hochschule mannheim

Comparison of compression tools for biological data and analysis of possible improvements

Gabriel Eichelkraut

Bachelor Thesis

for the acquisition of the academic degree Bachelor of Science (B.Sc.)

Course of Studies: Computer Science

Department of Computer Science

University of Applied Sciences Mannheim

01.12.22

Tutors

Prof. Elena Fimmel, Hochschule Mannheim

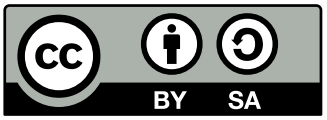
TBD

Eichelkraut, Gabriel:
Comparison of compression tools for biological data and analysis of possible improvements
/ Gabriel Eichelkraut. –
Bachelor Thesis, Mannheim: University of Applied Sciences Mannheim, 2022. 18 pages.

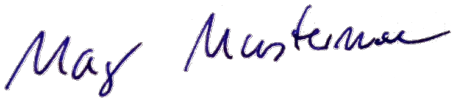
Eichelkraut, Gabriel:
Vergleich von Kompressionswerkzeugen für biologische Daten und Analyse von Verbesserungsmöglichkeiten
/ Gabriel Eichelkraut. –
Bachelor-Thesis, Mannheim: Hochschule Mannheim, 2022. 18 Seiten.

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.



Mannheim, 01.12.22



Gabriel Eichelkraut

Abstract

Comparison of compression tools for biological data and analysis of possible improvements

TBD.

Vergleich von Kompressionswerkzeugen für biologische Daten und Analyse von Verbesserungsmöglichkeiten

TBD.

Contents

Chapter 1

Introduction

Understanding how things in our cosmos work, was and still is a pleasure the human being always wants to fullfill. Getting insights into the rawest form of organic live is possible through storing and studying information, embeded in genetic codes. Since live is complex, there is a lot of information, which requires a lot of memory. With compression tools, the problem of storing information got restricted. Compressed data requires less space and therefore less time to be tranported over networks. This advantage is scaleable and since genetic information needs a lot of storage, even in a compressed state, improvements are welcomed. Since this field is, compared to others, like computer theorie and compression approchaes, relatively new, there is much to discover and new findings are not unusual. From some of this findings, new tools can be developed. They optimally increase two factors: the speed at which data is compressed and the compresseion ratio, meaning the difference between uncompressed and compressed data.

New discoveries in the universal rules of stochastical organisation of genomes might provide a base for new algoriths and therefore new tools or an improvement of existing ones for genome compression. The aim of this work is to analyze the current state of the art for probabilistic compression tools and their algorithms, and ultimately determine whether mentioned discoveries are already used. `might be thrown out due to time limitations` -> If this is not the case, there will be an analysation of how and where this new approach could be imprelented and if it would improve compression methods.

To reach a common ground, the first pages will give the reader a quick overview on the structure of human DNA. There will also be an superficial explanation for some basic terms, used in biology and computer science. The knowledge basis of this work is formed by describing differences in file formats, used to store genome data. In addition to this, a section relevant for compression will follow. This will go through the state of the art in coding theory.
In order to meassure an improvement, first a baseline must be set. Therefore the efficiency and effecitity of suiteable state of the art tools will be meassured. To be as precise as possible, the main part of this work focuses on setting up an environment, picking input data, installing and executing tools and finaly meassuring and documenting the results.
With this information, a static code analysis of mentioned tools follows. This will show where a possible new algorithm or an improvement to an existing one could be implemented. Running a proof of concept implementation under the same conditions and comparing runtime and compression ratio to the defined baseline shows the potential of the new approach for compression with probability algorithms.

Chapter 2

The Structure of the Human Genome and how its Digital Form is Compressed

2.1. Structure of Human DNA

To strengthen the understanding of how and where biological information is stored, this section starts with a quick and general rundown on the structure of any living organism.
All living organisms, like plants and animals, are made of cells (a human body can consist out of several trillion cells) Bianconi et al. 2013. A cell in itself is a living organism; The smalles one possible. It consists out of two layers from which the inner one is called nucleus. The nucleus contains chromosomes and those chromosomes hold the genetic information in form of DNA.

DNA is often seen in the form of a double helix. A double helix consists, as the name suggestes, of two single helix.

Each of them consists of two main components: the Suggar Phosphat backbone, which is not relevant for this work and the Bases. The arrangement of Bases represents the Information stored in the DNA. A base is an organic molecule, they are called Nucleotides WATSON and CRICK 1953.

For this work, nucleotides are the most important parts of the DNA. A Nucleotide can occour in one of four forms: it can be either adenine, thymine, guanine or cytosine. Each of them got a Counterpart with which a bond can be established: adenine can bond with thymine, guanine can bond with cytosine.
From the perspective of an computer scientist: The content of one helix must be

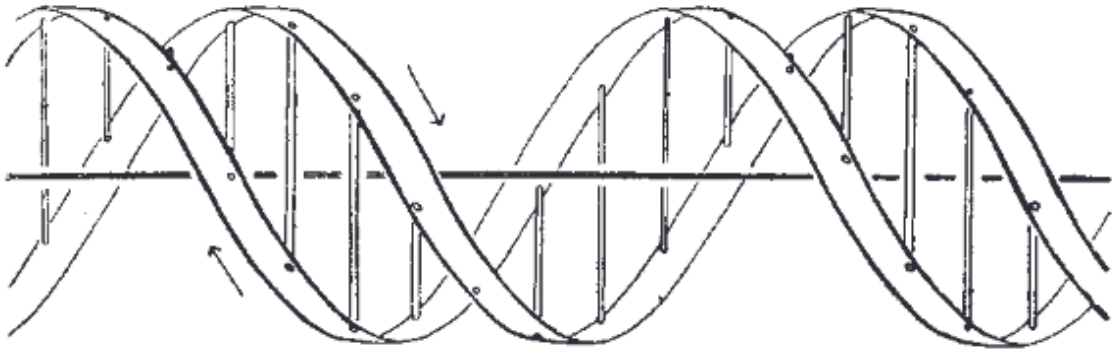


Figure 2.1.: A purely diagrammatic figure of the components DNA is made of. The smaller, inner rods symbolize nucleotide links and the outer ribbons the phosphate-suggar chains WATSON and CRICK 1953.

stored, to persist the full information. In more practical terms: The nucleotides of only one (entire) helix needs to be stored physically, to save the information of the whole DNA because the other half can be determined by "inverting" the stored one. An example would show the counterpart for e.g.: `adenine, guanine, adenine` chain which would be a chain of `thymine, cytosine, thymine`. For the sake of simplicity, one does not write out the full name of each nucleotide, but only its initiat. So the example would change to `AGA` in one Helix, `TCT` in the other.
This representation ist commonly used to store DNA digitally. Depending on the sequencing procedure and other factors, more information is stored and therefore more characters are required but for now 'A', 'C', 'G' and 'T' should be the only concern.

2.2. File Formats used to Store DNA

As described in previous chapters DNA can be represented by a string with the buildingblocks A,T,G and C. Using a common fileformat for saving text would be impractical because the ammount of characters or symbols in the used alphabet, defines how many bits are used to store each single symbol.
The **ascii!** (**ascii!**) table is a characterset, registered in 1975 and to this day still in use to encode texts digitally. For the purpose of communication bigger charactersets replaced **ascii!**. It is still used in situations where storage is short. Storing a single *A* with **ascii!** encoding, requires 8 bit (excluding magic bytes and the bytes used to mark End of File (EOF)) . Since there are at least 2^8 or 128 displayable symbols. The buildingblocks of DNA require a minimum of four letters, so two

bits are needed In most tools, more than four symbols are used. This is due to the complexity in sequencing DNA. It is not 100% preceice, so additional symbols are used to mark nucelotides that could not or could only partly get determined. Further a so called quality score is used to indicate the certainty, for each single nucleotide, that is was sequenced correctly as what was stored.
More common everyday-usage text encodings like unicode require 16 bits per letter. So settling with **ascii!** has improvement capabilitie but is, on the other side, more efficient than using bulkier alternatives like unicode.

Several people and groups have developed different fileformats to store genomes. Unfortunally for this work, there is no defined standard filetype or set of filetypes, therefore one has to gather information by themselve. In order to not go beyond scope, this work will focus only on fileformats that fullfill following criteria:

- the format has reputation, either through a scientific paper, that prooved its superiority to other relevant tools or through a broad ussage of the format.
- the format should no specialize on only one type of DNA.
- the format mainly stores nucleotide seuqences and does not neccesarily include International Union of Pure and Applied Chemistry (IUPAC) codes besides A, C, G and T (Johnson 2010).
- the format is open source. Otherwise improvements can not be tested, without buying the software and/or requesting permission to disassemble and reverse engineer the software or parts of it.
- the compression methode used in the format is based on probabilities.

Information on available formats where gathered through various Internet platforms (Flicek 2022; UCSC - University of California 2022; *Global Alliance for Genomics and Health* 2022). Some common fileformats found:

- BED
- CRAM
- FASTA
- FASTQ
- GFF

- SAM/Binary Alignment Map (BAM)
- twoBit
- VCF

Since methods to store this kind of Data are still in development, there are many more filetypes. The few, mentioned above are used by different organisations and are backed by scientific papers.
Considering the first criteria, by searching through anonymously accesable **ftp!** (**ftp!**) servers, only two formats are used commonly: FASTA or its extension FASTQ and the BAM Format.

2.2.1. FASTQ

Is a text base format for storing sequenced data. It saves nucleotides as letters and in extend to that, the quality values are saved. FASTQ files are split into multiples of four, each four lines contain the informations for one sequence. The exact structure of FASTQ format is as follows: `Line 1: Sequence identifier aka. Title, starting with an @ and an optional description.`
`Line 2: The seuqence consisting of nucleoids, symbolized by A, T, G and C.`
`Line 3: A '+' that functions as a seperator. Optionally followed by content of Line 1.`
`Line 4: quality line(s). consisting of letters and special characters in the ascii! scope.`

The quality values have no fixed type, to name a few there is the sanger format, the solexa format introduced by Solexa Inc., the Illumina and the QUAL format which is generated by the PHRED software. The quality value shows the estimated probability of error in the sequencing process. [...]

2.2.2. Sequence Alignment Map

Sequence Alignment Map (SAM) often seen in its compressed, binary representation BAM with the fileextension `.bam`, is part of the SAMtools package, a uitlity

tool for processing SAM/BAM and CRAM files. The SAM/BAM file is a text based format delimited by TABs. It uses 7-bit US-ASCII, to be precise Charset ANSI X3.4-1968 as defined in RFC1345. The structure is more complex than the one in FASTQ and described best, accompanied by an example:

```
Coor     12345678901234  5678901234567890123456789012345
ref      AGCATGTTAGATAA**GATAGCTGTGCTAGTAGGCAGTCAGCGCCAT

+r001/1        TTAGATAAAGGATA*CTG
+r002         aaaAGATAA*GGATA
+r003       gcctaAGCTAA
+r004                     ATAGCT..............TCAGC
-r003                           ttagctTAGGC
-r001/2                                        CAGCGGCAT
```

Figure 2.2.: SAM/BAM file structure example

2.3. Compression aproaches

The process of compressing data serves the goal to generate an output that is smaller than its input data. In many cases, like in gene compressing, the compression is idealy lossless. This means it is possible for every compressed data, to receive the full information that were available in the origin data, by decompressing it. Lossy compression on the other hand, might excludes parts of data in the compression process, in order to increase the compression rate. The excluded parts are typicaly not necessary to transmit the origin information. This works with certain audio and pictures files or network protocols that are used to transmit video/audio streams live. For DNA a lossless compression is needed. To be preceice a lossy compression is not possible, because there is no unnecessary data. Every nucleotide and its position is needed for the sequenced DNA to be complete. For lossless compression two mayor approaches are known: the dictionary coding and the entropy coding. Both are described in detail below.

2.3.1. Dictionary coding

Dictionary coding, as the name suggest, uses a dictionary to eliminate redundand occurences of strings. Strings are a chain of characters representing a full word or

just a part of it. This is explained shortly for a better understanding of dictionary coding but is of no great relevance to the focus of this work: Looking at the string 'stationary' it might be smart to store 'station' and 'ary' as seperate dictionary enties. Which way is more efficient depents on the text that should get compressed. The dictionary should only store strings that occour in the input data. Also storing a dictionary in addition to the (compressed) input data, would be a waste of resources. Therefore the dicitonary is made out of the input data. Each first occourence is left uncompressed. Every occurence of a string, after the first one, points to its first occurence. Since this 'pointer' needs less space than the string it points to, a decrease in the size is created.

Unfortunally, known implementations like the ones out of LZ Family, do not use probabilities to compress and are therefore out of scope for this work. Since finding repetations and their location might also be improved, this chapter will remain.

2.3.2. Shannons Entropy

The founder of information theory Claude Elwood Shannon described entropy and published it in 1948 (Shannon 1948). In this work he focused on transmitting information. His theorem is applicable to almost any form of communication signal. His findings are not only usefull for forms of information transmition.

Altering this figure shows how it can be used for other technology like compression. The Information source and destination are left unchanged, one has to keep in mind, that it is possible that both are represented by the same phyiscal actor. transmitter and receiver are changed to compression/encoding and decompression/decoding and inbetween ther is no signal but any period of time (Shannon 1948).

Shannons Entropy provides a formular to determine the 'uncertainty of a probability distribution' in a finite field.

$$H(X) := \sum_{x \in X, prob(x) \neq 0} prob(x) \cdot log_2(\frac{1}{prob(x)}) = -\sum_{x \in X, prob(x) \neq 0} prob(x) \cdot log_2(prob(x)).$$

Figure 2.3.: Shannons definition of entropy.

He defined entropy as shown in figure 2.3. Let X be a finite probability space. Then x in X are possible final states of an probability experimen over X. Every state that actually occours, while executing the experiment generates infromation which is meassured in *Bits* with the part of the formular displayed in 2.4(Delfs and Knebl 2007; Shannon 1948):

$$log_2(\frac{1}{prob(x)}) \equiv -log_2(prob(x)).$$

Figure 2.4.: The amount of information measured in bits, in case x is the end state of a probability experiment.

2.3.3. Arithmetic coding

Arithmetic coding is an approach to solve the problem of wasting memeory due to the overhead which is created by encoding certain lenghts of alphabets in binary. Encoding a three-letter alphabet requires at least two bit per letter. Since there are four possilbe combinations with two bits, one combination is not used, so the full potential is not exhausted. Looking at it from another perspective, less storage would be required, if there would be a possibility to encode two letters in the alphabet with one bit and the other one with a two byte combination. This approache is not possible because the letters would not be clearly distinguishable. The two bit letter could be interpreted either as the letter it should represent or as two one bit letters. Arithmetic coding works by translating a n-letter alphabet into a n-letter binary encoding. This is possible by projecting the input text on a floatingpoint number. Every character in the alphabet is represented by an intervall between two floating point numbers in the space between 0.0 and 1.0 (exclusively). This intervall is determined by its distribution in the input text (intervall start) and the the start of the next character (intervall end). To encode a sequence of characters subdividing is used. This means the intervall start of the character is noted, its intervall is split into smaller intervalls with the ratios of the initial intervalls between 0.0 and 1.0. With this, the second character is choosen. This process is repeated for until a intervall for the last character is choosen.
To encode in binary, the binary floating point representation of a number inside the intervall, for the last character is calculated, by using a similar process, described above, called subdividing.

2.3.4. Huffman encoding

The well known Huffman coding, is used in several Tools for genome compression. This subsection should give the reader a general impression how this algorithm works, without going into detail. To use Huffman coding one must first define an alphabet, in our case a four letter alphabet, containing `A, C, G and T` is sufficient. The basic structure is symbolized as a tree. With that, a few simple rules apply to the structure:

- every symbol of the alphabet is one leaf
- the right branch from every not is marked as a 1, the left one is marked as a 0
- every symbol got a weight, the weight is defined by the frequency the symbol occours in the input text
- the less weight a node has, the higher the probability is, that this node is read next in the symbol sequence

The process of compressing starts with the nodes with the lowest weight and buids up to the hightest. Each step adds nodes to a tree where the most left branch should be the shortest and the most right the longest. The most left branch ends with the symbol with the highest weight, therefore occours the most in the input data. Following one path results in the binary representation for one symbol. For an alphabet like the one described above, the binary representation encoded in ASCI is shown here `A -> 01000001, C -> 01000011, G -> 01010100, T -> 00001010`. An imaginary sequence, that has this distribution of characters `A -> 10, C -> 8, G -> 4, T -> 2`. From this information a weighting would be calculated for each character by dividing one by the characters occurence. With a corresponding tree, created from with the weights, the binary data for each symbol would change to this `A -> 0, C -> 11, T -> 100, G -> 101`. Besides the compressed data, the information contained in the tree msut be saved for the decompression process.

misc

Arithmetic Asymmetric numeral systems Golomb Huffman Adaptive Canonical Modified Range Shannon Shannon–Fano Shannon–Fano–Elias Tunstall Unary Universal Exp-Golomb Fibonacci Gamma Levenshtein

2.4. Implementations in Relevant Tools

2.4.1.

2.4.2.

2.4.3.

Chapter 3

Environment and Procedure to Determine the State of The Art Efficiency and Compressionratio of Relevant Tools

Since improvements must be meassured, defining a baseline which would need to be beaten bevorhand is neccesary. Others have dealt with this task several times with common algorithms and tools, and published their results. But since the test case, that need to be build for this work, is rather uncommon in its compilation, the available data are not very usefull. Therefore new testdata must be created.
The goal of this is, to determine a baseline for efficiendy and effectivity of state of the art tools, used to compress DNA. This baseline is set by two important factors:

- Efficiency: **Duration** the Process had run for
- Effectivity: The difference in **Size** between input and compressed data

As a third point, the compliance that files were compressed losslessly should be verified. This is done by comparing the source file to a copy that got compressed and than decompressed again. If one of the two processes should operate lossy, a difference between the source file and the copy a difference in size should be recognizeable.

3.1. Sever specifications and test environment

To be able to recreate this in the future, relevant specifications and the commands that reveiled this information are listed in this section.

Reading from /proc/cpuinfo reveals processor spezifications. Since most of the information displayed in the seven entrys is redundant, only the last entry is shown. Below are relevant specifications listed:

```
cat /proc/cpuinfo
```

- available logical processors: 0 - 7
- vendor: GenuineIntel
- cpu family: 6
- model nr, name: 58, Intel(R) Core(TM) i7-3770 CPU @ 3.40GHz
- microcode: 0x15
- MHz: 2280.874
- cache size: 8192 KB
- cpu cores: 4
- fpu and fpu exception: yes
- address sizes: 36 bits physical, 48 bits virtual

Full CPU secificaiton can be found in appendix.

The installed Random Access Memory (RAM) was offering a total of 16GB with four 4GB instances. For this paper relevant specifications are listed below: Command used to list

```
dmidecode --type 17
```

- Total/Data Width: 64 bits
- Size: 4GB
- Type: DDR3
- Type Detail: Synchronous
- Speed/Configured Memory Speed: 1600 Megatransfers/s

3.2. Operating System and Additionally Installed Packages

To leave the testing environment in a consistent state, not project specific processes running in the background, should be avoided. Due to following circumstances, a current Linux distribution was choosen as a suiteable operating system:

- factors that interfere with a consistent efficiency value should be avoided
- packages, support and user experience should be present to an reasonable ammount

Some backround processes will run while the compression analysis is done. This is owed to the demand of an increasingly complex operating system to execute complex programs. Considering that different tools will be exeuted in this environment, minimizing the backround processes would require building a custom operating system or configuring an existing one to fit this specific use case. The boundary set by the time limitation for this work rejects named alternatives. Choosing **Debian GNU/Linux** version **11** features enough packages to run every tool without spending to much time on the setup.
The graphical user interface and most other optional packages were ommited. The only additional package added in the installation process is the ssh server package. Further a list of packages required by the compression tools were installed. At last, some additional packages were installed for the purpose of simplifying work processes and increasing the safety of the environment.

- installation process: ssh-server
- tool requirements:, git, libhts-dev, autoconf, automake, cmake, make, gcc, perl, zlib1g-dev, libbz2-dev, liblzma-dev, libcurl4-gnutls-dev, libssl-dev, libncurses5-dev, libomp-dev
- additional packages: ufw, rsync, screen, sudo

A complete list of installed packages as well as individual versions can be found in appendix.

3.3. Selection, Receivement, and Preperation of Testdata

Following criteria is requiered for testdata to be appropriate:

- The testfile is in a format that all or at least most of the tools can work with.

- The file is publicly available and free to use.

Since there are multiple open File Transfere Protocol (FTP) servers which distribute a varety of files, finding a suiteable one is rather easy. The ensembl databse featured defined criteria, so the first suiteable were choosen: Homo_sapiens.GRCh38.dna.chromosome. This sample includes over 20 chromosomes, whereby considering the filenames, one chromosome was contained in a single file. After retrieving and unpacking the files, write priviledges on them was withdrawn. So no tool could alter any file contents.
Following tools and parameters where used in this process:

```
\$ wget http://ftp.ensembl.org/pub/release-107/fasta/homo_sapiens/dna/
    Homo_sapiens.GRCh38.dna.chromosome.{2,3,4,5,6,7,8,9,10}.fa.gz
\$ gzip -d ./*
\$ chmod -w ./*
```

The choosen tools are able to handle the File Format for Storing Genomic Data (FASTA) format. However some, like samtools, require to convert FASTA into another format like SAM.
Simply comparing the size is not sufficient, therefore both files are temporarly stripped from metadata and formating, so the raw data of both files can be compared.

Chapter 4

Results and Discussion

Table 4.1.: Multi-column table

Multi-column	
X	X

Table 4.2.: Multi-column table

ratio			
h: tool v: taks	GeCo	samtools to BAM	samtools to CRAM
method/taks	geco	sam -> bam	sam -> cram

Compression time			
	GECO	Samtools	
File 1	235005	15178	
File 2	246503	15211	
File 3	20169	12526	
File 4	194081	11986	
File 5	183878	11436	
File 6	173646	10738	
File 7	159999	9995	
File 8	148288	9142	
File 9	12304	8276	
File 10	134937	8460	
File 11	136299	8508	
File 12	134932	8467	
File 13	999022	6770	
File 14	924753	6309	
File 15	852555	5959	
File 16	827651	5481	
File 17	820814	5151	
File 18	798429	5012	
File 19	586058	3662	
File 20	645884	4025	
File 21	411984	2783	

File sizes			
	Source file	GECO	Samtools CRAM
File 1	253105752	46364770	55769827
File 2	136027438	27411806	32238052
File 3	137338124	27408185	32529673
File 4	135496623	27231126	32166751
File 5	116270459	20696778	23568321
File 6	108827838	18676723	21887811
File 7	103691101	16804782	20493276
File 8	91844042	16005173	19895937
File 9	84645123	15877526	20177456
File 10	81712897	16344067	19310998
File 11	59594634	10488207	14251243
File 12	246230144	49938168	58026123
File 13	65518294	13074402	15510100
File 14	47488540	7900773	9708258
File 15	51665500	41117340	47707954
File 16	201600541	39248276	45564837
File 17	193384854	37133480	43655371
File 18	184563953	35355184	40980906
File 19	173652802	31813760	38417108
File 20	162001796	30104816	34926945
File 21	147557670	23932541	29459829

List of Abbreviations

DNA Deoxyribonucleic Acid
EOF End of File
FASTA File Format for Storing Genomic Data
FASTQ File Format Based on FASTA
FTP File Transfere Protocol
GECO Genome Compressor
IUPAC International Union of Pure and Applied Chemistry
RAM Random Access Memory
SAM Sequence Alignment Map
BAM Binary Alignment Map

List of Tables

List of Figures

Listings

Bibliography

Bianconi, Eva et al. (July 2013). “An estimation of the number of cells in the human body”. In: *Annals of Human Biology* 40.6, pp. 463–471. DOI: 10.3109/03014460.2013.807878.

Delfs, Hans and Helmut Knebl (2007). *Introduction to Cryptography. Principles and Applications (Information Security and Cryptography)*. Springer, p. 368.

Flicek, Paul (Oct. 24, 2022). *ENSEMBL Project*. URL: http://www.ensembl.org/.

Global Alliance for Genomics and Health (Oct. 10, 2022). URL: https://github.com/samtools/hts-specs..

Johnson, Andrew D. (Mar. 2010). “An extended IUPAC nomenclature code for polymorphic nucleic acids”. In: *Bioinformatics* 26.10, pp. 1386–1389. DOI: 10.1093/bioinformatics/btq098.

Shannon, C. E. (July 1948). “A Mathematical Theory of Communication”. In: *Bell System Technical Journal* 27.3, pp. 379–423. DOI: 10.1002/j.1538-7305.1948.tb01338.x.

UCSC - University of California, Santa Cruz (Oct. 28, 2022). *UCSC Genome Browser*. URL: https://genome.ucsc.edu/ (visited on 10/28/2022).

WATSON, J. D. and F. H. C. CRICK (Apr. 1953). “Molecular Structure of Nucleic Acids: A Structure for Deoxyribose Nucleic Acid”. In: *Nature* 171.4356, pp. 737–738. DOI: 10.1038/171737a0.

Appendix A

Erster Anhang: Lange Tabelle

Hier ein Beispiel für einen Anhang. Der Anhang kann genauso in Kapitel und Unterkapitel unterteilt werden, wie die anderen Teile der Arbeit auch.

Table A.1.: Lange Tabelle mit ISO-Ländercodes

Country	A 2	A 3	Number
AFGHANISTAN	AF	AFG	004
ALBANIA	AL	ALB	008
ALGERIA	DZ	DZA	012
AMERICAN SAMOA	AS	ASM	016
ANDORRA	AD	AND	020
ANGOLA	AO	AGO	024
ANGUILLA	AI	AIA	660
ANTARCTICA	AQ	ATA	010
ANTIGUA AND BARBUDA	AG	ATG	028
ARGENTINA	AR	ARG	032
ARMENIA	AM	ARM	051
ARUBA	AW	ABW	533
AUSTRALIA	AU	AUS	036
AUSTRIA	AT	AUT	040
AZERBAIJAN	AZ	AZE	031
BAHAMAS	BS	BHS	044
BAHRAIN	BH	BHR	048
BANGLADESH	BD	BGD	050
BARBADOS	BB	BRB	052
BELARUS	BY	BLR	112
BELGIUM	BE	BEL	056
BELIZE	BZ	BLZ	084
BENIN	BJ	BEN	204
BERMUDA	BM	BMU	060
BHUTAN	BT	BTN	064
BOLIVIA	BO	BOL	068
BOSNIA AND HERZEGOWINA	BA	BIH	070
BOTSWANA	BW	BWA	072
BOUVET ISLAND	BV	BVT	074
BRAZIL	BR	BRA	076
BRITISH INDIAN OCEAN TERRITORY	IO	IOT	086
BRUNEI DARUSSALAM	BN	BRN	096
BULGARIA	BG	BGR	100
BURKINA FASO	BF	BFA	854
BURUNDI	BI	BDI	108
CAMBODIA	KH	KHM	116
CAMEROON	CM	CMR	120

CANADA	CA	CAN	124
CAPE VERDE	CV	CPV	132
CAYMAN ISLANDS	KY	CYM	136
CENTRAL AFRICAN REPUBLIC	CF	CAF	140
CHAD	TD	TCD	148
CHILE	CL	CHL	152
CHINA	CN	CHN	156
CHRISTMAS ISLAND	CX	CXR	162
COCOS (KEELING) ISLANDS	CC	CCK	166
COLOMBIA	CO	COL	170
COMOROS	KM	COM	174
CONGO	CG	COG	178
COOK ISLANDS	CK	COK	184
COSTA RICA	CR	CRI	188
COTE D'IVOIRE	CI	CIV	384
CROATIA (local name: Hrvatska)	HR	HRV	191
CUBA	CU	CUB	192
CYPRUS	CY	CYP	196
CZECH REPUBLIC	CZ	CZE	203
DENMARK	DK	DNK	208
DJIBOUTI	DJ	DJI	262
DOMINICA	DM	DMA	212
DOMINICAN REPUBLIC	DO	DOM	214
EAST TIMOR	TP	TMP	626
ECUADOR	EC	ECU	218
EGYPT	EG	EGY	818
EL SALVADOR	SV	SLV	222
EQUATORIAL GUINEA	GQ	GNQ	226
ERITREA	ER	ERI	232
ESTONIA	EE	EST	233
ETHIOPIA	ET	ETH	210
FALKLAND ISLANDS (MALVINAS)	FK	FLK	238
FAROE ISLANDS	FO	FRO	234
FIJI	FJ	FJI	242

Beachten Sie, dass die Tabelle manchmal erst nach dreimaligem Lauf durch LaTeX richtig angezeigt wird.