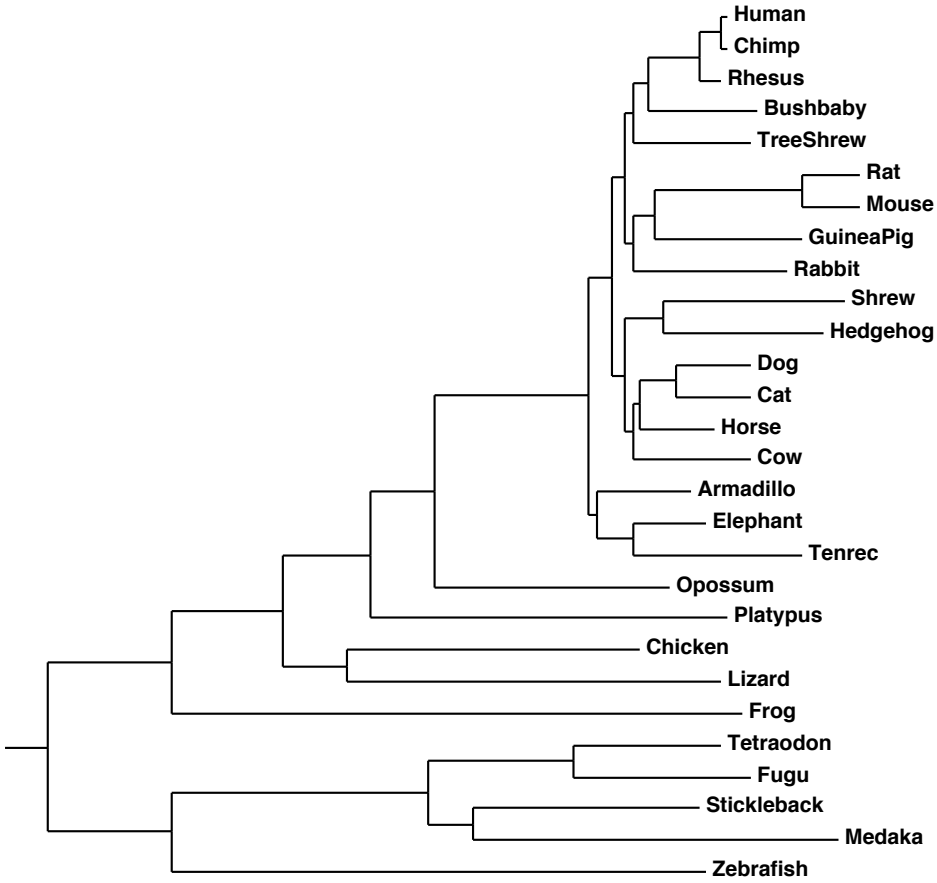


Figure 4.2: Phylogenetic tree that indicates the assumed evolutionary relationships among the sequences of several species in the 28-way alignment. The branch lengths are proportional to the average number of substitutions per site (based on [146]).

4.1.1 Multiple Alignment Format (MAF)

The multiple alignment format (or MAF) is used for storing a series of multiple alignments in a format that is easy to parse and relatively easy to read. This format is used to store multiple alignments at the DNA level between entire genomes. A MAF file is composed by several MSABs (Multiple Sequence Alignment Blocks), as can be seen in Figure 4.3. Each one of those MSABs contain several types of lines. Figure 4.4 depicts an example of a MSAB. As it can be seen, the MSAB always starts with an 'a' line that contains the score information. Usually, this line type defines the beginning of a new MSAB. The 's' lines are the most important lines, containing information about the sequence alignment (DNA bases and gaps). The first 's' line of each MSAB is the reference sequence from which the alignment was made. More details about this format and the line types that it uses can be found at Appendix C.

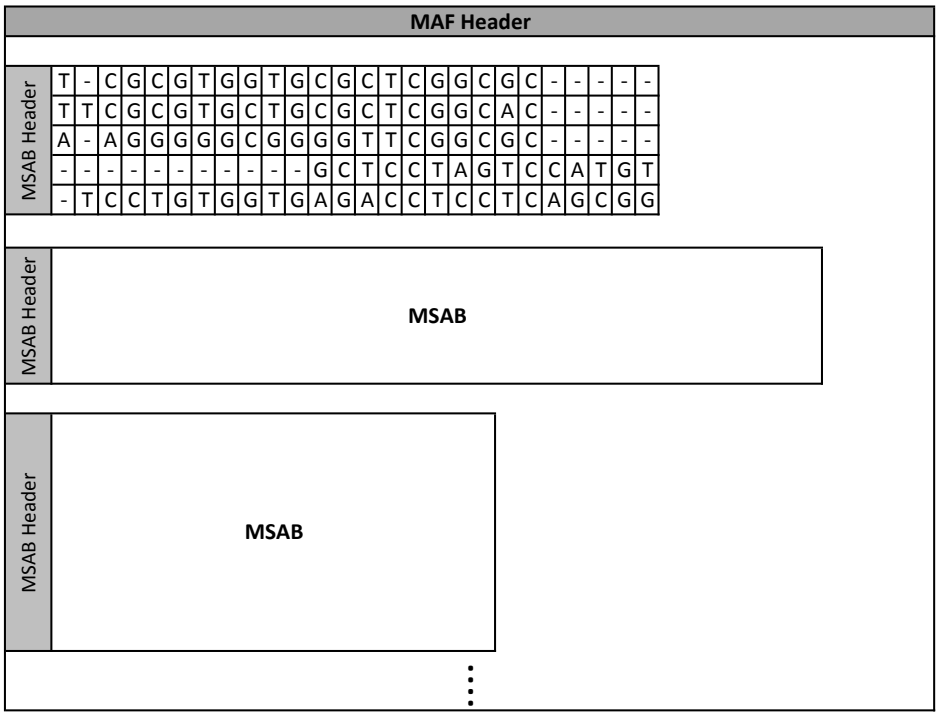


Figure 4.3: A basic description of a MAF file. Each Multiple Sequence Alignment Block (MSAB) contains the DNA bases for a set of species properly aligned, using an additional gap '-' symbol. The MSABs can also contain additional information that although not mentioned in this figure, it is mentioned latter in this section.

```
a score=-5291.000000
s hg19.chrM                     237 40 + 16571 -----ATAACAATTGAATGTCTGCACAGCCGC------TTTCCACACAGAC
s turTru1.scaffold_98585      25144 31 - 72340 -----ATTGTAATTATAAACTTGCACA---------------CATATTATC
q turTru1.scaffold_98585                       -----99999999999999999999---------------999999999
i turTru1.scaffold_98585      N 0 C 0
s galGal3.chrM                  895 35 + 16775 ------------TTATCAATTTTCACTTCCTC---TATTTTCTTCACAAA-
q galGal3.chrM                                 ------------FFFFFFFFFFFFFFFFFFFF---FFFFFFFFFFFFFFF-
i galGal3.chrM                C 0 C 0
s xenTro2.scaffold_19023       2456 51 +  3053 ATTTAACTACCATAATGAATTCTCAGCTTTTTACCTATTTTCCACCCGGGG
i xenTro2.scaffold_19023      C 0 C 0
e gorGor1.Supercontig_0439211   236 99 +   616 I
```

Figure 4.4: An example of a MSAB with 's', 'q', 'i', and 'e' lines.

4.1.2 Genomic data sets

In order to assess the performance of the compression methods presented in this work, we used four data sets retrieved from the USCSC Genome Bioinformatics Browser (see Table 4.1). The four data sets used are aligned taking as a reference the human genome, although the compression methods proposed in this work are compatible with other data sets (for example, those that have a non-human alignment reference). The data set *multiz28wayB* was created from the "multiz28wayAnno.tar.gz" file, which contains alignments similar (but not equal) to the ones in *multiz28way*, with the optional 'q', 'i' and 'e' lines that are not present in the *multiz28way* data set.

Table 4.1: Data sets information used in this work. The four data sets are aligned using the human specie as a reference. The data set were retrieved from the USCSC database. The *multiz28wayB* was created from the "multiz28wayAnno.tar.gz" file which contains almost (the alignments are slight different in some MSABs) the same alignments of the *multiz28way* for the same species for all chromosomes, with additional annotations to indicate gap context, genomic breaks, and quality scores for the sequence in the underlying genome assemblies (optional 'q', 'i' and 'e' lines).

Data set name	URL	Download date
multiz28way	`http://hgdownload-test.cse.ucsc.edu/goldenPath/hg18/multiz28way`	March, 2014
multiz28wayB	`http://hgdownload-test.cse.ucsc.edu/goldenPath/hg18/multiz28way`	March, 2014
multiz46way	`http://hgdownload-test.cse.ucsc.edu/goldenPath/hg19/multiz46way`	March, 2014
multiz100way	`http://hgdownload-test.cse.ucsc.edu/goldenPath/hg19/multiz100way`	March, 2014

The *multiz28way* data set was first used by Hanus *et al.* [160–162]. It contains 27 vertebrate genomes aligned with the human genome (a total of 28 species). The *multiz46way* and *multiz100way* contain 45 and 99 vertebrate genomes, respectively, also aligned with the human genome (a total of 46 and 100 species respectively). These data sets are quite different in terms of number of species and consequently in terms of size. Moreover, they are also different in terms of the line types that each one contains. The lines types that can be found in MAF files are described in Appendix C. In Table 4.2, we can see the different line types that each data set has. The 's' lines are the most important lines so they appear in all the four data sets. The *multiz28wayB* and *multiz46way* data sets are the only ones that have 'q' lines. The 'i' and 'e' lines can only be found in the *multiz28wayB*, *multiz46way*, and *multiz100way* data sets. These differences will affect the attained compression results, as we will explain latter. More information regarding the data sets mention in this section can be found in Appendix D.

Table 4.2: Approximate raw size of each data set, number of species, number of MSABs, and line types that each data set has. The check mark (✓) symbolizes the presence of a line type in the data set while the x mark (✗) symbolizes absence.

Data set	Uncompressed size (gigabytes)	Number of species	Number of MSABs	Line types			
				's'	'q'	'i'	'e'
multiz28way	45	28	23,120,374	✓	✗	✗	✗
multiz28wayB	106	28	23,387,797	✓	✓	✓	✓
multiz46way	252	46	33,429,985	✓	✓	✓	✓
multiz100way	716	100	109,850,940	✓	✗	✓	✓

4.2 Specialized compression methods for MAF files

Due to the size of genomic data that is generated and processed, efficient compression algorithms are essential. Usually, in order to overcome this problem popular general-purpose compression tools (such as gzip) are used. However, these tools were not specifically designed

to compress this kind of data, and often fall short when the intention is to reduce the data size as much as possible.

4.2.1 Hanus' method

At the time of writing this document, the only algorithm specially designed for compressing whole genome alignments was introduced by Pavol Hanus *et al.* [160–162]. Their method is based on well-established statistical evolutionary models and on prediction techniques, used for lossless binary image compression. In their proposed evolutionary-based compression scheme, the nucleotides (include DNA bases, the gap symbol '-' and other letters 'N'/'n') in a MSAB are compressed using the predictions obtained from a nucleotide substitution model, whereas the gaps are encoded independently using techniques from lossless binary image compression. According to them, encoding the nucleotides and the gaps separately is justified by the independence of the two underlying mutational processes and should not introduce an inherent loss to the achievable compression rate.

4.2.1.1 Nucleotides compression

Regarding the compression of the nucleotides, Hanus *et al.* proposed two encoding approaches, both relying on statistical evolutionary models that describe the evolutionary relationships between homologous nucleotides. The nucleotides in each MSAB column are homologous in the sense of sharing a common ancestor. In Figure 4.2 we have the phylogenetic tree for 28 species that can be found in the multiz28way data set. This phylogenetic tree represented by $\mathcal{T}$, contains 28 branches with length τ, describing the evolutionary relationship between the several species. For each MSAB, a subtree corresponding to the species that occur in the MSAB is created. Then, for each alignment column, j, the set of species leaf nodes, l, corresponding to the homologous nucleotides actually observed in this column is determined first. The gaps are removed, leading to the vector of homologous nucleotides, x_l^j, observed in the column. The evolutionary relationship of the species leaf nodes l is described by a subtree of the full phylogenetic tree $\mathcal{T}$.

In order to compute the probability, $p(x_l^j)$, of observing a set of homologous nucleotides in different species, given the evolutionary model relating the species, the Felsenstein algorithm [163] is used. This algorithm allows to calculate the likelihood on a tree, using an efficient iterative procedure. The obtained probabilities for each column can then be used for driving an arithmetic encoder.

This columnwise approach represents an optimal encoding strategy given the evolutionary model. However, it is only feasible for a small number of nucleotides per column. Therefore, the authors decided to use an alternative encoding scheme, based on a representative common ancestor. A representative common ancestor nucleotide, $\hat{x}_\tau^j$, is encoded for each column together with a set of conditional probabilities, $p(x_{l_i}^j|\hat{x}_\tau^j), \forall i = 1 \ldots N$, of all leaf nucleotides observed in that column. This representative common ancestor, $\hat{x}_\tau^j$, is a function of the column realization, x_l^j, and is chosen with the aim of minimizing the number of bits required to encode column j,

$$\hat{x}_\tau^j = \underset{x_\tau}{\operatorname{argmax}} \left(\prod_{i=0}^{N} p(x_{l_i}^j|x_\tau) \right). \tag{4.1}$$

The estimated common ancestor nucleotide, $\hat{x}_\tau^j$, corresponds to the maximum likelihood estimate under the assumption that the nucleotides have evolved independently.

4.2.1.2 Gaps compression

The gaps in the MSABs result from the alignment process. In [161], the gaps are compressed by considering each MSAB as a binary image, where the presence of a gap is signaled with one of the two possible pixel values (e.g., 1) and the four DNA bases with the other pixel value (e.g., 0). This binary image, also known as a puncturing matrix, is compressed using a template driven prediction compression algorithm and arithmetic entropy coding. In order to choose a suitable context template, the authors tried several context sizes and configurations and concluded that the best context should be of size 4 (depicted in Figure 4.5 as the "Main Context"). As we can see, there are different contexts to encode the first two rows and also the first column (see Figure 4.5). In order to be able to use the edge contexts, it is necessary to encode the first column and the first two rows, separately. Moreover, the puncturing matrix compressor (PMc) was also compared to other state-of-the-art approaches for the lossless compression of binary images, including JBIG and JBIG2, outperforming them.

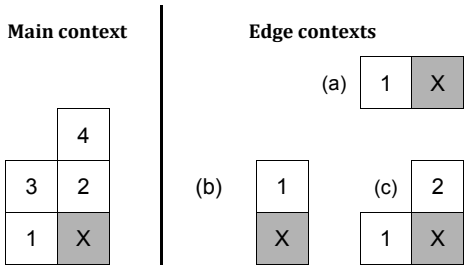


Figure 4.5: Left: best main context of depth 4. Right: edge contexts (a) first column; (b) first row; (c) second row. The position to be encoded is denoted by the "X" (adapted from [161]).

4.2.2 MAF-BGZIP

In 2012, Wheeler and Tarasov developed a plugin for BioRuby that offer support to bioinformatics to deal with MAF files [164–166]. This plugins provides a set of tools for indexed and sequential access to MAF data, as well as performing various manipulations on it and writing modified MAF files. In particular, this library provides support for BGZF (Blocked GZip Format) compressed MAF files, which combine Gzip compression with blocking for efficient random access. The maf-bgzip tool creates compressed MAF files that consist of concatenated 64 kilobytes blocks, each one as an independent Gzip stream. These files can be decompressed in its entirety with Gzip however, this library enables random access using "virtual offsets" as defined in SAM/BAM for fast access to a certain portion of the MAF file.

This compression tool is not optimized in terms of compression, instead, it improves the random access of Gzip files sacrificing compression performance for accessing performance.

4.3 Proposed method for the MSABs based on a mixture of finite-context models

In this section we present a new compression method for the MSABs of a given MAF file. This method only handles the DNA bases and the alignment symbol of the 's' lines. All the optional lines and header information (such as the source name, source size, etc.) are ignored. A full compression method is introduced in Section 4.4. As mention before in Section 4.2, the Hanus *et al*. method [160–162] method separates the MSAB into two different sources (DNA bases and gaps). The method that we proposed uses an alternative approach where both the DNA bases and alignment gaps are addressed at once. This new modeling strategy allows exploring additional data correlations not considered by the Hanus *et al*. approach, such as inter-column dependencies and base/gap relations, resulting in further coding gains.

4.3.1 Method description

The proposed approach is based on a mixture of finite-context models. Finite-context models have been used for single DNA sequence compression [70, 167, 168]. However, in this case we are dealing with multiple DNA sequences divided into several MSABs, where the size of these blocks ranges from 1 row to several rows. In our approach, we consider each MSAB as a special image, where each pixel/position can have only 5 different values from the alphabet $\mathcal{A} = \{\mathrm{A}, \mathrm{C}, \mathrm{G}, \mathrm{T}, \text{-}\}$. In order to compute the probability estimate for a certain symbol, we adopted a strategy very similar to the one described in Section 3.5.2.

4.3.2 Proposed models

4.3.2.1 Typical image templates

As mentioned before, we are treating each MSAB as a special image with 5 intensities (5 different symbols). We process the symbols of each block in a raster scan order, as in typical sequential image coding. Figure 4.6 shows the context templates used. Templates T4 and T10 are typically found in the context of image compression. Template T4 is smaller than T10, which allows to capture more reliable statistical information in smaller MSABs.On the contrary, template T10 is better at capturing statistical information in larger MSABs. The last template, T9, is more specific for this kind of data, trying to explore as much as possible the correlation along the columns (see Figure 4.4).

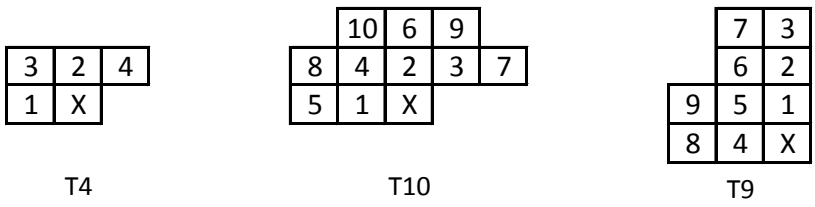


Figure 4.6: Context templates used. The "X" indicates the current symbol.

4.3.2.2 Ancestral Context Model (ACM)

The ancestral context model (ACM) is a special model that explores the correlation between the symbols of each column in the MSABs. This ancestral model uses the most frequent symbol per column as a context symbol that can be used for defining the conditioning states of this model. In order to understand more easily the ACM, we present in Figure 4.7 a small example. This figure shows a small portion of the MSAB presented at the top of Figure 4.4. The ancestral context model is basically an ancestral line with n symbols, where n is the number of columns of the current MSAB. For each block that is processed, the algorithm initializes a line with an arbitrary symbol (e.g. 'A'). After processing each symbol, the algorithm computes the most frequent symbol for the current column, from the first row until the current one. The most frequent symbol is inserted in the ancestral line to be used later. The information of the ancestral line after processing several rows will be very similar to the estimated ancestor line depicted in Figure 4.4.

Original MSAB

		1	2	3	4	5	6	7	8	…
human	1	C	C	G	C	T	G	C	T	•••
platypus	2	C	T	T	T	T	C	T	T	•••
chicken	3	C	A	A	T	C	G	T	T	•••
lizard	4	C	A	T	T	A	G	T	T	•••
zebrafish	5	C	C	G	T	T	T	T	T	•••
estimated ancestor		C	C	G	T	T	G	T	T	•••

Ancestral context model

		1	2	3	4	5	6	7	8	…
Initial ancestral		A	A	A	A	A	A	A	A	•••
After line	1	C	C	G	C	T	G	C	T	•••
After line	2	C	T	T	T	T	C	T	T	•••
After line	3	C	A	A	T	T	G	T	T	•••
After line	4	C	A	T	T	T	G	T	T	•••
After line	5	C	C	G	T	T	G	T	T	•••

Figure 4.7: Top: a small piece of the block presented in Figure 4.4; Bottom: the ancestral context model. After encoding each line, the most frequent symbol per column is computed and inserted in the ancestral line.

Algorithm 1 describes how the ancestral line is updated during the compression of each MSAB. The presented algorithm only computes the most frequent symbol in a specific column *col* when processing row *row*. In the end, it will return the most frequent symbol. As can be seen, the loop in line 4 computes the frequencies of each symbol in the current column, from the first row until the current row (including the current row). Since this update process is performed after encoding the current symbol, then the encoded symbol is also available at the decoder at this point. The loop in line 9 is responsible for obtaining the most frequent

symbol or in the case of a tie the symbol closest to the current row.

Algorithm 1 ComputeAncestral($msaBlock, row, col$)

Require: A $msaBlock \neq$ NULL.
Require: An integer $row > 0$.
Require: An integer $col > 0$.
Ensure: The most frequent symbol of the current column.

```
1: for all s ∈ {A, C, T, G, −} do
2:     freq[s] ← 0
3: end for
   {Loop the current column symbols}
4: for i = 1 to row do
5:     s ← GetSymbol(msaBlock, i, col)
6:     freq[s] ← freq[s] + 1
7: end for
   {r stores the most frequent symbol in the current column}
8: r ← A
9: for all s ∈ {C, T, G, −} do
10:    if freq[s] ≥ freq[r] then
11:        r ← s
12:    end if
13: end for
14: return r
```

During the compression of each MSAB, we have an ancestral context line with statistical information regarding the most frequent symbol per column. However, we only need a small portion of the ancestral line. In order to obtain the size of the ancestral context line that maximizes the compression ratio, we ran some simulations for different sizes using only the ACM. In Figure 4.8, we show an example of a MSAB and layout of the ancestral line.

After performing the simulations using the *multiz28way* data set, we obtained the results that are listed in Table 4.3. According to these results, the sizes of left- and right-hand side parts of the ACM that minimize the average number of bits per symbol is, respectively, 2 and 5. Therefore, we chose an ACM with size 8 (2 left + 5 right + 1 center) to perform the rest of our simulations.

4.3.2.3 Static Column Model (SCM)

Due to the fact that the arithmetic coding uses probability estimates of each symbol, we can define an order-0 model that explores the strong correlation that is present in each column. The SCM uses the probabilities of each symbol per column to encode the current symbol. In Figure 4.9, we show a small example that explains how this model works. In this example, we are processing row number 5, meaning that the probabilities of each symbol must be calculated using only the information of rows 1-4. Suppose that we were encoding the symbol positioned at the fifth row and third column. According to Figure 4.9, a relative frequency of 1/4 will be used by the probability estimator for compressing symbol 'G'.

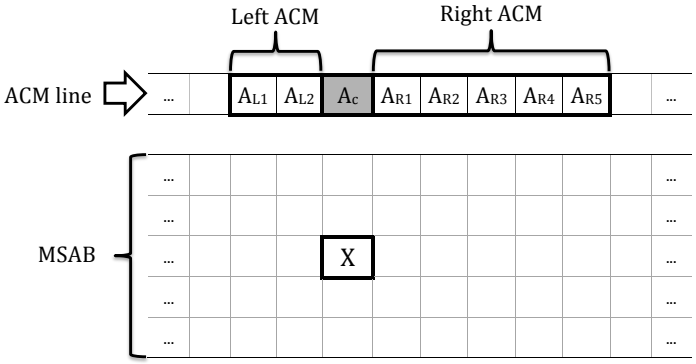


Figure 4.8: Portion of the ancestral line used. The "X" denotes the current symbol to be encoded. We used 2 symbols on the left-hand side part of ACM and 5 symbols on the right-hand side part. The combination of the left- and right-hand side parts and of the central symbol, denoted as A_C in the figure, results in an order-8 ACM.

Table 4.3: Simulation results, in bits per symbol, using only the ACM, for different context sizes for the *multiz28way* data set. The first column (denoted as "Left") indicates the number of symbols used on the left-hand side of the symbol to encode. The first row indicates the number of symbols used in the ACM on the right-hand side of the symbol to encode. The results show that for the *multiz28way* data set, the best sizes are Left-2, Right-5.

Left \ Right	0	1	2	3	4	5	6	7	8
0	1.374	1.347	1.328	1.317	1.313	1.312	1.312	1.313	1.319
1	1.327	1.296	1.277	1.266	1.261	1.260	1.260	1.266	1.281
2	1.324	1.294	1.274	1.263	**1.258**	**1.258**	1.263	1.277	1.309
3	1.319	1.291	1.272	1.264	1.261	1.267	1.281	1.310	1.364
4	1.319	1.290	1.271	1.262	1.263	1.276	1.307	1.361	1.441
5	1.318	1.290	1.273	1.271	1.282	1.310	1.364	1.442	1.521
6	1.316	1.288	1.274	1.278	1.305	1.358	1.438	1.519	-
7	1.316	1.292	1.292	1.313	1.369	1.448	1.521	-	-
8	1.320	1.304	1.317	1.360	1.436	1.515	-	-	-

4.3.2.4 Column Model 5 (CM5)

The SCM is a model that explores correlations along a MSAB column. However, it could be also interesting to reuse the statistical information of the previous columns to encode symbols of other MSA columns. For this purpose, we propose a Column Model of order 5 (CM5), which corresponds to the number of symbols of the alphabet. In this case, instead of using the neighboring symbols of the current symbol to define the conditioning context (see Section 4.3.2.1), the CM5 uses the frequency of each symbol of the current column to build a small context.

In order to understand more easily this model, we present a small example in Figure 4.10 that shows how the context for each column is created. Considering that we are processing

		1	2	3	4	5	6	7	8	…
Already processed	1	**C**	C	G	C	**T**	**G**	C	**T**	…
	2	**C**	T	**T**	**T**	**T**	C	**T**	**T**	…
	3	**C**	**A**	A	**T**	C	**G**	**T**	**T**	…
	4	**C**	**A**	**T**	**T**	A	**G**	**T**	**T**	…
Current	5	C	C	G	T	T	T	T	T	…

P('A') →	0/4	2/4	1/4	0/4	1/4	0/4	0/4	0/4
P('C') →	**4/4**	**1/4**	0/4	1/4	1/4	1/4	1/4	0/4
P('G') →	0/4	0/4	**1/4**	0/4	0/4	3/4	0/4	0/4
P('T') →	0/4	1/4	2/4	**3/4**	**2/4**	**0/4**	**3/4**	**4/4**
P('-') →	0/4	0/4	0/4	0/4	0/4	0/4	0/4	0/4

Figure 4.9: A Static Column Model example. After processing the first 4 rows, the algorithm computes the probability of each symbol per column (bottom). The computed probabilities are used to encoded the symbols of row 5.

the symbol at position (5, 2), the context that is used to encode symbol 'C' is denoted as "Context 2" in Figure 4.10. After analyzing column 2 of Figure 4.10, the symbols, sorted by frequency in a non-ascending order, are ACTG-. The sorting process that is used to create the context is described in the loop on line 11 of Algorithm 2. Using this model, it is possible to combine contexts that are very similar due to the statistical similarity of their columns. For example, in Figure 4.10, contexts 5 and 8 are similar, because after computing the statistics of each symbol in each column using Algorithm 2, the resulting context is the same.

4.3.3 Experimental results

In this section we will present the results obtained using the proposed method and we also compare its performance with several popular general compression methods such as gzip [169], bzip2 [170], ppmd (using the version implemented by the 7-Zip [171] archiver). The proposed compression tool is available at `http://bioinformatics.ua.pt/software/saco` or `https://github.com/lumiratos/saco` for testing, that includes the source code and two Windows binaries (32 and 64 bits). For the general compression methods we needed to extract the DNA bases and alignment gaps for each MAF file for a separate file. Moreover, all the non-ACGT symbols encountered were transformed into gaps ('-') and all lower case symbols were turned into upper case. Table 4.4 depicts the transformation that is done to each symbol in more detail. After this extraction process, we could compress the files using the popular general compression methods mention earlier. In terms of size, we present in Table 4.5 the raw size in gigabytes for each data set before and after applying the symbols transformation and extracted all optional lines.

The proposed method relies on several models that will be combined in order to obtain a probability estimate to be used in the arithmetic coder. In order to understand which models to use, we performed several simulations under three representative files for the *multiz28way*

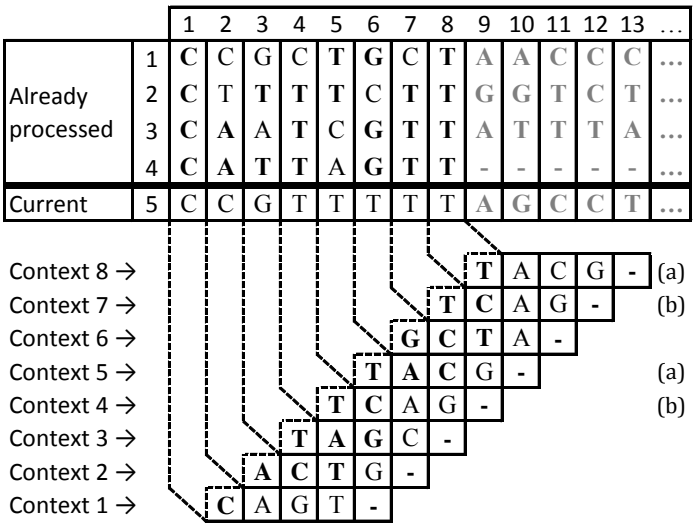


Figure 4.10: A CM5 example. Contexts 1-8 are used to encoded the first 8 symbols of row 5. Each context is built sorting the symbols of each column by its frequency in a non-ascending order.

Table 4.4: Symbol mapping used for the proposed method.

Input symbol	Output symbol
a/A	A
c/C	C
g/G	G
t/T	T
n/N	-
-	-

Table 4.5: The approximately raw size of each data set in gigabytes, for each data set, before and after applying the symbol transformation depicted in Table 4.4. The last columns corresponds to the size of the DNA symbols and alignment gaps, after the transformation, without consider any optional lines and header information.

Data set	Original size (GB)	Size after transformation (GB)
multiz28way	45	31
multiz28wayB	106	31
multiz46way	252	65
multiz100way	716	149

data set. In this case we decided to use the largest file ("chr2") an intermediate file ("chr9") and the smallest file ("chrY"), without considering the "chrM" file. In Figure 4.11 we present the results obtained using several model combinations. We only show the best model combi-

Algorithm 2 GetColumnContext($msaBlock, row, col$)

Require: A $msaBlock \neq$ NULL.
Require: An integer $row > 0$.
Require: An integer $col > 0$.
Ensure: The conditioned context of the current symbol.

```
1: nSymbols ← 5
2: for i = 1 to nSymbols do
3:     freq[s] ← 0
4: end for
5: symbols ← {A, C, T, G, −}
6: ids ← {1, 2, 3, 4, 5}
7: ctx ← {}
   {Loop the current column symbols and update frequencies of each symbol.}
8: for i = 1 to row − 1 do
9:     s ← GetSymbol(msaBlock, i, col)
10:    freq[s] ← freq[s] + 1
11: end for
   {Sort the symbols in a non-ascending order using its frequency.}
12: for i = 1 to nSymbols do
13:    max ← freq[ids[i]]
14:    for j = i + 1 to nSymbols do
15:        if freq[ids[j]] > max then
16:            max ← freq[ids[j]]
17:            tmp ← ids[i]
18:            ids[i] ← ids[j]
19:            ids[j] ← tmp
20:        end if
21:    end for
       {Build the context combining the symbols, first the most frequent and then the less
       frequent ones, using a concat operation (operator ||).}
22:    ctx ← ctx || symbols[ids[i]]
23: end for
24: return ctx
```

nations, due to the large number of all possible arrangements. T4, T9, and T10 denotes the models based on the templates illustrated in Figure 4.6. ACM, SCM, and CM5 correspond to the models described in Sections 4.3.2.2, 4.3.2.3, and 4.3.2.4, respectively. According to the results obtained, the best combination model seem to be the "T4T9T10+ACM". The variation in terms of performance for the first seven combinations is small but even so it can effect the compression results for larger data sets. We also performed the same experiment in the other three data sets in order to understand if we attain different results. In fact for the *multiz46way* and *multiz100way* the combination "T4T9T10+ACM" is the third best, but very close the the best combination in terms of performance. In this first approach we wanted to optimize the compression tool to the *multiz28way* data set and observe if the selected combination is reliable in terms of performance for other data sets. Due to the fact

that the selected combination ("T4T9T10+ACM") have a performance very close to the best combination for the *multiz46way* and *multiz100way* data sets we decided to use it for now on in our experiments.

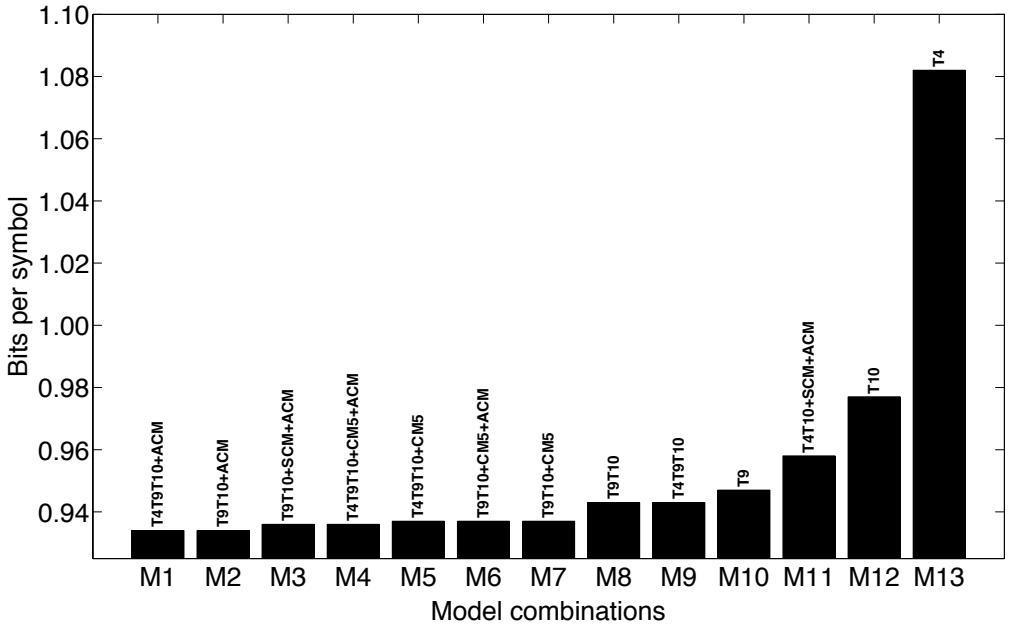


Figure 4.11: Average bits per symbol attained for several combination models, using three files of the *multiz28way* data set ("chr2", "chr9", and "chrY").

In Section 3.5.2, we have explained how the mixture of models is performed. Because this mixture depends on a parameter γ, we investigated how the performance is related to the value of γ. Figure 4.12 displays this relation, where it can be observed that the γ providing the best results is between 0.95 and 0.96. Since the variation of the compression results between $\gamma = 0.955$ and $\gamma = 0950$ is minimal, we decided to use $\gamma = 0.95$ as default.

Table 4.6 contains the overall compression ratio in bits per symbol of the four data sets used in this work for several compression methods. We included results for several general compression methods such as gzip [169], bzip2 [170], ppmd, lzma, and for Hanus *et al*. method [161]. Tables E.1-E.4 from Appendix E contain the results in a more detail for each MAF file. The proposed method attained on average from 0.52 up to 0.94 bits per symbol, including the additional information required by the decoder for recovering the MSABs, such as the size of each block. When compared to the Hanus *et al*. method [161], the proposed approach attained approximately 7% better results. If we analyze the compression performance through the four data sets, we can notice that the results for the first two data sets (*multiz28way* and *multiz28wayB*) are very similar because they are very similar in terms of 's' lines. Regarding the other two data sets, it is visible an improvement in the compression ratio when compared to the *multiz28way* and *multiz28wayB* data sets. The compression ratio tends to improve when the data set size increases, regardless of the compression method used. Despite the model combination used was tuned for the *multiz28way* data set, if we look to the compression gains we verify that the combination model remains effective for other data sets, i.e., it does not reveal over-fitting.

In Figure 4.13 we can find the frequency of each symbol in percentage for the four data

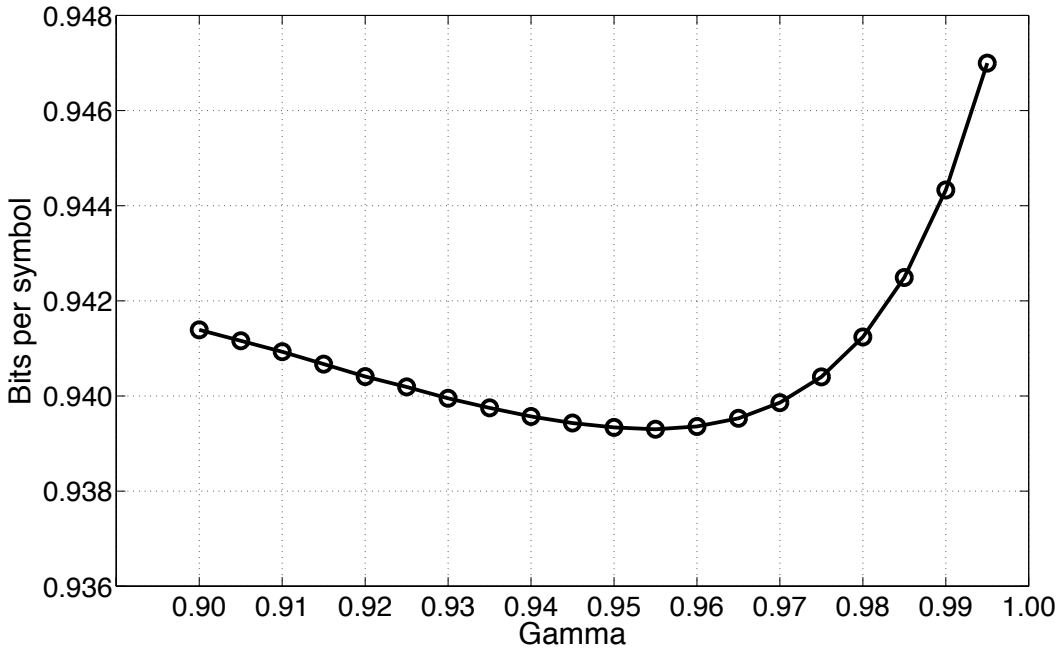


Figure 4.12: Relation between the average number of bits per symbol for the *multiz28way* data set and the parameter γ. This evaluation was performed using the combination "M1" depicted in Figure 4.11 (T4+T9+T10+ACM).

Table 4.6: Performance of several compression methods in the four data sets used in this work. The results are presented in bits per symbol (bps) which means that the results were computed taking into account the number of bytes of the compressed file divided by total amount of DNA based and alignment gaps. Results for the Hanus *et al.* method [161] were not included for the last three data sets, because the method is not able to process the files with optional lines.

Data set	**Gzip**	**Bzip2**	**PPMd**	**LZMA**	**Hanus [161]**	**Proposed [13]**
multiz28way	1.70	1.88	1.81	1.20	1.01	**0.94**
multiz28wayB	1.70	1.87	1.81	1.19	–	**0.94**
multiz46way	1.40	1.60	1.71	0.94	–	**0.73**
multiz100way	1.06	1.26	1.55	0.68	–	**0.52**
Total	1.29	1.48	1.65	0.86	–	**0.67**

sets used in this work. It is visible that C and G are the symbol less frequent in the four data set. On the contrary, the gap symbol is the most frequent regardless of the data set. The increase in the gap symbol frequency that is observed is probably caused by the fact that with more species involved, it is more difficult to align sequence thus, resulting in more alignment gaps.

Figure 4.14 presents the contribution of each symbol for the final number of bits required to store each data set (without considering the overhead associated with the additional information, such as the size of each MSAB). After analyzing the chart, it is clear that the

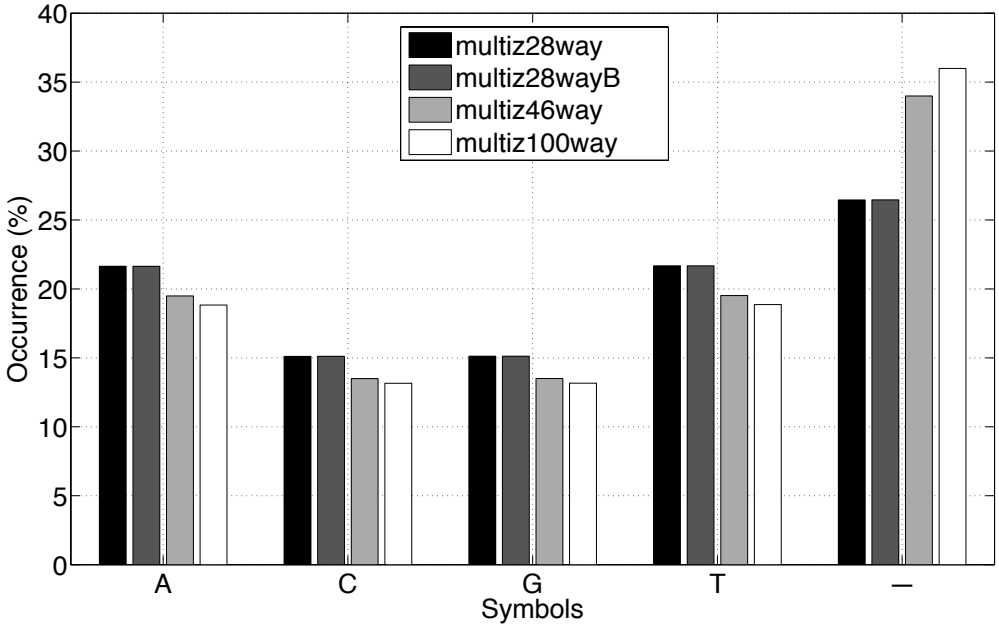


Figure 4.13: Occurrence percentage of each symbol for the four data sets used in this work.

symbols 'C' and 'G' are the ones with the worse compression results. The reason behind this is probably because they are the least frequent symbols. On the other hand, the gap symbol is the one with the best compression ratio. The great performance attained in the gap symbol is due to its high frequency in each data set. Moreover, it is also justified by the larger cluster of alignment gaps that are very typical for this type of files. Combined, these two aspects have a decisive influence in the final compression ratio, as can be observed specially for the *multiz46way* and *multiz100way* data set that have $\approx 7.5\%$ and $\approx 9.5\%$ more alignment gaps than the *multiz28way* data set.

In order to understand more clearly the performance of each model, we have collected statistical information of each one during the compression of the *multiz28way* data set. The goal was to count how many times a certain model was the best one in the mixture. For each symbol, the best model is the one that alone would generate better compression results than the others. These results are shown in Figure 4.15. As can be seen, on average the model associated with the context template T9 was $\approx 38\%$ of the time the best one, whereas the model with context template T10 was the best one $\approx 31\%$ of the time. The template associated with the context template T4 was the best $\approx 17\%$ of the time. Finally the model that had less influence in the mixture was the ACM with only $\approx 14\%$ of use.

4.3.4 Complexity

According to the models presented in Section 4.3.2 and the experimental results presented in the previous Section, the proposed algorithm relies on finite context models with an alphabet $\mathcal{A} = \{\mathrm{A}, \mathrm{C}, \mathrm{G}, \mathrm{T}, \text{-}\}$, with five symbols. The model order is defined by the alphabet and by the number of symbols used to build the context. Since the number of symbols to build the context is associated to the template sizes illustrated in Figure 4.6 and the ACM model used Left-2 and Right-5 context symbols, as described in Table 4.3, the number of counter

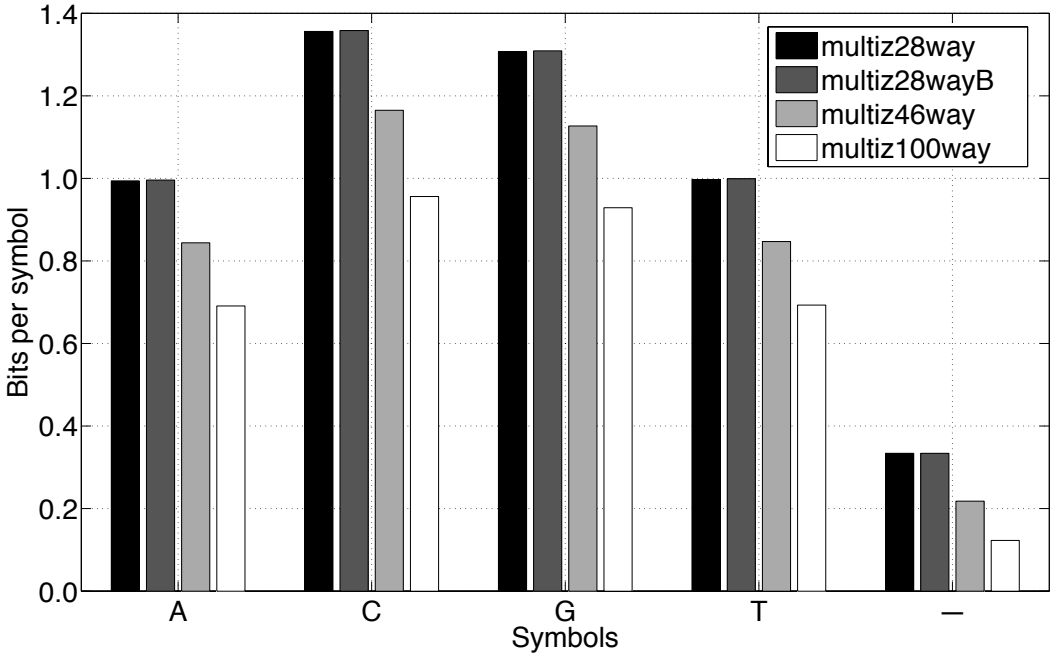


Figure 4.14: Average bits per symbol used to store each symbols of the four data sets used in this work.

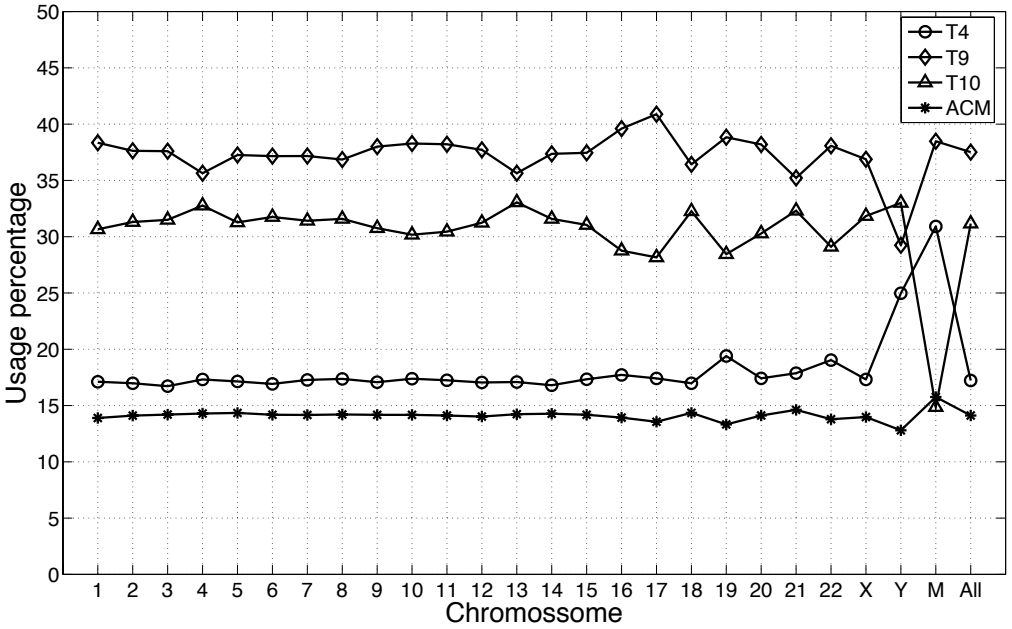


Figure 4.15: The performance of the 4 models: Using template T4, T9, T10 and ACM. The performance is based on the number of times that, during compression, each model was the best one. The best model in the mixture is the one that, if considered alone and for each symbol, could generate the best compression results among the four.

for each model is:

- T4 - $5 \times 5^4 = 3,125$ counters.
- T9 - $5 \times 5^9 = 9,765,625$ counters.
- T10 - $5 \times 5^{10} = 48,828,125$ counters.
- In ACM we have two left symbols, five right symbols and the symbol in the same column as the current symbol being encoded denoted A_c in Figure 4.8. A total of $2 + 1 + 5 = 8$ context symbols.
  - ACM - $5 \times 5^8 = 1,953,125$ counters.

The total amount of counters is then 60,550,000. Taking into account that in the implementation each counter is stored in two bytes, the total amount of computer memory required to store the counters is about 115 megabytes. Tables 4.7 and 4.8 we present in a compact way the encoding and decoding time in seconds for each data set for the methods used in our experiments in Section 4.3.3. The encoding and decoding time were obtained using the *time* [1] command, available in Linux. The presented compression tool took about 22 hours to compress the *multiz28way* data set. The decoding time is about 18 hours. The encoding/decoding results presented in Tables 4.7 and 4.8 for the Hanus *et al.* method [161] was computed using a tool provided by the authors that does compress all the information (including header information). We only presented the results to be used as a reference. Furthermore, it is important to indicate that the provided tool does not decode some files of the *multiz28way* data set ("chr15", "chr16", "chr17", and "chr19") so the decoding time indicated in Table 4.8 does not take into account those files.

Regarding the other compression methods they are much faster when compared to the proposed method and to Hanus *et al.* method [161]. However, the compression performance for those other methods is lower. We can conclude that compression gains usually come with some costs in terms of memory and computation time.

Table 4.7: Performance of several compression methods in terms of encoding time in seconds for the four data sets used in this work.

Data sets	**Gzip**	**Bzip2**	**PPMd**	**LZMA**	**Hanus [161]**	**Proposed [13]**
multiz28way	4,763	6,413	3,160	69,527	* 77,899	80,600
multiz28wayB	4,766	6,186	3,342	70,648	–	80,880
multiz46way	8,249	12,530	5,864	131,830	–	167,540
multiz100way	13,241	25,924	12,797	246,613	–	383,907
Total	31,019	51,053	25,163	518,618	–	712,927
* The encoding time was obtained by the tool provided by Pavol Hanus that compresses the entire MAF files (including header information).						

In Figure 4.16 we present the relation between the size of each data set and the corresponding encoding/decoding time. As can be seen there is a linear behavior through the four

[1] `http://linux.die.net/man/1/time`

Table 4.8: Performance of several compression methods in terms of decoding time in seconds for the four data sets used in this work.

Data sets	Gzip	Bzip2	PPMd	LZMA	Hanus [161]	Proposed [13]
multiz28way	403	3,422	3,597	890	* 78,691	63,364
multiz28wayB	408	3,460	3,619	900	–	63,358
multiz46way	783	5,672	6,585	1,587	–	132,474
multiz100way	1,683	9,758	14,665	2,833	–	302,710
Total	3,277	22,312	28,466	6,210	–	561,906

* The decoding time was obtained by the tool provided by Pavol Hanus that compresses the entire MAF files (including header information). Moreover, the obtained decoding time was computed without taking into account files "chr15", "chr16", "chr17", and "chr19".

data sets both encoding and decoding time. Furthermore, we can also observe that the time difference between encoding and decoding tends to increase for larger data sets.

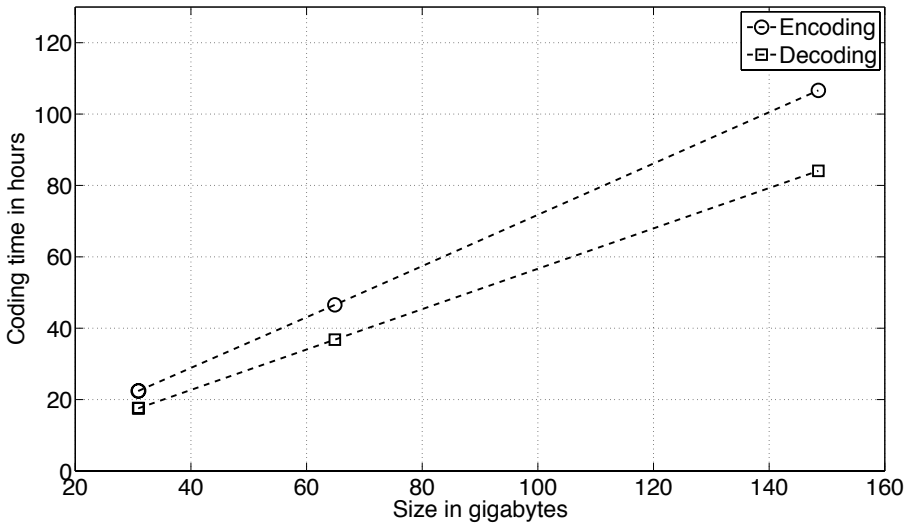


Figure 4.16: Relation between the size of each data set and the encoding/decoding time. Each marker symbolizes one data set. The *multiz28way* and *multiz28wayB* have a similar size and coding time after applying the transformation illustrated in Table 4.4 and discarding all the optional lines and header information. It is visible a linear behavior along the data sets both in term of encoding and decoding time.

4.4 MAFCO: a compression tool for MAF files

In Section 4.3 we described a compression method based on a mixture of finite-context models and arithmetic coding, for compressing the DNA bases and the alignment gaps of the MSABs. This initially approach did not process the optional lines and header information

that can be found in MAF files. Furthermore, the method presented in Section 4.3 did not take into consideration the lowercase DNA bases as well for the n/N characters. Taking also into consideration the coding time obtained for the previous approach, we intended to reduce it by using more simple models that could attained a similar compression performance.

MAFCO is a full lossless compressor, capable of processing all line types that are part of MAF. Instead of using a mixture of finite-context models, we opted by a single model, in order to improve the encoding/decoding time. Similar to the previous approach, this compression tool relies on probabilistic models, know as finite-context models, that are quite effective for DNA data compression [70, 167, 168]. The goal of these probabilistic models is to assign probability estimates for the next symbol, taking into account a recent past context. The size k of that context can vary ($k > 0$) and it will define the model order. Assuming that the k past outcomes are given by $x_{n-k+1..n} = x_{n-k+1} \dots x_n$ (order-k model), the probability estimates $P(x_{n+1}|x_{n-k+1..n})$ are computed using the symbol counts that are accumulated while the information source is processed, with

$$P(s|x_{n-k+1..n}) = \frac{c(s|x_{n-k+1..n}) + \alpha}{c(x_{n-k+1..n}) + |\mathcal{A}|\alpha}, \tag{4.2}$$

where $c(s|x_{n-k+1..n})$ represents the number of times that, in the past, symbol s was found having $x_{n-k+1..n}$ as the conditioning context, $|\mathcal{A}|$ denotes the size of the coding alphabet $\mathcal{A}$, and where

$$c(x_{n-k+1..n}) = \sum_{a \in \mathcal{A}} c(a|x_{n-k+1..n}) \tag{4.3}$$

is the total number of events that has occurred so far in association with context $x_{n-k+1..n}$. Parameter α allows balancing between the maximum likelihood estimator and a uniform distribution (when the total number of events, n, is large, it behaves as a maximum likelihood estimator). For $\alpha = 1$, (4.2) is the well-known Laplace estimator.

MAFCO uses several different types of single finite-context models. They are different in terms of order (size of the context) and in terms of alphabets that they handle. In order to find the optimal size of each finite-context model, we performed some experiments using three representative MAF files of each data set. These optimal sizes are used by default. In the following sections, we address the compression of each one of the four line types, that can be found in a MAF file, in more detail.

4.4.1 Compression of the 's' lines

Similar to the previous approach, described in Section 4.3, MAFCO treats each MSAB as a special image type with five intensities (alphabet $\mathcal{A}_s = \{\mathrm{A}, \mathrm{C}, \mathrm{G}, \mathrm{T}, \text{-}\}$) that correspond to the four DNA bases and the alignment gap ('-'). MAFCO provides a set of context templates that can be used for compressing the $\mathcal{A}_s$ symbols of the 's' lines. To improve speed, the proposed tool uses only a single model to encode the alignments of the 's' lines, so only one of the context templates is used (see Figure 4.17).

The 's' lines of each MSAB can also have other symbols than the ones indicated earlier in alphabet $\mathcal{A}_s$. Those other symbols include lower case symbols $\{\mathrm{a}, \mathrm{c}, \mathrm{g}, \mathrm{t}\}$ and non-ACGT symbols $\{\mathrm{N}, \mathrm{n}\}$. So, in the 's' lines the set of symbols that can be found is $\{\mathrm{A}, \mathrm{a}, \mathrm{C}, \mathrm{c}, \mathrm{G}, \mathrm{g}, \mathrm{T}, \mathrm{t}, \mathrm{N}, \mathrm{n}, \text{-}\}$ which are mapped according to Table 4.9.

After observing the MSAB depicted in Figure 4.4, we can see that each 's' line contains some header information that needs to be handled. The format of the 's' lines is explained

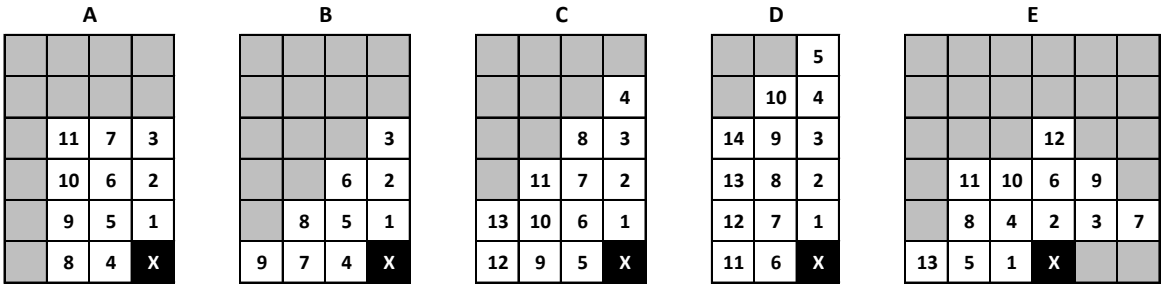


Figure 4.17: Set of 2D context templates available in MAFCO. The 'X' denotes the current symbol that is being encoded/decoded. Template 'C' with depth 10 is the default.

in more detail in Appendix C.3. The first time a given *source name* and *source size* field appears, the encoder needs to encode the entire string of *source name* and the number that corresponds to the *source size*. For the other times that this same *source name* needs to be coded, the encoder only needs to send a number that will identify this *source name*. During the decoding process, this number will be used to obtain the *source name* and *source size* fields that will be stored in an auxiliary table, with the already decoded *source names* and *source sizes*. These two fields are both encoded using a finite-context model (FCM) with an uniform distribution. The *size* field can be obtained by the number of non-gaps that are in the sequence alignment of the MSAB. The *start* field is also encoded using FCM with a uniform distribution. However, this field can be also represented by an offset that can be computed as

$$\text{startOffset} = \text{start}_x - \text{start}_{x-1} - \text{size}_{x-1}, \tag{4.4}$$

where start_x represents the current start of a given source, start_{x-1} indicates the previous start of the same source and size_{x-1} represents the size of the align sequence of the previous 's' line of the same source. Instead of encoding the absolute *start* value, MAFCO encodes the offset using (4.4). This technique allows the encoder to spend less bits encoding this field, due to the fact that usually this offset is zero most of the time. This approach is only used if a 's' line of a given source was already processed (a reference *start* value is necessary to compute the offset). There are also situations where the obtained offset is negative. This situation is caused because the alignment made of a given source can start at a position after the *start* position of the last alignment. In this case, MAFCO also encodes the *start* field as an absolute value. An auxiliary binary stream is needed to differentiate an absolute start value from a offset start value. This auxiliary stream is encoded using a 5-order FCM. The model order of the previous stream and all the other models that are used in MAFCO can be specified by the user, however default values are defined. The *strand* field can only have two different values, '+' or '-'. This field is encoded using a 3-order FCM.

In Table 4.9 we can find the symbols mapping for the DNA bases and alignment gaps along the three streams used to encode the alignment sequence. The {} represents a stream that is not present to encode a given symbol. According to Figure 4.18, MAFCO splits the DNA alignments into two or three different streams. The *main stream* (always present) is a 5-symbol information source $\mathcal{A}_s$, which conveys the information of the DNA bases and alignment gaps. This stream is encoded using one of the five templates depicted in Figure 4.17. By default, MAFCO uses the template 'C' with depth 10 (model order). The second stream depicted in Figure 4.18 as *extra stream*, is present in absence of ACGT symbols (N's and n's).

In case of having an alignment gap ('-'), this *extra stream* is not necessary. This particular stream must be present to disambiguate the occurrence of the "1" symbol in the *main stream*. A "0" in this stream represents a c/C base, whereas a "1" means an extra symbol n/N. This stream is encoded by default using a 5-order FCM. As mentioned before, the MSABs may contain upper and lower case DNA bases. In order to encode this information, a third stream, called *case stream*, is necessary. This binary stream is associated to each symbol of the *main stream* (except the alignment gap symbol '-'), indicating the respective case type. Similar to the previous streams, this stream is also compressed using a order-5 FCM.

Table 4.9: Symbols mapping for each one of the streams illustrated in Figure 4.18. The {} represents a stream that is not present for a particular symbol.

Symbol	Main stream	Extra stream	Case stream
A	0	{}	0
a	0	{}	1
C	1	0	0
c	1	0	1
G	2	{}	0
g	2	{}	1
T	3	{}	0
t	3	{}	1
N	1	1	0
n	1	1	1
-	4	{}	{}

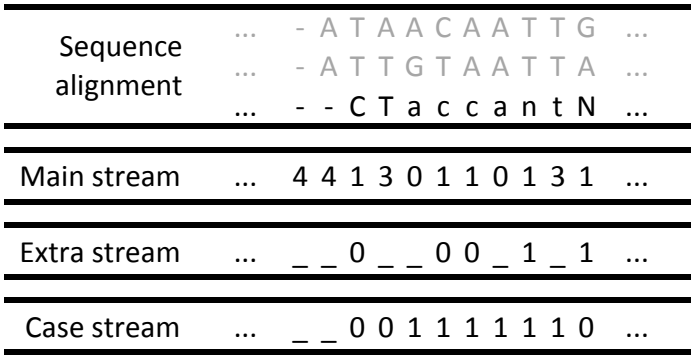


Figure 4.18: Illustration of each one of the streams used for encoding the DNA alignments information. In this example we are processing a MSAB with three lines and currently the third line is the one being processed. Depending of the symbol, the sequence can be split into at most three streams. The first stream corresponds to the DNA bases and gaps. The second stream represents the extra symbols (N's and n's). The last stream is used to process the upper/lower case information. All these streams are encoded using FCM and arithmetic coding.

4.4.2 Compression of the 'q' lines

As mentioned in Appendix C.4, the 'q' lines contain information about the quality of each aligned base of the immediately preceding 's' line. The *source name* is the same as the previous 's' line, so that information was already processed before and it is not necessary to encode it again. Regarding the quality values, in this case we have an alphabet of 13 symbols: $\mathcal{A}_q = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, \mathrm{F}, -, .\}$. In fact, the size of the alphabet is 12 symbols, because the gap positions are the same as in the immediately preceding 's' line, so the alignment gaps are not encoded here. This stream of quality values is encoded using a single order-5 FCM. A different model order can be specified by the user, but for a large alphabet such as $\mathcal{A}_q$, higher model orders require more memory. Furthermore, the presence of these 'q' lines needs to be encoded as well, because there are MAF files that may not have these optional lines. A binary stream is then necessary to indicate, for each MSAB, if it contains 'q' lines. Inside each MSAB, it is also necessary to indicate for each 's' line if there is a 'q' line associated (having the same source name). The first 's' line (reference source) does not have any 'q' line associated. However, for the remaining 's' lines, it is necessary a second binary stream that indicates if it has a 'q' line associated. These two binary streams are encoded also using a 5-order FCM.

4.4.3 Compression of the 'i' lines

Similar to the 'q' lines, the 'i' lines are also associated to a 's' line. This means that the source name of a 'i' line is the same as the immediately preceding 's' line. The only information that needs to be encoded corresponds to the four fields described in Appendix C.5. The counts are compressed using a single FCM with a uniform distribution. The status symbols are encoded using a 4-order FCM.

After analyzing the contents of some of the MAF files, we noticed that there is a correlation between the left and right status and counts of the same source. Basically, the left status symbol and count of a given source is the same as the last right status symbol and count. This means that we only need to encode both left and right status and count for the first time that a given source of a 'i' line appears. After the first occurrence of a 'i' line of a given source, the proposed method only encodes the right status and count. The left status and count can be obtained by the previous right status and count already processed. However, there are some irregularities in this "rule". Sometimes, the left status or/and count are different from the previous right status and count. In order to overcome this irregularities, we created two auxiliary binary streams that represent these irregularities (one stream for the irregular counts and the other for the irregular status symbols). Both streams are independent, because it is possible to have only an irregular count and not an irregular status symbol. By default, these two streams are encoded using a 5-order FCM. Similar to the 'q' lines the 'i' lines also have a binary stream that indicates if the current MSAB has 'i' lines. This stream is also encoded using a 5-order FCM.

4.4.4 Compression of the 'e' lines

The 'e' lines are quite different, when compared to the 'q' and 'i' lines. They are not associated with any of the 's' line of the MSAB that is being encoded. However, they only appear in a MSAB if in any of the previous MSABs a 's' or a 'e' line of the same source occurred. Because the 'e' lines are not associated to any 's' lines of the current MSAB, all

header fields (source name, start position, etc.) need to be encoded. This header information is compressed in the same way as the header information of the 's' lines. The 'e' lines have a status field that the 's' lines do not have. The status field of a given 'e' line usually has the same value of the status field as the 'e' line of the immediately preceding MSAB or the last 'i' line processed of the same source. Two auxiliary binary streams are necessary to encode the status symbol of a given 'e' line. These streams indicate if an irregularity occurred between the current status value of a given 'e' line and the previous 'e' or 'i' line status value of the same source. Both streams are encoded using a 5-order FCM.

4.4.5 Parallel processing and partial decoding

Genomic data files are growing in size every day, a growth that leads to both storage and access issues. In order to overcome these issues, MAFCO uses an approach that allows large files to be split into several parts that can be compressed/decompressed in parallel. This approach reduces the compression/decompression time. However, some compression ratio loss might occur, because statistics gathered in one part of the file may not be available in other parts. Despite this, the compression ratio loss in large files is usually small, and largely compensated by the gain in compression/decompression time. Furthermore, this splitting approach allows the user to decode only some parts of the encoded file, without needing to decode the full compressed file.

The proposed compression tool allows parallel compression/decompression of MAF files that contain several MSABs. After the splitting process, each part contains an integer number of MSABs. By default, the compression tool splits the input MAF file into four parts and, by consequence, uses also four threads. The number of parts in which the MAF file can be split may be specified by the user (-ng flag), as well as the maximum number of threads (-nt flag). We call to each part of the split file a GOBs (Group Of Blocks). By default, in the decoder the entire compressed file is decoded. However, the user can specify a range of GOBs to decode (-ng flag). Note that it is only possible to decode a range of GOBs if the file was initially encoded using a multi-part approach. This approach is quite useful, because it can reduce the decompression time, by skipping the decoding of some unneeded GOBs. Furthermore, the number of threads that is used in the decoder does not have to be the same as in the encoder. This allows, for example, the compression of large files using multi-core computers, while being able to decompress them in more modest machines, if needed. This capability is helpful, because usually the compression of MAF files is done only once by a powerful multi-core computer. The decoding is done many times by the research community, using computers with very different capabilities.

4.4.6 Experimental results

In this section we will present the compression results attained using the proposed compression tool and we also compare its performance with several popular general compression methods such as gzip [169], bzip2 [170], ppmd and lzma (the last two using the version implemented by the 7-Zip [171] archiver). Furthermore, we also include results for the Hanus *et al*. method [161] and for the maf-bgzip tool [164–166] cited in Section 4.2.2. The MAFCO tool is available at `http://bioinformatics.ua.pt/software/mafco` or `https://github.com/lumiratos/mafco` for testing, that includes the source code and two Windows binaries (32 and 64 bits). There are also three small files available at this site, as well

as instructions of how to quickly test the compression tool.

The performance of each compression method can be found summarized in Table 4.10. Alternative, in Tables E.5-E.8 we can find the same results with more detail for each MAF file for the four data sets used in this work. The results are presented in bytes for gzip and in percentage compression gain in relation with gzip, for the other methods. The compression gain in relation with gzip was computed as:

$$G_M = 100 \times \frac{\text{NBytes}_{gzip} - \text{NBytes}_M}{\text{NBytes}_{gzip}} \tag{4.5}$$

where M denotes a compression method and the "NBytes" corresponds to the size of the compressed file in bytes. The presented results were obtained using a single thread and without splitting the MAF file in several GOBs, and will be used later to evaluate the performance loss when the compressor splits the input MAF file into several GOBs. For the *multiz28way*, the proposed method attained about 9% better results, when compared to the Hanus *et al*. method [161]. It was not possible to obtain the compression results for the other data sets, using the Hanus *et al*. method [161], due to compatibility problems. When compared with gzip, MAFCO attained a compression gain of 51.7% for the *multiz28way* data set. For the *multiz28wayB* and *multiz46way* data sets, the compression gain is about 54.3% and 57.3%, respectively. It seems that the compression gain increases with the size of each data set. Regarding the *multiz100way* data set, the MAFCO compression gain is lower when compared with the other data sets (about 34.1%). The reason for this lower performance is due to the small average number of columns of the MSABs (see Figure D.2 of Appendix D.1), suggesting that the model presented in Section 4.4.1 to compress the 's' lines is less effective when the MSABs have a small number of columns.

If we look again at Table 4.10, we can see that the performance of bzip2 increases as the size of the data sets increases. The ppmd and maf-bgzip performance have a different behavior: it decreases as the data sets increase in size, even reaching "negative" performances for the *multiz100way* in case of the ppmd. The maf-bgzip has the worst performance for all data sets, when compared to gzip. The reason to this low performance is due to the nature of the compression method. As mentioned earlier in Section 4.2.2, the goal of this tool is to provide fast random access to gzip files, sacrificing compression performance. The Hanus *et al*. method [161] works only for data sets having exclusively 's' lines (e.g., the *multiz28way* data set). Despite the good results attained when compared to gzip (about 46.8%), the compression tool provided by Hanus *et al.* does not work in some files of the *multiz28way* data set namely "chr15", "chr16", "chr17", and "chr19". The encoder is capable of encoding those files, however it was not possible to decompress them. In terms of global coding time (compression plus decompression), our method is slower when compared to all the other methods, except method [161]. However, it seems that in the encoding phase lzma is the slowest method among all the others. Despite all this, the proposed compression tool is ≈ 5 times faster than method [161]. In the decoding phase, MAFCO is ≈ 4 times faster, when compared to the Hanus *et al*. method [161]. These conclusions were only made based on the results of Table E.5 for the *multiz28way* data set, using a single thread and without splitting the MAF file in several GOBs. Figure 4.20 illustrates the relation between the size of each data set and the corresponding encoding/decoding time using a single thread.

As mentioned earlier in Section 4.4.5, MAFCO implementation allows the user to split the MAF file into several GOBs and also to encode/decode them in parallel, using several threads. Table 4.11 presents the performance of MAFCO compared to gzip, when we split

Table 4.10: Performance of several compression methods in the four data sets used in this work. Size is indicated in bytes, whereas the percentages indicate the amount of reduction attained in comparison to gzip. Results for the Hanus *et al.* method [161] were not included for the last three data sets, because the method is not able to process the files with optional lines.

Data set	Original size	Gzip size	Bzip2	PPMd	LZMA	BGZIP	Hanus [161]	MAFCO
multiz28way	48,510,921,185	10,443,713,974	8.7%	10.9%	23.1%	-21.1%	46.8%	51.7%
multiz28wayB	113,528,207,035	16,216,614,098	16.6%	13.7%	20.7%	-35.1%	–	54.3%
multiz46way	270,579,509,536	32,523,764,993	18.1%	5.1%	21.0%	-49.1%	–	57.3%
multiz100way	794,243,994,061	72,086,319,647	21.2%	-8.5%	20.7%	-79.7%	–	34.1%
Total	1,226,862,631,817	131,270,412,712	18.9%	-0.8%	21.0%	-61.9%	–	43.7%

the MAF file into 1, 2, 4, and 8 GOBs. It is easy to conclude that even when the MAF file is split into 8 GOBs, the compression loss is lower than 1%, when compared to the results where the MAF file was not split.

Table 4.11: Performance of MAFCO using 1, 2, 4, and 8 threads for the four data sets used in this work. The size in bytes correspond to the compressed size when splitting the input file in 1, 2, 4, and 8 GOBs. The performance in percentage is in relation to Gzip and was computed according to Equation (4.5) illustrated in page 98.

Data sets	Measure	Number of threads			
		1	2	4	8
multiz28way	Size (bytes)	5,040,456,423	5,064,458,333	5,096,918,160	5,140,548,936
multiz28wayB		7,404,223,927	7,431,756,156	7,469,502,762	7,520,788,321
multiz46way		13,892,203,210	13,928,245,189	13,978,585,203	14,048,576,802
multiz100way		47,540,555,498	47,575,545,450	47,624,037,726	47,691,302,758
Total		73,877,439,058	74,000,005,128	74,169,043,851	74,401,216,817
multiz28way	Performance (%)	51.74	51.51	51.20	50.78
multiz28wayB		54.34	54.17	53.94	53.62
multiz46way		57.29	57.18	57.02	56.81
multiz100way		34.05	34.00	33.93	33.84
Total		43.72	43.63	43.50	43.32

In Table 4.12, we can find the results regarding the total amount of time needed to encode/decode the four data sets using 1, 2, 4, and 8 threads. Individual results for each data set are depicted in Tables E.9-E.12 of Appendix E.2. The obtained results were obtained by splitting the input MAF file into 1, 2, 4, and 8 GOBs. We used the same number of threads as the number of GOBs in our simulations, although MAFCO is capable of encoding/decoding a MAF file with a number of parallel processes that is different from the number of parts in which the file was split. Furthermore, the decoder can also use a number of threads that is different from the number used during encoding. In this particular case, we used the same number of threads, in order to be easier to analyze the results obtained. In the previous mentioned tables, we can find the encoding and decoding time in seconds in three different forms. The "CPU time", which corresponds to the total system and user time obtained by

Table 4.12: Performance of MAFCO using 1, 2, 4, and 8 threads for the four data sets used in this work. The "CPU time" corresponds to the total CPU time obtained by the *time* command in Linux. The "Optimal CPU time" corresponds to the "CPU time" divided by the number of threads. The speedup was computed by dividing the "Optimal CPU time" for one thread (sequential execution) by the "Optimal CPU time" for n threads (calculated according to (4.6)). Finally, the efficiency is obtained by dividing the speedup with the number of threads (according to (4.7)).

Measure	**Encoding**				**Decoding**			
Number of threads	**1**	**2**	**4**	**8**	**1**	**2**	**4**	**8**
CPU time (secs)	171,325	172,534	175,087	184,358	236,992	239,751	241,855	243,996
Optimal CPU time (secs)	171,325	86,267	43,772	23,045	236,992	119,876	60,464	30,500
Speedup	1.00	1.99	3.91	7.43	1.00	1.98	3.92	7.77
Efficiency	1.00	0.99	0.98	0.93	1.00	0.99	0.98	0.97

the Linux *time* [2] function. The "Optimal CPU time", computed by dividing "CPU time" by the number of threads used. The previous time measures were used to compute the *speedup* and *efficiency* metrics. The first metric is defined as

$$S_p = \frac{T_1}{T_p}, \tag{4.6}$$

where p corresponds to the number of parallel processes, T_1 is the execution time using one thread (sequential algorithm) and T_p is the execution time using p parallel processes. Linear or ideal speedup is obtained if $S_p = p$. The efficiency metric can be computed as

$$E_p = \frac{S_p}{p} = \frac{T_1}{pT_p}. \tag{4.7}$$

This metric is a value between zero and one, that indicates how well-used the processors are in executing the algorithm. Efficiency values close to one correspond to linear speedup algorithms. On the other hand, values close to zero indicate that the processors are not being well-used (poor parallelization).

The previous 2 metric (*speedup* and *efficiency*) are also presented in Table 4.12 and Tables E.9-E.12, in the last two rows of each table. Analyzing the obtained results, we can see that MAFCO has a linear speedup, up to 8 parallel processes. It seems that the efficiency of our method is similar between the encoding and decoding phases, regardless of the number of parallel processes used (up to 8 in this case).

Figure 4.19 depicts the encoding/decoding memory usage in megabytes for all the methods (except maf-bgzip). The memory has been estimated with *valgrind*, using *massif*. Because *valgrind* is very slow, we decided to assess the memory usage using three files from the *multiz28way* data set (*chr2*, *chr9*, and *chrY*). We computed the average value between the three files and plotted the obtained results in Figure 4.19. As can be seen, along the methods evaluated, gzip, bzip2 and ppmd are the ones that require less memory. Method [161], denoted in Figure 4.19 as "MSAc", is the one that requires more memory. The proposed method requires less memory than method [161] but more than lzma.

[2] `http://linux.die.net/man/1/time`

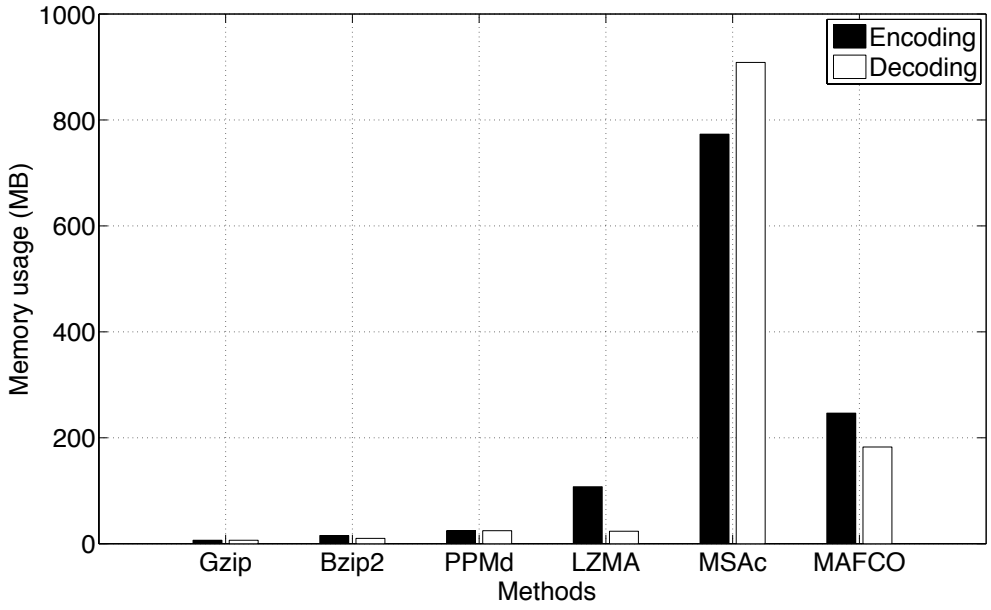


Figure 4.19: Encoding and decoding memory usage in megabytes for all methods used (except maf-bgzip). The memory has been estimated with *valgrind*, using *massif*. The presented values were computed considering only three representative MAF files (*chr2*, *chr9*, and *chrY*) from the *multiz28way* data set.

4.4.7 Complexity

The MAFCO compression tool uses several finite-context models that differ in terms of order and alphabet size, as was explained in Section 4.4. Similar to the previous method described in Section 4.3, the alphabet $\mathcal{A} = \{\mathrm{A}, \mathrm{C}, \mathrm{G}, \mathrm{T}, \text{-}\}$ is associated with the sequence alignments of each MSAB. Instead of using a mixture of finite-context models MAFCO relies on a single model using as a context template the one designated as 'C' in Figure 4.17, with order 10. Moreover, there are several binary streams that need to be coded using also finite-context models. Furthermore, we need robust models to encode the quality values of the 'q' lines and the status symbols of the 'i' and 'e' lines. Taking into account this, the number of counters for each model type is given by:

- Sequence alignment - $|\mathcal{A}_1| = 5$ and model order = 5, thus $5 \times 5^{10} = 48,828,125$ counters.
- Strand information - $|\mathcal{A}_2| = 2$ and model order = 3, thus $2 \times 2^3 = 16$ counters.
- Quality values - $|\mathcal{A}_3| = 12$ and model order = 5, thus $12 \times 12^5 = 2,985,984$ counters.
- Status symbols for the 'i' lines - $|\mathcal{A}_4| = 6$ and model order = 4, thus $6 \times 6^4 = 7,776$ counters.
- Status symbols for the 'e' lines - $|\mathcal{A}_5| = 5$ and model order = 4, thus $5 \times 5^4 = 3,125$ counters.
- 12 binary models - $|\mathcal{A}_6| = 2$ and model order = 5, thus $12 \times (2 \times 2^5) = 768$ counters.

The $|\mathcal{A}_n|$ denotes the alphabet size (number of symbols). The total amount of counters is then 51,825,794. Because each counter is stored in two bytes, the total amount of computer memory required to store the counters is about 99 megabytes. Taking into account the obtained results depicted in Table 4.12 and Tables E.9-E.12, we can conclude that MAFCO took almost 5 hours to encode the *multiz28way* data set. The decoding took a lit more (about 5 and a half hours) because as already mentioned earlier the decoder needs to write considerably more data in the disk than the encoder due to the fact that the encoded files are much smaller than the decoded files. Figure 4.20 illustrates the relation between the size of each data set and the corresponding encoding/decoding time. The plot is not as linear as the previous approach due to the fact that each data set is different in terms of not only size but also line types which will affect the obtained results. Moreover, we can also observe that the time difference between encoding and decoding tends to increase for larger data sets.

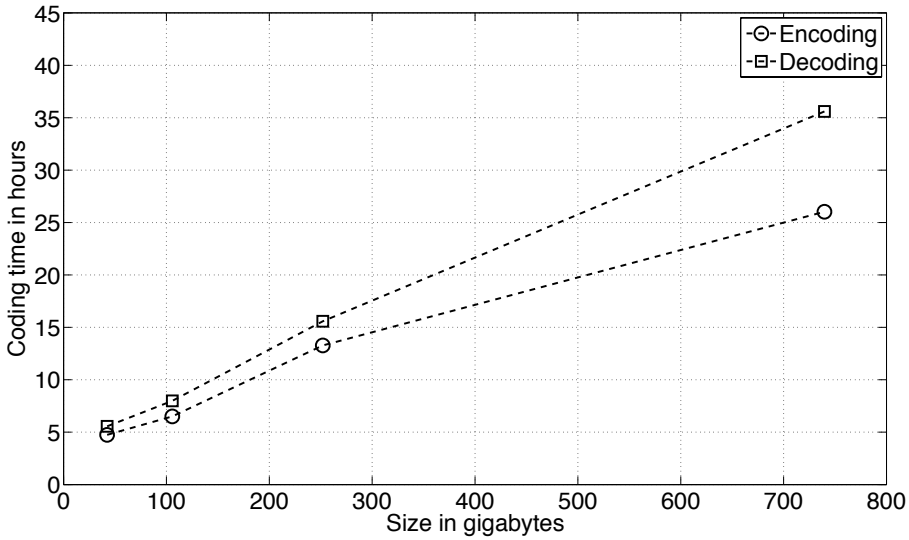


Figure 4.20: Relation between the size of each data set and the encoding/decoding time. Each marker symbolizes one data set. The linear behavior through the four data sets is less regular due to the difference between each data set in term of line types.

4.5 Summary

This chapter presents two compression methods for MAF files. MAF files store a series of multiple alignments in a format that is easy to parse and relatively easy to read. These files store multiple alignments at the DNA level between entire genomes. Both methods presented consider the sequence alignments as a special image, where each pixel/position can have only five different values (DNA bases and the alignment gap).

The first method described in Section 4.3 was developed to deal only with the DNA bases and the alignment gaps that can be found in the 's' lines. All the remain data (header information and optional lines) was not processed. This first method is based on a mixture of finite-context models and arithmetic coding. Several specialized models were purposed in order to obtain a more effective model mixture. After some experiments, we obtained an ideal mixture that maximizes the compression ratio for the *multiz28way* data set. We used

that same optimized mixture for the other data sets and we verified that the combination model remains effective, i.e., it does not reveal over-fitting. Regarding the compression results this method requires on average to store the sequence alignments from 0.52 up to 0.94 bits per symbol, including the additional information required by the decoder for recovering the MSABs, such as the size of each block. When compared to the Hanus *et al*. method [161], this method produced 7% better results, for the *multiz28way* data set. Despite the obtained results, this method is considerably slow specially for large data sets. In our experiments, the proposed method took ≈ 22 hours to encode the *multiz28way* data set and about four and a half days to encode the *multiz100way* data set. Despite all this, Hanus *et al*. method [161] method attained similar results in terms of coding time for the *multiz28way*.

In order to optimize the compression method proposed in Section 4.3, we introduced MAFCO (`https://github.com/lumiratos/mafco`), a compression tool for MAF files, in Section 4.4. This tool is capable of handling all the information that can be found in MAF files. The MAFCO core is based on simple models without any mixture. This approach with more simple models is the key aspect to improve the coding time. When compared to gzip, MAFCO attained a compression gain between 34.1% to 57.3%, depending on the data set. In terms of speed, the encoding time for the *multiz28way* data set is about 5 hours which is 4 times faster when compared to the first method described in Section 4.3. Furthermore, MAFCO allows parallel compression/decompression of MAF files which means that the coding time can be further improved, at a cost of some compression loss.

Finally, we should mention that the presented compression tools work for any kind of MAF files, regardless of the reference species used in the alignment. The MAFCO tool attained a lower performance on the *multiz100way* data set due to the small average number of columns of the MSABs of this data set. In order to overcome this situation, an alternative model should be implemented as future work. Moreover, there is also some space left to improve the splitting process of the MAF files. For now MAFCO uses a sequential split process without looking to the content of the MAF file. It should be interesting to study other splitting modes in order to offer users the possibility to decode certain regions of interest of the MAF files.

"Everything that has a beginning, has an end."

Andy and Larry Wachowski

5 Conclusion and future work

5.1 Conclusions

The goal of this research work was to study and develop lossless compression methods for microarray images and Whole Genome Alignments (WGAs).

In Chapter 2, we provided an overview of the most common image coding standards namely JBIG, PNG, JPEG-LS, and JPEG2000. We also described the intra-mode of H.264/AVC and HEVC, when applied to still images. However, both standards lack support for images with 16 bits per pixel. We also described the most common image decomposition approaches that are commonly used in image compression. Moreover, we explained the models used in this research work, known as finite-context models. In the last part of the chapter, we present a state of the art overview of the most relevant lossless compression tools for microarray images.

DNA microarray technology is an important tool that is used in the study of gene function, regulation and interaction across a large number of genes, and even across entire genomes. The raw data of a microarray experiment consists on a pair of 16 bits per pixel grayscale images that have a high spacial resolution, due to the microscopic size of the spots. These images are analyzed using several different software tools that allow the extraction of relevant information, such as the intensity of the spots and the background level. Despite the main goal of these experiments is to extract the genetic information from the expression levels, it is recommended to keep both the genetic information extracted and the original microarray experiments. The reasons of this recommendation is due to the fact that the analysis techniques are still evolving and also because repeating the microarray experiment is expensive and sometimes even impossible. The beginning of Chapter 3 provides a brief description of the microarray images data sets used in this research work. After that, some experimental simulations where performed in order to evaluate the most common image coding standards namely JBIG, PNG, JPEG-LS, and JPEG2000 in several microarray image sets (a total of 298 images). Among the previous mentioned image coding standards, JPEG-LS is the standard that provides the best compression results. In Section 3.4, we introduced some modifications to method [107] in order to improve the compression results. The first modification is a

segmentation step before the encoding procedure. The second modification implemented is bitplane reduction, where some redundancy in the pixel precision can be used to improve the compression results. According to the results, some improvements were obtained for some data sets, but globally the results are very close to the original method [107]. In Section 3.5, a simple bitplane coding compression tool based on the pixel value estimates was introduced, to be applied to microarray images. This method was inspired in the work done by Kikuchi *et al*. [140]. A second approach using a mixture of two models was also implemented and tested. One of the models used in the mixture is the method described in Section 3.5, whereas the other model is similar to the one described in [107]. According to the results, the method based on pixel value estimates attained better compression when compared to JPEG-LS and methods [106, 107]. The only exception is the *Full* mode of method [107]. For the second approach based on a mixture of models, the proposed method outperform JPEG-LS and the methods in [106, 107] in both modes (*SA-256* and *Full*). Section 3.6 presents a compression algorithm based on a binary tree decomposition that attained $\approx 9\%$ better results when compared to JPEG-LS. On the other hand, the *Full* mode of method [107] attained $\approx 8\%$ better results when compared to JPEG-LS. At the end of Chapter 3, we present a rate-distortion study that evaluates four methods, including JBIG and JPEG2000. The other evaluated methods are those of [16] and [107]. In terms of RMSE and MAE, it seems that the method [16] is the one that attained better ratio-distortion results when compared to the other three methods evaluated. A recent microarray specific distortion metric was also used in this study. According to the obtained results, it seems that methods based on bitplane decomposition such as JBIG and the method in [107], are the ones that attained better results in terms of MDM when compared to JPEG2000 and the method in [16].

Whole Genome Alignments WGAs are particular voluminous data sets in molecular genomics, that gained a considerable importance over the last years. These WGAs provide an opportunity to study and analyze the evolutionary process of several species. In this context, the MAF format is used to store a series of multiple alignments of several species and chromosomes. These MAF files can be very large requiring several hundreds of gigabytes to be stored in raw format. In Chapter 4, we addressed two compression methods for these MAF files. The first method described in Section 4.3 only deals with the DNA bases and alignment gaps that can be found in the MAF files. All the remaining optional lines and header information was not considered. This first method, based on a mixture of finite-context models and arithmetic coding attained a compression gain of $\approx 7\%$ when compared to a recent method proposed by Hanus *et al*. [160]. The second compression tool is an optimized version of the method introduced in Section 4.3. The compression tool, designated as MAFCO in Section 4.4, is a full lossless compressor, capable of handling all information that can be found in MAF files. This tools is based on several single finite-context models of several orders. According to the attained results, MAFCO provides a compression gain between $\approx 34\%$ to $\approx 57\%$, when compared to gzip. Furthermore, MAFCO allows parallel compression/decompression of MAF files which means that the coding time can be reduced depending on the number of parallel processes and also allowing the user to decode only a portion of the file, without requiring to decode the entire file.

5.2 Future work

As many research works, there is always space for more work to be done. In this particular case, the compression tools developed can be further improved in terms of encoding/decoding speed and also in terms of compression performance. There is also the possibility to developed new and more efficient methods to compress microarray images and MAF files. In the following, we will briefly discuss the possible directions for future research:

- Improve the compression method for microarray images based on binary-tree decomposition explained in Section 3.6 using alternative strategies for building and traversing the binary tree, different context modeling approach and adding multi-resolution support.
- Implement a rate-distortion mechanism, based on the MDM described in Section 3.7, to be applied to a specific compression method based on a bitplane decomposition approach, for microarray images. The goal is to provide a reconstruction image with higher MDM values at lower bitrates.
- Improve the encoding/decoding time of the proposed methods for microarray images using a multi-threading approach. This parallel approach can be implemented in two ways. In the first one, the input image is split into several blocks with the same size and coded in parallel. The alternative approach could be encoding each bitplane of a given image in parallel.
- Study the relation between each pair of microarray images and develop a new compression method that can take advantage of this relation.
- Apply the methods presented in Chapter 3 to other types of images (medical, hyperspectral, etc.).
- Improve the MAFCO compression tool introduced in Section 4.4, mainly for the *multiz100way* data set. The goal would be to study the reason to the lower performance of this tool in this particular data set and develop an alternative model to overcome this issue.
- Add a mechanism to MAFCO in order to be able to decompress sections of each chromosome. The idea is also to provide random access to the compressed stream using a mechanism similar to the one described in Section 4.2.2. This mechanism needs to be efficient in order to increase the access speed without jeopardizing the compression performance.
- Study alternative models to compress the optional lines and header information of MAF in order to improve the compression results of MAFCO.

5.3 Acknowledgments

We would like to thank Miguel Hernández-Cabronero for providing some of the microarray images data sets used in this work and also for providing his implementation of the Microarray Distortion Metric as described in [127]. Moreover, we also like to thank Pavol Hanus for providing an implementation of his algorithm [161] and Clayton Wheeler for the support given in the use of his maf-bgzip tool [164–166].

"The skill of writing is to create a context in which other people can think."

Edwin Schlossberg

A

Microarray images data sets

This Appendix provides some representative microarray images that were used to evaluate the performance of the compression methods described in this thesis. The data sets were collected from different publicly available sources however, currently some of these data sets are not available in their original location anymore. We decided then to upload them to an alternative public location that we indicate in each data set in this Appendix. We can also find these alternative download locations at `http://bioinformatics.ua.pt/software/ment` or `https://github.com/lumiratos/ment`. The presented images were enhanced through an intensity adjustment process in order to be more easy to observe the spots.

In Table A.1, we can find supplementary information regarding the data sets used in this work. The properties presented show that the *Omnibus* data set (considering both modes) is the one with more pixels among the data sets presented (about 77.7%). The spot layout and the number of spot regions are other interesting property that is also depicted in Table A.1, that can affect the compression performance.

A.1 ApoA1 data set

The *ApoA1* data set from the Terry Speed Microarray data analysis group can be found at `http://www.stat.berkeley.edu/users/terry/zarray/Html/apodata.html`. Alternative it can also be downloaded at `http://sweet.ua.pt/luismatos/microarrays/apoa1.html`. This data set contains 32 images with approximate $\geq 1044 \times 1041$ pixels. In Figure A.1, we can find a representative pair of images from this data set. The remaining images are very similar in terms of entropy and size.

A.2 Arizona data set

The *Arizona* data set was provided by Megan T. Sweeney from the David Galbraith laboratory and can be found at `http://deic.uab.es/~mhernandez/media/imagesets/arizona.tar.bz2`. Alternative, the data set can also be found at `http://sweet.ua.pt/luismatos/`

Table A.1: Supplementary information regarding the microarray image data sets used in this work. The overall percentage of pixels is computed taking into account the amount of pixels of each data set and the total number of pixels for all data sets.

Data sets	**Number pixels**	**Overall pixel %**	**Spot layout**	**Number of spot regions**
ApoA1	34,813,224	1.03	square	16
Arizona	364,320,000	10.74	hexagonal	4
ISREC	14,000,000	0.41	square	4
IBB	553,892,460	14.07	square	48
Omnibus-LM	1,317,600,000	38.85	hexagonal	4
Omnibus-HM	1,317,600,000	38.85	hexagonal	4
Stanford	207,590,280	6.12	square	16
Yeast	114,294,784	3.17	square	4
YuLou	21,329,553	0.63	square	* 16; 48; 32
Overall	3,938,110,748	100.00	–	–
*** 16 spot regions for "array1", 48 for "array2", and 32 for "array3".**				

`microarrays/arizona.html`. This data set contains a total of 6 images with 13800×4400 pixels. In Figure A.2, we present a representative pair of images from this data set. The remaining images are very similar in terms of entropy and size.

A.3 IBB data set

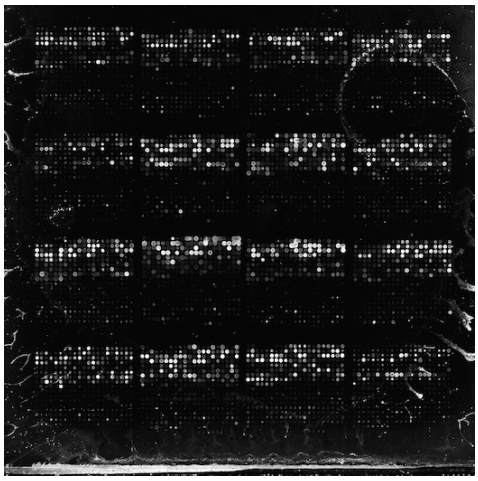
The *IBB* microarray image data set was provider by Antonio Casamayor from the Institut de Biotecnologia i Biomedicina (IBB) at the Universitat Autnoma de Barcelona (UAB). The data set can be downloaded at `http://deic.uab.es/~mhernandez/media/imagesets/ibb.tar.bz2`. Alternative, we can also found it at `http://sweet.ua.pt/luismatos/microarrays/ibb.html`.

A.4 ISREC data set

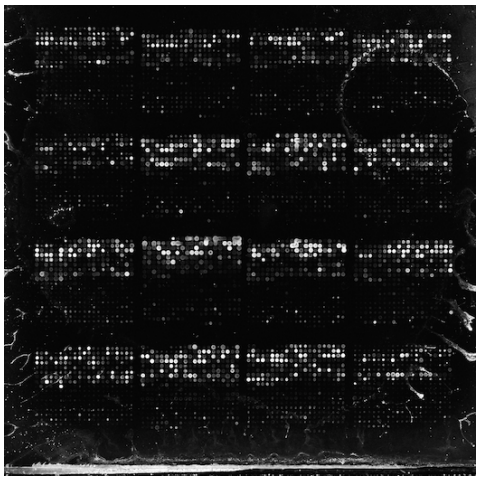
The *ISREC* data set from the Swiss Institute for Bioinformatics (SIB) was originally downloaded from `http://www.isrec.isb-sib.ch/DEA/module8/P5_chip_image/images` however, this site is currently offline. Despite this, the *ISREC* data set is available for download at `http://sweet.ua.pt/luismatos/microarrays/isrec.html`. This data set contains 14 images with 1000×1000 pixels. In Figure A.4, we present a representative pair of images from this data set. The remaining images are very similar in terms of entropy and size.

A.5 Omnibus data set

The *Omnibus* data set can be obtained at the Gene Expression Omnibus (GEO) of the National Center for Biotechnology Information (NCBI) at the following location `ftp://ftp.ncbi.nlm.nih.gov/geo/samples`. The previous URL contains a considerable



1230c1G



1230c1R

Figure A.1: A pair of microarray images from the *ApoA1* data set with 1044×1041 pixels.

amount of folders and files so we decided to put the used images at `http://sweet.ua.pt/luismatos/microarrays/omnibus-lm.html` and `http://sweet.ua.pt/luismatos/microarrays/omnibus-hm.html`. We separate this data set into two sub-data sets: Low Mode images and High Mode images. The two sub-data sets are associated with the same experiment but they were scanned using two different modes: high and low. Each sub-data set contains a total of 25 images (total of 50 images) with 12200×4320 pixels. In Figure A.5, we present a representative pair of images from this data set, scanned in High Mode. The remaining images, scanned in High Mode are very similar in terms of entropy and size. On the other hand, the images scanned in Low Mode are different in terms of entropy but have the same size. For more information consult Table 3.1 of Section 3.1.

A.6 Stanford data set

The *Stanford* data set was originally downloaded from `ftp://smd-ftp.stanford.edu/pub/smd/transfers/Jenny`. It seems that the data set is not available at that location anymore. However, it can be found at `http://sweet.ua.pt/luismatos/microarrays/stanford.html`. This data set contains a total of 40 images with different sizes, $> 1900 \times 2000$ pixels. Figure A.6 depicts a representative pair of images from this data set. The remaining images are very similar in terms of entropy and size.

A.7 Yeast data set

The *Yeast* data set from the Stanford Yeast Cell-Cycle Regulation Project, is available online at `http://genome-www.stanford.edu/cellcycle/data/rawdata/individual.html`. Alternative, it can be obtained at `http://sweet.ua.pt/luismatos/microarrays/yeast.html`. This data set contains 109 images with 1024×1024 pixels. In Figure A.7, we present a representative pair of images from this data set. The remaining images are very similar in terms of entropy and size.

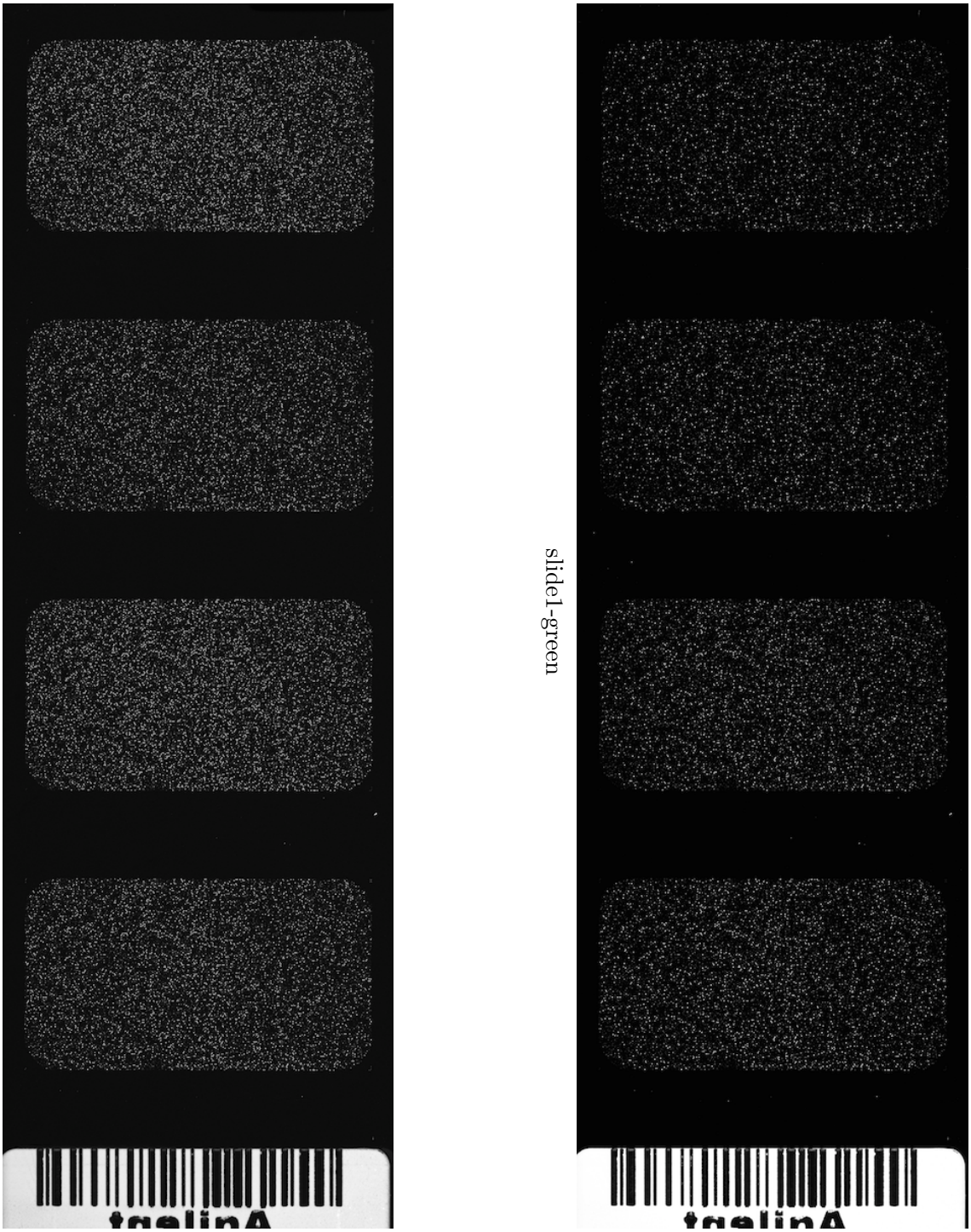
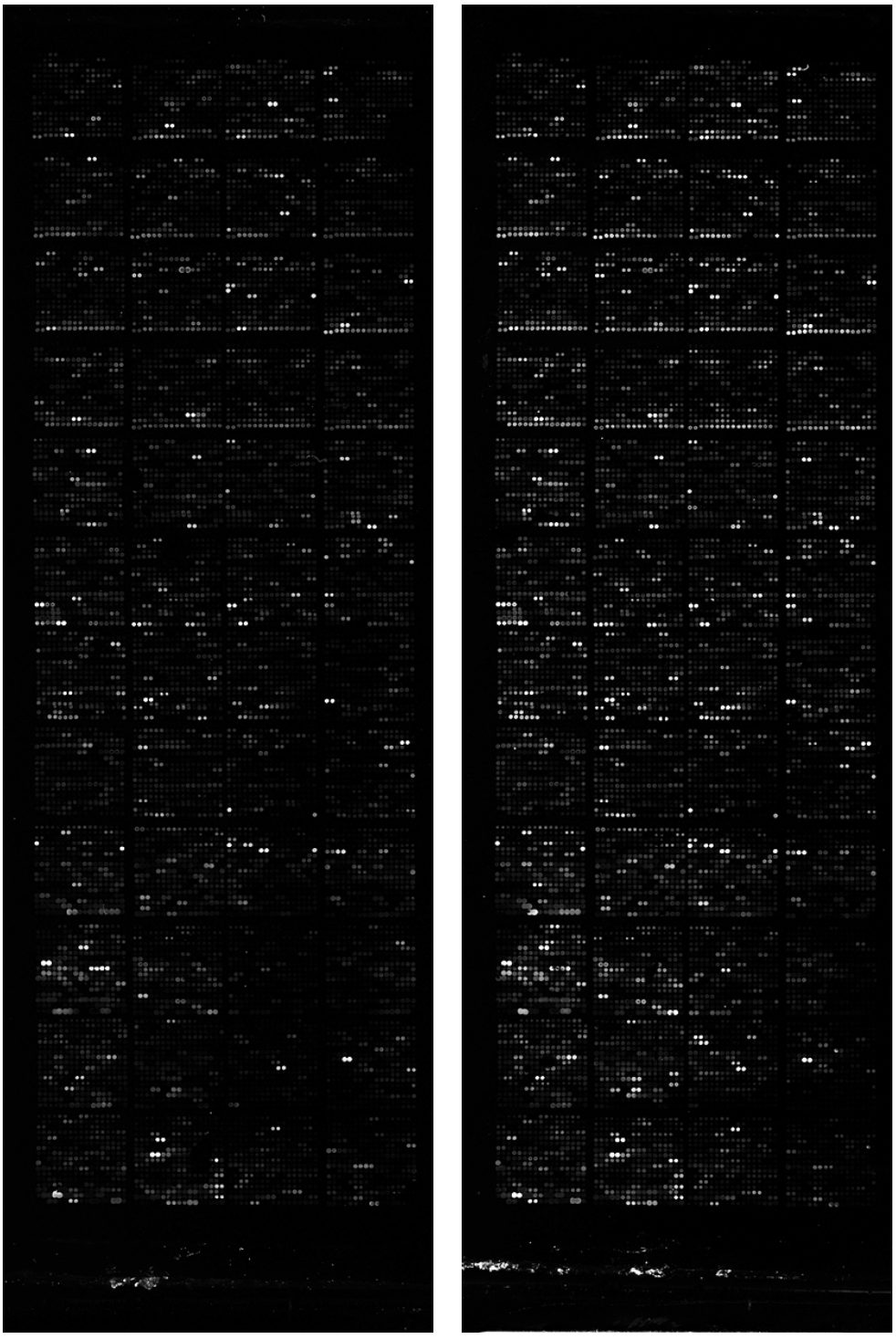


Figure A.2: A pair of microarray images from the *Arizona* data set with 13800 × 4400 pixels.

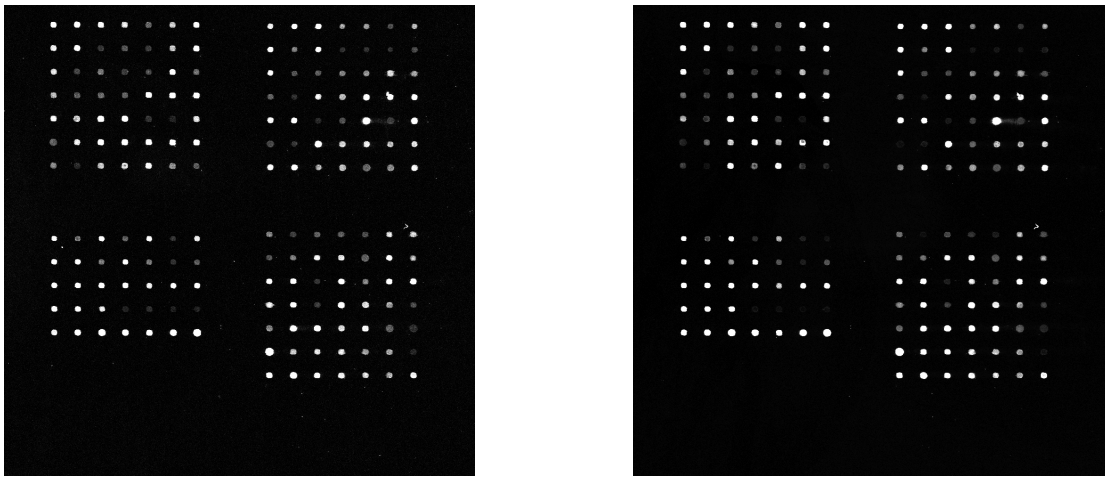
A.8 YuLou data set

The *YuLou* microarray image set was first used by the "MicroZip" tool proposed by Lonardi and Luo in 2004 [96] and latter in 2005 by Zhang *et al*. [99]. Usually in the literature this data set is also designated as *MicroZip* however, we decided to use the name *YuLou* in order to avoid confusion with the "MicroZip" compression tool. The *YuLou* microarray image set was originally downloaded from `www.cs.ucr.edu/yuluo/MicroZip`, but currently this site is offline. However, it can be downloaded from `http://sweet.ua.pt/luismatos/microarrays/yuluo.html`. This data set contains only 3 images with different sizes ($> 1800 \times 1900$) pixels. Figures A.8-A.10 depict the 3 images from this data set.



wt32h-Cy3-532 wt32h-Cy5-635

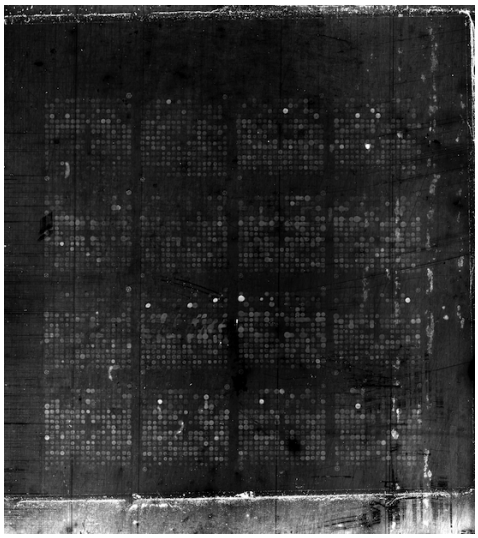
Figure A.3: A pair of microarray images from the *IBB* data set with 2019×6235 pixels.



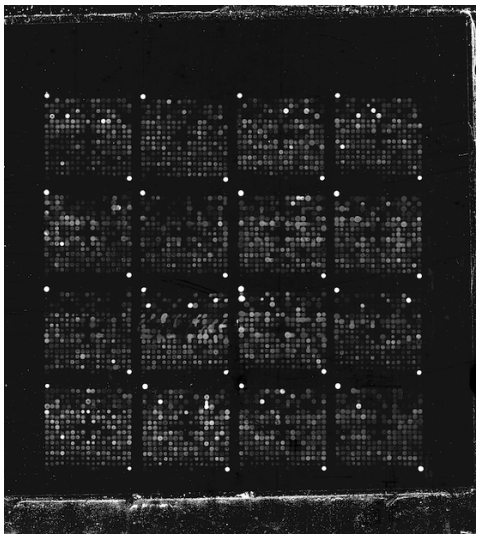
Def661Cy3 Def661Cy5

Figure A.4: A pair of microarray images from the *ISREC* data set with 1000×1000 pixels.

Figure A.5: A pair of microarray images from the *Omnibus* data set with 12200×4320 pixels.

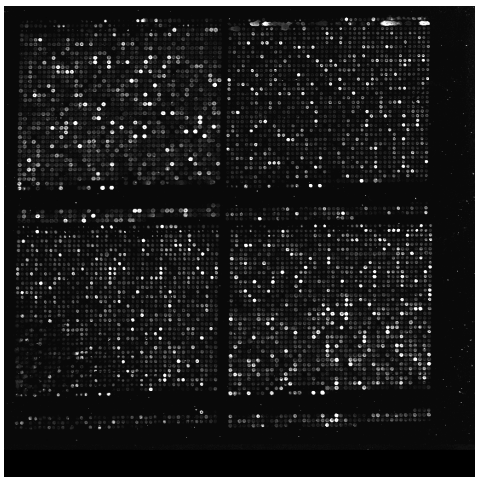


2001-01-18-0008-ch1

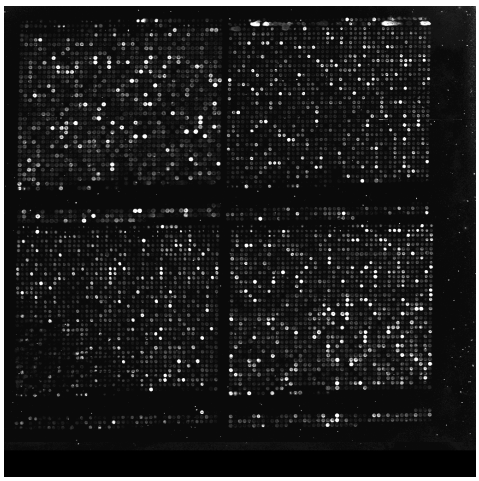


2001-01-18-0008-ch2

Figure A.6: A pair of microarray images from the *Stanford* data set with 2200×2467 pixels.



y744n32-ch1



y744n32-ch2

Figure A.7: A pair of microarray images from the *Yeast* data set with 1024×1024 pixels.

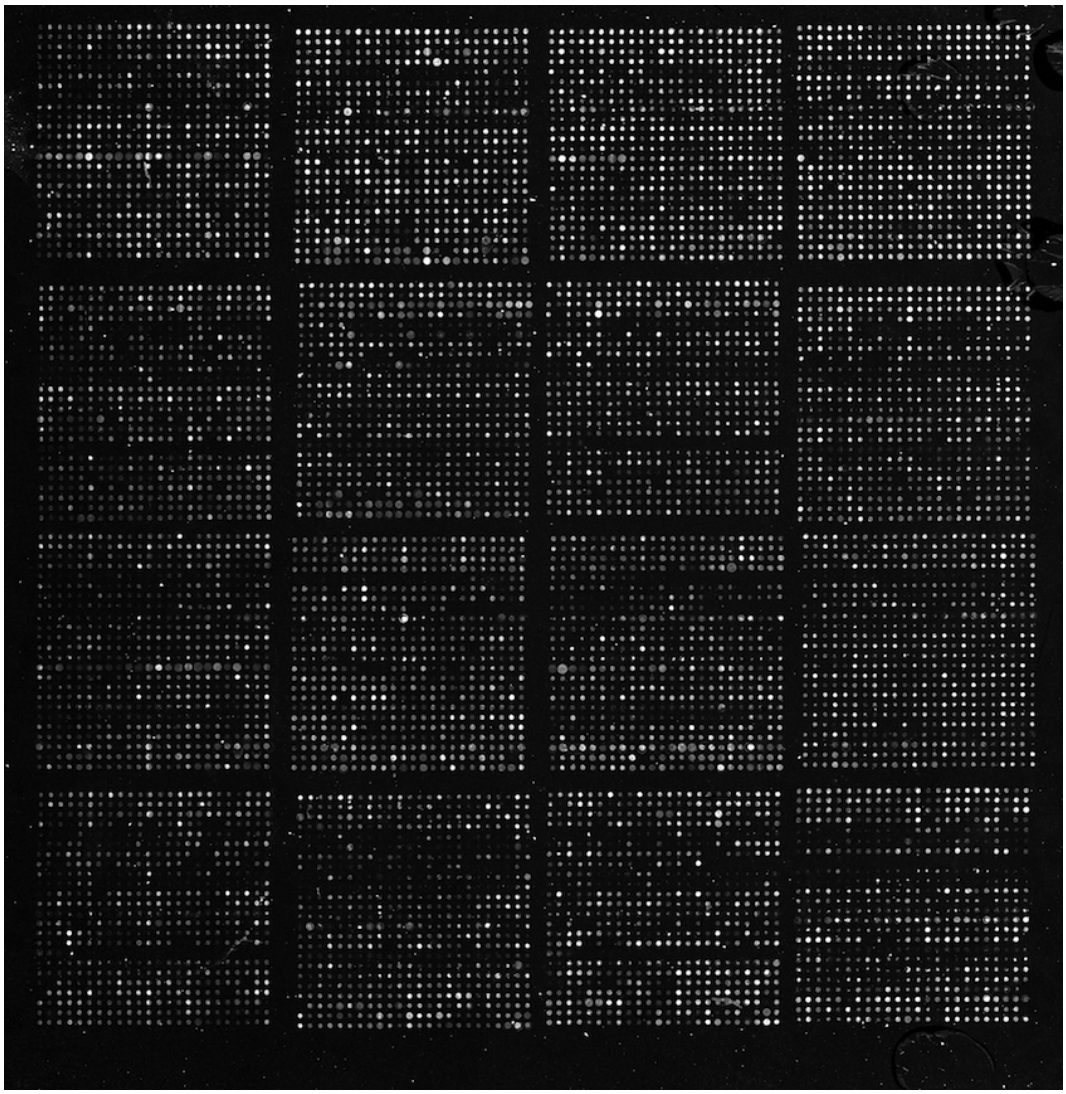


Figure A.8: Image "array1" from the *YuLou* data set with 1872×1916 pixels.

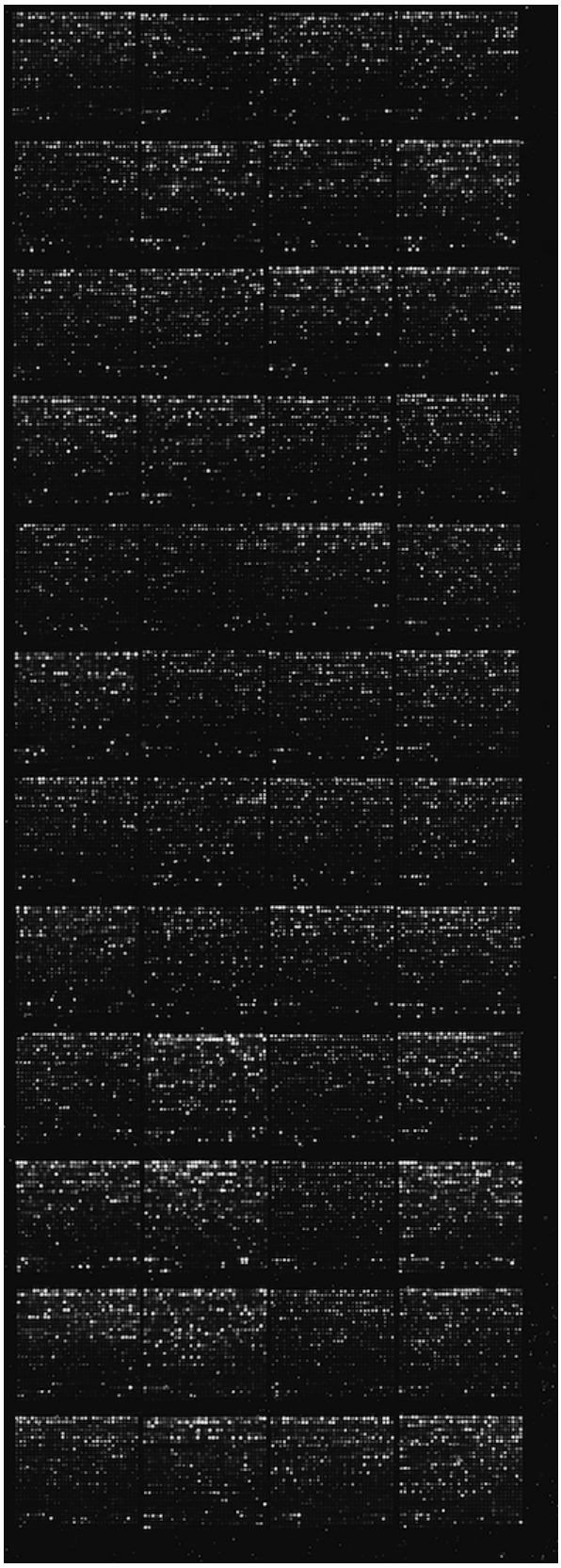


Figure A.9: Image "array2" from the *YuLou* data set with 1956×5496 pixels.

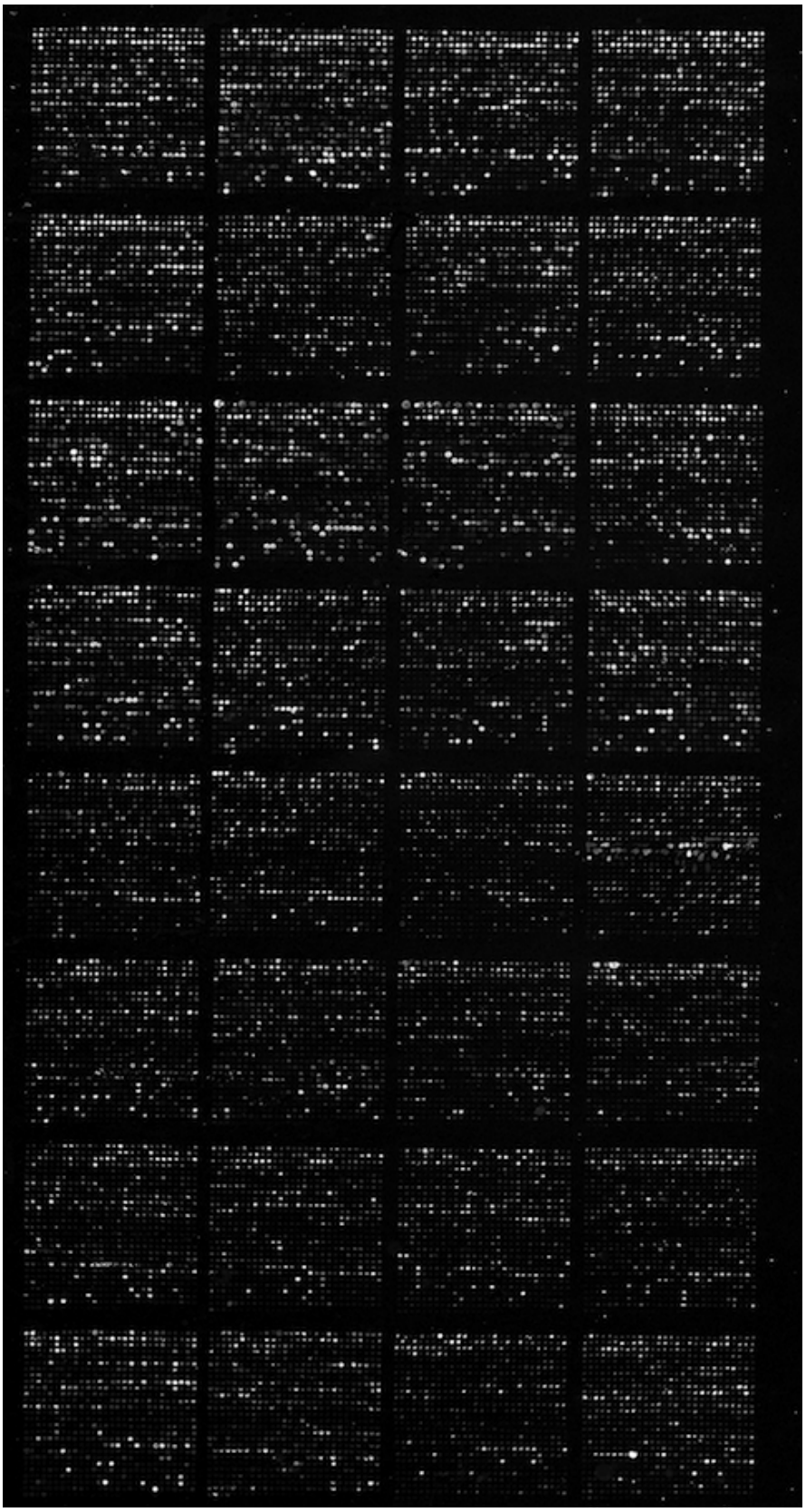


Figure A.10: Image "array3" from the *YuLou* data set with 1929×3625 pixels.

"No act of kindness,
no matter how small,
is ever wasted."

Aesop

B

Global microarray image compression results

In this Appendix we present a global view of the obtained results for the microarray compression tools proposed in this thesis. In Table B.1, we can find the compression results in bits per pixel for all the methods proposed in Chapter 3.

Table B.1: Compression results in bits per pixel (bpp) for all the proposed compression methods described in Chapter 3. Results for methods [106, 107] were also included. The best results are in bold. In the last four rows we present the encoding/decoding time ("CTime" and "DTime") in hours and speed ("CSpeed" and "DSpeed") in kilobytes per second, globally for all data sets.

	Compression methods																	
Data sets	**Neves [106]**	**SBC**		**Context search area (256 × 256)**						**Context search area (Full)**						**BTD**	
		Greedy	**Best**	**Neves [107]**	**BFS**	**HC**	**SBR**	**SBC-Mix**	**Neves [107]**	**BFS**	**HC**	**SBR**	**SBC-Mix**	**Greedy**	**Best**
ApoA1	10.280	10.205	10.205	10.225	10.265	10.259	10.263	10.149	10.194	10.234	10.231	10.232	**10.142**	10.199	10.194
Arizona	8.394	8.308	8.308	8.293	8.291	8.300	8.297	8.238	8.242	8.245	8.244	8.243	8.219	**8.186**	**8.186**
IBB	8.063	8.537	8.537	8.039	8.041	8.041	8.041	8.001	7.974	7.982	7.978	7.978	7.966	**7.943**	**7.943**
ISREC	10.217	10.260	10.260	10.199	10.215	10.239	10.235	10.169	10.159	10.193	10.195	10.199	**10.148**	10.200	10.198
Omnibus (LM)	5.309	4.645	4.645	5.679	5.637	5.652	5.661	4.646	4.567	4.570	4.561	4.565	4.545	4.540	**4.539**
Omnibus (HM)	7.047	6.581	6.581	7.744	7.616	7.743	7.738	6.571	6.471	6.479	6.473	6.472	6.443	6.401	**6.400**
Stanford	7.664	7.403	7.403	7.468	7.415	7.433	7.436	7.331	7.379	7.349	7.350	7.349	7.305	7.306	**7.303**
Yeast	5.610	5.492	5.492	5.511	5.430	5.601	5.506	5.354	5.453	5.395	5.527	5.466	5.326	5.323	**5.318**
YuLou	8.840	8.669	8.669	8.667	8.675	8.667	8.668	8.609	8.619	8.641	8.626	8.626	**8.591**	8.593	8.592
Average	6.772	6.437	6.437	7.101	7.041	6.092	7.094	6.343	6.284	6.288	6.286	6.285	6.257	6.236	**6.235**
CTime (hours)		3.07	21.40	60.95	6.24	8.33	6.32	6.99	40.09	643.19	964.29	676.07	727.33	686.63	63.10	168.33
DTime (hours)		4.30	4.91	4.90	3.67	3.55	4.65	4.04	18.83	6.00	5.63	7.44	6.71	23.64	14.42	15.70
CSpeed (KB/s)		539	100	35	343	257	338	306	53	3	2	3	3	3	34	13
DSpeed (KB/s)		498	434	437	583	602	460	529	114	357	380	289	319	91	148	136

Legend:

- SBC: Simple Bitplane Coding (Section 3.5)
- BFS: Background/Foreground Separation (Section 3.4.1)
- HC: Histogram Compaction (Section 3.4.2)
- SBR: Scalable Bitplane Reduction (Section 3.4.2)
- BTD: Binary-Tree Decomposition (Section 3.6)

"Two roads diverged in a wood, and I –
I took the one less traveled by,
And that has made all the difference."

Robert Frost

C Multiple Alignment Format (MAF)

In this Appendix we will address the line types used in the Multiple Alignment Format (MAF). According to [172], the MAF is used for storing a series of multiple alignments in a format that is easy to parse and easy to read. This format stores multiple alignments at the DNA level between entire genomes.

The general structure of this format is line-oriented. Each alignment ends with a blank line. Each alignment sequence is stored on a single line that has no length limit (the size of each line is limited by the disk size and the file system that is used). Each word in a line is separated by any white space. Comment lines are identified by a '#' symbol in the beginning of the line. Lines starting with '##' are usually ignored by most of the programs, however they contain meta-data of one form or another.

Each MAF file is divided into paragraphs that terminate in a blank line. Within a paragraph, the first letter of the line will indicate its type. Each multiple alignment is in a separated paragraph that begins with an 'a' line and contains 's', 'q', 'i', and 'e' lines. This set of lines within the same paragraph is known as a Multiple Sequence Alignment Block (MSAB).

As mention earlier in the main document (Section 4.1.1), the 's' lines are the most important lines that contains all the information about the sequence alignment (DNA bases and gaps). Regarding the optional lines, they are usually ignored by parsers but in this case we are not ignoring them. In what follows, we will describe the format of each line type and its respectively fields. We also present some examples of MAF files in the end of this Appendix.

C.1 The header lines

The header of a MAF file begins with a "##maf". The previous word is followed by a series of "variable=value" pairs. No white spaces surrounding the "=" symbol.

```
##maf <variable>=<value> <variable>=<value> ... <variable>=<value>
```

The currently defined variables are:

- `version` (required) - the alignment version currently set to 1.
- `scoring` (optional) - the name of the scoring scheme used to create the alignments.
  - bit - roughly corresponds to blast bit values (roughly 2 points per aligning base minus penalties for mismatches and inserts).
  - blastz - correspond to the blastz scoring scheme (roughly 100 points per aligning base).
  - probability - some score normalized between 0 and 1.
- `program` (optional) - name of the program that generated the alignment.

Other variables that might occur are usually ignored by most of the parsers. After the "##maf" some optional lines are possible. These optional lines start by a '#' and are usually the parameters that were used to run the alignment program. This lines are not mandatory so there are some files that do not have them. After the optional header lines, the same '##maf' line presented in the beginning of the file, is displayed. See Figure C.1 and C.2 for better understanding the MAF header structure.

C.2 The 'a' lines

After the MAF header, each MSAB starts with an 'a' line. Each 'a' line is followed by a series of "variable=value" pairs. Currently, the variables defined are:

- `score` (optional) - it is a floating point score associated with the MSAB.
- `pass` (optional) - a positive integer value. For programs that do multiple pass alignments such as blastz [173, 174], this shows which pass this alignment came from. Typically, pass 1 will find the strongest alignments genome-wide, and pass 2 will find weaker alignments between two first-pass alignments.

None of the above variables are mandatory, however the `score` variable is present in all the data sets used in this work.

C.3 The 's' lines

The format of a 's' line is:

```
s <source> <start> <size> <strand> <srcSize> <MSA>
```

The 's' lines have several fields which are described next:

- `<source>` contains the name of one of the source sequences for the alignment. This field is usually defined as `<specieName.chromossome>` or `<specieName.scaffold>`.
- `<start>` the start position of the aligning region in the source sequence.
- `<size>` the size of the aligning region in the source sequence. This number corresponds to the number of non-gaps in the alignment.

- `<strand>` defines if the alignment is done to the reverse-complement source.
- `<srcSize>` the size of the entire source sequence, not just the parts involved in the alignment.
- `<MSA>` the nucleotides and gaps in the alignment.

C.4 The 'q' lines

The 'q' lines contain information regarding the quality of each aligned base for the current source sequence. This quality information is a compressed version of the actual raw quality data, where the quality of each aligned DNA base is represented using a single character that can be a number between 0 and 9 or 'F' which represents a finish sequence (see Table C.1 for more details). These 'q' lines are always associated with the previous 's' lines and therefore they share the same source name. The first 's' line (reference source sequence) of each MSAB does not have a 'q' line associated. The 'q' lines have the following structure:

`q <source> <qualityValues>`

The 'q' lines only have two fields:

- `<source>` contains the name of the source sequence. It should be the same source name as the 's' line immediately preceding this line.
- `<qualityValues>` the MAF quality values that correspond to the aligning DNA bases in the previous 's' line. The alignment gap ('-') that is present in the previous 's' line is replicated to the 'q' line as well. The quality values are computed using

$$\text{MAFQV} = \min\left(\left\lfloor \frac{\text{AQV}}{5} \right\rfloor, 9\right), \tag{C.1}$$

where MAFQV denotes the MAF quality value that is used in the 'q' lines and AQV represents the actual quality value. Table C.1 shows the range of quality values for the 'q' lines, as well as the mapping of the raw quality values to the MAF quality values.

Table C.1: Mapping of MAF quality values. The quality values can be 'F' (finished sequence) or a number derived from the actual quality scores (which ranges from 0 to 97) or the manually assigned score of 98. The MAF quality values are computed according to (C.1).

MAF quality value	**Raw quality score range**	**Quality level**
0 - 8	0 - 44	Low
9	45 - 97	High
0	98	Manually assigned
F	99	Finished

C.5 The 'i' lines

The 'i' lines contain information about what is happening before and after the current MSAB in the aligning source. These informative lines have information about the context of the source sequence lines immediately preceding them. The 'i' lines have the following format:

```
i <source> <lStatus> <lCount> <rStatus> <rCount>
```

Similar to the 'q' lines, the source name of an 'i' line is the same one of the preceding 's' line. The possible 'i' line fields are:

- `<source>` contains the name of the source sequence. Should be the same source name as in the 's' line immediately preceding this line.
- `<lStatus>` left status is represented by a single character that describes the relationship between the sequence in this MSAB and the sequence that appears in the previous MSAB.
- `<lCount>` left count is usually the number of DNA bases in the aligning source between the start of this alignment and the end of the previous one.
- `<rStatus>` right status is represented by a single character that describes the relationship between the sequence in this MSAB and the sequence that appears in the subsequent MSAB.
- `<rCount>` right count is usually the number of DNA bases in the aligning source between the end of this alignment and the start of the next one.

The status characters that can be found in 'i' lines can only have one of the following values:

- 'C' - the sequence before or after is contiguous to this MSAB.
- 'I' - there are bases between the bases in this MSAB and the one before or after it.
- 'N' - this is the first sequence from this source.
- 'n' - this is the first sequence from this source but it is bridged by another alignment from a different chromosome/scaffold.
- 'M' - there is missing data before or after this block ('N' or 'n' symbols in the sequence).
- 'T' - the sequence in this MSAB has been used before in a previous MSAB (likely a tandem duplication).

C.6 The 'e' lines

The 'e' lines contain information about empty parts of the MSAB. The 'e' lines indicate if there is not aligning DNA for the specified source, but the current MSAB is bridged somehow by a chain that connects MSABs before and after this MSAB. The 'e' line has the following structure:

```
e <source> <start> <size> <strand> <srcSize> <status>
```

The fields of the 'e' lines are described next. There are some fields that are very similar to the ones of the 's' lines described earlier.

- `<source>` contains the name of one of the source sequences for the alignment.
- `<start>` the start position of the non-aligning region in the source sequence.
- `<size>` the size of the non-aligning region in the source sequence.
- `<strand>` defines if the previous alignment is done to the reverse-complement source.
- `<srcSize>` the size of the entire source sequence, not just the parts involved in the alignment.
- `<status>` a character that specifies the relationship between the non-aligning sequence in this MSAB and the sequence that appears in the previous and subsequent MSABs.

The status characters that can be found in 'e' lines can only have one of the following values:

- 'C' - the sequence before or after is contiguous implying that this region was either deleted in the source or inserted in the reference sequence.
- 'I' - there are non-aligning bases in the source species between chained MSABs before and after this MSAB.
- 'N' - there are non-aligning bases in this source and the next MSAB starts in a new chromosome or scaffold that is bridged by a chain between still other MSABs.
- 'M' - there is missing data before or after this block ('N' or 'n' symbols in the sequence).
- 'T' - the empty region of this MSAB has been used before in a previous MSAB (likely a tandem duplication).

C.7 MAF file examples

In Figures C.1 and C.2, we can find a small portion of file "chrM.maf" from the *multiz28way* and *multiz46way* data sets, respectively. In the first example (Figure C.1), the MSABs only contain 's' lines. On the other hand, we can find optional lines in the MSABs of Figure C.2. Notice that the MAF header (the comment lines in the beginning that start with a '#') is different in the two cases in terms of number of lines.

```
##maf version=1 scoring=autoMZ.v1
# autoMZ.v1 + T=chrM__0.maf
# maf_project.v12 hg18
# multiz.v11.1 M=1
##maf version=1 scoring=autoMZ.v1
a score=4683.000000
s hg18.chrM               193 33 +   16571 CCTACTAAAGTGTGTTAATTAATTAATGCTTGT
s anoCar1.scaffold_364 458122 28 - 1526782 -----TACATGGTATTATTTAATTAATGCTTGT
s galGal3.chrM            851 33 +   16775 CCCGGTAAATGGTGCTATTTAGTGAATGCTTGT

a score=-5749.000000
s hg18.chrM               226 70 +   16571 AGGACATAATAATAACAATTGAATGTCTGCACAGCCGCTTTCCACACAGACATCATAACAAAAAATTTCC
s ornAna1.Contig3497    32985 67 +   39338 AGGACACAGTTCAAGCAATGG--Tatttactgagt-gcttaccctgtgcacagcactgtactagcattgt
s galGal3.chrM            884 60 +   16775 CGGACATATTTTTATCAATTT----TCACTTCCTCTATTTTCTTCACAAAACTAGGAAA------TTCAC
s anoCar1.scaffold_364 458150 63 - 1526782 TAGACATAAATTTTTTAAAG-------GATTGACCTGTTTTTTATTTAAACCTCCGAAAACAGGCCTCAC

a score=-4292.000000
s hg18.chrM                 296 40 +   16571 ACCAA----------ACCCCCCC-CTCCCCCCGC----TTCTGGCCACAGCACTT
s echTel1.scaffold_154587   199 40 +    2023 AGCAA----------ACCCCCCTACCCCCCCTAT----TACTTAATA-AGTACCT
s ornAna1.Contig116526       11 43 +    3072 ATTAA----------ACCCCCTA-CCCCCTTAAAAATGTCATTACCA-AAGAACT
s anoCar1.scaffold_364   458213 42 - 1526782 AATGACCAGATTAAAAAACCCTA-CACCAATAAC----TATCCATTT--------

a score=1985.000000
s hg18.chrM                 336 30 +   16571 AA-ACA-------CATCTCTGCCAAACCCCAAAAACAA
s echTel1.scaffold_154587   239 30 +    2023 AATATA-------CATC-CTGCCAAACCCCAAAAACAA
s ornAna1.Contig116526       54 29 +    3072 A--GTA-------TTTCTCCGTCAAACCCCAAAACCGG
s galGal3.chrM              990 26 +   16775 AA-ACA-------TTTT-TTAAAAAACT---AAATTAC
s anoCar1.scaffold_364   458255 36 - 1526782 AA-TCAAGTTTACTTTC-TTGCCAAACCCCAAAAGTTG

a score=-6101.000000
s hg18.chrM               366 25 +   16571 AGAAC--------------CCTAACACCAGCCTAACCAG
s ornAna1.Contig116526     83 35 +    3072 ACAACAAA---TCTTTAGCTAAACATCTTATCATCTAG
s galGal3.chrM           1016 33 +   16775 -----ATACAAACTACCGCATAAAATCCCTCAAACTAT
s anoCar1.scaffold_364 458291  3 - 1526782 -----AGA--------------------------------
```

Figure C.1: The first 5 MSABs of file "chrM.maf" from the *multiz28way* data set.

```
##maf version=1 scoring=autoMZ.v1
a score=15057.000000
s hg19.chrM                  0 187 + 16571
GATCACAGGTCTATCACCCTATTAACCACTCACGGGAGCTCTCCATGCATTTGGTATTTT-CGTCTGGGGGGTGTGCACGCGATAGCATTGCGAGACGCTGGAGCCGGAGCACCCTATGTCGCA
GTATCTGTCTTTGATTCCTGCCTCATTCTATTATTTATCGCACCTACGTTCAATATTACAGGCG
s gorGor1.Supercontig_0439211 0 184 +   616
GATCACAGGTCTATCACCCTATTAACCACTCACGGGAGCTCTCCATGCATTTGGTATTTTTCGTC-GGGGGGGTGTGCACGCGATAGCATCGCGAAACGCTGGAACCGGAGCACCACATGTCGCA
GTATCTGTCTTTGATTCCTGCCTCATACCATTATTTATCGCACCTACGTTCAATATTACAG---
q gorGor1.Supercontig_0439211
578899999999999999999999999999999999999999999999999999999999999-99999999999999999999999999999999999999999999999999999999
99999999999999999999999999999999999999999999999999999999999999---
i gorGor1.Supercontig_0439211 N 0 I 21

a score=4190.000000
s hg19.chrM                   187 50 + 16571 AACATACCTACTAAA-GTGTGTTAATTAATTAATGCTTGTAGGACATAATA
s xenTro2.scaffold_19023     2405 51 +  3053 AACAGATGTATTAGGCATGTGTACATTCATTAATGCATGACTGACATAAAA
i xenTro2.scaffold_19023     N 0 C 0
s gorGor1.Supercontig_0439211  205 31 +   616 -------------------GTGTTAATTAATTCATGCTTGTTGGACATAA--
q gorGor1.Supercontig_0439211                -------------------99999999999999999999999999999--
i gorGor1.Supercontig_0439211 I 21 I 99
s galGal3.chrM                 851 44 + 16775 ------CCCGGTAAA-TGGTGCTATTTAGTGAATGCTTGTCGGACATATTT
q galGal3.chrM                               ------FFFFFFFFF-FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
i galGal3.chrM               N 0 C 0

a score=-5291.000000
s hg19.chrM                   237 40 + 16571 -----ATAACAATTGAATGTCTGCACAGCCGC------TTTCCACACAGAC
s turTru1.scaffold_98585    25144 31 - 72340 -----ATTGTAATTATAAACTTGCACA---------------CATATTATC
q turTru1.scaffold_98585                     -----99999999999999999999---------------999999999
i turTru1.scaffold_98585     N 0 C 0
s galGal3.chrM                 895 35 + 16775 -------------TTATCAATTTTCACTTCCTC---TATTTTCTTCACAAA-
q galGal3.chrM                               -------------FFFFFFFFFFFFFFFFFFF---FFFFFFFFFFFFFF-
i galGal3.chrM               C 0 C 0
s xenTro2.scaffold_19023    2456 51 +  3053 ATTTAACTACCATAATGAATTCTCAGCTTTTTACCTATTTTCCACCCGGGG
i xenTro2.scaffold_19023     C 0 C 0
e gorGor1.Supercontig_0439211   236 99 +   616 I
```

Figure C.2: The first 3 MSABs of file "chrM.maf" from the *multiz46way* data set.

"Any intelligent fool can make things bigger, more complex, and more violent. It takes a touch of genius – and a lot of courage to move in the opposite direction.‘"

E. F. Schumacher

D

Multiple alignments data sets

In this Appendix we will present some statistical information regarding the MAF data sets used in this work. There are some important differences among the data set that are worth to describe in order to justify some of the obtained results.

D.1 Statistics regarding the average number of columns and rows of each MSAB

One interesting and important statistic (depicted in Figures. D.1 and D.2) is the size of each MSAB. From Figure D.1, we conclude that, in terms of percentage of occurrence, there is a similar statistical pattern in the four data sets. The maximum number of rows of the first two data sets is 28, because it corresponds to the number of species in these data sets. For the other two data sets, the maximum number of rows is 46 and 100, respectively. Despite the first two data sets have the same number of species and the species are also similar, the statistics regarding the number of rows in each MSAB are different. We can see in Figure D.1 that the *multiz28wayB* has more MSABs with one row, compared to the *multiz28way* (about 1.1% more). There are also small differences (lower than 0.1%) between those two data sets for higher number of rows.

Regarding the number of columns of each MSAB, in Figure D.2 we can find some statistics for the data sets used in this work. For the first three data sets (*multiz28way*, *multiz28wayB*, and *multiz46way*), we observe a similar statistical pattern. However, the last data set (*multiz100way*) has a very different statistical pattern. This difference affects the compression results (in the results section of Chapter 4 we address this subject again). Contrarily to the first three data sets, there is a high percentage of MSABs (about 13.8%) than only have one column in *multiz100way*. These MSABs with few columns could also affect the compression results of the methods proposed in Chapter 4. It is important to note that the range of the number of columns for each MSAB is quite large. For example, in *multiz100way* the maximum number of columns for a MSAB can go up to $11,236,519$. In order to create the charts of Figure D.2, we filtered the statistics in order to contain about 90% of the occurrences. By

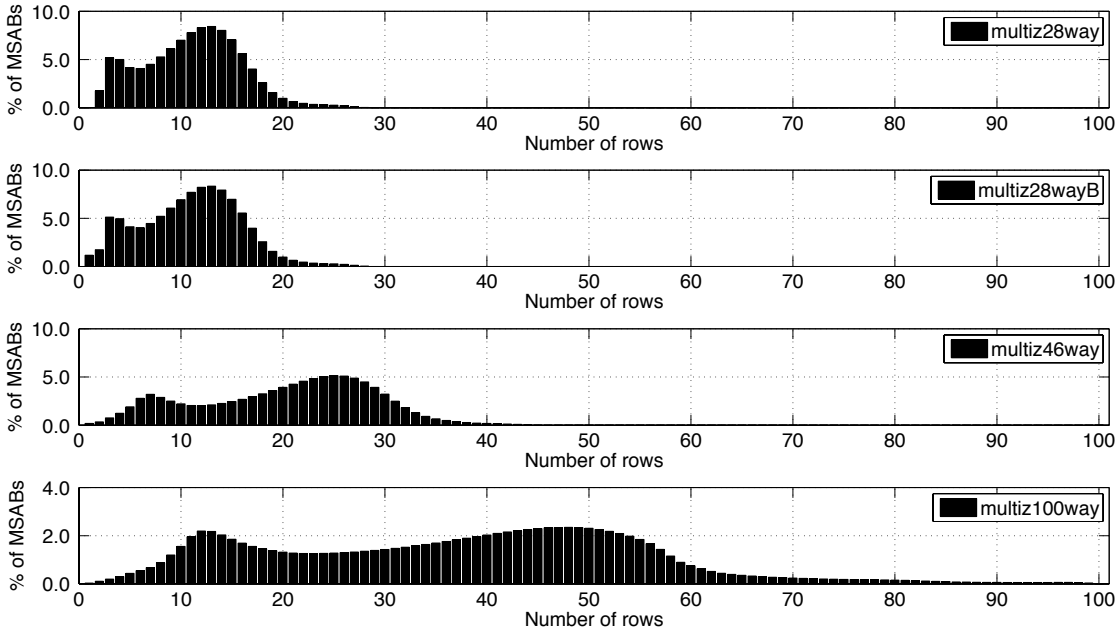


Figure D.1: Number of rows of each MSAB for the *multiz28way*, *multiz28wayB*, *multiz46way*, and *multiz100way* respectively. The results are depicted in terms of percentage of occurrence in order to be more easy to analyze them. Taking into account the maximum number of rows per MSAB allowed for each data set is 28, 28, 46, and 100 respectively, there is a similar statistic pattern between the four data sets.

doing that, we can analyze more easily the obtained results. Regarding the first two data sets, there are again some minor statistical differences, lower than 0.1%.

D.2 Statistics regarding the symbols of 's', 'q', 'i', and 'e' line types

The 's' lines contain the DNA alignments. These alignments are usually DNA bases that include the nucleotides {A, C, G, T}. However, there are some other symbols in these particular data sets which may include some non-ACGT characters (N's and the alignment gap '-'), as well as upper and lower case characters. Figure D.3 shows the statistics for the set of characters that occur in the 's' lines. We can see that the N's characters are the less frequent. On the other hand, the gap symbol ('-') is the most frequent symbol. The frequency of the gap symbol tends to increase when the number of species increases. This is justified by the increased difficulty in creating alignments with more species, leading to more alignment gaps.

Regarding the 'q' lines and according to what we mentioned in Appendix C.4, there are 11 different quality values that may occur. There are also other possible characters like the alignment gap '-' and a dot character '.', that indicates a missing quality value. In Figure D.4, we can find the statistics of each one of the 11 quality values present in the *multiz28wayB* and *multiz46way* data sets. Notice that the other two data sets do not have 'q' lines. According to Figure D.4, the quality value 9 occurs more than 92%, meaning that the majority of the

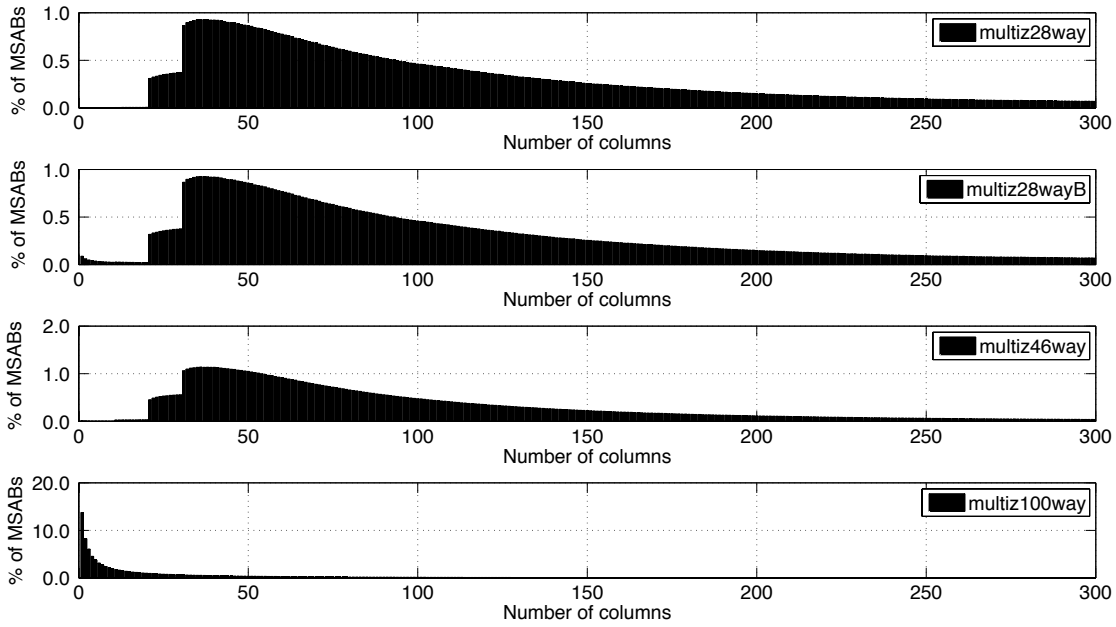


Figure D.2: Number of columns of each MSAB for the *multiz28way*, *multiz28wayB*, *multiz46way*, and *multiz100way* respectively. The results are depicted in term of percentage of occurrence and only cover about 90% of the occurrences in order to be more easy to make some conclusions. The first three data sets have a similar statistical pattern. On the contrary, in the last data set the statistical pattern is quite different when compared to the first three data sets.

alignments have a high quality level (see Table C.1 in Appendix C.4).

The 'i' lines have two fields called left and right status. According to what we have described in Appendix C.5, there are six different status characters. In Figure D.5, we can find the percentage of occurrence of each status character. We did not take into account the left and right status, since we have simply computed the overall percentage (left and right together). The most frequent status character is 'C'. This means that there is an high percentage of contiguous sequences between entire MSABs, i.e., it exists some redundancy between several MSABs.

Finally in the 'e' lines, there is also a status character field, but there are only five possible characters in this case (Figure D.6 depicts their percentage of occurrence). As we can see, the most frequent status character is 'I', which symbolizes the presence of non-aligning bases in the source species between chained MSABs before and after this MSAB.

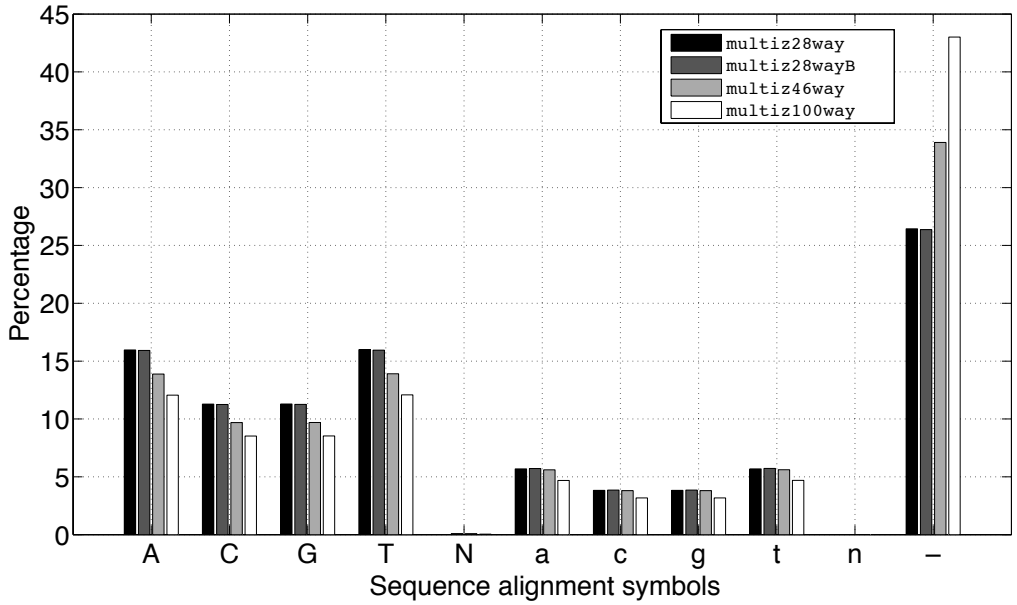


Figure D.3: Statistic of the set of characters that occur in the 's' lines for the four data sets used in this work. Results are presented in percentage.

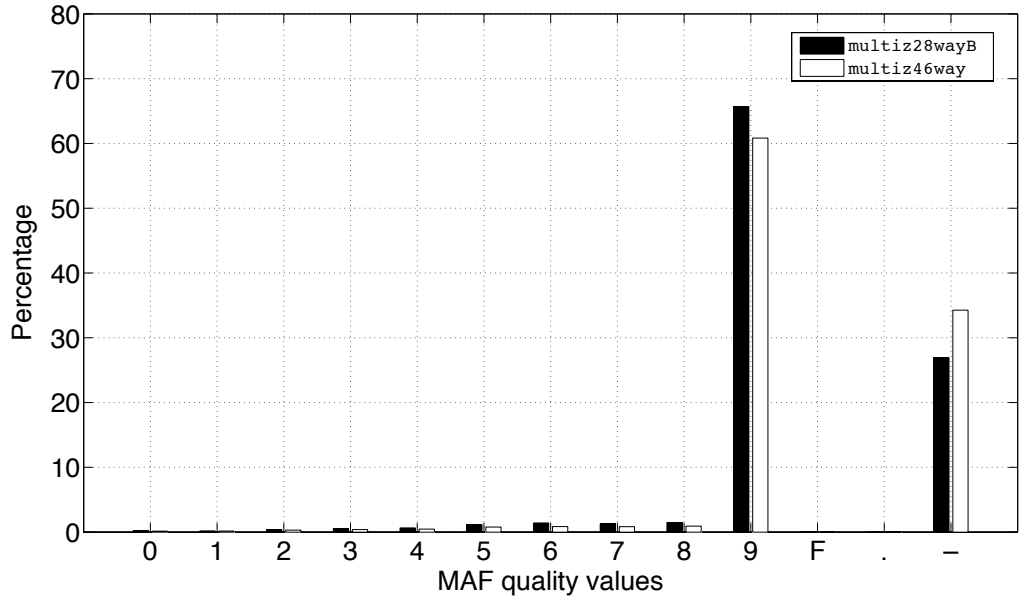


Figure D.4: Percentage of occurrence of each one of the 11 possible quality values in the 'q' lines for the *multiz28wayB* and *multiz46way* data sets. The *multiz28way* and *multiz100way* do not have 'q' lines.

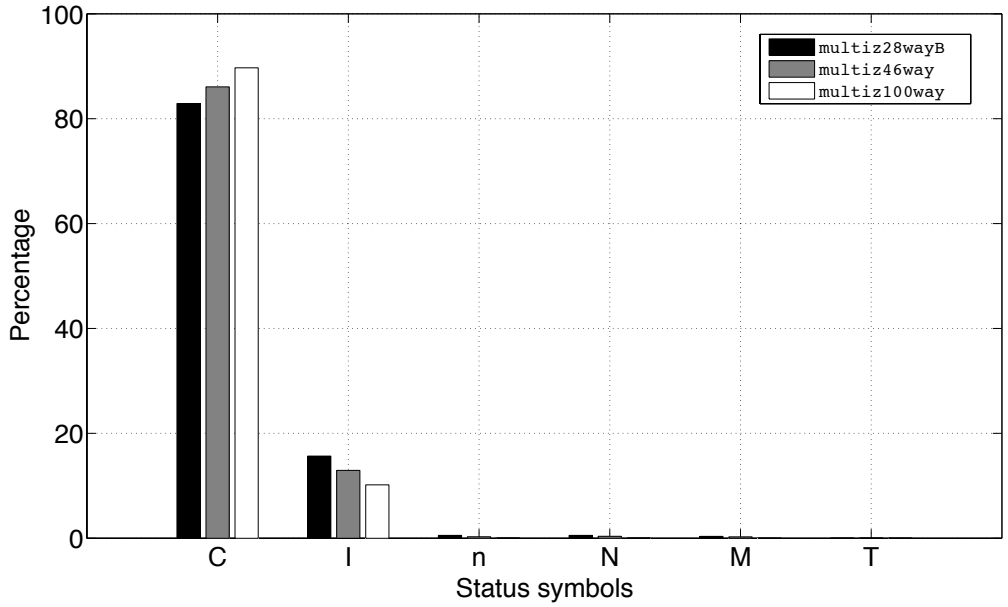


Figure D.5: Percentage of occurrence for each of the six possible status symbols in the 'i' lines for the *multiz28wayB*, *multiz46way*, and *multiz100way* data sets. The *multiz28way* does not have 'i' lines.

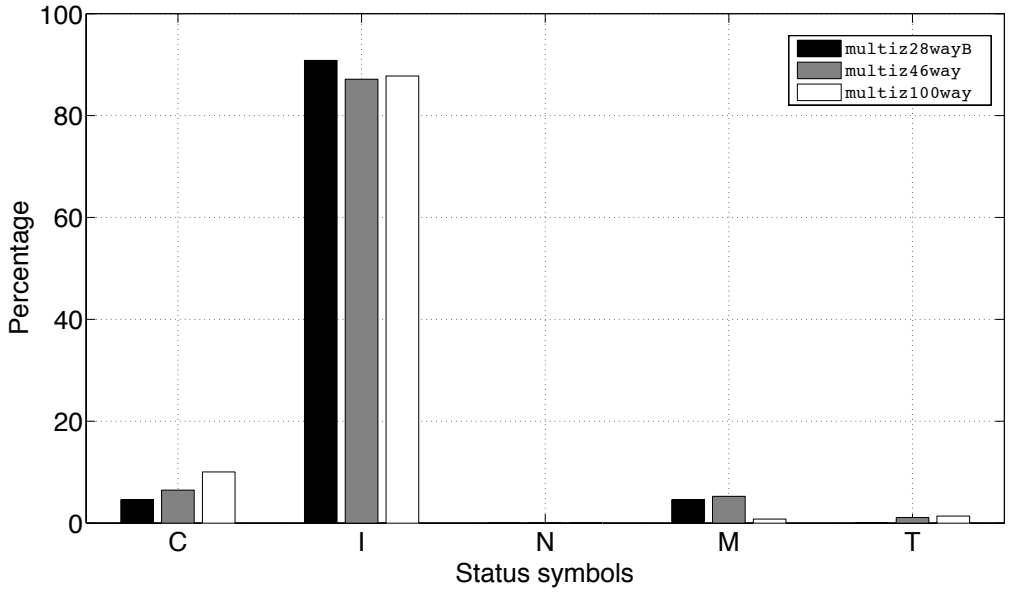


Figure D.6: Percentage of occurrence for each of the five possible status symbols in the 'e' lines for the *multiz28wayB*, *multiz46way*, and *multiz100way* data sets. The *multiz28way* does not have 'e' lines.

"Once you eliminate the impossible, whatever remains, no matter how improbable, must be the truth."

Arthur Conan Doyle

E

Detailed results of the proposed compression methods for MAF files

In this Appendix we intended to present in more detail the compression results obtained for the proposed compression methods and for other general compression methods such as gzip [169], bzip2 [170], ppmd (using the version implemented by the 7-Zip [171] archiver) for each MAF file of the data sets presented in Section 4.1.2 and Appendix D. We also included compression results for the Hanus *et al*. method [161] and for the maf-bgzip tool [164–166].

E.1 Results of the compression algorithm for the MSABs based on a mixture of finite-context models

Table E.1: Performance of several compression methods for the *multiz28way* data set. The results are presented in bits per symbol (bps) which means that the results were computed taking into account the number of bytes of the compressed file divided by total amount of DNA bases and alignment gaps. The result of file "chrM.maf" for the Hanus method [161] was not considered because the authors did not include that file in their results. "CTime" and "DTime" indicate the compression and decompression times in seconds, respectively.

MAF file	**Gzip**	**Bzip2**	**PPMd**	**LZMA**	**Hanus [161]**	**Proposed [13]**
chr1	1.69	1.86	1.81	1.19	1.01	**0.93**
chr2	1.69	1.87	1.81	1.19	1.00	**0.93**
chr3	1.70	1.88	1.81	1.19	1.00	**0.93**
chr4	1.71	1.89	1.82	1.20	1.01	**0.95**
chr5	1.70	1.88	1.81	1.19	1.00	**0.94**
chr6	1.70	1.88	1.81	1.20	1.00	**0.94**
chr7	1.70	1.88	1.81	1.20	1.01	**0.95**
chr8	1.71	1.89	1.82	1.21	1.02	**0.95**
chr9	1.69	1.87	1.81	1.18	1.01	**0.94**
chr10	1.71	1.88	1.81	1.20	1.02	**0.95**
chr11	1.70	1.88	1.81	1.20	1.02	**0.95**
chr12	1.69	1.87	1.81	1.19	1.00	**0.94**
chr13	1.71	1.89	1.81	1.21	1.01	**0.95**
chr14	1.68	1.86	1.81	1.18	1.00	**0.93**
chr15	1.69	1.87	1.81	1.19	1.01	**0.94**
chr16	1.70	1.87	1.81	1.21	1.03	**0.96**
chr17	1.67	1.84	1.79	1.18	1.02	**0.94**
chr18	1.71	1.89	1.82	1.21	1.01	**0.95**
chr19	1.60	1.83	1.79	1.20	1.08	**1.00**
chr20	1.71	1.88	1.81	1.21	1.03	**0.95**
chr21	1.73	1.91	1.82	1.24	1.04	**0.98**
chr22	1.72	1.88	1.81	1.23	1.07	**0.99**
chrX	1.67	1.87	1.83	1.16	0.99	**0.93**
chrY	1.77	1.95	1.90	1.15	1.19	**1.14**
chrM	1.91	1.99	1.92	1.27	–	**1.08**
Total	1.70	1.88	1.81	1.20	1.01	**0.94**
CTime (secs)	4,763	6,413	3,160	69,527	77,899	80,600
DTime (secs)	403	3,422	3,597	890	* 78,691	63,364
The total decoding time was computed without taking into account files "chr15", "chr16", "chr17", "chr19", and "chrM".						

Table E.2: Performance of several compression methods for the *multiz28wayB* data set. The results are presented in bits per symbol (bps) which means that the results were computed taking into account the number of bytes of the compressed file divided by total amount of DNA bases and alignment gaps. Results for the Hanus method [161] were not included, because the method is not able to process the files of this data set.

MAF file	**Gzip**	**Bzip2**	**PPMd**	**LZMA**	**Proposed [13]**
chr1	1.69	1.86	1.81	1.19	**0.94**
chr2	1.69	1.87	1.81	1.19	**0.93**
chr3	1.70	1.88	1.81	1.19	**0.93**
chr4	1.71	1.89	1.82	1.20	**0.95**
chr5	1.70	1.88	1.81	1.19	**0.94**
chr6	1.70	1.88	1.81	1.20	**0.94**
chr7	1.70	1.88	1.81	1.20	**0.95**
chr8	1.71	1.89	1.82	1.21	**0.95**
chr9	1.69	1.87	1.81	1.18	**0.94**
chr10	1.70	1.88	1.81	1.20	**0.95**
chr11	1.70	1.87	1.81	1.20	**0.95**
chr12	1.69	1.87	1.81	1.19	**0.94**
chr13	1.71	1.89	1.81	1.21	**0.95**
chr14	1.69	1.86	1.81	1.18	**0.93**
chr15	1.69	1.87	1.81	1.19	**0.94**
chr16	1.70	1.87	1.81	1.20	**0.96**
chr17	1.67	1.84	1.79	1.18	**0.94**
chr18	1.71	1.89	1.82	1.21	**0.95**
chr19	1.68	1.83	1.79	1.20	**1.01**
chr20	1.71	1.88	1.81	1.21	**0.96**
chr21	1.73	1.91	1.82	1.24	**0.98**
chr22	1.72	1.88	1.81	1.23	**1.00**
chrX	1.67	1.87	1.83	1.15	**0.94**
chrY	1.78	1.94	1.89	1.15	**1.16**
chrM	1.91	1.99	1.92	1.27	**1.08**
Total	1.70	1.87	1.81	1.19	**0.94**
CTime (secs)	4,766	6,186	3,342	70,648	80,880
DTime (secs)	408	3,460	3,619	900	63,358

Table E.3: Performance of several compression methods for the *multiz46way* data set. The results are presented in bits per symbol (bps) which means that the results were computed taking into account the number of bytes of the compressed file divided by total amount of DNA bases and alignment gaps. Results for the Hanus method [161] were not included, because the method is not able to process the files of this data set.

MAF file	**Gzip**	**Bzip2**	**PPMd**	**LZMA**	**Proposed [13]**
chr1	1.39	1.59	1.70	0.94	**0.72**
chr2	1.39	1.59	1.70	0.94	**0.72**
chr3	1.40	1.59	1.70	0.94	**0.72**
chr4	1.41	1.62	1.72	0.95	**0.73**
chr5	1.40	1.60	1.71	0.94	**0.72**
chr6	1.40	1.60	1.70	0.94	**0.72**
chr7	1.40	1.60	1.71	0.94	**0.73**
chr8	1.42	1.62	1.72	0.95	**0.74**
chr9	1.39	1.59	1.70	0.93	**0.72**
chr10	1.41	1.60	1.71	0.95	**0.73**
chr11	1.40	1.60	1.70	0.94	**0.73**
chr12	1.39	1.59	1.71	0.93	**0.72**
chr13	1.42	1.61	1.71	0.95	**0.73**
chr14	1.39	1.58	1.70	0.93	**0.72**
chr15	1.40	1.59	1.71	0.94	**0.73**
chr16	1.41	1.60	1.70	0.95	**0.74**
chr17	1.38	1.57	1.69	0.93	**0.73**
chr18	1.42	1.61	1.71	0.95	**0.73**
chr19	1.41	1.59	1.71	0.96	**0.78**
chr20	1.41	1.61	1.71	0.95	**0.74**
chr21	1.44	1.64	1.72	0.98	**0.76**
chr22	1.43	1.62	1.71	0.98	**0.77**
chrX	1.36	1.59	1.74	0.90	**0.71**
chrY	1.68	1.87	1.87	1.10	**1.04**
chrM	1.76	1.89	1.87	1.14	**0.96**
Total	1.40	1.60	1.71	0.94	**0.73**
CTime (secs)	8,249	12,530	5,864	131,830	167,540
DTime (secs)	783	5,672	6,585	1,587	132,474

Table E.4: Performance of several compression methods for the *multiz100way* data set. The results are presented in bits per symbol (bps) which means that the results were computed taking into account the number of bytes of the compressed file divided by total amount of DNA bases and alignment gaps. Results for the Hanus method [161] were not included, because the method is not able to process the files of this data set.

MAF file	**Gzip**	**Bzip2**	**PPMd**	**LZMA**	**Proposed [13]**
chr1	1.06	1.25	1.54	0.68	**0.52**
chr2	1.06	1.25	1.54	0.68	**0.52**
chr3	1.05	1.25	1.54	0.67	**0.52**
chr4	1.07	1.28	1.56	0.69	**0.53**
chr5	1.06	1.26	1.55	0.68	**0.52**
chr6	1.06	1.26	1.55	0.68	**0.52**
chr7	1.07	1.27	1.56	0.69	**0.53**
chr8	1.08	1.29	1.57	0.69	**0.54**
chr9	1.06	1.26	1.54	0.68	**0.52**
chr10	1.07	1.28	1.56	0.69	**0.53**
chr11	1.06	1.26	1.54	0.68	**0.53**
chr12	1.05	1.25	1.55	0.68	**0.52**
chr13	1.08	1.28	1.56	0.69	**0.53**
chr14	1.05	1.25	1.54	0.67	**0.52**
chr15	1.06	1.26	1.55	0.68	**0.53**
chr16	1.08	1.28	1.56	0.70	**0.54**
chr17	1.05	1.24	1.53	0.68	**0.53**
chr18	1.08	1.28	1.57	0.69	**0.53**
chr19	1.09	1.28	1.55	0.71	**0.57**
chr20	1.08	1.27	1.56	0.69	**0.53**
chr21	1.11	1.32	1.59	0.72	**0.56**
chr22	1.10	1.30	1.56	0.72	**0.56**
chrX	1.03	1.25	1.57	0.66	**0.51**
chrY	1.34	1.57	1.75	0.87	**0.74**
chrM	1.44	1.55	1.67	0.91	**0.73**
Total	1.06	1.26	1.55	0.68	**0.53**
CTime (secs)	13,241	25,924	12,797	246,613	383,907
DTime (secs)	1,683	9,758	14,665	2,832	302,710

E.2 Results for the MAFCO tool

Table E.5: Performance of several compression methods for the *multiz28way* data set including the maf-bgzip tool [164–166], the Hanus *et al.* method [161] and MAFCO [12]. Size is indicated in bytes, whereas the percentages indicate the amount of reduction attained in comparison to gzip. "CTime" and "DTime" indicate the compression and decompression times in seconds, respectively. It was not possible to obtain the decompression time for the Hanus method for files "chr15", "chr16", "chr17", and "chr19" due to the inability to decompress the previous compressed files. The decoder crashed for those files.

MAF file	Original size	Gzip size	Bzip2	PPMd	LZMA	BGZIP	Hanus [161]	MAFCO
chr1	4,022,835,658	861,506,627	8.9%	10.9%	23.0%	-21.2%	46.7%	52.0%
chr2	4,296,853,493	920,575,545	8.7%	10.8%	23.0%	-21.2%	46.9%	52.2%
chr3	3,593,090,866	770,554,934	8.8%	10.9%	23.1%	-21.2%	47.1%	52.3%
chr4	3,073,992,930	666,614,267	8.4%	11.0%	23.2%	-21.0%	46.7%	51.7%
chr5	3,128,634,223	672,397,209	8.5%	10.8%	23.3%	-21.2%	47.0%	52.0%
chr6	2,966,559,585	639,038,581	8.8%	11.2%	22.9%	-21.2%	46.8%	52.1%
chr7	2,571,319,115	554,087,318	8.6%	11.0%	23.0%	-21.1%	46.6%	51.6%
chr8	2,411,928,601	523,333,170	8.7%	11.3%	22.9%	-20.9%	46.8%	51.8%
chr9	2,035,160,128	438,031,722	8.7%	10.9%	23.4%	-21.3%	46.9%	51.8%
chr10	2,311,437,571	498,321,150	8.8%	11.2%	23.0%	-21.0%	46.7%	51.7%
chr11	2,300,561,341	496,531,704	8.8%	11.2%	22.9%	-21.1%	46.7%	51.8%
chr12	2,248,000,862	481,915,693	8.9%	10.9%	23.0%	-21.2%	46.8%	51.9%
chr13	1,633,031,694	353,912,582	8.5%	11.2%	22.7%	-21.1%	46.9%	51.6%
chr14	1,580,086,460	336,436,718	8.7%	10.6%	23.1%	-21.4%	46.9%	51.9%
chr15	1,412,071,845	302,768,849	8.7%	10.7%	23.3%	-21.2%	46.8%	51.6%
chr16	1,349,153,222	292,005,300	9.2%	11.7%	22.7%	-21.0%	46.6%	51.3%
chr17	1,396,039,123	297,292,238	9.3%	11.2%	22.5%	-21.3%	46.5%	51.4%
chr18	1,310,902,389	284,591,318	8.3%	11.1%	22.6%	-20.8%	47.0%	51.7%
chr19	676,876,951	146,384,630	9.3%	11.6%	23.0%	-21.5%	44.4%	48.5%
chr20	1,057,369,122	229,980,184	8.8%	11.4%	22.6%	-20.7%	47.0%	51.6%
chr21	531,345,954	115,778,317	8.3%	11.4%	22.9%	-20.7%	46.5%	50.3%
chr22	514,129,056	112,644,622	8.8%	11.9%	22.4%	-20.5%	45.6%	49.7%
chrX	1,986,854,069	424,471,629	7.7%	9.1%	24.1%	-21.7%	46.9%	51.5%
chrY	102,175,276	24,416,245	3.9%	6.9%	34.0%	-23.2%	42.4%	41.7%
chrM	511,651	123,422	5.9%	9.3%	31.3%	-23.7%	51.4%	41.4%
Total	48,510,921,185	10,443,713,974	8.7%	10.9%	23.1%	-21.1%	46.8%	51.7%
CTime (secs)	–	5,264	7,466	6,972	91,305	6,149	77,899	15,640
DTime (secs)	–	579	4,008	7,505	1,485	684	* 78,691	19,675
* The total decoding time was computed without taking into account the files "chr15", "chr16", "chr17", and "chr19".								

Table E.6: Performance of several compression methods for the *multiz28wayB* data set including the maf-bgzip tool [164–166], the Hanus *et al.* method [161] and MAFCO [12]. Size is indicated in bytes, whereas the percentages indicate the amount of reduction attained in comparison to gzip. "CTime" and "DTime" indicate the compression and decompression times in seconds, respectively.

MAF file	**Original size**	**Gzip size**	**Bzip2**	**PPMd**	**LZMA**	**BGZIP**	**MAFCO**
chr1	9,456,408,599	1,342,681,738	16.8%	13.6%	20.6%	-35.4%	54.5%
chr2	10,035,496,567	1,434,499,715	16.7%	13.7%	20.7%	-35.3%	54.8%
chr3	8,373,240,249	1,201,302,812	16.8%	13.8%	20.8%	-35.3%	54.9%
chr4	7,183,676,250	1,031,352,562	16.4%	14.0%	20.8%	-34.8%	54.4%
chr5	7,254,861,853	1,044,286,909	16.6%	13.9%	21.0%	-35.1%	54.8%
chr6	6,959,446,887	993,429,088	16.7%	13.8%	20.7%	-35.3%	54.6%
chr7	6,016,336,798	859,676,471	16.5%	13.8%	20.7%	-35.0%	54.3%
chr8	5,645,003,774	811,244,637	16.5%	14.1%	20.7%	-34.7%	54.4%
chr9	4,740,179,020	679,117,241	16.5%	13.7%	21.0%	-35.4%	54.4%
chr10	5,442,352,172	773,751,273	16.6%	13.7%	20.6%	-35.4%	54.4%
chr11	5,349,129,368	770,001,734	16.6%	13.9%	20.7%	-34.9%	54.5%
chr12	5,297,984,347	750,607,032	16.7%	13.4%	20.6%	-35.3%	54.3%
chr13	3,830,323,865	548,929,150	16.4%	13.9%	20.5%	-34.9%	54.3%
chr14	3,703,885,642	525,081,467	16.7%	13.5%	20.7%	-35.6%	54.5%
chr15	3,325,339,186	468,823,996	16.5%	13.3%	20.8%	-35.6%	54.1%
chr16	3,193,134,985	451,671,587	16.9%	13.7%	20.5%	-35.3%	53.8%
chr17	3,328,605,396	464,547,555	17.0%	13.1%	20.2%	-36.0%	53.8%
chr18	3,071,045,071	441,484,106	16.5%	13.9%	20.6%	-34.8%	54.4%
chr19	1,593,854,855	223,816,606	16.2%	13.0%	20.1%	-34.5%	51.1%
chr20	2,499,025,345	356,359,925	16.8%	13.9%	20.5%	-34.9%	54.1%
chr21	1,234,442,414	179,686,209	16.3%	14.9%	19.9%	-34.0%	53.8%
chr22	1,216,663,730	173,590,445	16.4%	14.0%	19.9%	-34.2%	52.6%
chrX	4,578,159,108	658,211,900	15.8%	13.0%	21.3%	-34.2%	53.7%
chrY	198,823,596	32,309,311	10.7%	13.0%	29.0%	-28.2%	46.3%
chrM	787,958	150,629	11.3%	13.3%	29.8%	-29.6%	44.2%
Total	113,528,207,035	16,216,614,098	16.6%	13.7%	20.7%	-35.1%	54.3%
CTime(secs)	–	6,064	15,691	12,364	118,780	9,634	21,769
DTime(secs)	–	1,118	7,003	14,273	2,544	1,364	28,155

Table E.7: Performance of several compression methods for the *multiz46way* data set including the maf-bgzip tool [164–166], the Hanus *et al.* method [161] and MAFCO [12]. Size is indicated in bytes, whereas the percentages indicate the amount of reduction attained in comparison to gzip. "CTime" and "DTime" indicate the compression and decompression times in seconds, respectively.

MAF file	**Original size**	**Gzip size**	**Bzip2**	**PPMd**	**LZMA**	**BGZIP**	**MAFCO**
chr1	22,289,900,039	2,671,725,720	18.3%	5.1%	21.0%	-49.3%	57.3%
chr2	23,979,059,901	2,887,410,802	18.2%	5.2%	21.0%	-49.2%	57.7%
chr3	19,951,275,007	2,404,628,324	18.3%	5.2%	21.1%	-49.5%	57.8%
chr4	17,385,458,346	2,097,348,419	18.0%	5.4%	21.1%	-48.8%	57.4%
chr5	17,460,910,896	2,105,904,100	18.2%	5.2%	21.2%	-49.2%	57.7%
chr6	16,846,725,002	2,022,965,508	18.2%	5.2%	21.0%	-49.4%	57.6%
chr7	14,331,680,235	1,722,705,842	18.0%	5.2%	21.0%	-49.0%	57.2%
chr8	13,510,075,640	1,633,975,982	18.1%	5.6%	21.1%	-48.6%	57.4%
chr9	11,226,415,586	1,353,977,957	18.1%	5.1%	21.2%	-49.4%	57.4%
chr10	12,942,820,709	1,552,637,090	18.2%	5.1%	21.0%	-49.3%	57.3%
chr11	12,740,205,486	1,540,212,852	18.2%	5.2%	21.0%	-49.0%	57.4%
chr12	12,566,774,840	1,500,453,504	18.2%	4.8%	20.9%	-49.3%	57.2%
chr13	9,286,984,575	1,120,839,406	18.0%	5.4%	21.0%	-48.9%	57.3%
chr14	8,827,611,676	1,053,920,610	18.3%	4.9%	21.1%	-49.5%	57.5%
chr15	7,807,895,916	930,759,100	18.1%	4.7%	21.0%	-49.5%	57.1%
chr16	7,517,689,886	897,393,497	18.3%	5.3%	20.8%	-49.2%	56.7%
chr17	7,804,679,196	923,889,218	18.4%	4.5%	20.5%	-49.9%	56.7%
chr18	7,441,813,018	895,957,980	18.1%	5.2%	21.0%	-49.1%	57.3%
chr19	3,729,750,952	441,348,418	17.4%	4.6%	20.2%	-47.5%	54.0%
chr20	5,957,764,721	716,197,554	18.3%	5.2%	20.9%	-48.9%	57.1%
chr21	3,030,897,971	370,890,043	17.7%	6.5%	20.4%	-47.6%	56.9%
chr22	2,922,015,039	351,726,805	17.8%	5.6%	20.3%	-48.1%	55.7%
chrX	10,689,220,328	1,280,631,838	17.2%	4.0%	21.4%	-47.6%	56.6%
chrY	330,360,222	46,005,984	12.6%	9.6%	25.1%	-33.9%	48.6%
chrM	1,524,349	258,440	13.6%	11.6%	27.6%	-35.8%	46.7%
Total	270,579,509,536	32,523,764,993	18.1%	5.1%	21.0%	-49.1%	57.3%
CTime(secs)	–	12,630	40,778	28,200	239,483	18,529	45,033
DTime(secs)	–	2,410	17,474	31,387	5,315	2,814	60,653

Table E.8: Performance of several compression methods for the *multiz100way* data set including the maf-bgzip tool [164–166], the Hanus *et al.* method [161] and MAFCO [12]. Size is indicated in bytes, whereas the percentages indicate the amount of reduction attained in comparison to gzip. "CTime" and "DTime" indicate the compression and decompression times in seconds, respectively.

MAF file	**Original size**	**Gzip size**	**Bzip2**	**PPMd**	**LZMA**	**BGZIP**	**MAFCO**
chr1	67,162,951,212	6,030,068,459	21.6%	-8.7%	20.8%	-81.1%	33.8%
chr2	68,025,078,760	6,238,915,887	21.1%	-8.1%	20.8%	-78.9%	34.8%
chr3	56,062,041,375	5,173,469,684	21.1%	-8.1%	20.8%	-78.6%	35.1%
chr4	48,652,269,066	4,493,889,075	20.8%	-7.8%	20.5%	-77.4%	34.7%
chr5	47,845,267,924	4,471,431,587	20.8%	-7.6%	20.8%	-77.3%	35.3%
chr6	48,147,663,279	4,383,126,368	21.3%	-7.9%	20.6%	-79.5%	34.4%
chr7	42,044,759,280	3,799,717,025	21.1%	-8.6%	20.7%	-80.3%	33.8%
chr8	37,791,061,249	3,476,932,152	20.8%	-7.9%	20.4%	-78.2%	34.5%
chr9	32,570,813,239	3,000,403,465	21.1%	-8.1%	20.8%	-78.9%	34.5%
chr10	38,505,619,437	3,454,638,773	21.3%	-8.8%	20.7%	-80.5%	33.6%
chr11	37,659,224,635	3,443,176,075	21.4%	-8.1%	20.7%	-79.4%	34.7%
chr12	37,927,240,766	3,391,695,491	21.6%	-9.0%	20.8%	-80.8%	33.4%
chr13	26,152,598,730	2,390,417,195	20.9%	-8.3%	20.5%	-78.3%	34.5%
chr14	25,559,002,235	2,323,696,435	21.2%	-8.6%	20.9%	-80.3%	33.9%
chr15	24,558,909,404	2,168,462,320	21.6%	-9.6%	21.0%	-83.0%	33.2%
chr16	24,643,657,412	2,148,225,674	21.7%	-9.8%	20.7%	-83.6%	32.0%
chr17	26,536,651,254	2,276,291,369	22.5%	-10.2%	20.7%	-86.1%	31.3%
chr18	20,590,055,969	1,878,798,058	20.7%	-8.4%	20.7%	-78.9%	34.3%
chr19	15,067,437,124	1,260,754,986	23.2%	-10.7%	19.9%	-86.1%	29.6%
chr20	17,791,214,840	1,595,639,895	21.1%	-9.2%	20.8%	-80.7%	33.6%
chr21	8,581,616,277	812,767,967	20.5%	-6.6%	20.5%	-74.9%	33.6%
chr22	10,086,105,053	870,530,942	21.8%	-9.9%	20.6%	-84.5%	31.1%
chrX	30,254,911,291	2,814,652,183	20.6%	-8.9%	20.5%	-74.8%	34.8%
chrY	2,022,479,943	187,856,648	20.7%	-4.9%	20.7%	-67.5%	30.4%
chrM	5,364,307	761,934	22.0%	10.4%	27.6%	-42.8%	44.1%
Total	794,243,994,061	72,086,319,647	21.2%	-8.5%	20.7%	-79.7%	34.1%
CTime(secs)	–	25,977	184,072	80,953	558,021	69,920	88,883
DTime(secs)	–	7,383	51,949	86,091	13,152	8,997	128,509

Table E.9: Performance in terms of coding time of MAFCO [12] using 1, 2, 4, and 8 threads for the *multiz28way* data set. The "CPU time" corresponds to the total CPU time obtained by the *time* command in Linux. The "Optimal CPU time" corresponds to the "CPU time" divided by the number of threads. The speedup was computed by dividing the "Optimal CPU time" for one thread (sequential execution) by the "Optimal CPU time" for n threads. Finally, the efficiency is obtained by dividing the speedup by the number of threads.

Measure	**Encoding**				**Decoding**			
Number of threads	**1**	**2**	**4**	**8**	**1**	**2**	**4**	**8**
CPU time (secs)	15,640	15,960	16,080	16,603	19,675	19,691	20,094	20,391
Optimal CPU time (secs)	15,640	7,980	4,020	2,075	19,675	9,846	5,024	2,549
Speedup	1.00	1.96	3.89	7.54	1.00	2.00	3.92	7.72
Efficiency	1.00	0.98	0.97	0.94	1.00	1.00	0.98	0.96

Table E.10: Performance in terms of coding time of MAFCO [12] using 1, 2, 4, and 8 threads for the *multiz28wayB* data set. The "CPU time" corresponds to the total CPU time obtained by the *time* command in Linux. The "Optimal CPU time" corresponds to the "CPU time" divided by the number of threads. The speedup was computed by dividing the "Optimal CPU time" for one thread (sequential execution) by the "Optimal CPU time" for n threads. Finally, the efficiency is obtained by dividing the speedup by the number of threads.

Measure	**Encoding**				**Decoding**			
Number of threads	**1**	**2**	**4**	**8**	**1**	**2**	**4**	**8**
CPU time (secs)	21,769	21,860	22,142	23,027	28,155	28,376	28,852	29,197
Optimal CPU time (secs)	21,769	10,930	5,536	2,878	28,155	14,188	7,213	3,650
Speedup	1.00	1.99	3.93	7.56	1.00	1.98	3.90	7.71
Efficiency	1.00	1.00	0.98	0.95	1.00	0.99	0.98	0.96

Table E.11: Performance in terms of coding time of MAFCO [12] using 1, 2, 4, and 8 threads for the *multiz46way* data set. The "CPU time" corresponds to the total CPU time obtained by the *time* command in Linux. The "Optimal CPU time" corresponds to the "CPU time" divided by the number of threads. The speedup was computed by dividing the "Optimal CPU time" for one thread (sequential execution) by the "Optimal CPU time" for n threads. Finally, the efficiency is obtained by dividing the speedup by the number of threads.

Measure	**Encoding**				**Decoding**			
Number of threads	**1**	**2**	**4**	**8**	**1**	**2**	**4**	**8**
CPU time (secs)	45,033	45,375	45,930	47,897	60,653	61,139	61,832	62,695
Optimal CPU time (secs)	45,033	2,688	11,482	5,987	60,653	30,570	15,458	7,837
Speedup	1.00	1.98	3.92	7.52	1.00	1.98	3.92	7.74
Efficiency	1.00	0.99	0.98	0.94	1.00	0.99	0.98	0.97

Table E.12: Performance in terms of coding time of MAFCO [12] using 1, 2, 4, and 8 threads for the *multiz100way* data set. The "CPU time" corresponds to the total CPU time obtained by the *time* command in Linux. The "Optimal CPU time" corresponds to the "CPU time" divided by the number of threads. The speedup was computed by dividing the "Optimal CPU time" for one thread (sequential execution) by the "Optimal CPU time" for n threads. Finally, the efficiency is obtained by dividing the speedup by the number of threads.

Measure	**Encoding**				**Decoding**			
Number of threads	**1**	**2**	**4**	**8**	**1**	**2**	**4**	**8**
CPU time (secs)	88,883	89,339	90,935	96,831	128,509	130,545	131,077	131,713
Optimal CPU time (secs)	88,883	44,670	22,734	12,104	128,509	65,273	32,769	16,464
Speedup	1.00	1.99	3.91	7.34	1.00	1.97	3.92	7.81
Efficiency	1.00	0.99	0.98	0.92	1.00	0.98	0.98	0.98

Bibliography

[1] K. Sayood, *Introduction to data compression*, 4th edition, Morgan Kaufmann, October 2012.

[2] D. S. Taubman and M. W. Marcellin, *JPEG2000: image compression fundamentals, standards and practice*, Kluwer Academic Publishers, 2002.

[3] P. Hegde, R. Qi, K. Abernathy, C. Gay, S. Dharap, R. Gaspard, J. Earle-Hughes, E. Snesrud, N. Lee, and J. Quackenbush, "A concise guide to cDNA microarray analysis", *Biotechniques*, vol. 29, no. 3, pp. 548–562, September 2000.

[4] S. K. Moore, "Making chips to probe genes", *IEEE Spectrum*, vol. 38, no. 3, pp. 54–60, March 2001.

[5] S. Satih, N. Chalabi, N. Rabiau, R. Bosviel, L. Fontana, Y.-J. Bignon, and D. J. Bernard-Gallon, "Gene expression profiling of breast cancer cell lines in response to soy isoflavones using a pangenomic microarray approach", *OMICS: A Journal of Integrative Biology*, vol. 14, pp. 231–238, June 2010.

[6] M. S. Giri, M. Nebozhyn, L. Showe, and L. J. Montaner, "Microarray data on gene modulation by HIV-1 in immune cells: 2000-2006", *Journal of Leukocyte Biology*, vol. 80, no. 5, pp. 1031–1043, 2006.

[7] R. C. Hardison, "Conserved noncoding sequences are reliable guides to regulatory elements", *Trends in Genetics*, vol. 16, no. 9, pp. 369–372, September 2000.

[8] A. Siepel and D. Haussler, "Computational identification of evolutionarily conserved exons", in *Proceedings of the Eighth Annual International Conference on Research in Computational Molecular Biology*, ser. RECOMB '04, pp. 177-186, New York, NY, USA, March 2004.

[9] S. S. Gross and M. R. Brent, "Using multiple alignments to improve gene prediction", *Journal of Computational Biology*, vol. 13, no. 2, pp. 379–393, March 2006.

[10] J. S. Pedersen, G. Bejerano, A. Siepel, K. Rosenbloom, K. Lindblad-Toh, E. S. Lander, J. Kent, W. Miller, and D. Haussler, "Identification and classification of conserved RNA secondary structures in the human genome", *PLoS Computational Biology*, vol. 2, no. 4, p. e33, March 2006.

[11] **L. M. O. Matos**, A. J. R. Neves, and A. J. Pinho, "Lossy-to-lossless compression of biomedical images based on image decomposition", in *Applications of Digital Signal Processing through Practical Approach*, Ed. S. Radhakrishnan, InTech, pp. 125–158, October 2015.

[12] **L. M. O. Matos**, A. J. R. Neves, D. Pratas, and A. J. Pinho, "MAFCO: a compression tool for MAF files", *PLoS ONE*, vol. 10, no. 3, pp. e0116082, March 2015.

[13] **L. M. O. Matos**, D. Pratas, and A. J. Pinho, "A compression model for DNA Multiple Sequence Alignment Blocks", *IEEE Transactions on Information Theory*, vol. 59, no. 5, pp. 3189–3198, May 2013.

[14] **L. M. O. Matos**, D. Pratas, and A. J. Pinho, "Compression of whole genome alignments using a mixture of finite-context models", in *Proceedings of International Conference on Image Analysis and Recognition, ICIAR 2012*, ser. LNCS, Eds. A. Campilho and M. Kamel, pub. Springer, vol. 7324, pp. 359–366, Aveiro, Portugal, June 2012.

[15] **L. M. O. Matos**, A. J. R. Neves, and A. J. Pinho, "A rate-distortion study on microarray image compression", in *Proceedings of the 20th Portuguese Conference on Pattern Recognition, RecPad 2014*, Covilhã, Portugal, October 2014.

[16] **L. M. O. Matos**, A. J. R. Neves, and A. J. Pinho, "Compression of microarray images using a binary tree decomposition", in *Proceedings of the 22nd European Signal Processing Conference, EUSIPCO-2014*, pp. 531–535, Lisbon, Portugal, September 2014.

[17] **L. M. O. Matos**, A. J. R. Neves, and A. J. Pinho, "Compression of DNA microarrays using a mixture of finite-context models", in *Proceedings of the 18th Portuguese Conference on Pattern Recognition, RecPad 2012*, Coimbra, Portugal, October 2012.

[18] **L. M. O. Matos**, A. J. R. Neves, and A. J. Pinho, "Lossy-to-lossless compression of microarray images using expectation pixel values", in *Proceedings of the 17th Portuguese Conference on Pattern Recognition, RecPad 2011*, Porto, Portugal, October 2011.

[19] **L. M. O. Matos**, A. J. R. Neves, and A. J. Pinho, "Lossless compression of microarray images based on background/foreground separation", in *Proceedings of the 16th Portuguese Conference on Pattern Recognition, RecPad 2010*, Vila Real, Portugal, October 2010.

[20] ISO/IEC, *Information technology - Coded representation of picture and audio information - progressive bi-level image compression*, International Standard ISO/IEC 11544 and ITU-T Recommendation T.82, March 1993.

[21] H. Hampel, R. B. Arps, C. Chamzas, D. Dellert, D. L. Duttweiler, T. Endoh, W. Equitz, F. Ono, R. Pasco, I. Sebestyen, C. J. Starkey, S. J. Urban, Y. Yamazaki, and T. Yoshida, "Technical features of the JBIG standard for progressive bi-level image compression", *Signal Processing: Image Communication*, vol. 4, no. 2, pp. 103–111, April 1992.

[22] A. N. Netravali and B. G. Haskell, *Digital pictures: representation, compression and standards*, 2nd edition, Plenum, New York, 1995.

[23] D. Salomon, *Data compression - The complete reference*, 4th edition, Springer, 2007.

[24] M. Abdat and M. G. Bellanger, "Combining Gray coding and JBIG for lossless image compression", in *Proceedings of the IEEE International Conference on Image Processing, ICIP-94*, vol. III, pp. 851–855, Austin, TX, November 1994,

[25] B. Fowler, R. Arps, A. E. Gamal, and D. Yang, "Quadtree based JBIG compression", in *Proceedings of the Conference on Data Compression, DCC '95*, pp. 102–111, March 1995.

[26] ISO/IEC, *JBIG2 bi-level image compression standard*, International Standard ISO/IEC 14492 and ITU-T Recommendation T.88, 2000.

[27] F. Ono, W. Rucklidge, R. Arps, and C. Constantinescu, "JBIG2-the ultimate bi-level image coding standard", in *Proceedings of the 2000 International Conference on Image Processing*, vol. 1, pp. 140–143, 2000.

[28] J. Seward, "Portable Network Graphics homepage", http://www.libpng.org/pub/png (last accessed on August 24, 2014).

[29] International Standard ISO/IEC 15948:2004, *Information technology - Computer graphics and image processing Portable Network Graphics (PNG): Functional specification*, 2004.

[30] G. Roelofs, *PNG: The Definitive Guide* (http://www.libpng.org/pub/png/book), Ed. O'Reilly, Greg Roelofs, 2003.

[31] J. Ziv and A. Lempel, "A universal algorithm for sequential data compression", *IEEE Transactions on Information Theory*, vol. 23, pp. 337–343, 1977.

[32] Network Working Group, *RFC 1951: DEFLATE compressed data format Specification*, 2006.

[33] ISO/IEC, *Information technology - Lossless and near-lossless compression of continuous-tone still images*, ISO/IEC 14495-1 and ITU Recommendation T.87, 1999.

[34] ISO/IEC, *Information technology - Lossless and near-lossless compression of continuous-tone still images: extensions*, ISO/IEC 14495-2, 2000.

[35] M. J. Weinberger, G. Seroussi, and G. Sapiro, "The LOCO-I lossless image compression algorithm: principles and standardization into JPEG-LS", *IEEE Transactions on Image Processing*, vol. 9, no. 8, pp. 1309–1324, August 2000.

[36] W. Xiaolin and N. Memon, "CALIC - a context based adaptive lossless image codec", in *1996 IEEE International Conference on Acoustics, Speech, and Signal Processing, ICASSP-96*, vol. 4, pp. 1890–1893, May 1996.

[37] P. G. Howard and J. S. Vitter, "Fast and efficient lossless image compression", in *Proceedings of the Data Compression Conference, DCC-93*, pp. 351–360, Snowbird, Utah, March 1993.

[38] A. Zandi, J. D. Allen, E. L. Schwartz, and M. Boliek, "CREW: Compression with reversible embedded wavelets", in *DCC '95: Data Compression Conference*, pp. 212–221, Los Alamitos, CA, March 1995.

[39] ISO/IEC, *Information technology - JPEG 2000 image coding system*, ISO/IEC International Standard 15444-1, ITU-T Recommendation T.800, 2000.

[40] C. Christopoulos, A. Skodras, and T. Ebrahimi, "The JPEG2000 still image coding system: an overview", *IEEE Transactions on Consumer Electronics*, vol. 46, no. 4, pp. 1103–1127, November 2000.

[41] A. Skodras, C. Christopoulos, and T. Ebrahimi, "The JPEG 2000 still image compression standard", *IEEE Signal Processing Magazine*, vol. 18, no. 5, pp. 36–58, September 2001.

[42] K. R. Rao and Y. Huh, "JPEG 2000", in *Video/Image Processing and Multimedia Communications 4th EURASIP-IEEE Region 8 International Symposium on VIPromCom*, pp. 1–6, 2002.

[43] A. Agarwal, A. H. Rowberg, and Y. Kim, "Fast JPEG 2000 decoder and its use in medical imaging", *IEEE Transactions on Information Technology in Biomedicine*, vol. 7, no. 3, pp. 184–190, September 2003.

[44] D. Taubman, E. Ordentlich, M. Weinberger, G. Seroussi, I. Ueno, and F. Ono, "Embedded block coding in JPEG 2000", in *Proceedings of the IEEE International Conference on Image Processing, ICIP-2000*, vol. II, pp. 33–36, Vancouver, Canada, September 2000.

[45] P. Jorgensen, http://homepage.cs.uiowa.edu/∼jorgen (last accessed on August 2014).

[46] Q. Cai, L. Song, G. Li, and N. Ling, "Lossy and lossless intra coding performance evaluation: HEVC, H.264/AVC, JPEG 2000 and JPEG LS", in *Signal Information Processing Association Annual Summit and Conference (APSIPA ASC), 2012 Asia-Pacific*, pp. 1–9, December 2012.

[47] T. Nguyen and D. Marpe, "Performance analysis of HEVC - based intra coding for still image compression", in *Picture Coding Symposium (PCS), 2012*, pp. 233–236, May 2012.

[48] D. F. Simonea, M. Ouareta, F. Dufauxa, A. G. Tescherb, and T. Ebrahimia, "A comparative study of JPEG2000, AVC/H.264, and HD photo", in *SPIE Optics and Photonics, Applications of Digital Image Processing XXX*, vol. 6696, 2007.

[49] A. Al, B. P. Kudva, S. Babu, D. Sumam, and A. V. Rao, "Quality and complexity comparison of H.264 intra mode with JPEG2000 and JPEG", in *2004 International Conference on Image Processing, ICIP '04*, vol. 1, pp. 525–528, October 2004.

[50] ITU-T and ISO/IEC, *H.264: Advanced video coding for generic audiovisual services*, ITU-T Rec. H.264 and ISO/IEC 14496-14, 2014.

[51] T. Wiegand, G. J. Sullivan, G. Bjøntegaard, and A. Luthra, "Overview of the H.264/AVC video coding standard", *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 7, pp. 560–576, July 2003.

[52] I. E. G. Richardson, *H.264 and MPEG-4 video compression*, John Wiley & Sons, Ltd., 2003.

[53] Y. Ye and M. Karczewicz, "Improved H.264 intra coding based on bi-directional intra prediction, directional transform, and adaptive coefficient scanning", in *15th IEEE International Conference on Image Processing, ICIP 2008*, pp. 2116–2119, October 2008.

[54] G. J. Sullivan, P. N. Topiwala, and A. Luthra, "The H.264/AVC advanced video coding standard: overview and introduction to the fidelity range extensions", in *In SPIE Conference on Applications of Digital Image Processing XXVII*, vol. 5558, pp. 454–474, 2004.

[55] **L. M. O. Matos**, "Study and applications of the H.264 video coding standard", Master's thesis, University of Aveiro, July 2009.

[56] G. J. Sullivan, J. Ohm, W.-J. Han, and T. Wiegand, "Overview of the High Efficiency Video Coding (HEVC) standard", *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, no. 12, pp. 1649–1668, December 2012.

[57] ITU-T and ISO/IEC, *H.265: High Efficiency Video Coding (HEVC)*, ITU-T H.265 and ISO/IEC 23008-2 MPEG-H Part 2, 2013.

[58] J. Ohm, G. J. Sullivan, H. Schwarz, T. K. Tan, and T. Wiegand, "Comparison of the coding efficiency of video coding standards - Including High Efficiency Video Coding (HEVC)", *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, no. 12, pp. 1669–1684, December 2012.

[59] V. Sanchez and J. Bartrina-Rapesta, "Lossless compression of medical images based on HEVC intra coding", in *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 6622–6626, May 2014.

[60] V. Sze, M. Budagavi, and G. J. Sullivan, *High Efficiency Video Coding (HEVC): Algorithms and Architectures*, ser. Integrated Circuits and Systems, Eds. V. Sze, M. Budagavi, and G. J. Sullivan, Springer International Publishing, July 2014.

[61] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*, 2nd edition, Prentice Hall, January 2002.

[62] P. Buonora and F. Liberati, "A format for digital preservation of images: A study on JPEG 2000 file robustness", *D-Lib Magazine*, vol. 14, no. 7/8, August 2008.

[63] Y. Yoo, Y. G. Kwon, and A. Ortega, "Embedded image-domain compression using context models", in *Proceedings of the IEEE International Conference on Image Processing, ICIP-99*, vol. I, pp. 477–481, Kobe, Japan, October 1999.

[64] S. Baase and A. V. Gelder, *Computer Algorithms: Introduction to Design and Analysis*, 3rd edition, Addison Wesley, November 1999.

[65] X. Chen, S. Kwong, and J.-F. Feng, "A new compression scheme for color-quantized images", *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 12, no. 10, pp. 904–908, October 2002.

[66] A. J. Pinho and A. J. R. Neves, "Lossy-to-lossless compression of images based on binary tree decomposition", in *Proceedings of the IEEE International Conference on Image Processing, ICIP-2006*, pp. 2257–2260, Atlanta, GA, October 2006.

[67] A. J. Pinho and A. J. R. Neves, "L-infinity progressive image compression", in *Proceedings of the Picture Coding Symposium, PCS-07*, Lisbon, Portugal, November 2007.

[68] A. J. Pinho and A. J. R. Neves, "Progressive lossless compression of medical images", in *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing, ICASSP-2009*, Taipei, Taiwan, April 2009.

[69] A. J. R. Neves and A. J. Pinho, "Compression of microarray images", in *Signal Processing*, Ed. S. Miron, InTech, pp. 429–448, March 2010.

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

[71] A. J. Pinho, A. J. R. Neves, D. A. Martins, C. A. C. Bastos, and P. J. S. G. Ferreira, "Finite-context models for DNA coding", in *Signal Processing*, Ed. S. Miron, InTech, pp. 117–130, March 2010.

[72] A. J. Pinho, A. J. R. Neves, V. Afreixo, C. A. C. Bastos, and P. J. S. G. Ferreira, "A three-state model for DNA protein-coding regions", *IEEE Transactions on Biomedical Engineering*, vol. 53, no. 11, pp. 2148–2155, November 2006.

[73] T. C. Bell, J. G. Cleary, and I. H. Witten, *Text compression*, Prentice Hall, 1990.

[74] K. Sayood, *Introduction to data compression*, 3rd edition, Morgan Kaufmann, 2006.

[75] G. Lidstone, "Note on the general case of the Bayes-Laplace formula for inductive or a posteriori probabilities", *Transaction of the Faculty of Actuaries*, vol. 8, pp. 182–192, 1920.

[76] P. S. Laplace, *A philosophical essay on probabilities* (translated from the sixth French edition by F. W. Truscott and F. L. Emory, 1902), John Wiley & Sons, New York, 1814.

[77] H. Jeffreys, "An invariant form for the prior probability in estimation problems", *Proceedings of the Royal Society (London) A*, vol. 186, pp. 453–461, 1946.

[78] R. E. Krichevsky and V. K. Trofimov, "The performance of universal encoding", *IEEE Transactions on Information Theory*, vol. 27, no. 2, pp. 199–207, March 1981.

[79] J. Rissanen and G. G. Langdon, Jr., "Arithmetic coding", *IBM Journal of Research and Development*, vol. 23, no. 2, pp. 149–162, 1979.

[80] D. A. Huffman, "A Method for the Construction of Minimum-Redundancy Codes", *Proceedings of the Institute of Radio Engineers*, vol. 40, no. 9, pp. 1098–1101, September 1952.

[81] D. Kambhampati, *Protein Microarray Technology*, Wiley-Backwell, March 2004.

[82] E. M. Southern, "Detection of specific sequences among DNA fragments separated by gel electrophoresis", *Journal of Molecular Biology*, vol. 98, no. 3, pp. 503–517, 1975.

[83] S. P. Fodor, J. L. Read, M. C. Pirrung, L. Stryer, A. T. Lu, and D. Solas, "Light-directed, spatially addressable parallel chemical synthesis", *Science*, vol. 251, no. 4995, pp. 767–773, February 1991.

[84] R. Jörnsten and B. Yu, "Comprestimation: microarray images in abundance", in *Proceedings of the Conference on Information Sciences*, Princeton, NJ, March 2000.

[85] R. Jörnsten, B. Yu, W. Wang, and K. Ramchandran, "Compression of cDNA and inkjet microarray images", in *Proceedings of the IEEE International Conference on Image Processing, ICIP-2002*, vol. 3, pp. 961–964, Rochester, NY, September 2002.

[86] R. Jörnsten, Y. Vardi, and C.-H. Zhang, "On the bitplane compression of microarray images", in *Proceedings of the 4th International Conference on Statistical Data Analysis on the L1-norm and Related Methods*, Ed. Y. Dodge, Neuchâtel, Switzerland, August 2002.

[87] R. Jörnsten, B. Yu, W. Wang, and K. Ramchandran, "Microarray image compression and the effect of compression loss", in *Proceedings of the Workshop on Genomic Signal Processing and Statistics, GENSIPS-2002*, Raleigh, NC, October 2002.

[88] R. Jörnsten and B. Yu, "Compression of cDNA microarray images", in *Proceedings of the IEEE International Symposium on Biomedical Imaging, ISBI-2002*, pp. 38–41, Washington, DC, July 2002.

[89] R. Jörnsten, W. Wang, B. Yu, and K. Ramchandran, "Microarray image compression: SLOCO and the effect of information loss", *Signal Processing*, vol. 83, pp. 859–869, 2003.

[90] J. Hua, Z. Xiong, Q. Wu, and K. Castleman, "Fast segmentation and lossy-to-lossless compression of DNA microarray images", in *Proceedings of the Workshop on Genomic Signal Processing and Statistics, GENSIPS-2002*, Raleigh, NC, October 2002.

[91] J. Hua, Z. Liu, Z. Xiong, Q. Wu, and K. Castleman, "Microarray BASICA: background adjustment, segmentation, image compression and analysis of microarray images", in *Proceedings of the IEEE International Conference on Image Processing, ICIP-2003*, vol. 1, pp. 585–588, Barcelona, Spain, September 2003.

[92] J. Hua, Z. Liu, Z. Xiong, Q. Wu, and K. Castleman, "Microarray BASICA: background adjustment, segmentation, image compression and analysis of microarray images", *EURASIP Journal on Applied Signal Processing*, vol. 2004, no. 1, pp. 92–107, January 2004.

[93] E. R. Dougherty, "An introduction to morphological image processing", *Tutorial texts in optical engineering*, 1992.

[94] N. Faramarzpour, S. Shirani, and J. Bondy, "Lossless DNA microarray image compression", in *Proceedings of the 37th Asilomar Conference on Signals, Systems, and Computers, 2003*, vol. 2, pp. 1501–1504, November 2003.

[95] N. Faramarzpour and S. Shirani, "Lossless and lossy compression of DNA microarray images", in *Proceedings of the Data Compression Conference, DCC-2004*, p. 538, Snowbird, Utah, March 2004.

[96] S. Lonardi and Y. Luo, "Gridding and compression of microarray images", in *Proceedings of the IEEE Computational Systems Bioinformatics Conference, CSB-2004*, Stanford, CA, August 2004.

[97] M. Burrows and D. J. Wheeler, *A block-sorting lossless data compression algorithm*, Digital Systems Research Center, May 1994.

[98] A. Said and W. A. Pearlman, "A new, fast, and efficient image codec based on set partitioning in hierarchical trees", *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 6, no. 3, pp. 243–250, June 1996.

[99] Y. Zhang, R. Parthe, and D. Adjeroh, "Lossless compression of DNA microarray images", in *Proceedings of the IEEE Computational Systems Bioinformatics Conference, CSB-2005*, Stanford, CA, August 2005.

[100] Y. Zhang and D. Adjeroh, "Prediction by partial approximate matching for lossless image compression", in *Proceedings of the Data Compression Conference, DCC-2005*, p. 494, Snowbird, Utah, 2005.

[101] A. Neekabadi, S. Samavi, S. A. Razavi, N. Karimi, and S. Shirani, "Lossless microarray image compression using region based predictors", in *Proceedings of the IEEE International Conference on Image Processing, ICIP-2007*, vol. 2, pp. 349–352, San Antonio, Texas, USA, September 2007.

[102] P. W. Holland and R. E. Welsch, "Robust regression using iteratively reweighted least-squares", *Communications in Statistics - Theory and Methods*, vol. 6, no. 9, pp. 813–827, 1977.

[103] K. Sayood, *Introduction to data compression*, 2nd edition, Morgan Kaufmann, 2000.

[104] S. Battiato and F. Rundo, "A bio-inspired CNN with re-indexing engine for lossless DNA microarray compression and segmentation", in *Proceedings of the IEEE International Conference on Image Processing, ICIP-2009*, vol. 1-6, pp. 1737–1740, Cairo, Egypt, November 2009.

[105] S. Battiato, F. Rundo, and F. Stanco, "Self organization motor maps for color-mapped image re-indexing", *IEEE Transactions on Image Processing*, vol. 16, no. 12, pp. 2905–2915, December 2007.

[106] A. J. R. Neves and A. J. Pinho, "Lossless compression of microarray images", in *Proceedings of the IEEE International Conference on Image Processing, ICIP-2006*, pp. 2505–2508, Atlanta, GA, October 2006.

[107] A. J. R. Neves and A. J. Pinho, "Lossless compression of microarray images using image-dependent finite-context models", *IEEE Transactions on Medical Imaging*, vol. 28, no. 2, pp. 194–201, February 2009.

[108] R. Lukac, K. N. Plataniotis, B. Smolka, and A. N. Venetsanopoulos, "A data-adaptive approach to cDNA microarray image enhancement", in *Proceedings of the 5th International Conference on Computational Science - Volume Part II*, ser. ICCS'05, Springer-Verlag, Berlin, Heidelberg, pp. 886–893, 2005.

[109] B. Smolka and K. N. Plataniotis, "Ultrafast technique of impulsive noise removal with application to microarray image denoising", in *Proceedings of the Second International Conference on Image Analysis and Recognition*, ser. ICIAR'05, Springer-Verlag, pp. 990–997, Berlin, Heidelberg, 2005.

[110] D. Adjeroh, Y. Zhang, and R. Parthe, "On denoising and compression of DNA microarray images", *Pattern Recognition*, vol. 39, pp. 2478–2493, February 2006.

[111] X. Chen and H. Duan, "A vector-based filtering algorithm for microarray image", in *IEEE/ICME International Conference on Complex Medical Engineering, CME 2007*, pp. 794–797, May 2007.

[112] T. J. Peters, R. Smolikova-Wachowiak, and M. P. Wachowiak, "Microarray image compression using a variation of singular value decomposition", in *29th Annual International Conference of the IEEE Engineering in Medicine and Biology Society, EMBS 2007*, pp. 1176–1179, August 2007.

[113] A. Zifan, M. Moradi, and S. Gharibzadeh, "Microarray image enhancement by denoising using decimated and undecimated multiwavelet transforms", *Signal, Image and Video Processing*, vol. 4, no. 2, pp. 177–185, 2010.

[114] M. R. N. Avanaki, A. Aber, and R. Ebrahimpour, "Compression of cDNA microarray images based on pure-fractal and wavelet-fractal techniques", *ICGST International Journal on Graphics, Vision and Image Processing, GVIP*, vol. 11, pp. 43–52, March 2011.

[115] Q. Xu, J. Hua, Z. Xiong, M. L. Bittner, and E. R. Dougherty, "The effect of microarray image compression on expression-based classification", *Signal, Image and Video Processing*, vol. 3, no. 1, pp. 53–61, 2009.

[116] R. Bierman, N. Maniyar, C. Parsons, and R. Singh, "MACE: Lossless compression and analysis of microarray images", in *Proceedings of the 21st Annual ACM Symposium on Applied Computing, SAC2006*, Dijon, France, April 2006.

[117] R. Bierman and R. Singh, "Influence of dictionary size on the lossless compression of microarray images", in *Twentieth IEEE International Symposium on Computer-Based Medical Systems, CBMS '07.*, pp. 237–242, June 2007.

[118] M. Hernández-Cabronero, J. Muñoz-Gómez, I. Blanes, J. Serra-Sagristà, and M. W. Marcellin, "DNA microarray image coding", in *Proceedings of the IEEE International Data Compression Conference, DCC-2012*, pp. 32–41, Snowbird, Utah, April 2012.

[119] M. Hernández-Cabronero, F. Aulí-Llinàs, J. Bartrina-Rapesta, I. Blanes, L. Jiménez-Rodríguez, M. W. Marcellin, J. Muñoz-Gómez, V. Sanchez, J. Serra-Sagristà, and Z. Xu, "Multicomponent compression of DNA microarray images", in *Proceedings of the CEDI Workshop on Multimedia Data Coding and Transmission, WMDCT2012*, September 2012.

[120] B. Koc, Z. Arnavut, and H. Kocak, "Lossless compression of DNA microarray images with inversion coder", in *Proceedings of the Data Compression Conference, DCC-2014*, pp. 411–411, Snowbird, Utah, March 2014.

[121] M. Hernández-Cabronero, I. Blanes, M. W. Marcellin, and J. Serra-Sagristà, "A review of DNA microarray image compression", in *Proceedings of International Conference on Data Compression, Communication and Processing, CCP-2011*, pp. 139–147, June 2011.

[122] M. Hernández-Cabronero, I. Blanes, M. W. Marcellin, and J. Serra-Sagristà, "Standard and specific compression techniques for DNA microarray images", *MDPI Algorithms*, vol. 4, pp. 30–49, 2012.

[123] R. Kothapalli, S. J. Yoder, S. Mane, and T. P. Loughran, Jr., "Microarray results: how accurate are they?" *BMC Bioinformatics*, vol. 3, no. 1, p. 22, 2002.

[124] Y. F. Leung and D. Cavalieri, "Fundamentals of cDNA microarray data analysis", *Trends on Genetics*, vol. 19, no. 11, pp. 649–659, November 2003.

[125] R. Sasik, C. H. Woelk, and J. Corbeil, "Microarray truths and consequences", *Journal of Molecular Endocrinology*, vol. 33, no. 1, pp. 1–9, August 2004.

[126] D. B. Allison, X. Cui, G. P. Page, and M. Sabripour, "Microarray data analysis: from disarray to consolidation and consensus", *Nature Reviews Genetics*, vol. 7, no. 1, pp. 55–65, January 2006.

[127] M. Hernández-Cabronero, V. Sanchez, M. W. Marcellin, and J. Serra-Sagristà, "A distortion metric for the lossy compression of DNA microarray images", in *Proceedings of the IEEE International Data Compression Conference, DCC-2013*, pp. 171–180, Cliff Lodge, Snowbird, Utah, 2013.

[128] A. J. Pinho, A. R. C. Paiva, and A. J. R. Neves, "On the use of standards for microarray lossless image compression", *IEEE Transactions on Biomedical Engineering*, vol. 53, no. 3, pp. 563–566, March 2006.

[129] N. Hurley and S. Rickard, "Comparing Measures of Sparsity", *IEEE Transactions on Information Theory*, vol. 55, no. 10, pp. 4723–4741, October 2009.

[130] D. Zonoobi, A. A. Kassim, and Y. V. Venkatesh, "Gini Index as Sparsity Measure for Signal Reconstruction from Compressive Samples", *IEEE Journal of Selected Topics in Signal Processing*, vol. 5, no. 5, pp. 927–932, September 2011.

[131] Y. Yoo, Y. G. Kwon, and A. Ortega, "Embedded image-domain adaptive compression of simple images", in *Proceedings of the 32nd Asilomar Conference on Signals, Systems, and Computers*, vol. 2, pp. 1256–1260, Pacific Grove, CA, November 1998.

[132] S. Battiato, G. D. Blasi, G. Farinella, G. Gallo, and G. Guarnera, "Ad-hoc segmentation pipeline for microarray image analysis", in *Proceedings of IS&T-SPIE 18th Annual Symposium Electronic Imaging Science and Technology 2006*, pp. 300–311, February 2006.

[133] S. Battiato, G. M. Farinella, G. Gallo, and G. C. Guarnera, "Neurofuzzy segmentation of microarray images", in *19th International Conference on Pattern Recognition, ICPR 2008*, pp. 1–4, December 2008.

[134] V. Uslan and I. Ö. Bucak, "Clustering-based spot segmentation of cDNA microarray images", in *2010 Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, pp. 1828–1831, August 2010.

[135] V. Uslan and I. Ö. Bucak, "Microarray image segmentation using clustering methods", *Mathematical and Computational Applications*, vol. 15, no. 2, pp. 240–247, August 2010.

[136] Z. Y. Li and G. R. Weng, "Segmentation of cDNA microarray spots using k-means clustering algorithm and mathematical morphology", in *International Conference on Information Engineering*, pp. 159–162, January 2011.

[137] I. Rezaeian and L. Rueda, "Sub-grid and spot detection in DNA microarray images using optimal multi-level thresholding", in *Proceedings of the 5th IAPR International Conference on Pattern Recognition in Bioinformatics*, ser. PRIB'10, Springer-Verlag, pp. 277–288, Berlin, Heidelberg, 2010.

[138] N. Karimi, S. Samavi, S. Shirani, and P. Behnamfar, "Segmentation of DNA microarray images using an adaptive graph-based method", *Image Processing, IET*, vol. 4, no. 1, pp. 19–27, February 2010.

[139] E. I. Athanasiadis, D. A. Cavouras, D. T. Glotsos, P. V. Georgiadis, I. K. Kalatzis, and G. C. Nikiforidis, "Segmentation of complementary DNA microarray images by wavelet-based markov random field model", *IEEE Transactions on Information Technology in Biomedicine*, vol. 13, no. 6, pp. 1068–1074, November 2009.

[140] H. Kikuchi, K. Funahashi, and S. Muramatsu, "Simple bit-plane coding for lossless image compression and extended functionalities", in *Proceedings of the Picture Coding Symposium, PCS-09*, pp. 1–4, Chicago, Illinois, USA, May 2009.

[141] H. Kikuchi, R. Abe, and S. Muramatsu, "Simple bitplane coding and its application to multi-functional image compression", *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E95.A, no. 5, pp. 938–951, 2012.

[142] H. Kikuchi, T. Deguchi, and M. Okuda, "Lossless compression of LogLuv32 HDR images by simple bitplane coding", in *Picture Coding Symposium (PCS), 2013*, pp. 265–268, December 2013.

[143] A. J. Pinho and A. J. R. Neves, "A context adaptation model for the compression of images with a reduced number of colors", in *Proceedings of the IEEE International Conference on Image Processing, ICIP-2005*, vol. 2, pp. 738–741, Genova, Italy, September 2005.

[144] T. M. Cover and J. A. Thomas, *Elements of Information Theory*, 2nd edition, Wiley-Interscience, 2006.

[145] T. J. Atherton and D. J. Kerbyson, "Size invariant circle detection", *Image and Vision Computing*, vol. 17, no. 11, pp. 795–803, 1999.

[146] W. Miller, K. Rosenbloom, R. C. Hardison, M. Hou, J. Taylor, B. Raney *et al.*, "28-way vertebrate alignment and conservation track in the UCSC genome browser", *Genome Research*, vol. 17, no. 12, pp. 1797–1808, November 2007.

[147] B. Lewin, *Genes VIII*, Benjamin Cumming, December 2003.

[148] P. J. Hastings, J. R. Lupski, S. M. Rosenberg, and G. Ira, "Mechanisms of change in gene copy number", *Nat Rev Genet*, vol. 10, no. 8, pp. 551–564, 2009.

[149] G. M. Cooper, M. Brudno, E. A. Stone, I. Dubchak, S. Batzoglou, and A. Sidow, "Characterization of evolutionary rates and constraints in three mammalian genomes", *Genome Research*, vol. 14, no. 4, pp. 539–548, April 2004.

[150] M. Blanchette, "Computation and analysis of genomic multi-sequence alignments", *Annual Review of Genomics and Human Genetics*, vol. 8, no. 1, pp. 193–213, May 2007.

[151] V. Cutello, G. Nicosia, M. Pavone, and I. Prizzi, "Protein multiple sequence alignment by hybrid bio-inspired algorithms", *Nucleic Acids Research*, vol. 39, no. 6, pp. 1980–1992, March 2011.

[152] M. R. Aniba, O. Poch, A. Marchler-Bauer, and J. D. Thompson, "AlexSys: a knowledge-based expert system for multiple sequence alignment construction and analysis", *Nucleic Acids Research*, vol. 38, no. 19, pp. 6338–6349, October 2010.

[153] L. Ye and X. Huang, "MAP2: multiple alignment of syntenic genomic sequences", *Nucleic Acids Research*, vol. 33, no. 1, pp. 162–170, January 2005.

[154] M. Blanchette, W. J. Kent, C. Riemer, L. Elnitski, A. F. A. Smit *et al.*, "Aligning multiple genomic sequences with the threaded blockset aligner", *Genome Research*, vol. 14, no. 4, pp. 708–715, April 2004.

[155] N. Bray and L. Pachter, "MAVID: constrained ancestral alignment of multiple sequences", *Genome Research*, vol. 14, no. 4, pp. 693–699, April 2004.

[156] M. Brudno, C. B. Do, G. M. Cooper, M. F. Kim, E. Davydov, NISC Comparative Sequencing Program, E. D. Green, A. Sidow, and S. Batzoglou, "LAGAN and Multi-LAGAN: efficient tools for large-scale multiple alignment of genomic DNA", *Genome Research*, vol. 13, no. 4, pp. 721–731, April 2003.

[157] R. Chenna, H. Sugawara, T. Koike, R. Lopez, T. J. Gibson, D. G. Higgins, and J. D. Thompson, "Multiple sequence alignment with the Clustal series of programs", *Nucleic Acids Research*, vol. 31, no. 13, pp. 3497–3500, July 2003.

[158] P. A. Fujita, B. Rhead, A. S. Zweig, A. S. Hinrichs, D. Karolchik, M. S. Cline, M. Goldman, G. P. Barber, H. Clawson, A. Coelho, M. Diekhans, T. R. Dreszer, B. M. Giardine, R. A. Harte, J. Hillman-Jackson, F. Hsu, V. Kirkup, R. M. Kuhn, K. Learned, C. H. Li, L. R. Meyer, A. Pohl, B. J. Raney, K. R. Rosenbloom, K. E. Smith, D. Haussler, and W. J. Kent, "The UCSC genome browser database: update 2011", *Nucleic Acids Research*, vol. 39, no. Suppl. 1, pp. D876–D882, January 2011.

[159] T. Hubbard, D. Barker, E. Birney, G. Cameron, Y. Chen, L. Clark, T. Cox, J. Cuff, V. Curwen, T. Down, R. Durbin, E. Eyras, J. Gilbert, M. Hammond, L. Huminiecki, A. Kasprzyk, H. Lehvaslaiho, P. Lijnzaad, C. Melsopp, E. Mongin, R. Pettett, M. Pocock, S. Potter, A. Rust, E. Schmidt, S. Searle, G. Slater, J. Smith, W. Spooner, A. Stabenau, J. Stalker, E. Stupka, A. Ureta-Vidal, I. Vastrik, and M. Clamp, "The

Ensembl genome database project", *Nucleic Acids Research*, vol. 30, no. 1, pp. 38–41, January 2002.

[160] P. Hanus, J. Dingel, G. Chalkidis, and J. Hagenauer, "Source coding scheme for multiple sequence alignments", in *Proceedings of the Data Compression Conference, DCC-2009*, pp. 183–192, Snowbird, Utah, March 2009.

[161] P. Hanus, J. Dingel, G. Chalkidis, and J. Hagenauer, "Compression of whole genome alignments", *IEEE Transactions on Information Theory*, vol. 56, no. 2, pp. 696–705, February 2010.

[162] P. Hanus, "Selected communications theoretic aspects in genetics", Ph.D. dissertation, Technische Universität München, Arcisstraße 21, 80333 Mnchen, Germany, July 2010.

[163] J. Felsenstein, "Evolutionary trees from DNA sequences: a maximum likelihood approach", *Journal of Molecular Evolution*, vol. 17, pp. 368–376, 1981.

[164] C. Wheeler and A. Tarasov, "Bioruby-maf", https://github.com/csw/bioruby-maf, https://github.com/csw/bioruby-bgzf (last accessed on April 2014).

[165] R. J. P. Bonnal, J. Aerts, G. Githinji, N. Goto, D. MacLean, C. A. Miller, H. Mishima, M. Pagani, R. Ramirez-Gonzalez, G. Smant, F. Strozzi, R. Syme, R. Vos, T. J. Wennblom, B. J. Woodcroft, T. Katayama, and P. Prins, "Biogem: an effective tool-based approach for scaling up open source software development in bioinformatics", *Bioinformatics*, vol. 28, no. 7, pp. 1035–1037, 2012.

[166] N. Goto, P. Prins, M. Nakao, R. Bonnal, J. Aerts, and T. Katayama, "BioRuby: bioinformatics software for the Ruby programming language", *Bioinformatics*, vol. 26, no. 20, pp. 2617–2619, 2010.

[167] 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, p. e27, February 2012.

[168] A. J. Pinho and D. Pratas, "MFCompress: a compression tool for FASTA and multi-FASTA data", *Bioinformatics*, vol. 30, no. 1, pp. 117–118, January 2014.

[169] J.-L. Gailly and M. Adler, "Gzip home page", http://www.gzip.org (last accessed on August 2014).

[170] J. Seward, "Bzip2 homepage", http://www.gzip.org (last accessed on August 2014).

[171] I. Pavlov, "7-Zip Archiver", http://www.7-zip.org (last accessed on August 2014).

[172] UCSC Genome Bioinformatics, "UCSC FAQ: MAF Format", http://genome.ucsc.edu/FAQ/FAQformat.html#format5 (last accessed on April 2014).

[173] S. Schwartz, J. W. Kent, A. Smit, Z. Zhang, R. Baertsch, R. C. Hardison, D. Haussler, and W. Miller, "HumanMouse Alignments with BLASTZ", *Genome Research*, vol. 13, no. 1, pp. 103–107, 2003.

[174] M. Lab, "BLASTZ alignment program", http://www.bx.psu.edu/miller_lab (last accessed on August 2014).