BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
FAKULTA ELEKTROTECHNIKY
A KOMUNIKAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF BIOMEDICAL ENGINEERING
ÚSTAV BIOMEDICÍNSKÉHO INŽENÝRSTVÍ

BIOLOGICAL SEQUENCE CLASSIFICATION UTILIZING LOSSLESS DATA COMPRESSION ALGORITHMS
KLASIFIKACE BIOLOGICKÝCH SEKVENCÍ S VYUŽITÍM BEZEZTRÁTOVÉ KOMPRESE

MASTER'S THESIS
DIPLOMOVÁ PRÁCE

AUTHOR — **Bc. Ondřej Kruml**
AUTOR PRÁCE

SUPERVISOR — **Ing. Helena Škutková, Ph.D.**
VEDOUCÍ PRÁCE

BRNO 2016

BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

Master's Thesis

Master's study field **Biomedical Engineering and Bioinformatics**
Department of Biomedical Engineering

Student: Bc. Ondřej Kruml ***ID:*** 147506
Year of study: 2 ***Academic year:*** 2015/16

TITLE OF THESIS:

Biological sequence classification utilizing lossless data compression algorithms

INSTRUCTION:

1) Prepare a literature review of lossless compression algorithm focusing on biological sequence compression. 2) Design an algorithm for genomic and proteomic sequence classification implementing at least three metrics using lossless character encoders. 3) Implement an algorithm for biological sequence classification based on Lempel-Ziv compression algorithm in Matlab. 4) Create software with graphical user interface for sequence classification by at least three metrics using character compression. 5) Assemble set of genomic and proteomic sequences from public databases suitable for statistical testing of developed methods. 6) Perform statistic evaluation of classification results with NCBI taxonomy and standard phylogenetics methods.

RECOMMENDED READING:

[1] OTU H. H. a K. SAYOOD. A new sequence distance measure for phylogenetic tree construction. Bioinformatics, Nov 2003, 19(16), 2122-2130.

[2] GIANCARLO R., D. SCATURRO a F. UTRO. Textual data compression in computational biology: a synopsis. Bioinformatics, July 1, 2009 2009, 25(13), 1575-1586.

Date of project specification: 8.2.2016 ***Deadline for submission:*** 20.5.2016

Leader: Ing. Helena Škutková, Ph.D.
Consultant Master's Thesis:

prof. Ing. Ivo Provazník, Ph.D., *Subject Council chairman*

ABSTRACT

This master thesis is developing the idea of using lossless compression algorithms as a mean of classification of biological sequences. At first an overview of lossless data compression algorithms is presented, based on which the dictionary algorithm created by A. Lempel and J. Ziv in 1976 (LZ77) has been selected. This algorithm, that commonly serves for data compression, has been modified in order to enable the classification of biological sequences. Further modifications have been introduced to enhance the classification capabilities of the algorithm. Several datasets of biological sequences have been collected enabling a correct assessment of the LZ algorithm capability. The algorithm was compared to the classical alignment based methods: Jukes-Cantor, Tamura and Kimura. It has been proven that the algorithm has comparable results in the field of classification of biological sequences and even surpasses the alignment methods in 20% of the datasets. Best results are especially achieved with distant sequences.

KEYWORDS

Data compression, DNA, Lempel-Ziv, LZ77, phylogenetic, classification

ABSTRAKT

Tato diplomová práce se zabývá možností využití bezeztrátových kompresních algoritmů ke klasifikaci biologických sekvencí. Nejdříve je představena literární rešerše o bezeztrátových kompresních algoritmech, která byla využita k výběru slovníkového algoritmu vytvořeného A. Lempelem a J. Zivem v roce 1976 (LZ77). Tento algoritmus je běžně používán k datové kompresi a v předkládané práci byl modifikován tak, aby umožnil klasifikaci biologických sekvencí. K algoritmu byly navrženy další modifikace, které rozvíjí jeho klasifikační možnosti. V průběhu práce byla sestavena sada datasetů biologických sekvencí, která umožnila podrobné testování algoritmu. Algoritmus byl porovnán s klasickými zarovnávacími metodami: Jukes-Cantor, Tamura a Kimura. Bylo ukázáno, že algoritmus dosahuje srovnatelných výsledků v oblasti klasifikace biologických sekvencí a dokonce je u 20% datasetů překonává. Lepší výsledky dosahuje zejména u sekvencí, jež jsou si vzájemně vzdálené.

KLÍČOVÁ SLOVA

Datová komprese, DNA, Lempel-Ziv, LZ77, fylogenetika, klasifikace

KRUML, O. Biological sequence classification utilizing lossless data compression algorithms. Brno: Brno university of technology, Faculty of electrical engineering and communication, 2016 78p. Supervisor Ing. Helena Škutková, Ph.D.

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma Klasifikace biologických sekvencí s využitím bezeztrátové komprese jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

V Brně dne 20. května 2016 .....

(podpis autora)

PODĚKOVÁNÍ

Mé díky patří doktorce Heleně Škutkové, vedoucí mé diplomové práce, za rady a doporučení, které mi během studia udělila. Velice trpělivě a rychle odpovídala na mé dotazy i přes to, že byla sama velice zaneprázdněna.

V Brně dne 20. května 2016 .....

(podpis autora)

INDEX

FIGURE INDEX

TABLE INDEX

INTRODUCTION

This master thesis discusses about the possibility of classifying biological sequences with the use of lossless compression techniques and is focused on the dictionary algorithm created by Abraham Lempel and Jacob Ziv during the year of 1976 – LZ77.

The way of determining biological sequence similarities has been in the past mostly alignment based, meaning the sequences gathered via the Sanger method had first to be aligned between themselves before the actual algorithm of comparison was used. [1, 5] With the evolution of sequencing technologies and the reads becoming longer and longer, these alignment algorithms of computing complexity of, at best, $o(n^2)$ became very time consuming. More important than time, alignment methods are very susceptible to sequence noise. The Next Generation Sequencing methods, which replaced the Sanger method of DNA sequence construction, such as Illumina, Roche 454 etc. are fast at building sequences. On the other hand they sometimes have to compensate their speed with the usage of a reference sequence, to fill the gap created during the sequence reconstruction. These manual additions create sequence noise and therefore the alignment can be inaccurate. [1, 8]

New approaches are being worked on to resolve presented problems creating a new category of sequence analysis – the alignment free methods [32]. These methods are often based on nucleotide (or group of nucleotides) frequencies, such as Yang's method, or compression technics, such as the LZ77 algorithm. [1, 2]

In order to be able to understand the reasoning behind the usage of the LZ77 algorithm for classifying biological sequences, the fundamentals of information theory need to be lay out. The basis of information theory is introduced in the first chapter, both from the classical point of view of Shannon theory and the later Kolmogorov complexity. The following chapters present different approaches of generalized compression algorithms and those specified for biological compression. The last theoretical chapter focuses on compression algorithms that are not only used for data compression but are also specialized in the analysis of biological data – concretely DNA and amino acid sequences.

Compression algorithms can estimate the evolutionary distance between two sequences by calculating the degree of compression of a sequence, when the algorithm is given another sequence as model. If the sequence is well compressed, the model sequence contained most of the information, which means the sequences are evolutionary close. The applied part of this master thesis is divided into two parts. The first part focuses on implementing the LZ77 algorithm in the MATLAB environment and testing it's functionality to classify biological sequences. Several datasets of biological sequences have been collected over the course of this study, enabling a correct assessment of the algorithm capacities. In the second part, based on the gathered results, new methods are implemented to improve the algorithm. The first modification changes the core of the algorithm itself: the dictionary used by the LZ algorithm is modified from dynamic to static – a necessity to avoid the falsification of results by sequence repeats. The rest of the modifications are weighting methods adjusting the outcome of the LZ algorithm.

1 GENERAL COMPRESSION THEORY

The first chapter starts by introducing the basis of compression theory. The information theory and the term of entropy is explained, followed by the Kolmogorov complexity and Huffman coding. The last part of the chapter goes through some examples of lossless compression algorithms.

1.1 Information theory

Information theory is concerned with the transmission of information between a sender and a receiver through a communication channel. As sending data is costly, it is natural that theories dealing with data compression have been developed, leading to two main categories, lossy and lossless data compression. Compression without a loss of information brings questions such as how much information can be encoded in a single word or by how much can a message be compressed so that no information is lost. Two different approaches to describe this phenomenon are reproduced in this paper, entropy via the standard Shannon Theory and Kolmogorov complexity. [3, 4]

1.1.1 Shannon Theory

Information theory as such has been created by the American mathematician Claud Elwood Shannon. His motivation was to describe means of measuring the quantity of information in a symbol (or group of symbols) based on their frequency of occurrence. His research led him to discover a connection between the logarithmic function and the amount of information - the Entropy of information. [4]

The Entropy of an information source is dependent on the statistical nature of the source: the relations of the characters of the alphabet between themselves and their likelihood of occurrence. For the goals of this paper, best is to consider the source to be of the first order, meaning the characters of such a source are statistically independent and each of them has its own probability of occurrence. The entropy of such a source is described by the formula below:

$$H(X) = -\sum_{x \in X} p(x) \log\left(\frac{1}{\mathrm{p(x)}}\right) \qquad (1)$$

In this formula $H(X)$ is the entropy of X where X is a discrete random variable that belongs to a finite alphabet X and $p(x)$ is the probability of appearance of a symbol belonging to the alphabet X. It is determined that if p(x) = 0, the term is considered as 0. The base of the logarithm is usually set as 2, as it enables to measure the quantity of information in bits.[3, 4]

It is possible to deduce from the formula, that if for some $x \in X$, p(x) = 1, then all the others $x \in X$ must have their probability of occurrence p(x) = 0 and thus there is no uncertainty and the entropy is equal to zero. A message sent by such a source wouldn't be of much use.

On the other hand, the highest entropy is achieved in the case that p(x) = 1/N, where N is the cardinality of the alphabet *X*. [3]

For lossless compression reasoning entropy is a very important entity as it provides a limit to the best possible compression of an information source. It is a value that cannot be exceeded and therefore a value that lossless compression algorithms try to reach. [3, 4]

1.2 Kolmogorov complexity

Kolmogorov complexity is a different point of view on the information content than Entropy. It is based on the difficulty of data description, meaning the length of a computational procedure or algorithm that has to be created to describe the data. One of the definitions of Kolmogorov complexity is simply: the length of the code of the shortest program that generates the string it is supposed to generate. It is important to keep in mind, that there is always a finite program that can generate any finite string – the simple print statement e.g. print("ACTG"). [3, 4]

It is interesting to note that the idea behind Kolmogorov's complexity was discovered independently by 3 scientists, Ray Solomonoff, Andrei Kolmogorov and Gregory Chatin at approximately the same time – the 1960s. Andrei Kolmogorov, being a renowned Russian mathematician, got his name attached to the theory. [3]

Since many compression algorithms and especially the DNA compression algorithms involve in some way or other the idea of Kolmogorov complexity and because the Kolmogorov complexity lead Lempel and Ziv to introduce the LZ complexity, discussed later, the basic principles of this theory will be explained in this chapter. [5, 13]

Since the Kolmogorov complexity is defined by the length of the program needed to describe the data, it could be said that it is dependent on the programming language that is going to be used. A number e.g. 1099511627776 that would take 40 bits to be simply printed out, could be also described by 2^{40} if the power function would be defined in the programing language. If said power function wouldn't be described, then the program generating the number would indeed be longer. Fortunately this difference turns out not be so great. It is certainly possible to define the power function (and other functions) in different languages and so the final difference in computing code length would only be a constant depending on the two programs we are comparing as described by the invariance theorem below [3]:

$$C_f(x) - C_g(x) \leq c_{f,g} \tag{2}$$

$C_f(x)$ and $C_g(x)$ are the Kolmogorov complexities of x defined by the programming languages f and g, $c_{f,g}$ is a constant that that depends only on f and g.

As the difference is only a constant, we can say that for large numbers, or large data input, the percentual difference will minimize. [3]

1.2.1 Conditional complexity

The Shannon information theory most commonly works with prefix codes. A prefix code is an organization of code words, that for all code words in the alphabet, none is the prefix of another. This property allows the sender to expedite the information string of words concatenated to the receiver and the message will still be uniquely decoded. Kolmogorov complexity can in general theory work with non-prefix codes, but it leads to complications. For this reason the prefix-free Kolmogorov complexity is introduced as K(x). For the means of sequence comparison it is important to introduce Kolmogorov's conditional complexity, K(x|y). This measure is to be understood as the Kolmogorov complexity of x when y is provided to the program for free. Meaning that if the program discovers similarities between the sequences x and y, it may use this knowledge to save computational time and space, by simply pointing to the provided sequence y. [1,3,5]

1.3 Relation between Shannon theory and Kolmogorov complexity

Even thought that these two theories have very different fundamentals, they are almost in complete agreement in the field of information content. Given a Source of information S that can generate a set of strings x_1, x_2 ... x_n, and their probabilities of occurrence being accordingly $P_1, P_2 \ldots P_n$ its information content can be described by both: $\mathrm{H(X)} = -\sum_{i=1}^{n} P_i \log(P_i)$ and $\overline{K}(S) = \sum_{i=1}^{n} p_i K(x_i)$. It can be proven that the relation between those two values is:

$$\lim_{n\to\infty}\left(\frac{\overline{K}(S)}{H(X)}\right) = 1 \qquad (3)$$

The mathematical proof itself is of no interest for this paper but can be found in [3].

The meaning of the equation (3) is that even though Shannon theory analysis the source based on probabilities of occurrences and Kolmogorov's complexity is based on the analysis of concrete strings, they yield almost same results and therefore can both be used as valuable indicators of information content. [3] This relation is the reason compression algorithms can be used to classify biological sequences, as it will be showed later in this paper.

1.4 Lossless Compression algorithms

In this chapter the most common data compression approaches are introduced. These techniques serve as the basis for genetic data compression and in order to compare compression algorithms the quality of compression has to be estimated. There are two different values that can describe the quality of compression. It can be described by how many bits it takes to encode a character or by the ratio of the resulting compressed file size to the original file size.

1.4.1 Huffman coding

For data to be transferred from the sender to the receiver through a channel, they need to be encoded into a set of 0s and 1s. Huffman coding attributes different code lengths to each symbol of the alphabet according to their probability of occurrence. In order to be able to decode the message flawlessly, the codebook that the Huffman algorithm creates is a prefix code. What is perhaps the most important is that from all the possible ways of creating a codebook for an alphabet, Huffman coding is the algorithm that finds the optimal codebook. The code length will be of minimal possible length. In lossless compression the goal of the code is to reach the entropy of the source, if the coding algorithm doesn't reach entropy, the difference in bits is called redundancy. It can be proven that Huffman code's redundancy is at most $0.086 + p_1$ where p_1 is the probability of the most-common symbol in the alphabet. [3, 4]

The Huffman algorithm follows several steps in order to create its codebook. The knowledge of the probability of occurrence of all characters is needed. In the first step the characters are sorted in descending order of their probabilities. Once this done, the algorithm starts constructing a tree with the symbols of the alphabet being the leaves. The algorithm choses the two symbols with the lowest probabilities and connects them together, creating a new branch. The algorithm continues until all of branches are interconnected and then goes from top to bottom, assigning a 1 to the top edges and a zero to the bottom edges. Following this way from top to bottom, a specific code is associated to each symbol, creating thus the codebook. A complete tree can be seen on figure 1 [3,4].

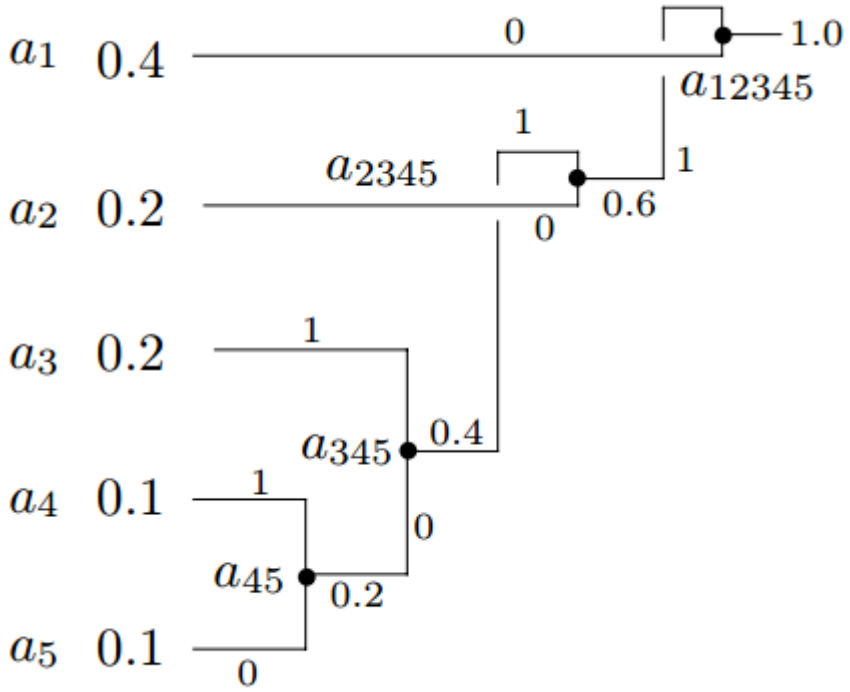

Figure 1: A tree created by the Huffman algorithm. The tree has been taken from [4].

The Huffman coding algorithm is very easy to implement and leads to very good

results. There are several modifications of this algorithm such as adaptive Huffman coding, but these won't be discussed in this text, as they are not related to the topic of this paper.

1.4.2 Dictionary algorithms

Dictionary algorithms store strings of symbols and encode each of these as code words. The dictionary can store these strings either permanently or the content of the dictionary can vary, leading to two categories: static and dynamic dictionaries. Static dictionaries allow additions of code words but no deletions, whereas the dynamic version allows even deletions. [3, 4]

Dictionary based compression methods can be considered to be entropy encoders for very large datasets. With H being the entropy of a string of n symbols, the dictionary should be able to compress the string to nH bits [3]. We can consider a dictionary supposed to compress English texts. Such a dictionary will be static and will contain about half million words. Since it needs to be coded in a binary table, a 19-bit token could seem to be ideal, since $2^{19} = 524288$, but the coding method should also consider the possibility that the word in the input string does not match any of the code words in the dictionary. Such a word has then to be coded manually, character by character. Considering this a better chosen token size would be 20-bits, with the first bit being an indicator of the presence or absence of the input string in the dictionary. In the case that a word is not found in the dictionary, the output will be encoded as 1|7bits to encode the number of characters in the unknown word | 8 bits for each ASCII character in the unknown word. By such encoding a 5 character word would be encoded by 20 bits if defined in the dictionary and 1+7+8*5 = 48bits if not. [4]

In general adaptive dictionaries are the better choice, because undefined words can appear often and bit length of such a word is larger as seen in the previous example.

1.4.3 Lempel-Ziv compression algorithm – LZ77

The LZ77 algorithm is the main topic of this master thesis and it also belongs to the group of dictionary based compression algorithms and will be described in this section. This method focuses on redundancies in the input string, finding parts of strings that match those that were already computed, and uses pointers to code the new incoming words.

LZ77 is a dynamic dictionary algorithm. The dictionary can be empty at first and is filled with 3 instances during the algorithm: a pointer to the location of the last occurrence of the currently analyzed string, the length of the string and the last added character. At each step of the analysis, the algorithm will check if the currently analyzed character of the input string can be found in the already analyzed string. If it is not, then the location pointer is set to 0, the length is set to 0 and the character is saved in the dictionary. Contrariwise if the character is found (possibly at multiple locations) then the next character of the input string is compared with the character following the position of the previous matches. This process repeats until no more matches are found, the location pointer is set up to the location of the first character in the already analyzed string, the length is set to the length of the matching string and lastly a new character is added. [3, 4] An example of the first 7 steps of the algorithm is described below:

Let us consider the sequence A: "LEA-LETS-LEAVE" – the spaces were replaced by – for better visibility

Table 1: Example of the first 7 steps of the LZ algorithm

	Processed string	Incoming string	Dictionary added content
Step 1		LEA-LETS-LEAVE	(0,0,L)
Step 2	L	EA-LETS-LEAVE	(0,0,E)
Step 3	LE	A-LETS-LEAVE	(0,0,A)
Step 4	LEA	-LETS-LEAVE	(0,0,-)
Step 5	LEA-	LETS-LEAVE	(4,2,T)
Step 6	LEA-LET	S-LEAVE	(0,0,S)
Step 7	LEA-LETS	-LEAVE	(5,3,A)

If we consider the production process of the processed string in the example and store the string that has been added at each step, the so called exhaustive history $H_E(A)$ is created. When the algorithm finishes the whole string, the exhaustive history would be [13]:

$H_E(A)$ = L►E►A►-►LET►S►-LEA►V►E

The number of components in the exhaustive is called the LZ complexity, noted *c*(A*)*, and for this example would be 9. The concept of exhaustive history and LZ complexity is the base of the biological comparison algorithm using LZ77 and will be used in the ensuing chapters.

There are several modifications of the LZ algorithm, but for biological data compression, the LZ77 algorithm is the preferred method. [13,15]

1.4.4 The arithmetic method

The Huffman coding is a simple and efficient method that provides the best coding for individual symbols. The problem with this method, however is, it can only assign an integer number of bits to each symbol. According to the definition of Entropy it would be ideal for a word with a probability of occurrence of 0.4 to be assigned a 1.32 bit code ($\log_2(0.4) = 1.32$) – this is something the Huffman coding can't do and so the word will be most probably assigned 1 or 2 bits. The arithmetic coding overcomes this flaw by assigning one code to the entire file. [3, 4, 7]

The arithmetic coding needs to have the input of the probability of occurrence of each symbol in the analyzed alphabet. A cumulative distribution can be created by adding these probabilities one by one. The sum of the distribution should be equal to 1, as it is the sum of probabilities of the symbols in the alphabet. In each step the cumulative distribution will see an increase its value:

$$c(m) = \sum_{s=0}^{m-1} p(s) \tag{4}$$

$c(m)$ stands for the cumulative probability considering m-1 characters and $p(s)$ represents the probabilities of symbols.

The cumulative probability enables to create an interval [0,1], with subintervals belonging to each symbol. The most probable symbol the bigger the interval. The coding system can be described by the following steps:

1) The algorithm starts with the full interval [0,1)
2) The current interval is divided into subintervals proportional to the probabilities.
3) Locating the subinterval belonging to the currently analyzed symbol and defining it as the new current interval and go to step 2. Do until all symbols of the analyzed string are read.
4) The output will be one number that defines the input string unmistakably.

With each step the interval becomes smaller and smaller so it takes more bits to express it, but what is important to realize is that the output is a single number e.g. 0.542642169841. The algorithm is explained on the figure taken from [7]. The example considers the probabilities of occurrences of the nucleotides as follows: A, T = 0.3, C, G = 0.2 the encoded sequence is TCA [3, 7].

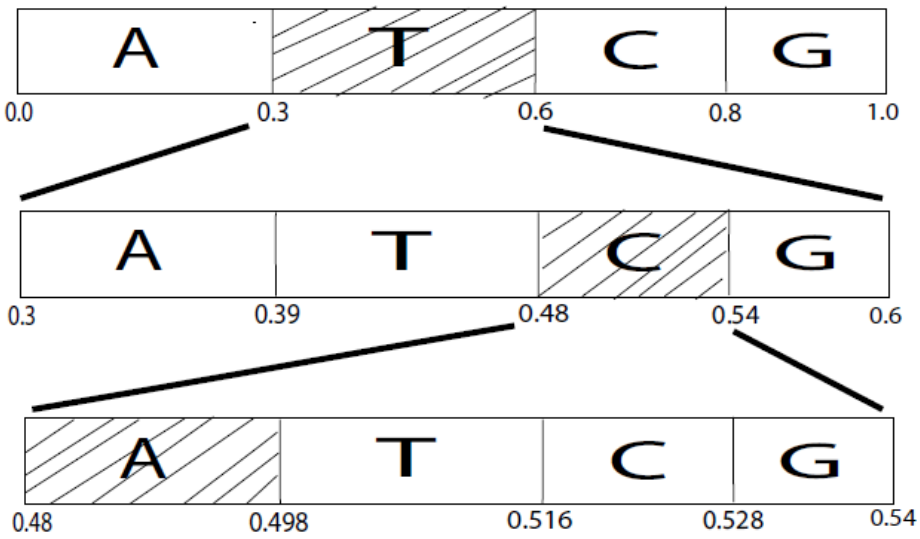

Figure 2: Example of arithmetic coding, image taken from [7]

As it can be seen on the figure 2, the unique interval of TCA is [0.48, 0.498) and so the sequence has been encoded in a unique way achieving lossless compression.

2 COMPRESSION OF BIOLOGICAL SEQUENCES

In the past years, the interest in processing biological sequences has been steadily growing and thus the need of transmitting and storing them has appeared. [6, 13] In general they consider DNA sequences and proteomic sequences. The length of the former varies a lot between species, the length of the human genome is around three billion symbols out of the alphabet {A, C, T, G} while the latter has an average length of only 450 symbols out of the alphabet of more than 20 amino acids [18].

Since the length of the analyzed DNA sequences is in general greater than those of protein sequences this paper considers the DNA string as input for the compressions. Having this in mind, it is important to realize that since the DNA contains only 4 different characters it is possible to encode each character with a 2 bits. Therefore the length of the binary computer code will be only two times longer than the actual sequence. The compression algorithms have to overcome this upper limit. [10]

2.1.1 Horizontal and Vertical mode

Two different approaches can be applied to the compression of biological sequences: horizontal and vertical mode. The horizontal mode can be understood as the compression of one sequence with no additional information. The sequence is compressed using only the information the sequence contains, working with substrings of the sequence itself. The Vertical mode compresses a sequence with an input information of a set of other sequences. In this mode the information contained in the other sequences helps to achieve a higher compression rate. If a long substring is found in one of the model sequences that matches the compressed sequence, the substring can simply be coded as a pointer to the model sequence substring, saving a lot of space. The vertical mode is of high importance for the classification of biological sequences using the compression algorithms, because the amount of similarity between the model sequence and the compressed sequence can be understood as a parallel to the evolutionary distance. [6, 9, 13]

2.1.2 Expert Model

The Expert Model (XM), developed by *Cao et al.* is a compression algorithm based on arithmetic encoding and Markov models. The arithmetic encoding method has been described in the previous chapter. A unique interval is attributed to the sequence based on the probability of occurrence of the next character. When only the arithmetic method is used the compression for genomic data is about 2 bits per character, which is unsatisfactory. For that reason *Cao et al.* include Markov models into the equation. In general, Markov models allow predicting the next state of a model, based on the current state. For the XM algorithm it means predicting the next character probability on the base of previous characters. The arithmetic compression using the XM model will have different interval ranges at each step. When the algorithm detects that an Adenine follows another Adenine, the probability intervals for the next nucleotides could be e.g.

A = 0.2, C = 0.35, G = 0.3, T = 0.15, meanwhile if an Adenine follows a Cytosine the intervals for the next nucleotide could be A = 0.3, C = 0.35, G = 0.2, T = 0.15. The attribution of these specific intervals is at each step calculated by the so called Expert Models. The algorithm starts with a population of Expert Models which are basically Markov models. Each expert has his own estimate for the probability distribution of the next character. These estimates are combined and are given to the arithmetic coder. The expert models also receive different weights based on their accuracy in estimating probability in the previous steps. On human genome the XM algorithm achieves a compression of 1.75 bits per character. [6, 7, 11, 16]

2.1.3 Biocompress, Biocompress-2, cFact

The Biocompress algorithms have been invented by Grumbach S. and Tahi F. and are based on two typical characteristics of the genome: tandem repeats and complementary palindromes. A tandem repeat is a short sequence of nucleotides that is repeated numerous times. The repeats are concatenated to each other. An example of a tandem sequence would be TAGTTTAGTTTAGTTTAGTT – the sequence TAGTT is repeated four times with the repeats being adjacent. Complementary palindromes are sequences of a certain length that match their reversed transcripts. For example TAGTTAACTA, when transcribed, leads to ATCAATTGAT, which matches the original sequence when reversed. The complementary palindromes can lead to hairpin structures of the DNA. Because these two features are common in the DNA, it is advantageous to encode them through a dictionary. The Biocompress methods are LZ77 based. First the maximum length of the tandems and palindrome is established. Then a complete 4-ary tree (there are 4 nucleotides) is formed that allows mapping the presence of palindromes and tandems. The difference between Biocompress and Biocompress-2 is on how the algorithms handle the parts of the sequences that do not contain repetitions. Biocompress simply codes the nucleotides using two bits per base while Biocompress-2 uses an arithmetic encoder. These two methods achieve up to 32% compression rate but only on the regions that are rich in described repeats. [3, 8, 9]

The research of the Biocompress algorithm led to the cFact algorithm. The approach is similar to its predecessor, but the maximum length of the repeats doesn't have to be input. The algorithm first constructs a suffix tree which allows discovering the longest palindrome and tandem repeat. The sequence is then encoding using the LZ77 algorithm and the non-repeat regions are encoded by two bits per base. [3, 8, 9]

2.1.4 Gencompress

The Gencompress algorithm by *Chen et al.* is also a dictionary based method that focuses on repeats in the human genome. The repeats do not have to be exact matches. This method comes from the fact that the human genome is rich with repeats that have a small percentage of differences. Most commonly these differences, called also edit operations, are only single nucleotide replacements, but insertions and deletions can also occur. The Gencompress algorithm finds all the approximate matches in LZ77 style, with a fixed maximum of differences (mutations). The algorithm searches the already compressed part for the longest x-nucleotide difference sequence coming up in the uncompressed part. The location and the length of the discovered string are saved, as well as the pointer to the mutation and the description of the mutation. [3, 8, 10] The

compression ratio can attain the value of 44%. [3]

If more than one mutation between the sequences is allowed the following problem occurs: The difference between a sequence ACTGT and ACAGT can be described as one replacement of the third character T to A or as one insertion and one deletion:

```
ACTGT
ACAGT
```

Or

```
ACT-GT
AC-AGT
```

With these possibilities the number of approximate matches could get out of hand and the algorithm could take a very long time to compute. For this reason a threshold is introduced, which limits the length of the compared strings and the number of edit operations allowed. For DNA the best time/compression results are length – 12 and maximum 3 edit operations. [10]

2.1.5 COMRAD – COMpression using RedundAncy of Dna

COMRAD is a vertical compression algorithm – meaning it uses a set of sequences to encode a concrete sequence. As it was stated, a single sequence possesses redundancies in the form of tandem repeats and complementary palindromes. The third type of genome redundancy is the redundancy between sequences. There is a high similarity in between species and even a higher similarity between individuals of one species. By analyzing the set of sequences it is possible to discover long strings, reaching even thousands of bases that are common to the majority of the sequences. The sequences may not be exact matches as the influence of evolution allows for mutations, which the COMRAD algorithm accounts for. The COMRAD algorithm is basically a copy of the RAY algorithm, except it is modified for DNA sequences. For that reason the RAY algorithm is presented below [11]:

The algorithm is divided into 4 steps that repeat until a terminating condition is reached. In the first step the frequencies of all adjacent symbols (words) of determined length is evaluated and even the overlapping occurrences are counted. During the second step the goal is to discover the word with the highest non-overlapping frequencies. The most frequent word from the first pass is chosen and the non-overlapping occurrences are counted for this word. If this new frequency doesn't drop below the word with the second highest overlapping frequency, the process is ended and the first word is chosen to encode the sequence. Else the non-overlapping frequency of the second highest word from the first step is counted and the process is repeated until the word with the highest non-overlapping frequency is determined. Once the word established, it is replaced by a symbol that doesn't belong to the alphabet of the analyzed string at every position of its occurrence. In the fourth and last step the word frequencies are updated to the new alphabet (since a new symbol has been added) and the algorithm repeats the steps 2-4 until the termination condition is reached. The termination condition is often chosen to be the moment when none of the words has the frequency of occurrence higher than 2. [11]

An example of the algorithm follows. Let us consider the input string "ACTCTACT" and the length of the words being 2.

Step 1) The overlapping frequencies are calculated: AC = 2; CT = 3; TC = 1; TA = 1;

Step 2) The highest frequency possess the word CT and it can be seen its non-overlapping frequency is still the highest.

Step 3) A new character is attributed to the word: x = CT and the string is modified: "AxxAx"

Step 4) The frequency occurrences are recalculated: Ax = 2; xx = 1; xA = 1;

Depending on the termination condition the algorithm would stop now or continue again at Step 2.

The COMRAD algorithm makes some modifications so that it is more suitable for use on DNA sequences. Because the DNA dataset, COMRAD works with, is a huge input, the minimal length of the word in the first iteration is set to 16 concrete nucleotides. The reasoning behind this step is that 16 nucleotides long words occur with a reasonable frequency in a large DNA collection and so by this step the algorithm saves numerous iterations. The second large modification is to account in the complementary palindromes. Other modifications handle sequence recognition when an evolutionary modification has affected the sequence. [11] This method achieves very different compression values depending on the dataset but even though the compression rate may vary the best published results achieve 0.04 bits per base when comparing around a thousand sequences of the human chromosome 20. [11]

2.2 Compression algorithms for classification of biological sequences

In this last theoretical chapter two methods that use compression algorithms for biological sequences classification are introduced. They are both based on the Kolmogorov complexity.

2.2.1 Universal Similarity Metric

The Universal Similarity Metric (USM) method is based on the Kolmogorov complexity. As stated in chapter 1.2, we can denote the conditional Kolmogorov complexity of a string x given y as K(x|y). This conditional complexity can be considered as a measure describing the distance between x and y. Unfortunately the Kolmogorov complexity isn't something computable therefore in order to establish K(x|y) an approximation needs to be used. [3, 4, 5]

As shown in the chapter 1.3, there is a close relation (limitedly equivalence for large strings) between the entropy of the information source and Kolmogorov complexity. This means that evaluating the entropy of the information source can provide an accurate estimate of the Kolmogorov complexity. Furthermore it can be proven that the conditional Kolmogorov complexity of x given y is the same as the Kolmogorov complexity of x concatenated with y up to a logarithmic precision.[5]

Therefore it is possible to estimate the Kolmogorov complexity by calculating the compression rate of the concatenated sequence. Even though that the entropy of the source information is defined as being the lowest possible limit of compression rate, the entropy isn't possible to be calculated as easily. The idea behind the USM algorithm is to use the best existing compression methods for biological sequences and estimate the entropy – the compression ratio becomes the compressive estimate of Entropy. [5]

The outcome of the USM analysis yields good results but is dependent on the quality of the compressor and if the dataset is well chosen for said compressor. [5]

2.2.2 Lempel-Ziv for biological sequences

The principle of the LZ77 (LZ) has been outlined in the chapter 1.4.3. There is basically no modification for the biological sequence analysis, apart from the difference in the alphabet used. In order to analyze the distance between two sequences, they need to be concatenated one to another and the number of steps needed to generate their exhaustive history can be calculated. The principle explaining why the measure of the number of steps can be used as an estimation of the evolutionary distance is described by the example below, taken from [13]:

Let us consider the following three sequences:

A = "AACGTACCATTG"; B = "CTAGGGACTTAT"; C = "ACGGTCACCAA"

Their individual exhaustive histories would be:

$H_E(A)$: ►A►AC►G►T►ACC►AT►TG

$H_E(B)$: ►C►T►A►G►GGA►CTT►AT

$H_E(C)$: ►A►C►G►GT►CA►CC►AA

As all of the histories contain 7 components their respective LZ complexities $c(A) = c(B) = c(C) = 7$. Let now be the concatenated sequences AC and BC with their respective exhaustive histories H_E (AC) and H_E (BC).

$H_E(AC)$: ►A►AC►G►T►ACC►AT►TG►ACGG►TC►ACCAA

$H_E(BC)$: ►C►T►A►G►GGA►CTT►AT►ACG►GT►CA►CC►AA

The LZ complexities are different this time with $c(AC) = 10$ and $c(BC) = 12$. The reason for this difference is that the sequence C has been encoded only in three steps when given the information of A, while it has been encoded in five steps given the sequence B. The cause is the sequence A and C share the strings ACG and ACCA and therefore are closer to each other. The number of steps S it takes to generate the sequence C using the sequence A can be calculated by the following simple equation:

$$S = c(AC) - c(A) \tag{5}$$

This is how the similarity between the sequences is estimated.

3 EXPERIMENT LAYOUT

The objective of this master thesis is to assess if the compression techniques based on the Lempel-Ziv (LZ) algorithm can be used for the classification of biological sequences. The focus is set on DNA sequences, but proteomic data are also briefly analyzed.

The experiment is divided into 4 parts. The first part takes 4 datasets of different length and similarity level. The compression algorithm is left as designed by the authors of [13], in order to gain a rough estimate to validate or deny the LZ approach. As the results were positive with ¾ of the datasets, the LZ approach has been validated.

The second part of testing has been designed in order to inquire how the similarity level of sequences influences the outcome of the algorithm. In this part 7 datasets of complete mitochondrial DNA have been used. The datasets were chosen according to the taxonomy tree of the NCBI database and are always comparing two groups of different taxonomy branch. The species in the main groups are chosen randomly. The first dataset compares two groups of bony vertebrates – *Euteleostomi*, and the next datasets descend the taxonomic tree till the last dataset which compares two groups of Primates.

The third part of testing works with fixed species, and concentrates on short sequences. The range goes from several hundreds of nucleobases to a thousand. Two proteomic sequences are also tested in this chapter, as they belong to the category of short sequences as well. The parts 2 & 3 were designed to determine the strength and weaknesses of the algorithm.

The fourth part implements modifications of the LZ77 algorithm, based on the previous results, in order to achieve higher precision in biological sequence classification.

3.1 Estimating the algorithm functionality

Four different datasets have been used to get a rough estimate of the possibilities of the LZ technique in the classification of biological sequences:

3.1.1 Datasets

Table 2: Datasets used for the first part of testing.

Dataset no.	Dataset specification	Dataset average length [bp]
1st dataset	16S rRNA sequences of 13 primates.	1500
2nd dataset	Mitochondrial DNA of chosen animals.	16500
3rd dataset	Hepatitis A virus variants from across the world - 25 sequences.	7500
4th dataset	Rhabdovirus variants, including 7 subgroups.	12000

The complete list of sequences can be found in the appendix of this paper.

3.1.2 Metric system

Four different, but similar, metrics have been used in this master thesis to estimate the evolutionary distance, the metrics were left as designed in [13]. The main idea behind the proposed metrics is that the number of steps necessary to create the exhaustive histories of concatenated sequences AB and BA are usually different, which needs to be taken into account.

Let's consider the individual and concatenated exhaustive histories of sequences A and B: *c*(*A*), *c*(*B*), *c*(*AB*), *c*(*BA*).

First metric:

$$d_1 = \max\{c(AB) - c(A), c(BA) - c(B)\} \tag{6}$$

The first metric, leading to the distance measure d_1, chooses the longest concatenated exhaustive history, subtracted by the provided sequence's history.

Second metric:

$$d_2 = \frac{\max\{c(AB) - c(A), c(BA) - c(B)\}}{\max\{c(A), c(B)\}} \tag{7}$$

It can be imagined that longer sequences will statistically have larger exhaustive histories and thus could falsify the result. This reasoning leads to the second metric. Compared to the first metric, the distance measure d_2 has been normalized and so the effect of variable sequence length has been diminished. [13]

Third metric:

$$d_3 = c(AB) - c(A) + c(BA) - c(B) \tag{8}$$

In the previous two cases one of the histories has been chosen to represent the relativeness of the sequences A and B. The d_3 approach is to take both of the variants, meaning *c*(*AB*) and *c*(*BA*), into consideration. This leads to the third metric system with the distance measure d_3. [13]

Forth metric:

$$d_4 = \frac{c(AB) - c(A) + c(BA) - c(B)}{\frac{1}{2}\{c(AB) + c(BA)\}} \tag{9}$$

The last metric is the normalized version of d_3. [13]

Those four metric systems have been used to analyze the datasets.

3.1.3 Algorithm specification

The algorithm is divided into 2 parts. In the first part the exhaustive history of each of the single input sequences ($c(A)$) is calculated. The number of steps needed for the construction of the exhaustive history is saved. The exhaustive history itself, the location pointer, length of the words and the new characters, as explained in the theoretical part of this work, are not saved. They would serve no purpose in the phylogenetical analysis. Even though the new character is not saved, it is still skipped at each step of the exhaustive history construction and therefore the next string matching step starts at the position "last character of the last longest common word +1". In the second part of the algorithm two sequences at a time are concatenated together and their conditional exhaustive history is calculated. As the first half of the sequence has been calculated in the first step, it is not necessary to recalculate it. The process starts with the first sequence (A) filling the left window and the second sequence (B) filling the right window. Once the number of steps to form the exhaustive history of B using A has been calculated, the number of steps of A known from the first step of the algorithm is added, and therefore $c(AB)$ is reached. These steps are repeated for all the combination of sequence pairs in the dataset. The resulting matrix is then used as an input for the metric systems, and the four phylogenetical trees are constructed. The flowchart of the algorithm is presented on the figure 3.

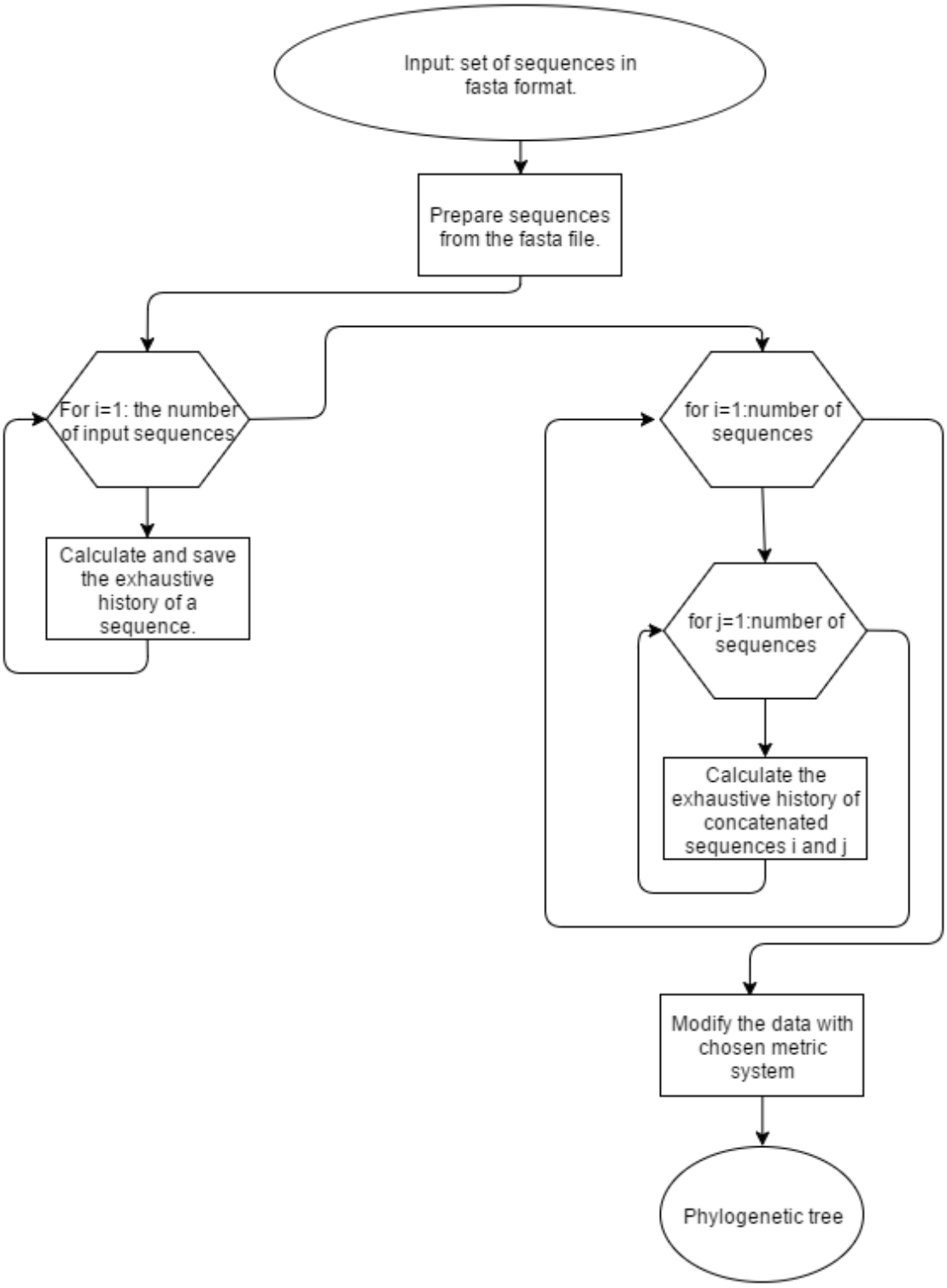

Figure 3: The flowchart of the program

3.1.4 Results and discussion

Thorough the testing of the algorithm it has been successfully confirmed that the LZ77 algorithm can be used for the classification of biological sequences. On the other hand the phylogenetical trees calculated with the four metrics possess all long branches from the leaf to the first nod and very short branches from nods to nods. This effect is due to the form of proposed metrics. The distances vary too little compared to their nominal value and so it appears the subjects are distant to each other. The reason is that the LZ77 algorithm isn't an algorithm to measure the exact evolutionary distance but a simple classification algorithm.

To compare the generated trees via LZ77 with the standard approach, a reference tree has been constructed from the set of sequences using the Jukes-Cantor distance [20]. In order to create a phylogenetic tree, a construction method had to be chosen. For simplicity in the first part of the experiment the UPGMA has been chosen [19]:

The reference tree for the first dataset (16S rRNA of 13 primates) is presented on the figure below:

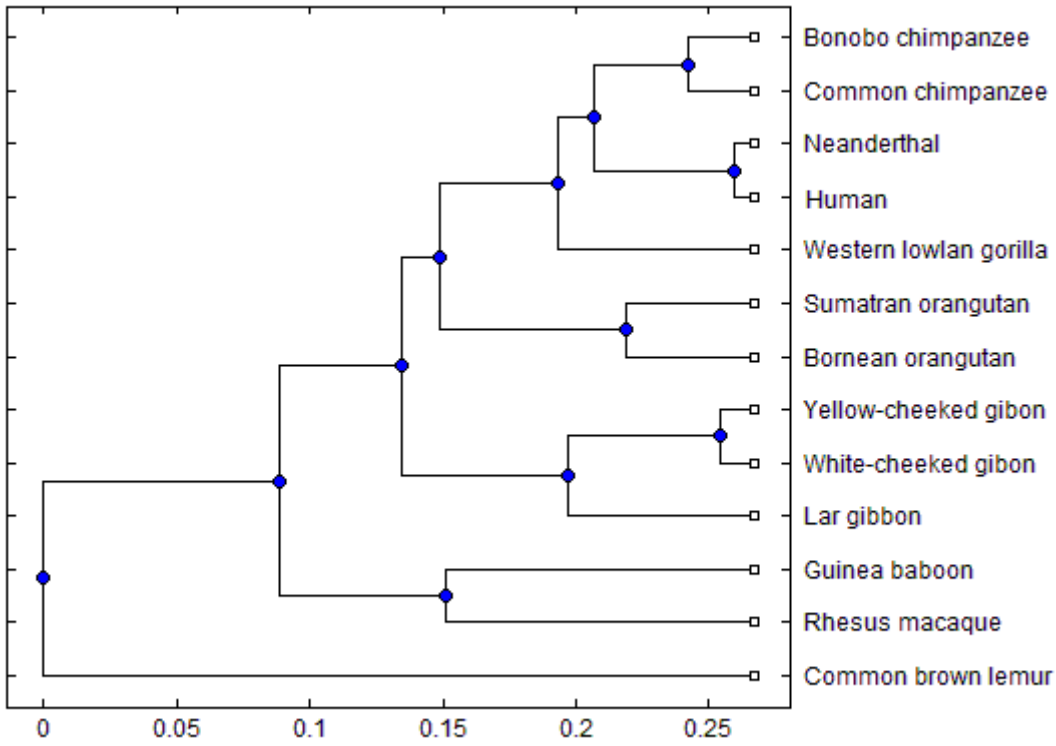

Figure 4: Reference phylogenetic tree constructed via J-C and UPGMA from the first dataset.

The two extreme values of the reference phylogenetic, from figure 4, are going to be examined first. The sequences of the Neanderthal and Human are the ones closest to each other. On the other hand the most distant sequence, connected directly to the root of the tree, is the Common brown lemur. The sequences were analyzed by the four metrics and the distance measures yielding the best phylogenetic tree for this concrete

dataset – the distance d_3, is presented below. The Common brown lemur, Neanderthal and Homo sapiens are stored under the variables 2, 10 and 11 respectively.

	1	2	3	4	5	6	7	8	9	10	11	12	13
1	0	0	0	0	0	0	0	0	0	0	0	0	0
2	421	0	0	0	0	0	0	0	0	0	0	0	0
3	356	412	0	0	0	0	0	0	0	0	0	0	0
4	363	417	380	0	0	0	0	0	0	0	0	0	0
5	386	415	387	353	0	0	0	0	0	0	0	0	0
6	386	405	389	357	166	0	0	0	0	0	0	0	0
7	366	407	394	327	256	190	0	0	0	0	0	0	0
8	362	411	374	211	358	362	349	0	0	0	0	0	0
9	391	408	391	337	260	259	323	359	0	0	0	0	0
10	380	400	380	328	256	256	314	357	178	0	0	0	0
11	377	402	386	332	255	261	318	364	183	44	0	0	0
12	366	409	391	320	334	330	357	356	264	257	259	0	0
13	383	411	390	315	256	256	317	358	155	143	149	203	0

Figure 5: The distance matrix d_3 with highlighted values of the closest and most distant individuals.

The distance measure based on LZ77 correctly distinguished that the Neanderthal and Human are the closest species from the dataset, as their relative distance value is the minimum of all the distances. The distance measure also agrees with the premise that the Common brown lemur is the most distant individual, as the column belonging to the sequence contains the highest numbers of the distance matrix. The phylogenetical tree created from this metric is presented on the figure 6:

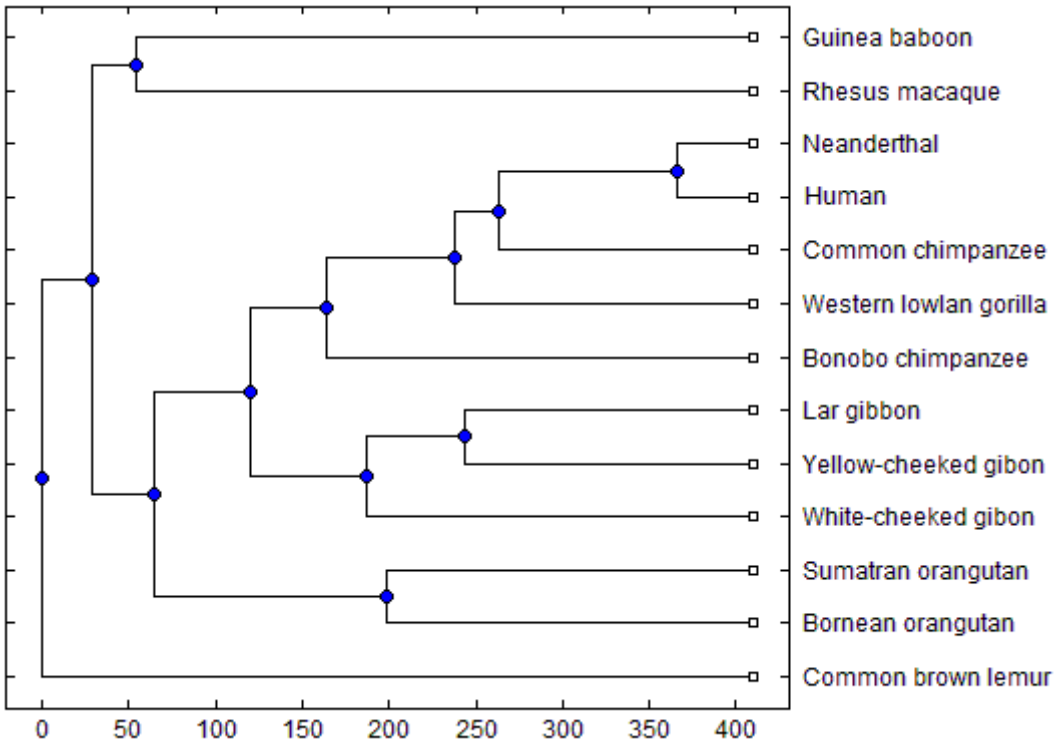

Figure 6: Phylogenetic tree constructed from the first dataset using the d_3 metric.

As explained in the introduction of this chapter, the distances between the leaves and the nods are considerable – longer than in the reference tree. The trees have been compared via the Robinson-Foulds metric (RF distance) [21], which describes the total of wrongly formed uncommon nods in between the two trees. The RF distance for the first dataset is equal to 8. Meaning that out of the 22 nodes (not counting the root) 14 are correct, yielding a 63% success rate. The sequences are analyzed in detail in the following figure. For an easier comparison the trees haven been turned into cladograms [22] – meaning the calculated distances have been ignored and the trees have been constructed out of the nods from the precedent figures:

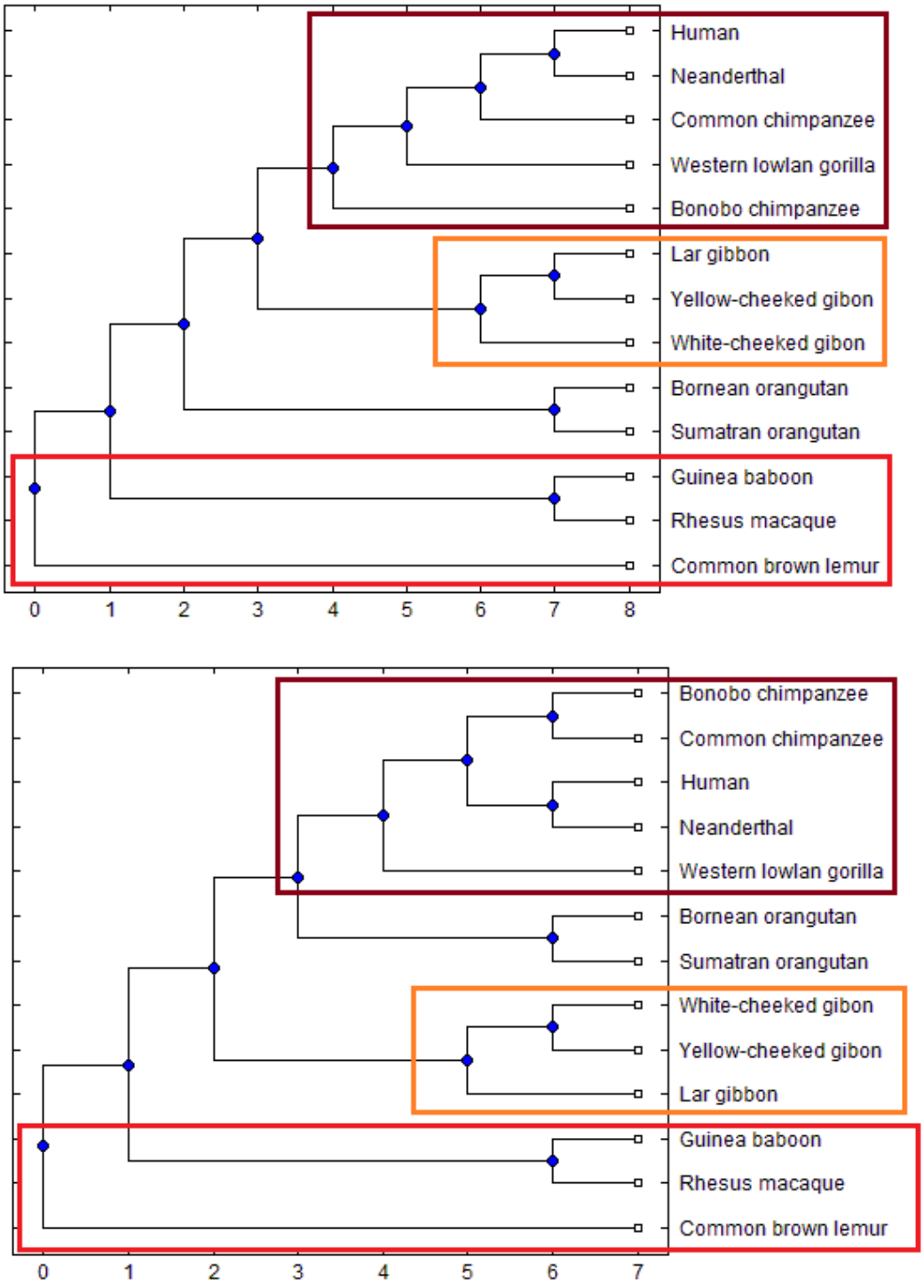

Figure 7: Comparison of the reference to the d3metric. On the top of the figure is the cladogram made out of d3 metric and at the bottom the reference. Three common large groups of Primates have been highlighted.

The first dataset didn't yield a very good result considering the RF distance but has proven that the algorithm is classifying in a correct manner. Figure 7 shows, that the LZ algorithm has correctly identified the three major groups of primates.

The second dataset containing the mitochondrial genome yields the best results with the distances d_3 and d_4. As the reference tree and the LZ d_3 tree is very similar, the comparison figure is showed directly:

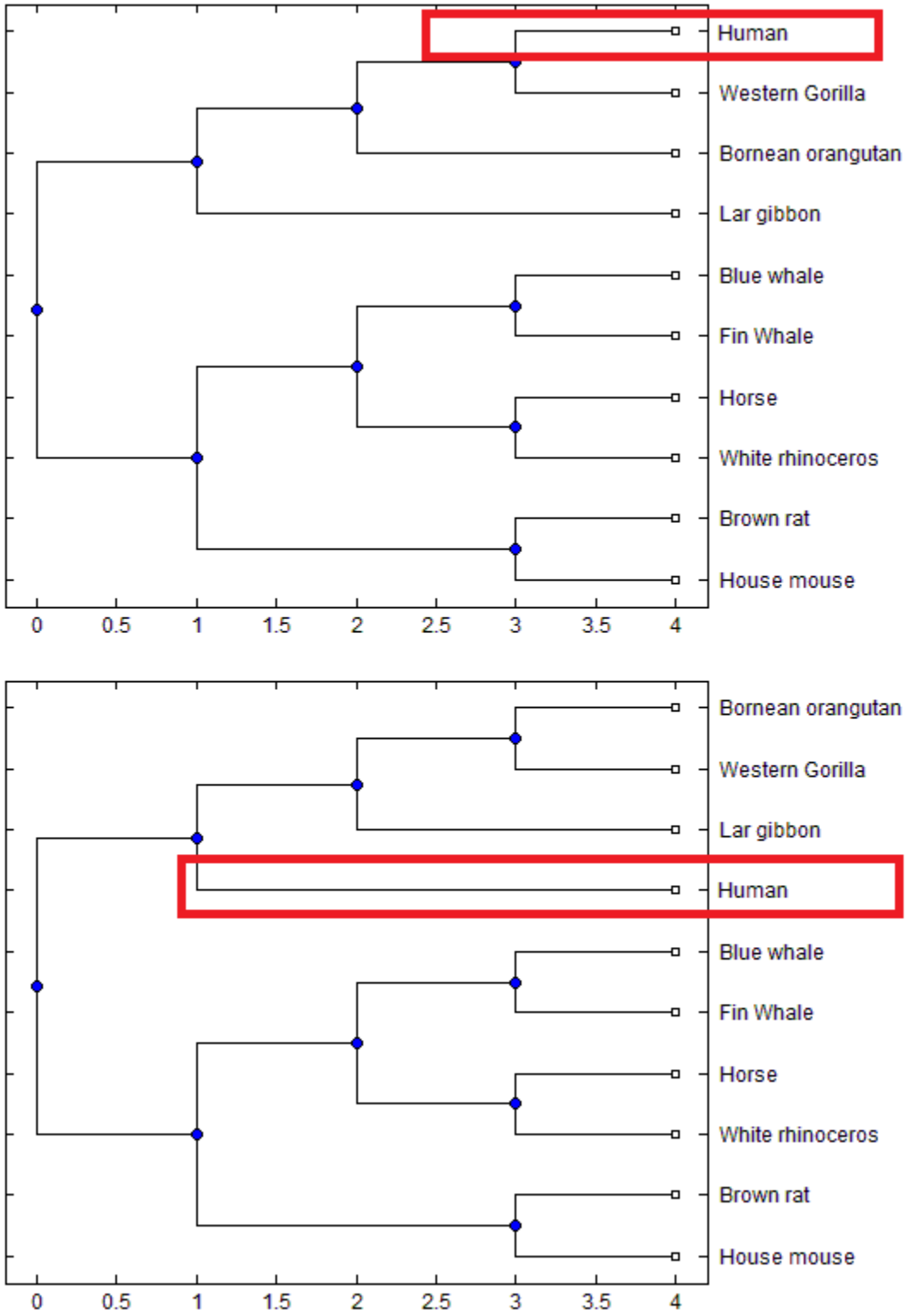

Figure 8: Cladograms of the third dataset. Distance d_3 on the top and reference on the bottom. The sole wrongly associated sequence is highlighted in the red box.

The second dataset contains only one misplaced sequence: the Human. Apart of that the phylogenetic tree has been generated correctly. The RF distance is 4, leading to a 75% success rate. It can be seen that even with a higher sequence length the LZ77 algorithm doesn't identify very similar sequences correctly but manages to group the close sequences together, not interfering with the other groups.

The reference phylogenetic tree for the third dataset can be viewed on the figure 8. The hepatitis A virus sequences are highly similar and the Jukes-Cantor algorithm doesn't classify the sequences correctly. For this reason the reference tree has been constructed as in the original paper [16].

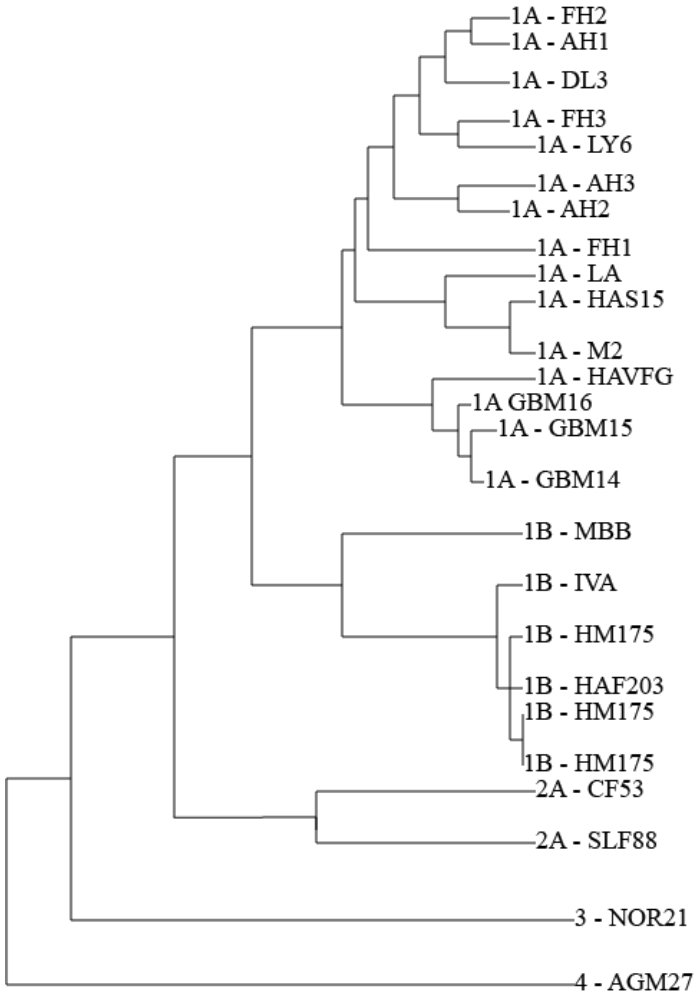

Figure 9: Reference tree for the hepatitis A virus as in [16]

This dataset contains 4 groups of the hepatitis virus differentiated by numbers on the figure 9. The group number 1 is abundant in sequences and is divided into two subgroups 1A and 1B. In order to highlight the closeness of the sequences the phylogenetical tree created via Jukes-Cantor and Neighbor-joining [23] is presented below:

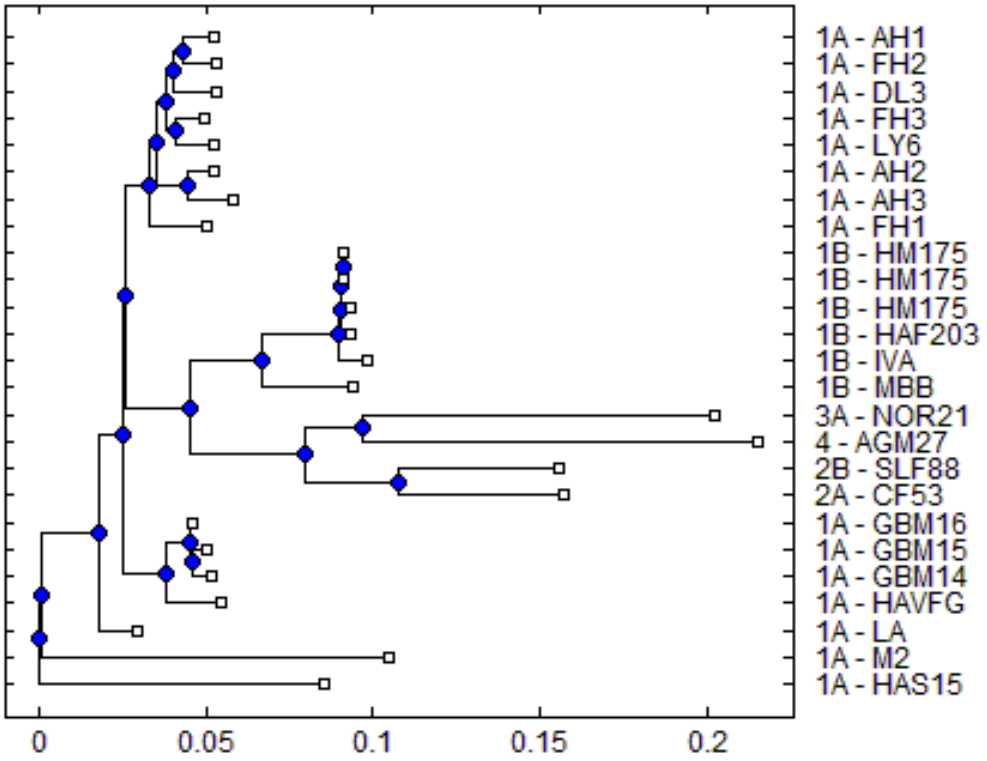

Figure 10: phylogenetic tree constructed based on JC from the second dataset.

The figure 10 shows that the distances are in many cases very similar, especially the sequences belonging to the groups 1A and 1B are very close to each other. The third dataset has been chosen to be a hard test for the LZ77 algorithm. The Jukes-Cantor algorithm fails to place the groups 2, 3 and 4 correctly, if compared to the reference from [16], and the group 1A is split into two parts, therefore this is a dataset that is challenging even for the alignment based methods.

The LZ77 algorithm has been tested for all of the four metrics. Metrics 1, 2 and 3 yield almost the same result with their maximum RF distance being 2. The fourth metric has a slightly worst result with the RF distance to the other 3 metric being 8. As the datasets are chosen to be difficult to classify, the simple UPGMA method is no longer sufficient, for this reason the phylogenetical trees in the rest of this paper are constructed via neighbor-joining. The resulting tree from the metric 3 is presented below:

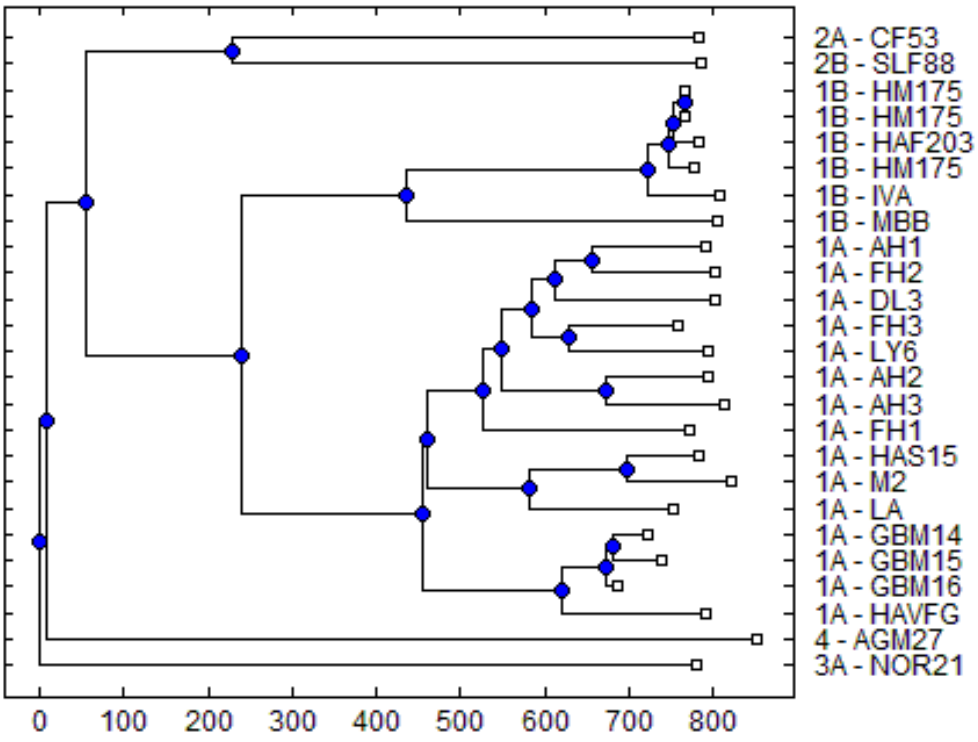

Figure 11: Phylogenetical tree of hepatitis A virus variants made from the 3rd metric.

It can be seen from the figure 11 that the LZ77 algorithm has managed to differentiate the main groups of hepatitis virus variants. The generated distances between the sequences (the length of the branches) are greater than in the tree generated by Jukes-Cantor, especially in the very similar groups such as 1A and 1B. This is a very positive discovery as the sequences that look similar to an alignment algorithm seem to be different enough for the LZ algorithm, which means the classification should be more precise. The next figure compares the reference from [16] with the phylogenetical tree created by the metric 3. For a better visibility the LZ tree has been turned into a cladogram.

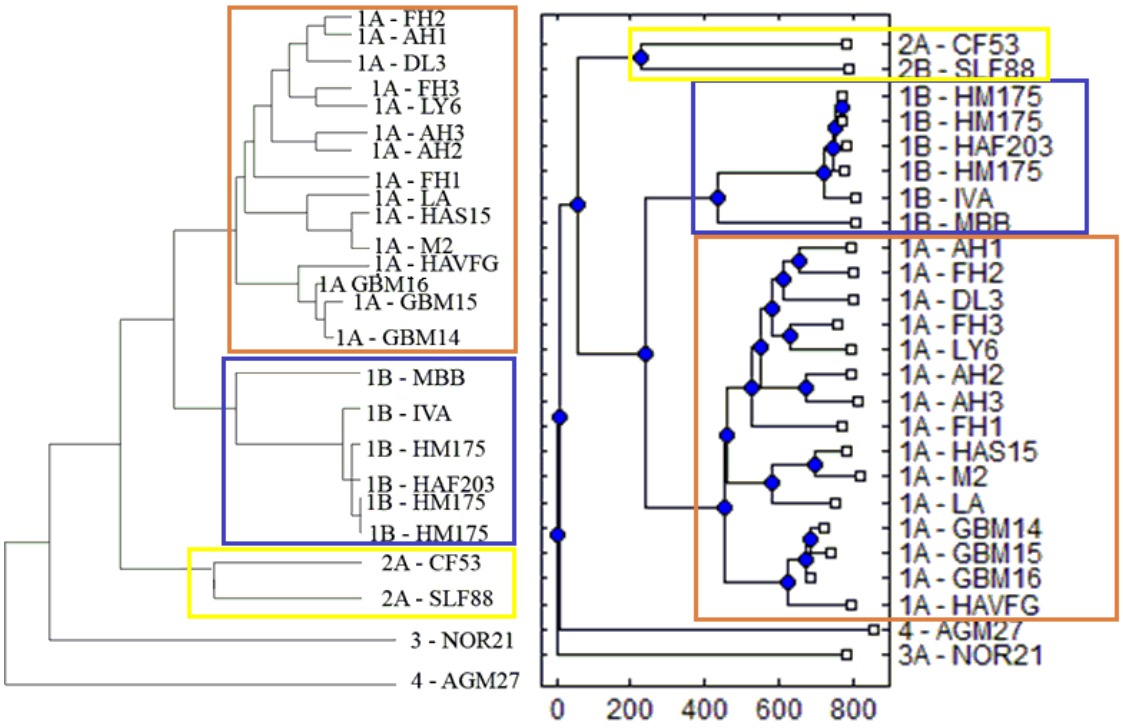

Figure 12: Comparison of the cladogram from d_3 and the reference for the second dataset. The three major traits have been highlighted.

The analysis of Figure 12 leads to the discovery that only 1 nods is classified incorrectly, leading to the RF distance of 2 between the reference and the LZ algorithm, which corresponds to 96 % success rate – a very good result for such a difficult dataset.

The fourth dataset consist of 35 different Rhabdoviruses belonging to 7 different subgroups. The specific sequences have been chosen from [17]. This is the most difficult dataset on which the LZ algorithm has been tested. The distances between the sequences are very similar. Before presenting the JC tree, the taxonomical tree of Rhabdoviruses as proposed by the authors of [17] is showed:

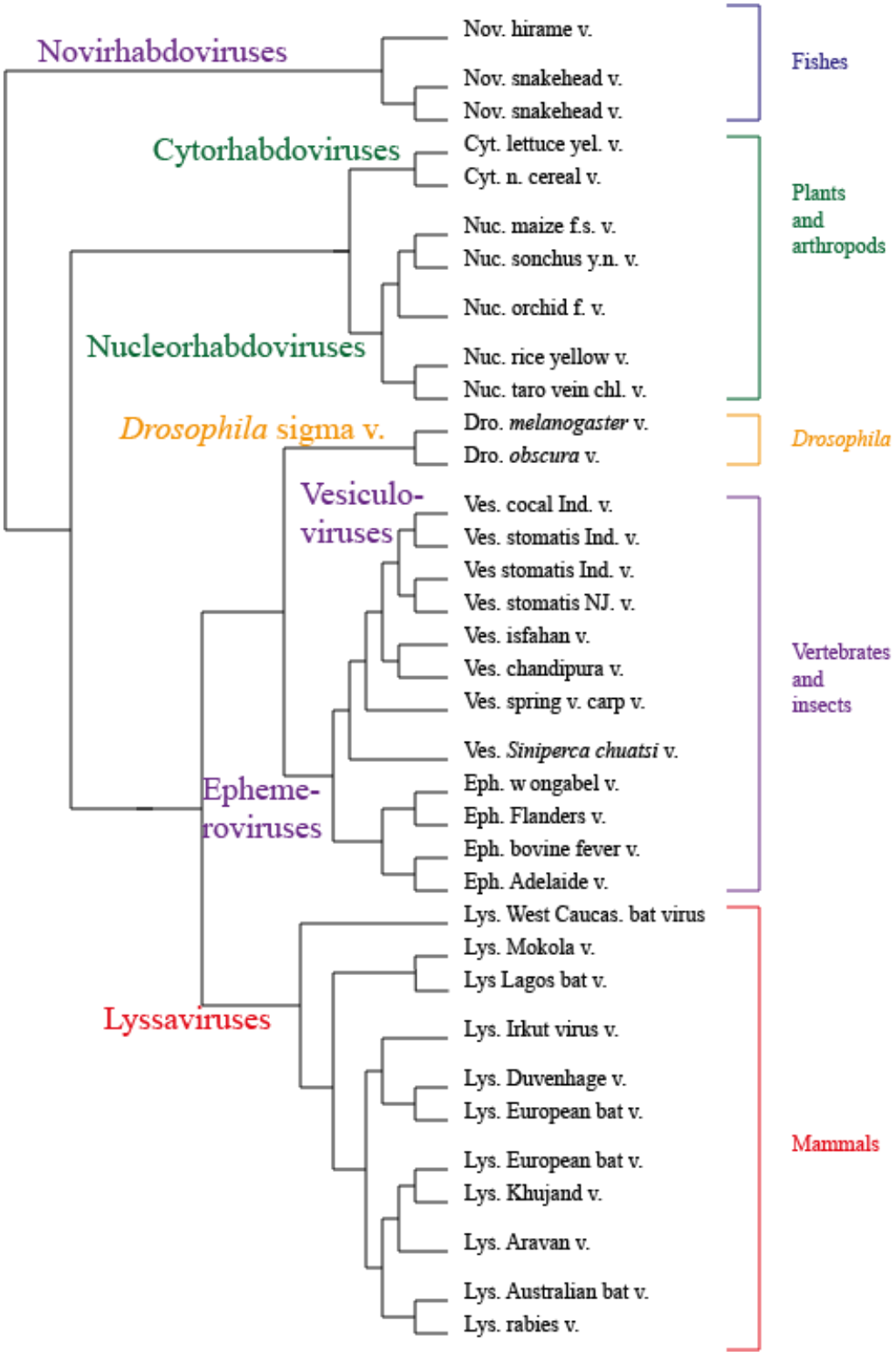

Figure 13: Reference phylogenetical tree of Rhabdoviruses [17]

The 7 different subgroups of the Rhabdovirus can be seen on the figure 13. The cladogram created via Jukes and neighbor joining can be seen below:

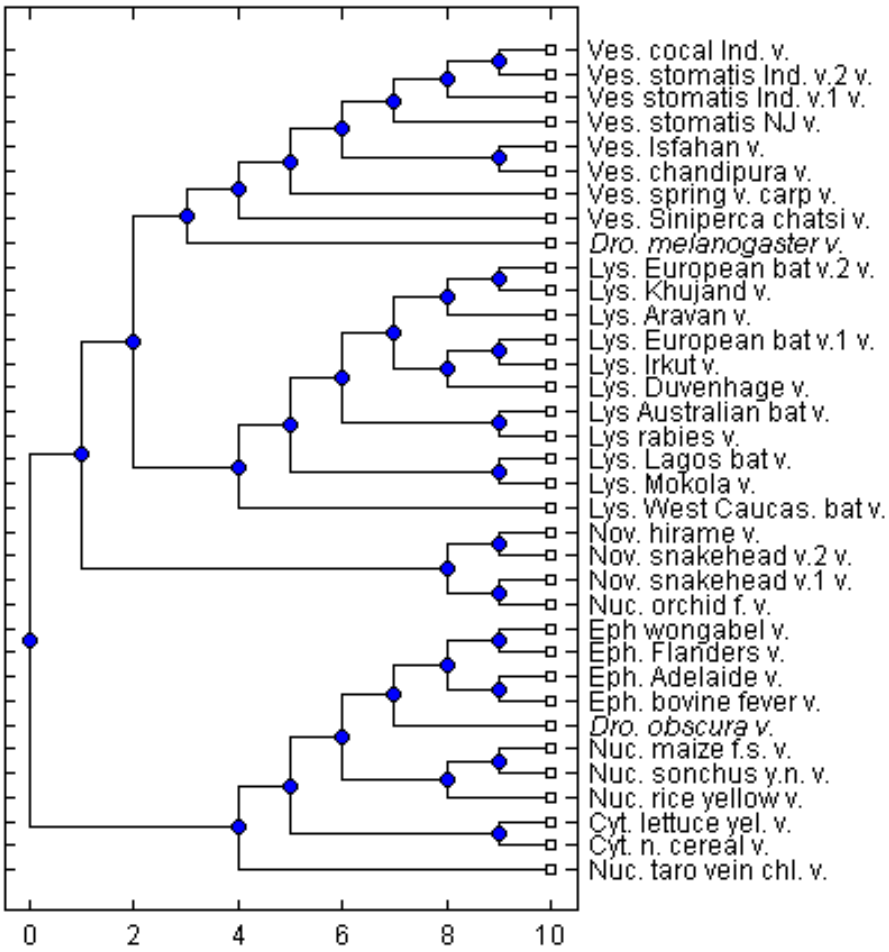

Figure 14: Jukes-Cantor's phylogenetical tree of the Rhabdoviruses variants.

The figure 14, shows that the classical Jukes-Cantor alignment based method manages to separate the Rhabdoviruses into the 7 subgroups almost correctly, only the drosophila sequences are not grouped together. On the other hand there are several mismatches compared to the reference, the biggest being the misplacement of the whole Ephemerovirus and Novirhabdovirus group. The Ephemerovirus should have the same ancestor as the Vesiculovirus, while the Novirhabdovirus subgroup should be the most distant one to the rest of the sequences. Once again this dataset is challenging even for the alignment based methods.

The dataset has been analyzed by the LZ algorithm and the table below shows the Robinson-Foulds distance between the 4 metrics:

Table 3: RF distances of the Rhabdovirus dataset between the 4 metrics

	Metric 1	Metric 2	Metric 3	Metric 4
Metric1	0	34	38	50
Metric2	34	0	36	52
Metric3	38	36	0	50
Metric4	50	52	50	0

The table 3 shows that the resulting trees from the 4 metrics differ largely between themselves. The reason of this failure is that the LZ complexity is very similar between the sequences. To back up this statement the first 11 entries from the variable, that contains the LZ complexity matrix modified by the metric 2, is showed below:

	1	2	3	4	5	6	7	8	9	10	11
1	0	0.8718	0.8976	0.8819	0.8682	0.8671	0.8810	0.8694	0.8673	0.8700	0.8904
2	0.8718	0	0.8889	0.8757	0.8658	0.8665	0.8767	0.8711	0.8679	0.8448	0.8810
3	0.8976	0.8889	0	0.8766	0.8848	0.8904	0.8792	0.8874	0.8874	0.8884	0.8802
4	0.8819	0.8757	0.8766	0	0.8723	0.8779	0.8664	0.8735	0.8695	0.8735	0.8763
5	0.8682	0.8658	0.8848	0.8723	0	0.8418	0.8767	0.8091	0.8500	0.8658	0.8816
6	0.8671	0.8665	0.8904	0.8779	0.8418	0	0.8696	0.8382	0.8373	0.8665	0.8792
7	0.8810	0.8767	0.8792	0.8664	0.8767	0.8696	0	0.8729	0.8696	0.8778	0.8767
8	0.8694	0.8711	0.8874	0.8735	0.8091	0.8382	0.8729	0	0.8505	0.8628	0.8810
9	0.8673	0.8679	0.8874	0.8695	0.8500	0.8373	0.8696	0.8505	0	0.8661	0.8690
10	0.8700	0.8448	0.8884	0.8735	0.8658	0.8665	0.8778	0.8628	0.8661	0	0.8769
11	0.8904	0.8810	0.8802	0.8763	0.8816	0.8792	0.8767	0.8810	0.8690	0.8769	0

Figure 15: Variable containing the LZ complexity modified by the metric 2, first 11 entries.

Only 11 entries have been shown for a good visibility, but the data are similar thorough the whole variable. Since the closeness of the sequences is this great and that the difference in the LZ complexity between the sequences is almost inexistent, the approach fails to classify the sequences properly. The high similarity can be visually seen on the phylogenetic tree of the distance 2 below:

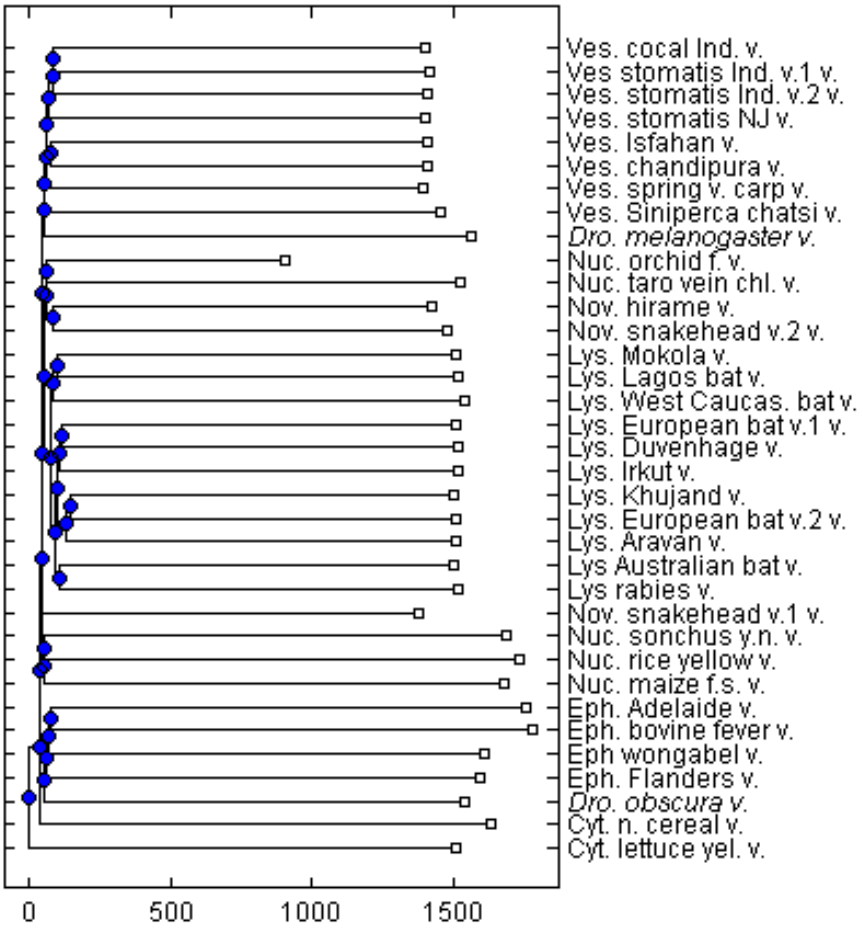

Figure 16: Graphical display of LZ complexity similarities between the sequences.

The result for the Rhabdoviruses is disappointing but expected. The LZ complexity with similar sequences is susceptible to variance. Even if the sequences are very similar, theoretically the ones closer to each other should still yield a better result. Unfortunately it cannot be said that there will always be more mutations, in slightly more distant sequences, which are the reason for an iteration of the algorithm to restart and thus increase the LZ complexity. Also in some cases a well-placed mutation can lead to a longer common word between two sequences. For this reason the sole LZ distance is not sufficient for sequences with similar distances in between them.

Throughout the whole testing the metric distance d_3 has had the best results and even in this case the same can be said. After analyzing all the phylogenetical trees, the distance d_3 manages to separate the Rhabdovirus families similarly to the JC tree, the cladogram is presented below:

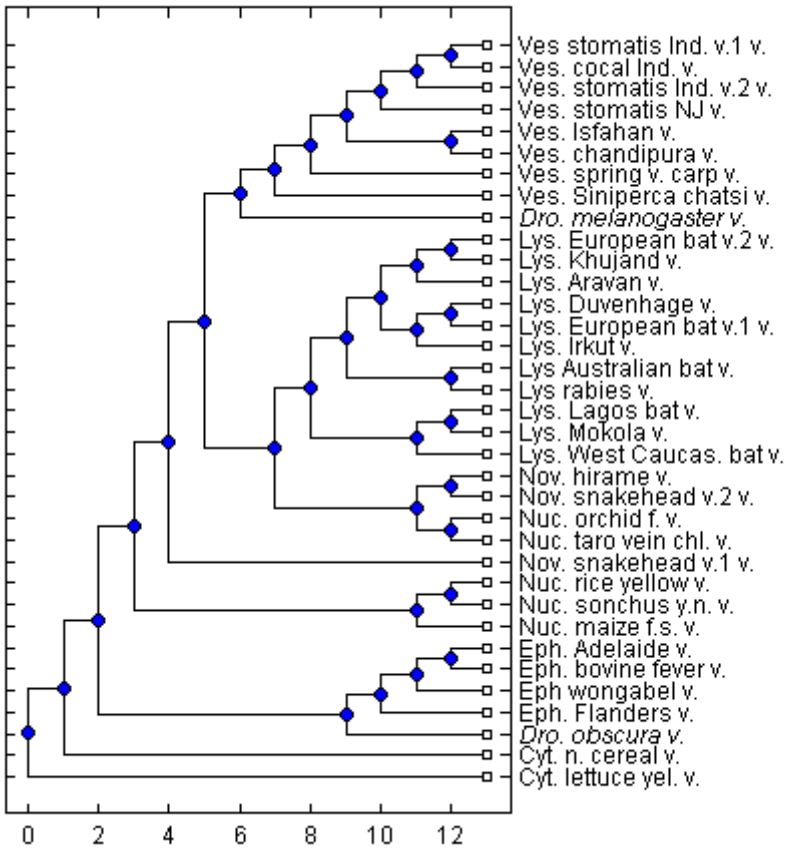

Figure 17: Cladogram of the Rhabdovirus dataset by the LZ alg. with the distance metric d_3.

The Ephemerovirus and the Novirhabdoviruses groups are not in the correct place, but overall the classification into groups has been successful.

3.1.5 Conclusion of the first part of testing

The direction of research in the field of compression data algorithms to be used to classify biological sequences is justified. Even though the algorithm doesn't show the best results with closely related sequences, it manages to group together the closest sequences and in the case of hepatitis A, the LZ algorithm outclasses the Jukes-Cantor alignment method. It is important to keep in mind that the algorithm used has been the LZ77 algorithm without any modification. It is a simple algorithm which hasn't been adapted for the classification of biological sequences apart from the fact of introducing the four different metric systems.

3.2 Sequence disparity testing

As it has been shown in the previous chapter, the algorithm as it stands cannot be used universally. The second part of testing has been designed in order to test the LZ algorithm's resolution. Species belonging to two different taxonomy branch groups are chosen in each datasets. The datasets are descending the taxonomic tree from very

distant to closer groups from the evolutionary point of view.

3.2.1 Algorithm specifications

The algorithm uses the same metric system and base algorithm as proposed in [17]. The tree construction the UPGMA method, which is old and unprecise, is changed to neighbor-joining. The reference trees used for this method will be based on the taxonomic classification of the NCBI database. For this part of testing the interest lies in the ability of the algorithm to separate the species in two groups and not necessarily to reach the perfect phylogenetical tree. As the NCBI tree is a cladogram, it doesn't use evolutionary distances. For this reason more species can be at the same taxonomic level. When evaluating the RF distance between the constructed tree and the reference tree, this fact could lead to falsification as the constructed trees are bifurcating trees. In order to correct this possible falsification the following rule, when evaluating the RF distance, is set. Both of the variants of the branches on figure 16 are going to be considered correct.

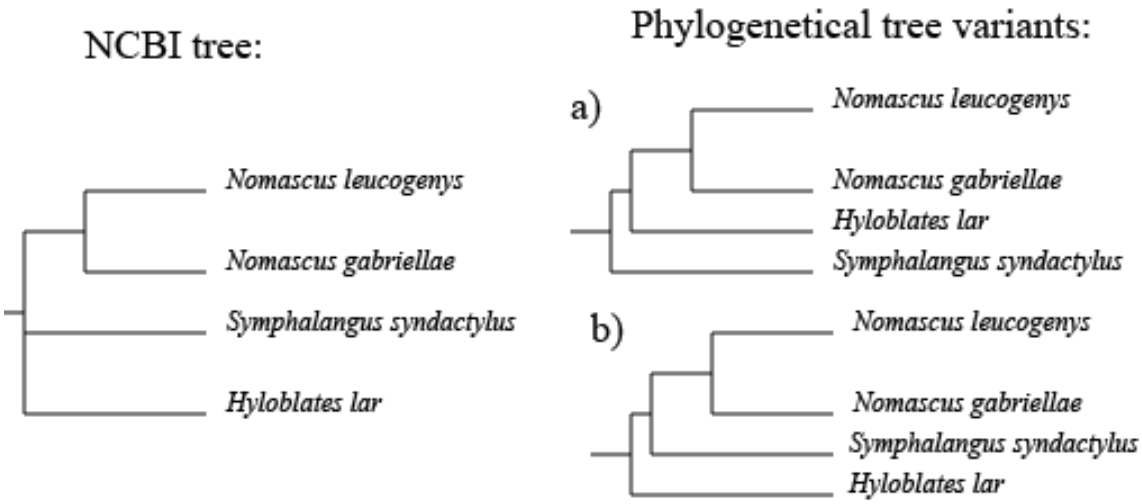

Figure 18: NCBI reference tree on the left side and the two correct variants that can occur on the right side.

3.2.2 Datasets

7 datasets of mitochondrial DNA have been prepared. Each dataset is composed of around 13 species belonging to two different groups of a taxonomical family. The number of sequences in a dataset varies due to the accessibility of sequenced species in the analyzed family. Once the main groups are chosen, the concrete species inside the groups are the ones that are accessible on NCBI, which means that they may differ a lot from each other, even inside of the groups. With this said, the difference shouldn't surpass the difference between the species from group to group. The last dataset is the concatenation of the last 3 datasets. The figure below displays the groups that have been chosen:

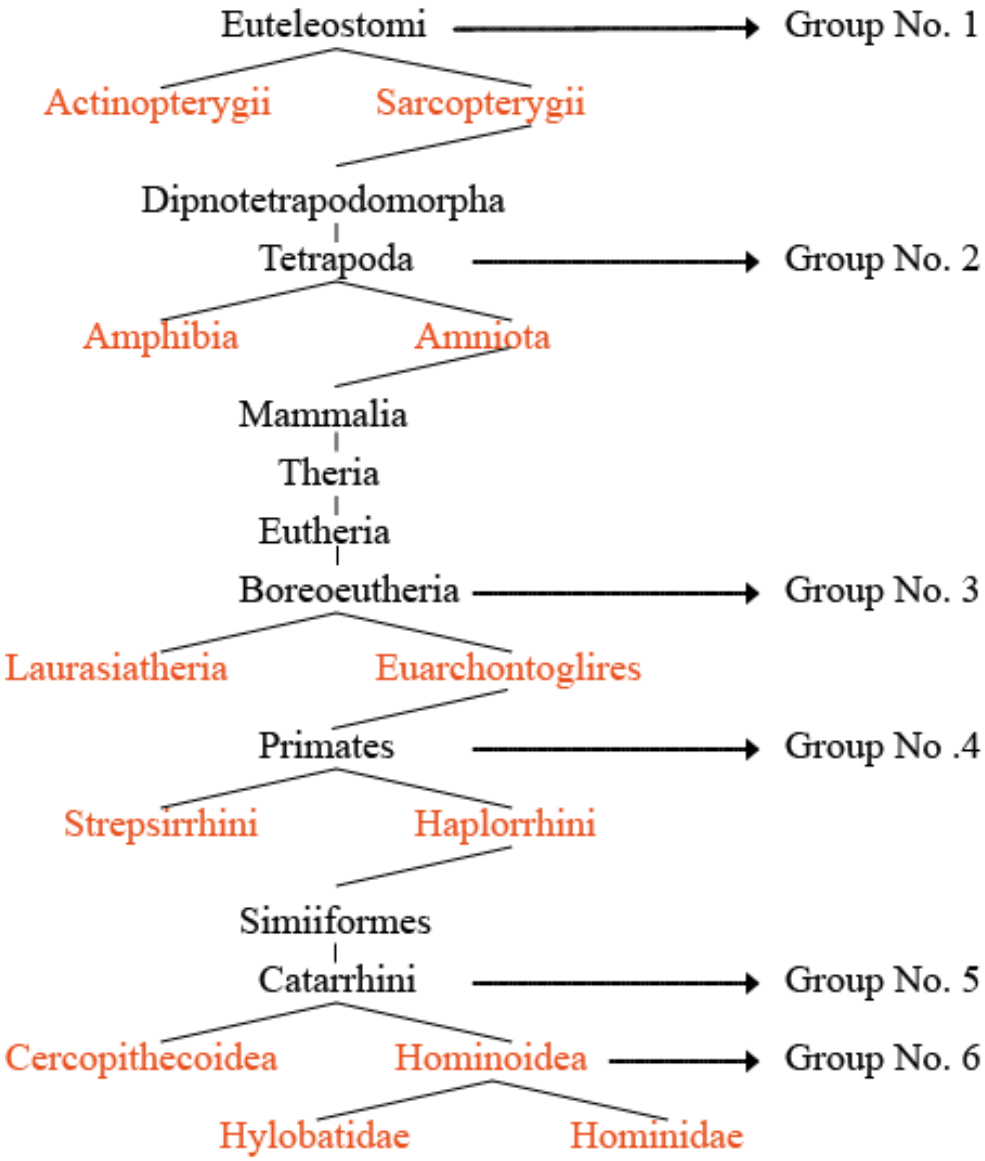

Figure 19: display of the families chosen for the second round of testing.

3.2.3 Results and discussion

The first taxonomic level contains species from the Actinopteri and Sarcopterigii groups.

The phylogenetical trees are presented in the following order: NCBI reference, Jukes-Cantor and LZ. The computed phylogenetical trees are displayed as cladograms for a better visibility:

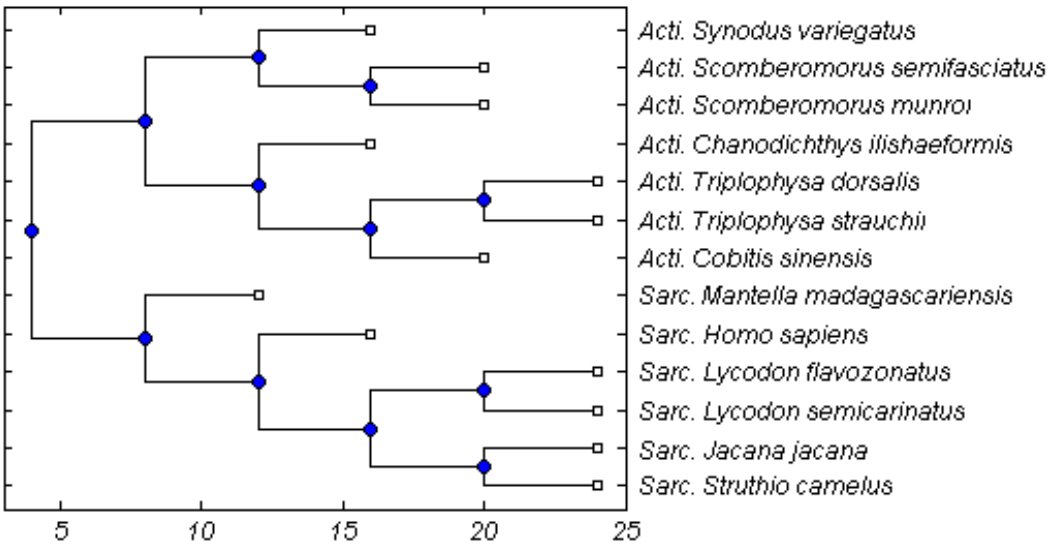

Figure 20: NCBI reference tree of Actinopteri and Sarcopterigii

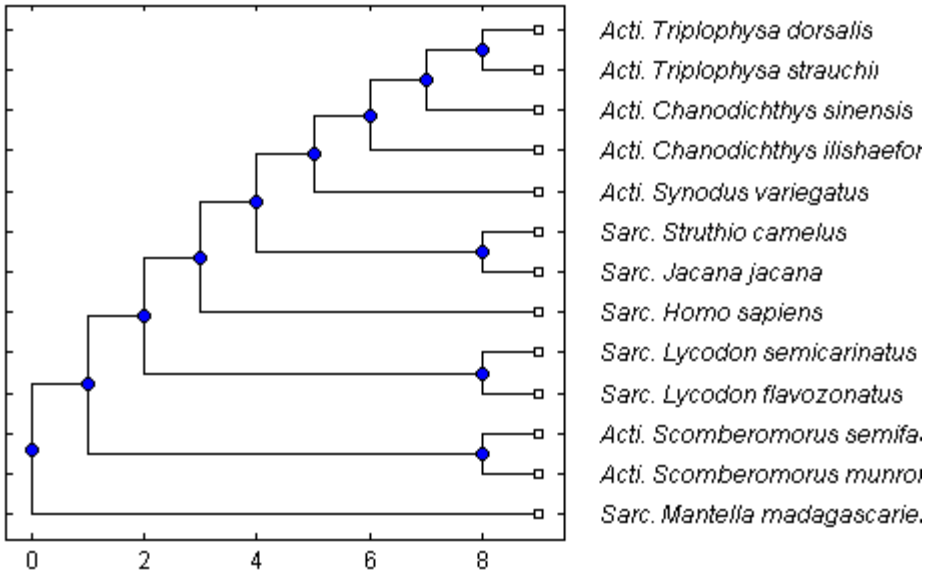

Figure 21: Jukes-Cantor cladogram of Actinopteri and Sarcopterigii

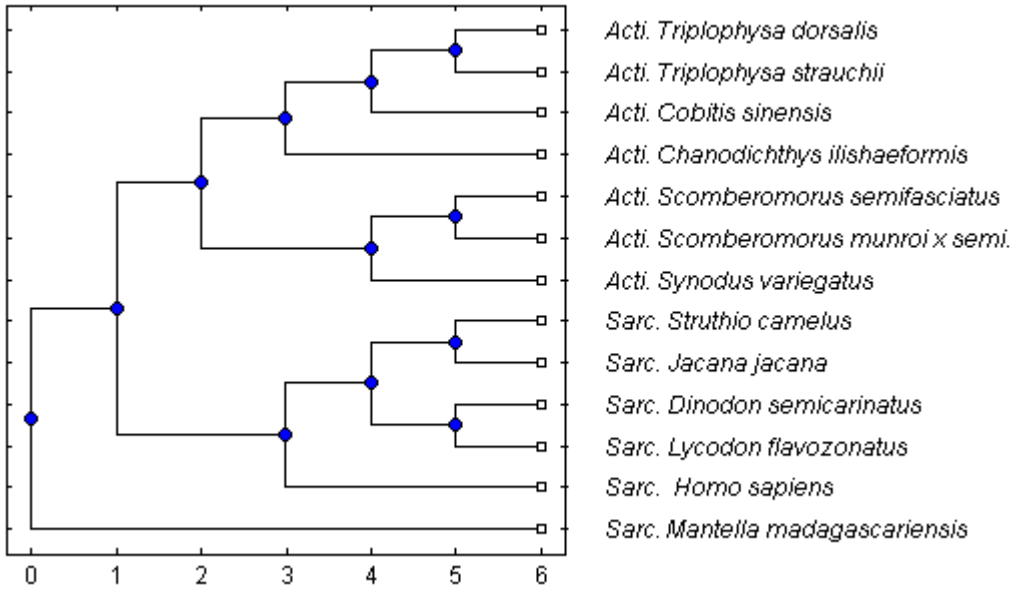

Figure 22: LZ cladogram based on metric no. 3. of Actinopteri and Sarcopterigii

The first dataset is made of two very foreign groups Actinopteri and Sarcopterigii. The tree using the metric no. 3 has been presented as it is the best for this group, but all the other metrics yield very similar results. The only sequence that poses problem is *Mantella madagascariensis*, which is sometimes classed with the wrong group or classified outside of the Amniota group. This is not too surprising as *Mantella Madagascariensis* belongs to the amphibia family while all the other sequences from Sarcopterygii belong to Amniota. *Mantella Madagascariensis* is very distant from both the Actinopteri and the Amnoita and therefore could be interpreted wrongly by the LZ algorithm. Apart from this sequence, all of the other species are classified in the correct groups, and so it can be confirmed that the algorithm works well at the taxonomic level of Euteleostomi. The table of RF distances of the four metrics, and three alignment algorithms from the NCBI reference (RFD_{REF}) is displayed below. Two alignment methods, Kimura [24] and Tamura [25] have been added in order to have a better comparison between the LZ algorithm and the alignment technics.

Table 4: RF distance between the 4 metrics and the alignment based algorithms compared to the NCBI reference of the Euteleostomi.

Methods compared:	LZ Metric 1	LZ Metric 2	LZ Metric 3	LZ Metric 4	Jukes-Cantor	Kimura	Tamura
$RFD_{REF:}$	4	4	2	6	10	4	4
Success rate:	82%	82%	91%	73%	55%	82%	82%

Out of the analyzed techniques the LZ algorithm in combination with the metric 3 has the best result. The Jukes-Cantor distance is the most inconsistent, the distances are too foreign from each other and therefore the necessary alignment becomes problematic.

The JC reference fails at classifying the sequences in the two groups.

The second dataset is at the taxonomic level of Tetrapoda and the analyzed groups are the Amphibia and the Amniota.

The phylogenetical trees from the second dataset are presented below, as the NCBI database is the reference, the alignment methods won't be displayed but their RF distance to the reference is going to be presented in the table 5:

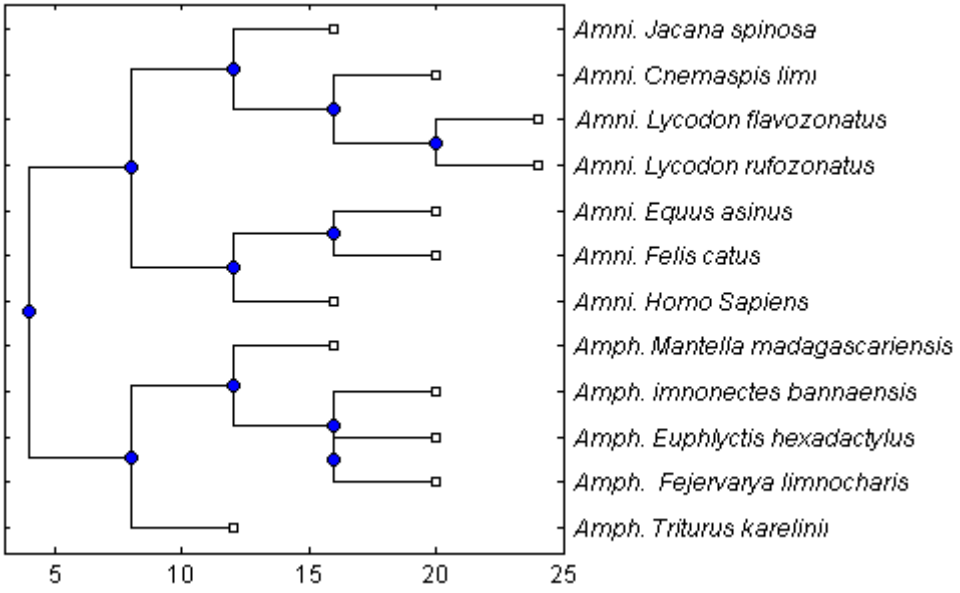

Figure 23: NCBI reference tree of Amphibia and Amniota

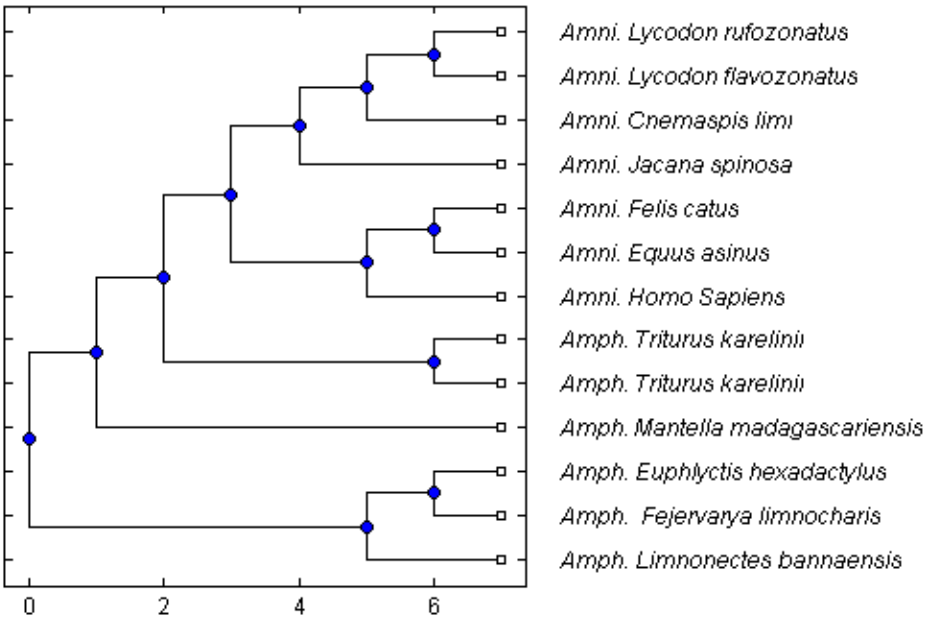

Figure 24: LZ cladogram based on metric no. 3. of Amphibia and Amniota.

Table 5: RF distance between the 4 metrics and the alignment based algorithms compared to the NCBI reference of the Tetrapoda.

Methods compared:	LZ Metric 1	LZ Metric 2	LZ Metric 3	LZ Metric 4	Jukes-Cantor	Kimura	Tamura
$RFD_{REF:}$	10	6	6	8	8	8	8
Success rate:	55%	73%	73%	64%	64%	64%	64%

The second dataset results are similar to the first dataset. The LZ algorithm manages to correctly recognize the Amniota group with all of the metrics but has a problem with the Amphibia group – it classifies the Triturus sequences wrongly, as if it would be a family on its own. The reason for this behavior is that once again the *Triturus karelinii* sequence is very distant from both – the Amniota and the rest of the Amphibia group. The alignment methods fail to classify the sequences correctly for the same reason. Once again the metric no. 3 has the best result.

The third dataset is at the taxonomical level of Boreoeutheria and compares the group of Laurasiatheria and Euarchontoglires.

The phylogenetical trees from the third dataset are presented below:

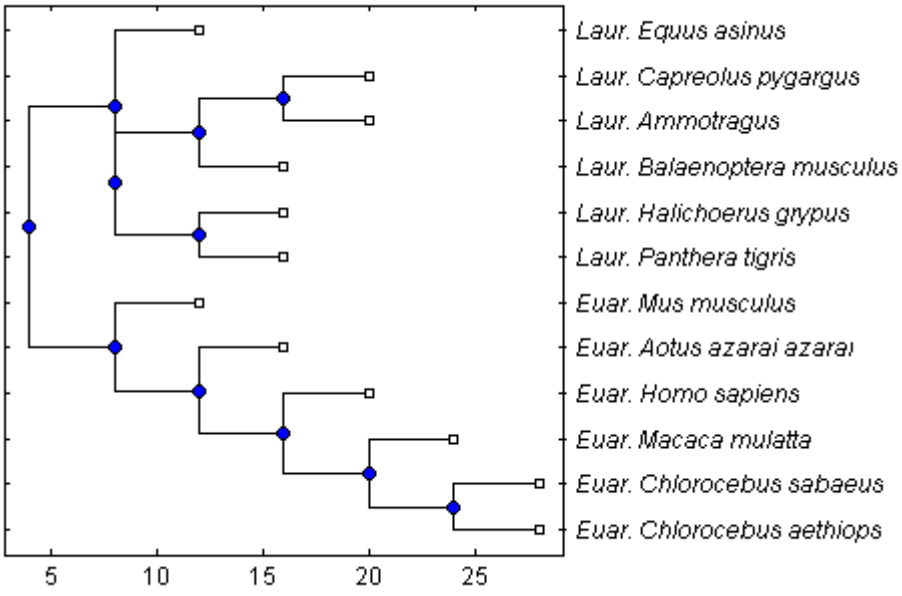

Figure 25: NCBI reference tree of Laurasiatheria and Euarchontoglires.

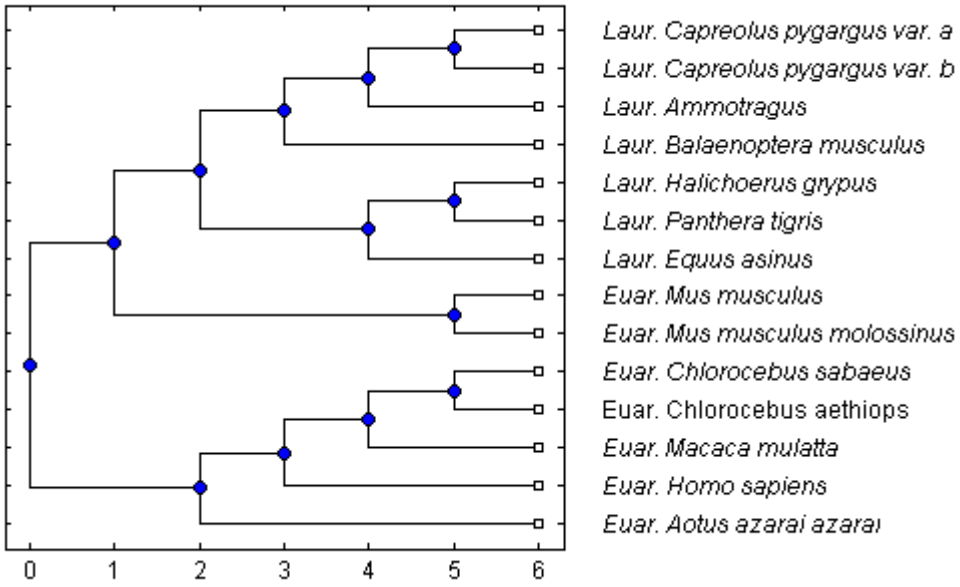

Figure 26: LZ cladogram based on metric no. 3 Laurasiatheria and Euarchontoglire.

Table 6: RF distance between the 4 metrics and the alignment based algorithms compared to the NCBI reference of the Boreoeutheria.

Methods compared:	LZ Metric 1	LZ Metric 2	LZ Metric 3	LZ Metric 4	Jukes-Cantor	Kimura	Tamura
$RFD_{REF:}$	2	2	2	2	8	2	2
Success rate:	92%	92%	92%	92%	66%	92%	92%

The third dataset leads to the same cladogram for all of the LZ metrics. The final tree is in accordance with the NCBI tree in every branch, but the *Mus musculus* sequences. It can be seen that the LZ algorithm can differentiate well between the groups and even inside of the families is able to classify the correct sequences together. The problem that repeats itself is when the algorithm has to classify a sequence that is at a similar distance to the two groups. In this case the *Mus musculus* sequence is the only one belonging to the subfamily of Glires, the immediate subdivision after Euarchontoglires. All of the other Euarchontoglires sequences belong to the group of Primates. As the Glires are distant from the Primates the LZ algorithm classifies it as a standalone group – the same case as with *mantilla madagascariensis* and the Titrus family. The alignment based methods of Kimura and Tamura yield the same result as the LZ algorithm, while the Jukes-Cantor method is failing to classify the sequences correctly, mixing the two taxonomical groups together.

The fourth dataset is at the taxonomical level of Primates and compares the group of Strepsirrhini and Haplorrhini.

The phylogenetical trees from the fourth dataset are presented below:

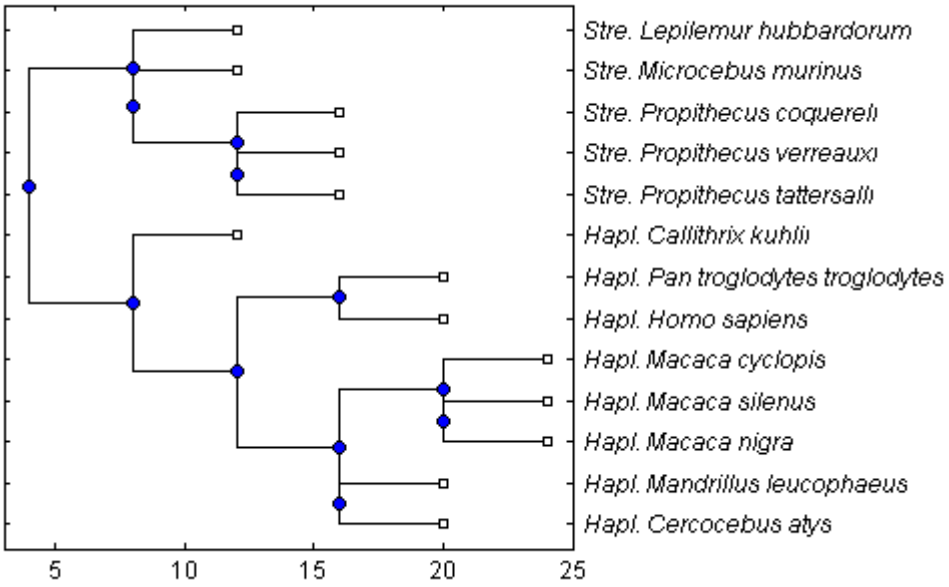

Figure 27: NCBI reference tree of Strepsirrhini and Haplorrhini.

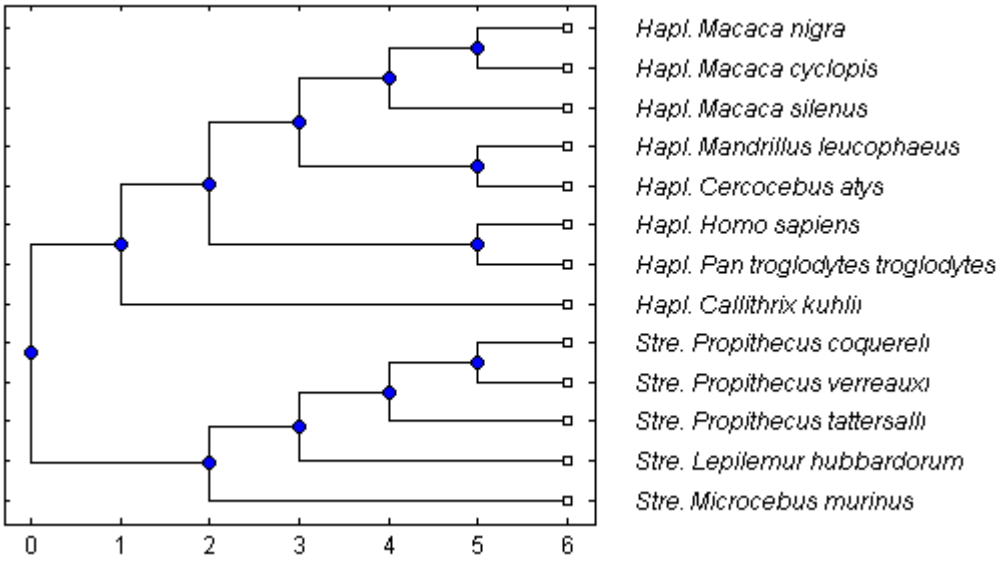

Figure 28: LZ cladogram based on metric no. 3 Strepsirrhini and Haplorrhini.

Table 7: RF distance between the 4 metrics and the alignment based algorithms compared to the NCBI reference of the primates.

Methods compared:	LZ Metric 1	LZ Metric 2	LZ Metric 3	LZ Metric 4	Jukes-Cantor	Kimura	Tamura
$RFD_{REF:}$	0	0	0	0	0	0	0
Success rate:	100%	100%	100%	100%	100%	100%	100%

The fourth dataset is in accordance in between the NCBI model and all of the computed phylogenetical trees, be it by the LZ algorithm or the alignment method. The LZ algorithm yields the same phylogenetical tree in ¾ metrics, but is at the same RF distance for all of the metrics. The only aberrance that occurs is in the classification of the *Lepilemur Hubbardorum* and *Microcerbus murinus*, which are at the same taxonomical level according to NCBI and therefore it is not considered an error.

The fifth dataset is at the taxonomical level of Catarrhini and compares the group of Cercopithecoidea and Hominoidea.

The phylogenetical trees from the fifth dataset are presented below:

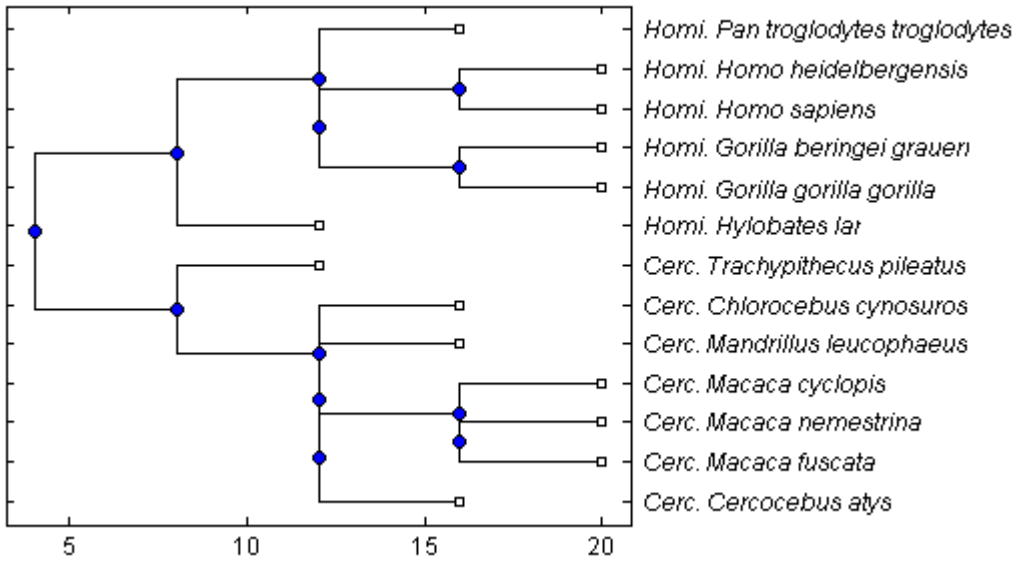

Figure 29: NCBI reference tree of Cercopithecoidea and Hominoidea.

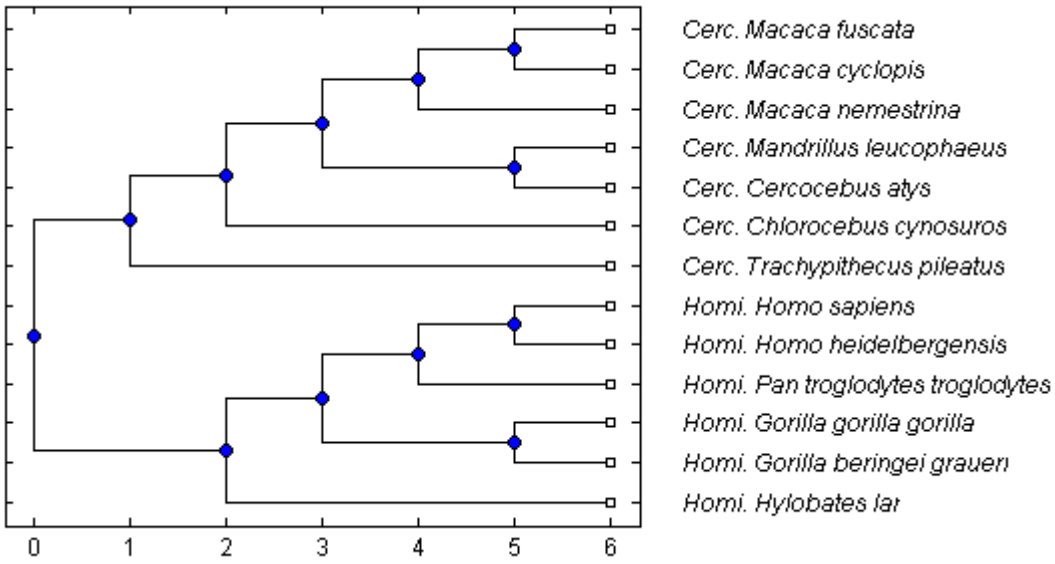

Figure 30: LZ cladogram based on metric no. 3 Cercopithecoidea and Hominoidea.

Table 8: RF distance between the 4 metrics and the alignment based algorithms compared to the NCBI reference of the Catarrhini.

Methods compared:	LZ Metric 1	LZ Metric 2	LZ Metric 3	LZ Metric 4	Jukes-Cantor	Kimura	Tamura
$RFD_{REF:}$	0	0	0	0	0	0	0
Success rate:	100%	100%	100%	100%	100%	100%	100%

The evolutionary distance of the fifth dataset sequences is fairly close but the results created by the different metrics are the same. The only missclassifications that occur in between the trees are the placement of *Pan troglodytes*, the Homo, the Gorilla subgroup and the placement of *Mandrillys leucophaeus*, *Cercocebus atys* and the Macaca group. According to the NCBI database the three species in both cases are a direct descendent of their common parent, meaning that for the NCBI database they are at the same level. For the reason described above, the results of the 5th dataset can be considered flawless.

The sixth dataset is at the taxonomical level of Hominoidea and compares the group of Hylobatidae and Hominidae. There are only 4 mitochondrial DNA sequence accessible from the family of the Hylobatidae and therefore the sixth dataset contains only 9 species.

The phylogenetical trees from the sixth dataset are presented below:

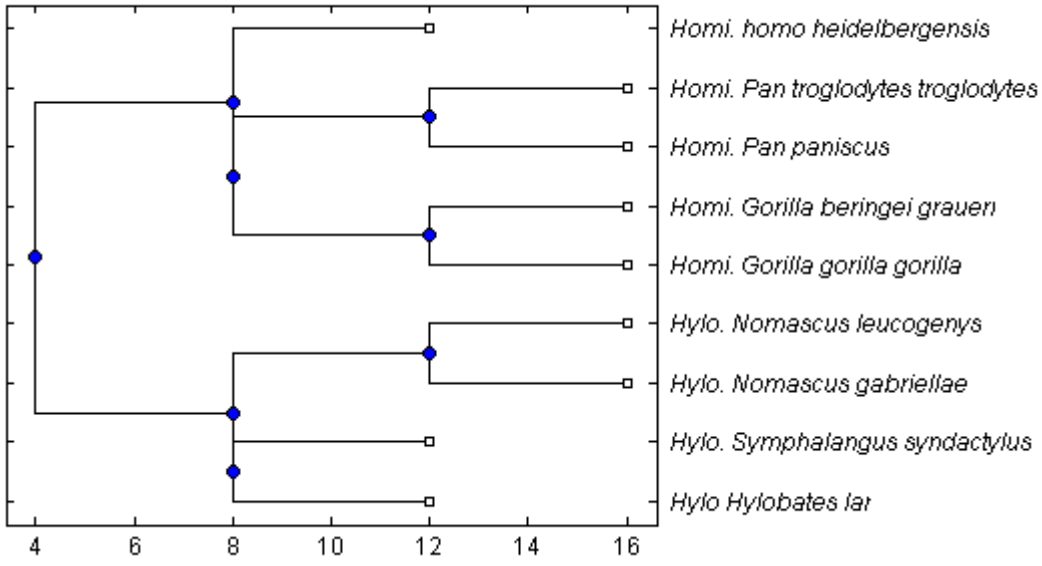

Figure 31: NCBI reference tree of Hylobatidae and Hominidae.

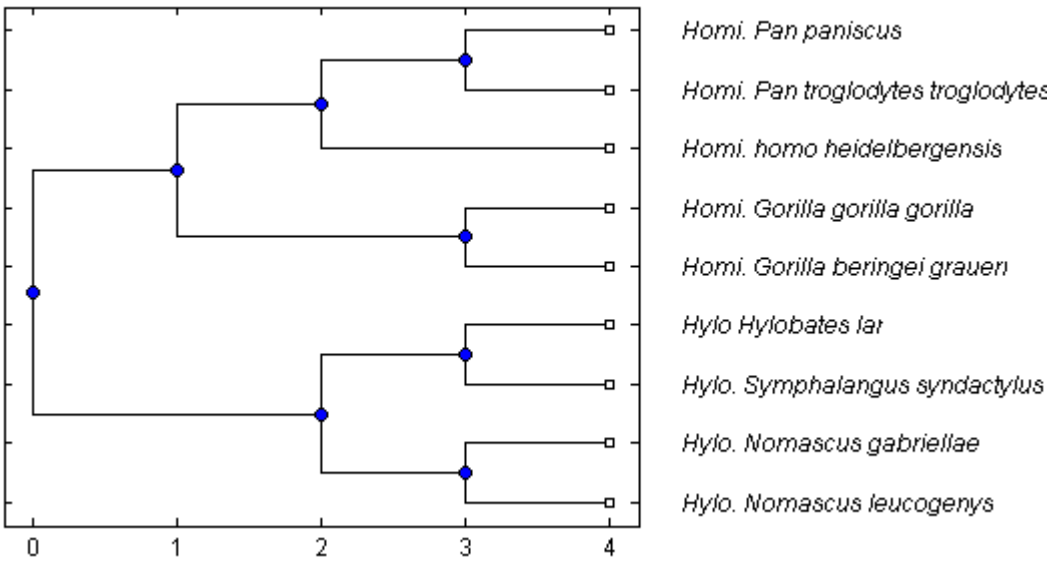

Figure 32: LZ cladogram based on metric no. 3 Hylobatidae and Hominidae.

Table 9: RF distance between the 4 metrics and the alignment based algorithms compared to the NCBI reference of the Hominoidea.

Methods compared:	LZ Metric 1	LZ Metric 2	LZ Metric 3	LZ Metric 4	Jukes-Cantor	Kimura	Tamura
$RFD_{REF:}$	0	0	0	0	0	0	0
Success rate:	100%	100%	100%	100%	100%	100%	100%

The results from the last dataset are similar to the previous one. The families are separated correctly with small variance inside of the families that are according to the NCBI reference: a flawless result.

The last dataset is the concatenation of the last three datasets, meaning it contains sequences belonging to the primate family. There are altogether 26 unique sequences.

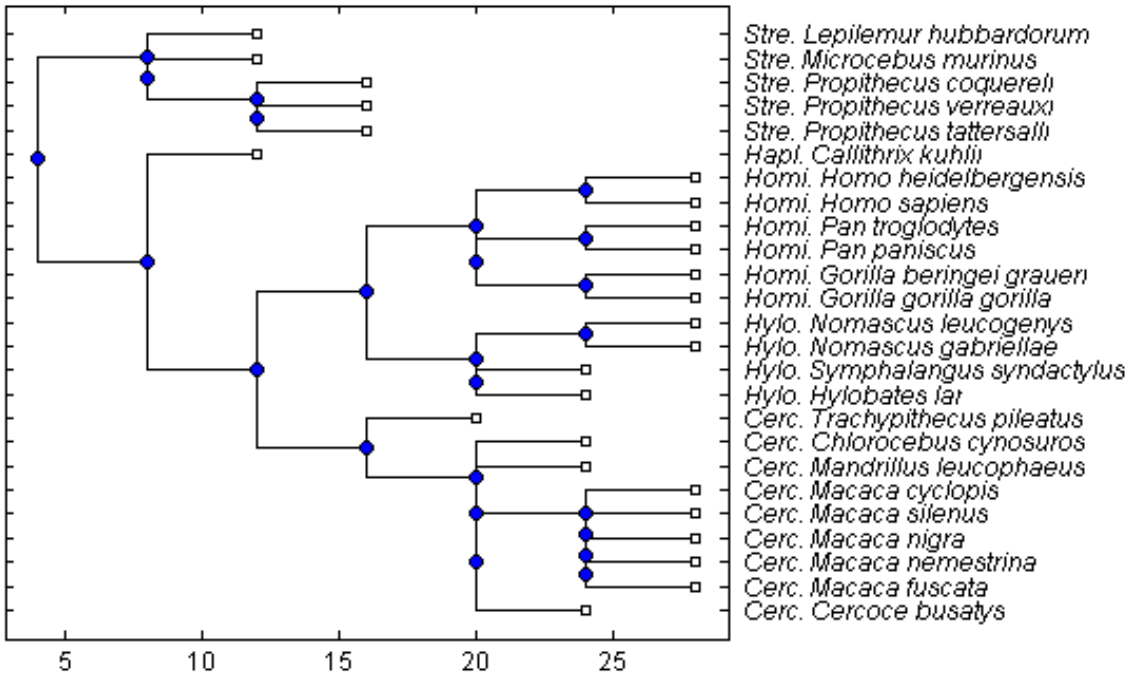

Figure 33: NCBI reference off all the sequences belonging to the taxonomic group of Primates.

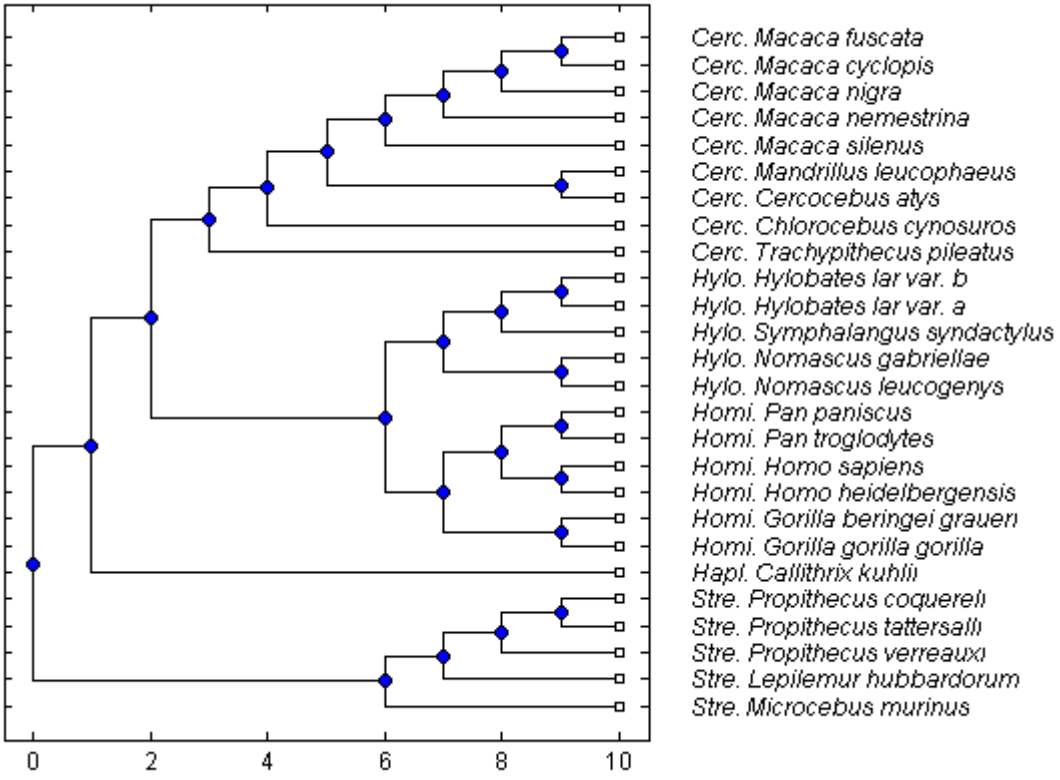

Figure 34: LZ cladogram based on metric no. 3 of all the sequences belonging to the taxonomic group of Primates.

The result of the last dataset is very positive. All of the sequences are classed correctly despite their large count:

Table 10: RF distance between the 4 metrics and the alignment based algorithms compared to the NCBI reference of all of the primates.

Methods compared:	LZ Metric 1	LZ Metric 2	LZ Metric 3	LZ Metric 4	Jukes-Cantor	Kimura	Tamura
$RFD_{REF:}$	0	0	0	0	0	0	0
Success rate:	100%	100%	100%	100%	100%	100%	100%

3.2.4 Conclusion of the second part of testing

In all of the different taxonomical level the LZ algorithm recognized the two different taxonomic branches and classed the majority of the sequences correctly. The LZ algorithm outclassed the alignment algorithms in the first three datasets, while equivalent in the last three datasets. The result of this testing shows, that the LZ algorithm as left as designed by [17] can distinguish different taxonomic families up to the level of Hominoidea.

While having positive results, an important flaw of the algorithm has been confirmed. The alignment based algorithms are able to estimate the evolution distance between the sequences by identifying the type of mutation that occurred in the sequence. This classification is simple as the sequences are aligned. This allows the alignment based algorithms such as Tamura or Kimura to surpass their predecessor Jukes-Cantor. They can weight the result by the type of mutation that occurred, which is additional information for the analysis, and leads to more precise result. The LZ algorithm as it stands doesn't have any such additional information. This leads to a misclassification when three groups are very distant from each other, even if two of them belong to the same taxonomic tree. In the second and the third dataset *Mus musculus*, *Triturs karelinii* and *Mantella madagascariensis* have been wrongly classified for this reason. Modifications to the LZ algorithm are described in the chapter 3.5.

3.3 Short sequences and proteomic sequences

It can be questioned, if the LZ algorithm performs consistently with different sequence length, especially if it is to be used on short proteomic sequences. With very short sequences the LZ complexity will be very small and therefore could be similar. In some cases due to variance a more distant sequence could yield a better result than a closer sequence. The table below displays the number of different combinations that can be created with *n* number of nucleotides and amino acids:

Table 11: Number of existing words for an *n* length string of nucleotides and amino acids

Number of nucleotides/amino acids:	Total nucleotide combinations:	Total amino acids combination:
1	4	21
2	16	441
3	64	9261
4	256	194481
5	1024	4084101
6	4096	85766121
7	16364	1801088541
8	65536	37822859361

The table 11 shows the number of different words that exist for a certain length of nucleotides/amino acids. The meaning of this table is, that if the biologically sequences were randomly generated, the probability of finding a certain word of a length of e.g. 6 in another sequence would be 1/4096, increased by the sequence length, for nucleotides and 1/85766121 for amino acids. As the number of combinations grows rapidly the algorithm is expected to work well even for short sequences.

3.3.1 Algorithm specification

The algorithm is left as designed by the authors of [17]. The reference sequences are taken from the NCBI database and are compared to the 4 metrics and the alignment methods.

3.3.2 Datasets

This dataset is composed of 5 sets of sequences of similar species. The sequences are coding regions of genes of apolipoprotein M (APOM), heat shock protein family A (HSPA8) and interleukin 2 (IL2). As the proteomic sequences of APOM and IL2 are available the algorithm will be tested on proteomic sequences as well. Some of the sequences are not sequenced yet and are only “predicted” sequences based on NCBI prediction algorithms. As the datasets were tested by the alignment algorithms and compared to the NCBI reference with positive result, this poses no problem. As some sequences are not available on the NCBI database, there are little differences, amongst the species, across the datasets.

The rounded average length of the sequences can be seen in the table below:

Table 11: Length of the sequences in the third part of testing

Gene	Nucleotide seq. length	Amino acid seq. length
APOM	576	191
IL2	466	154
HSPA8	1823	Unavailable

As the algorithm has been tested thoroughly with mitochondrial DNA, of approximately 16000bp, the sequences in this dataset have been chosen specifically short.

3.3.3 Results and discussion

As the species in the 5 datasets are similar, only one NCBI reference that includes all of the species will be shown and can be seen on the figure below:

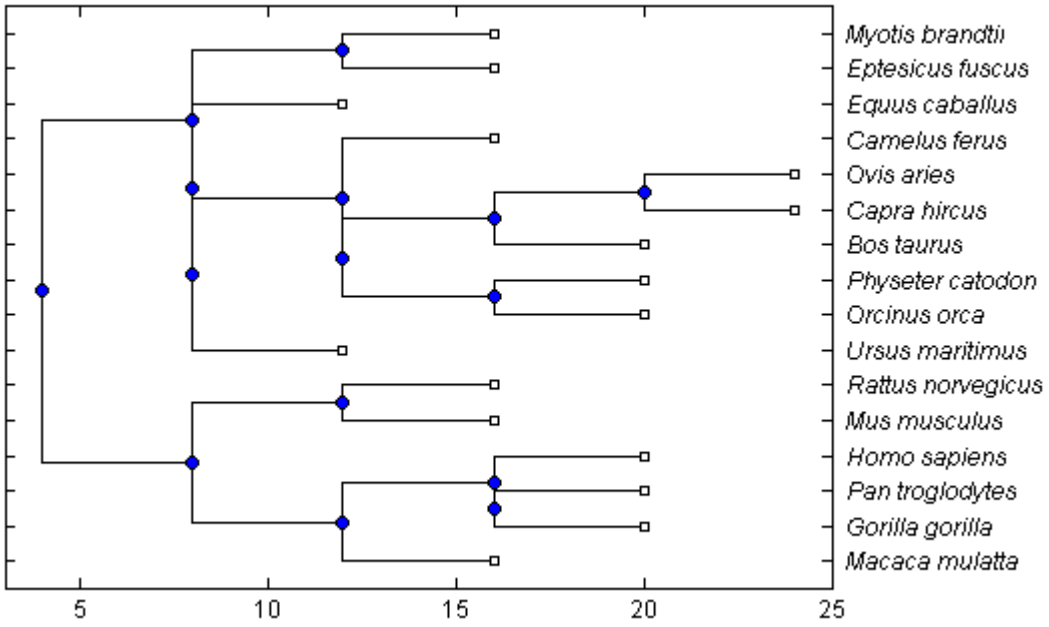

Figure 35: NCBI reference for the short sequence testing.

The species of this dataset belong to the class of Boreotheria. There are species such as the killer whale, the big brown bat, the house mouse or the gorilla, representing different subgroups. Amongst these groups several species have been added, therefore the dataset contains differences and similarities and can test the LZ algorithm on short sequences.

The phylogenetical tree of the APOM gene, constructed from nucleotides, is presented below, followed by the tables of the RF distances of nucleotide and proteomic sequences:

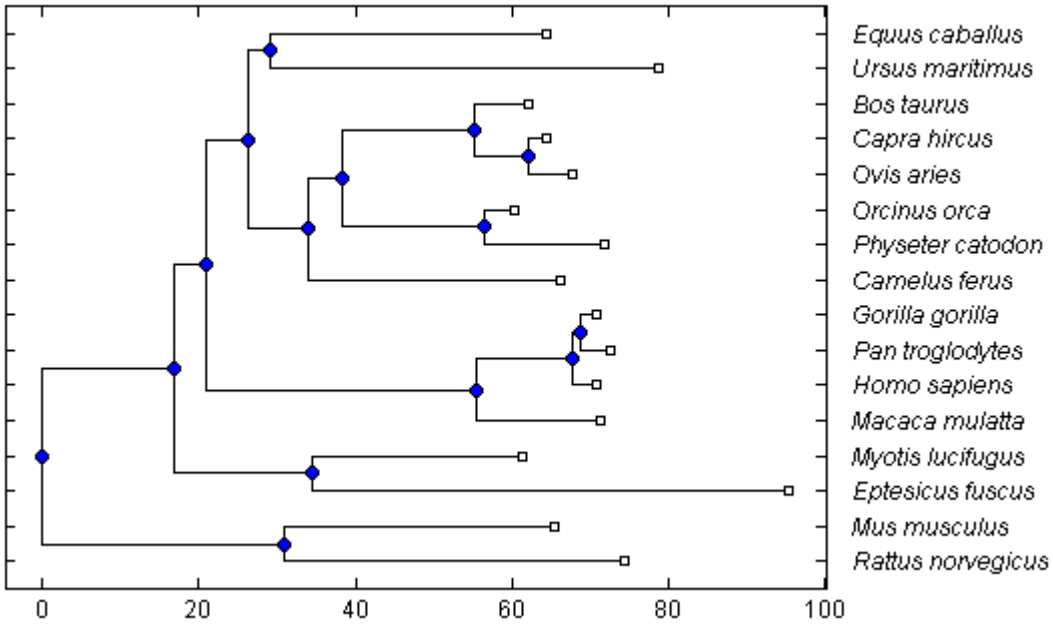

Figure 36: LZ phylogenetical tree based on the 3rd metric of the APOM gene.

Table 12: RF distance between the 4 metrics and the alignment based algorithms compared to the NCBI reference for nucleotide sequences of the APOM gene.

Methods compared:	LZ Metric 1	LZ Metric 2	LZ Metric 3	LZ Metric 4	Jukes-Cantor	Kimura	Tamura
RFD_{REF}:	4	4	4	4	4	4	4
Success rate:	85%	85%	85%	85%	85%	85%	85%

Table 13: RF distance between the 4 metrics and the Jukes-Cantor algorithm compared to the NCBI reference for proteomic sequences APOM gene.

Methods compared:	LZ Metric 1	LZ Metric 2	LZ Metric 3	LZ Metric 4	Jukes-Cantor
$RFD_{REF:}$	4	4	4	4	4
Success rate:	85%	85%	85%	85%	85%

The phylogenetical trees of all the LZ metrics and alignment algorithms are the same for both, the nucleotide and proteomic sequences. The RF distance from the NCBI reference is 4 in all of the cases. The reason for the difference being, the misplacement of the two groups of Glires (*Mus musculus* and *Rattus norvegicus*) and Chirpoteras (*Myotis lucifugus* and *Eptesicus furscus*). The trees constructed from the nucleotide and amino acids are compared in the next figure:

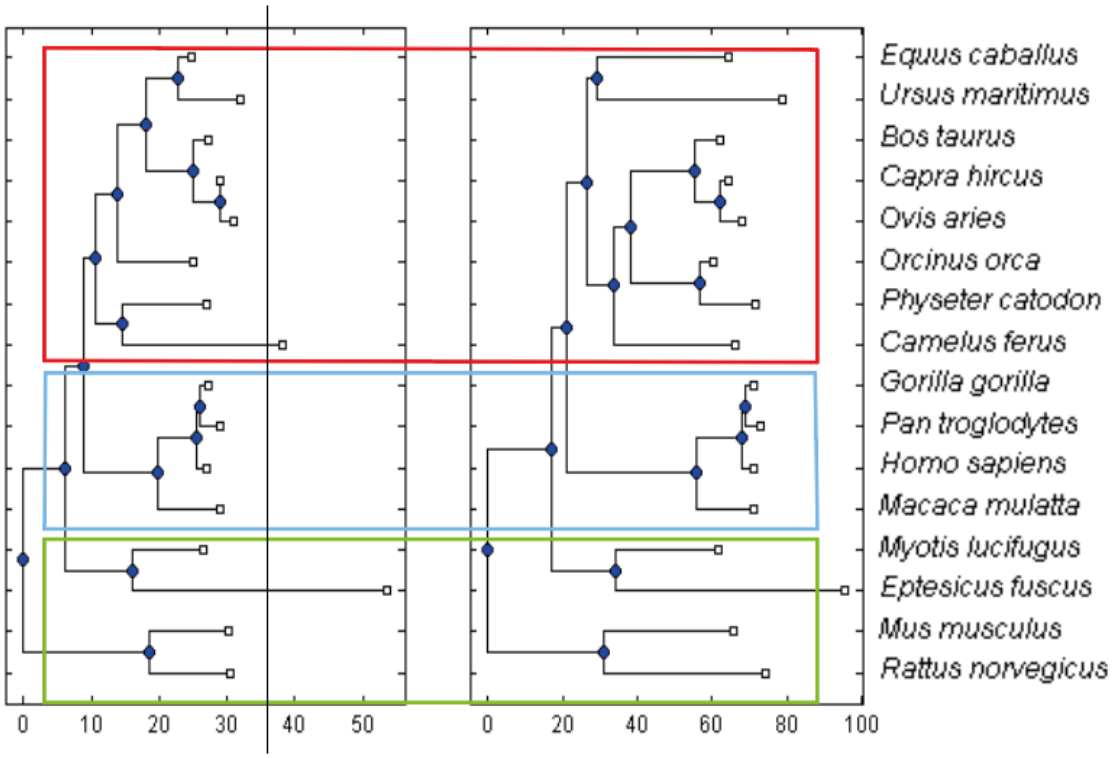

Figure 37: Comparison of the tree of the APOM gene constructed based on the proteomic sequences (left) and the nucleotide sequences (right) by the LZ algorithm and the distance metric d_3.

The phylogenetical trees are in accordance nods to nods and by the distances evaluated by the LZ algorithm, leading to two similarly shaped trees. The Person correlation coefficient [26] has been calculated based on the non-null values of the distance matrix'. The Person correlation coefficient is 0.9526 with the p-value of $8.11*10^{-63}$, meaning the two variables (distance matrix') are directly proportional with a high significance value.

The phylogenetical tree of the IL2 gene, constructed from nucleotides, is presented below, followed by the tables of the RF distances of nucleotides and proteomic sequences:

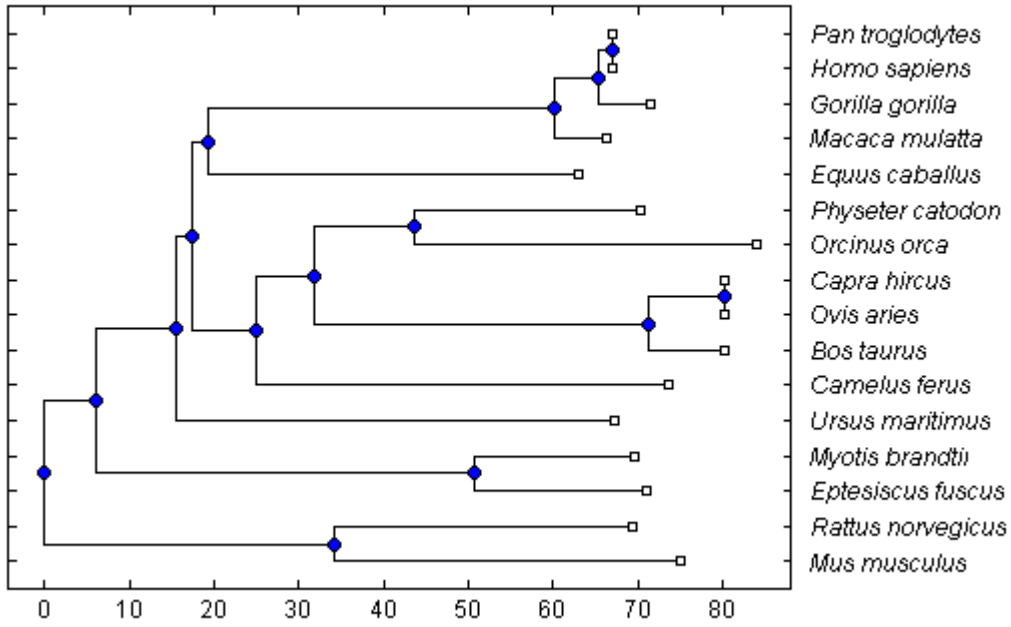

Figure 38: LZ phylogenetical tree based on the 3rd metric of the IL2 gene.

Table 14: RF distance between the 4 metrics and the alignment based algorithms compared to the NCBI reference for nucleotide sequences of the IL2 gene.

Methods compared:	LZ Metric 1	LZ Metric 2	LZ Metric 3	LZ Metric 4	Jukes-Cantor	Kimura	Tamura
$RFD_{REF:}$	8	8	8	8	0	4	4
Success rate:	72%	72%	72%	72%	100%	85%	85%

Table 15: RF distance between the 4 metrics and the Jukes-Cantor algorithm compared to the NCBI reference for proteomic sequences IL2 gene.

Methods compared:	LZ Metric 1	LZ Metric 2	LZ Metric 3	LZ Metric 4	Jukes-Cantor
$RFD_{REF:}$	6	6	6	6	4
Success rate:	79%	79%	79%	79%	85%

The result of the IL2 gene is worse than for the APOM gene. The groups of Glires and Chirportes are placed outside of their respective groups as previously but this time even the specie of Equus Cabellus is misplaced for the nucleotide generated tree. The trees constructed from the nucleotide and amino acids are compared in the next figure:

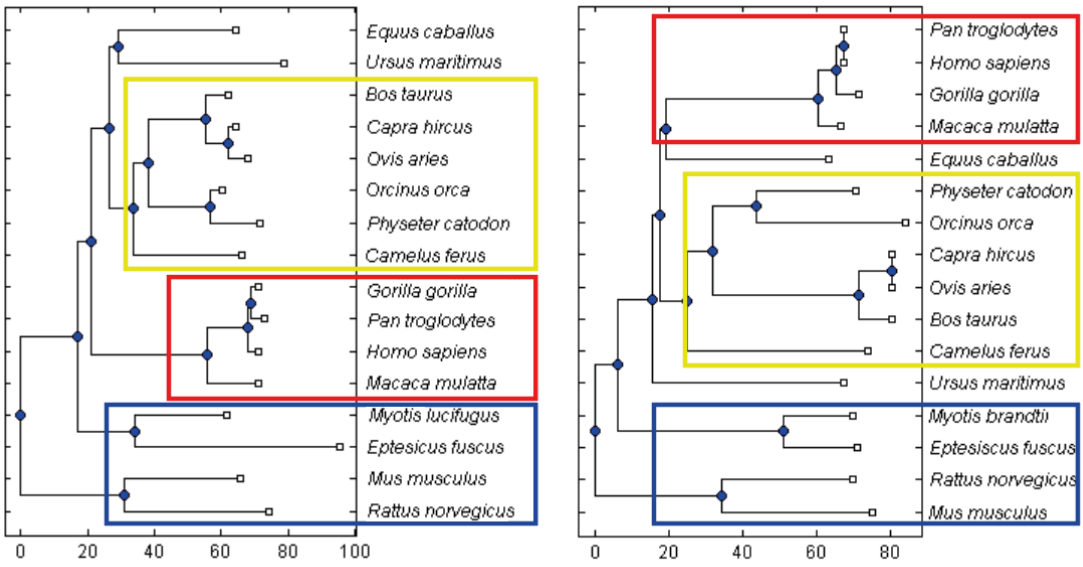

Figure 39: Comparison of the tree of the IL2 gene constructed based on the proteomic sequences (left) and the nucleotide sequences (right) by the LZ algorithm and the distance metric d_3.

Apart from the different placement of *Equus caballus*, the trees are in accordance. The Person correlation coefficient is 0.9724 with the p-value of $2.18*10^{-9}$ meaning the two variables (distance matrix') are directly proportional with a high significance value.

The last dataset of the HSPA8 gene corresponds to the reference in all of the cases but the Glires group. As this result has already been reported before only the table of the RF distances is presented:

Table 16: RF distance between the 4 metrics and the alignment based algorithms compared to the NCBI reference for nucleotide sequences of the HSPA8 gene.

Methods compared:	LZ Metric 1	LZ Metric 2	LZ Metric 3	LZ Metric 4	Jukes-Cantor	Kimura	Tamura
RFD_{REF}	2	2	2	2	2	2	2
Success rate:	93%	93%	93%	93%	93%	93%	93%

3.3.4 Conclusion of the third part of testing

The third phase of testing confirmed that the LZ algorithm is able to classify even short sequences with length around 400bp for nucleotides and 150 for amino acids. The resulting phylogenetical trees are not identical, but share the same baseline with their Person correlation coefficient being over 0.95 in both cases. The figures 37 and 39 prove that the algorithm is able to work with proteomic sequences as well as with nucleotide sequences. The reason being, as showed in the preface of this chapter, that even thought that the proteomic sequences are overall shorter, the size of the alphabet

compensates for this lack of length.

The best results were for the HSPA8 sequence, while the worst for the proteomic sequence of IL2. The main reason why the data are not exactly in accordance with the reference is the placement of the *Mus musculus* and *Rattus norvegicus* sequence.

Altogether it has been confirmed, throughout the first 3 parts of testing, that the LZ algorithm can distinguish between sequences up to the taxonomical level of Hominoidae, can work with proteomic sequences and is able to classify both, long and short sequences. In all of the cases the distance metric d_3 has had the best results. For this reason, in the following part of the paper, only the metric d_3 is going to be considered. The weaknesses of the algorithm are especially its inability to predict real evolutionary distance. This leads to misclassifications when sequences are as close as the Rhabdoviruses or when a group of sequences have a similar LZ complexity between themselves. These flaws will be looked upon in the next part of this paper.

3.4 Modifications of the LZ algorithm

This chapter proposes modifications to the original algorithm from [17]. Two different approaches were used. The first approach changes the algorithm itself to specialize it for the classification of biological sequences while the second introduces possibilities to weight the result of the algorithm

3.4.1 Static dictionary

This paper's main idea, as stated earlier, is that the conditional Kolmogorov complexity can be considered a measure describing the distance between two sequences. Moreover the equation (3) proves that there is a connection between Kolmogorov complexity and entropy for long strings and therefore the Kolmogorov complexity can be estimated via compression ratio. The chapter 2.2.1 points out via the Universal Similarity matrix that better the compression algorithm better the entropy estimate. This is the logic behind the algorithm in [17], which has been used for the previous analysis.

This statement is applicable to all compression algorithms, however it is possible to modify the compression algorithm to suit better the biological sequences. The LZ algorithm as used by the authors of [17], adds new words into the dictionary at every step and extends the left window sequence. With this configuration it will take less steps to generate the second sequence, as if the same word appears again, it will already be in the dictionary. The figure 40 demonstrates this phenomenon.

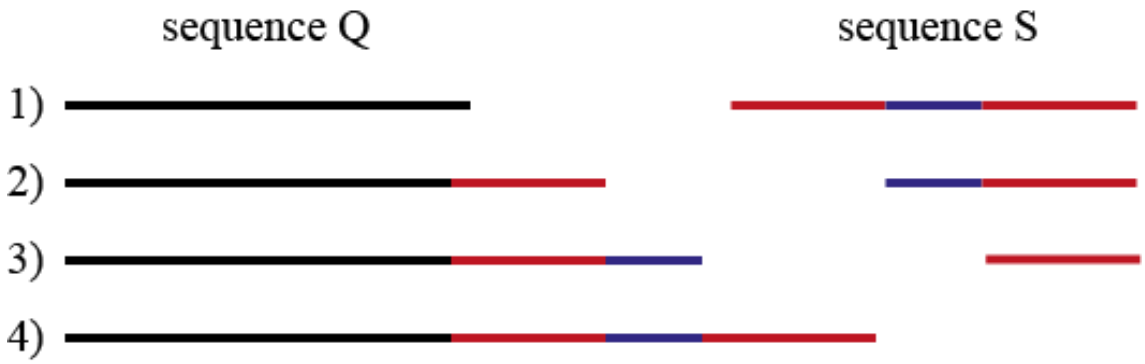

Figure 40: Two sequences being encoded by the original LZ algorithm

The black line on the figure represents the input sequence Q and the red and blue line represents the second sequence S, which is to be compressed. The red part of the sequence S symbolizes a string of the sequence that repeats itself twice at different locations. As the repeated sequence isn't part of the sequence Q, it will take many steps to the algorithm to generate the first red part of the sequence and reach the status in 2). The algorithm continues and adds the blue part of the sequence S, which also takes several steps, depending on the similarities between the two sequences. Once the blue part is generated, the red string is to be coded again, only this time it will only take one step to generate it, as it is already present in the dictionary. While this approach is very advantageous in compression, it can be seen that for biological comparison this approach is flawed. As the sequence Q did not possess the red part of the S sequence, the number of steps to generate S from Q should be larger.

This flaw can lead to the fact that if very distant sequences that are repetitive are to be compressed, their LZ distance will be small. For sequence comparison this feature is unwanted and therefore the following modification has been proposed:

The algorithm will no longer add new words to the dictionary from the second sequence (S) but will work only with a static dictionary generated from the sequence Q. This way even if the same sequence repeats many times, the algorithm will take the same amount of steps to generate each repeat, not leading to data falsification. The other benefit of this approach is that the algorithm become less time-consuming and is overall simpler. As the dictionary is no longer dynamic, new string search algorithms can be used for coding the dictionary, for instance the sequence Q could be encoded as a suffix tree in the first step, which would then lead to a very fast pattern search and could open a doorway to whole genome comparison [27, 28, 29, 30, 31].

This approach has been tested on all of the sequence presented in the first three parts and it leads to same or slightly better results than the algorithm used previously. The computing time difference between the two algorithms depends on the sequence length, as the implemented string pattern search isn't linear. The table below displays a selection of sequences used previously. The relevant parameters of the computer the algorithm run on were: processor: AMD FX(tm)-6100 Six-Core Processor 3.30GHz and RAM: 6,00GB Dual-Channel DDR3.

Table 17: Table of speed and accuracy of the LZ algorithm, modified LZ algorithm and Jukes-Cantor.

Sequence name:	Average sequence length:	Original alg. (1) Time [s]	% of correct nods based on RF distance of (1) to JC. Ref.	Modified algorithm (2) time [s]:	% of correct nods based on RF distance of (2) to JC. Ref.	Alignment algorithm time [s]:
Mitochondrial DNA of Primates 13seq	16743	552	92%	215	92%	259
1st mitochondrial 10 species	16468	241	100%	131	100%	149
Hepatitis A virus variants, 25 sequences	7400	441	96%	213	96%	218
MT 16S rRNA of 13 primates	1559	7.8	82%	5.1	82%	2.44
APOM 16 seq	575	3.63	85%	2.4	85%	0.7
APOM protein	190	1.1	85%	0,7	85%	0.25
IL2protein 16seq	154	0.87	72%	0.74	72%	0.27

The table 17 proves that the modified algorithm has a better computing time and comparable results to the original algorithm. For this reason the modified algorithm is going to be used in the next parts of this paper. A deeper analysis shows, that the modified algorithm has overall same result with “simple” sequences, meaning that the distances between the species varies and is not to close. The problematic sequence amongst these datasets is the Rhabdovirus variants. The figure below displays the outcome of the modified LZ algorithm:

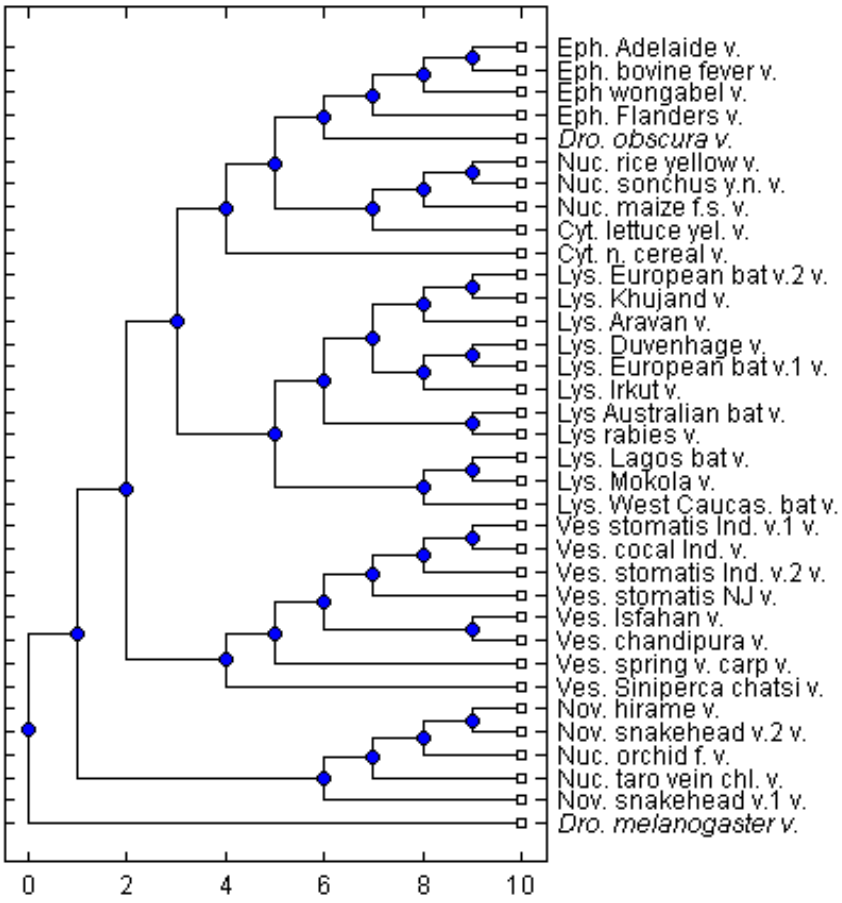

Figure 41: Cladogram of the rhabdovirus dataset computed by the modified LZ algorithm.

Both the original algorithm and the modified algorithm manage to separate the main classes of viruses, but fail to group the classes correctly between themselves. In the modified algorithm case the Nucleorhabdoviruses are wrongly conencted to the Ephemeroviruses and the vesiculoviruses and lysavirues are no longer clearly separeted from the other groups. On the other hand the Ephemeroviruses are no longer the most distant group, which is an improvement to the original.

Overall the modified algorithm has proved consistency on all the datasets with same results as the original. The computing time varies depending on the sequence, but for mitochondrial sequences is about two times faster than the original and even slightly faster than the alignment method.

3.4.2 Weighting the LZ complexity

For complex datasets the LZ algorithm can lead to very similar distance estimates in-between the sequences. The LZ complexities being close can generate flaws in the construction of the tree. In these cases the algorithm needs some other way to differentiate between the sequences. It is natural to question what other information can be mined from the LZ algorithm. This paper works with two approaches: the maximum length of words and the tranisition/transversion ration value.

The principle of the maximum length of words is very simple. At each step of the computation, when the algorithm finds the longest common string in both of the sequences, it saves the length of the string. At the end of the whole algorithm the words are ordered by their length and the *n* largest values are averaged. The algorithm shows best results for 5 longest words, therefore n is 5. At this point each couple of sequences possesses new information: the value of their averaged 5 longest words (the method is going to be referred as AV5). An AV5 matrix is created for all combinations of two sequences in the dataset. Once the whole dataset processed, the largest value of AV5 is selected, and the whole AV5 matrix is divided by the largest value, meaning that the values are in the range of 0 to 1, with 1 representing the combination of sequences with the highest AV5 value. In the next step the values are inverted, and therefore their range becomes from 1 to a theoretical infinite. The LZ complexity matrix is then multiplied by the AV5 matrix. After these steps the two sequences with the highest AV5 have their distance unchanged while the others are weighted. The sequences that had their AV5 value similar to the maximum AV5 value will see their LZ distance modified just slightly, while the sequences with a low value will see their LZ distance modified greatly. It is important to state that the premise for this modification is that the sequences that are closer from the evolutionary view point will share longer identical strings than the ones that are more distant. This can occur only if the mutations are localized on certain parts of the DNA sequence. The following figure shows the phylogenetical tree of the Rhabdovirus dataset after the implication of the AV5 for the third metric.

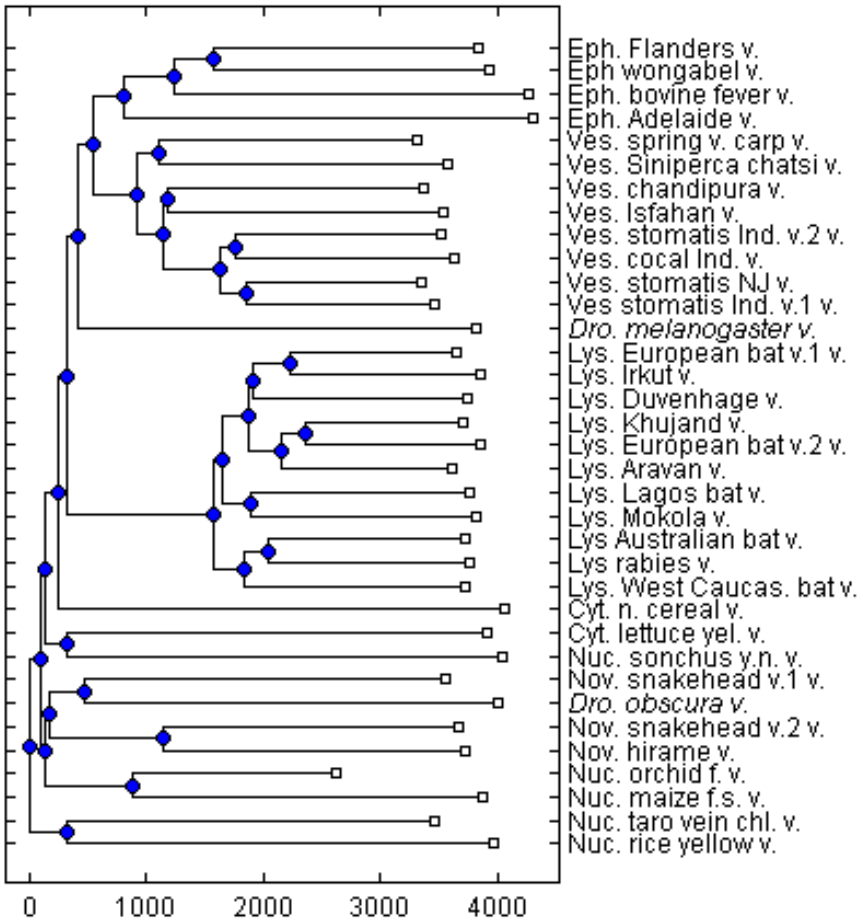

Figure 42: Cladogram of the rhabdovirus dataset computed by the modified LZ algorithm

The figure 42 should be compared with the figure 13 from chapter 3.1. where the original LZ distances can be seen. The families of Rhabdoviruses on the figure 42 are better differentiated than the ones on figure 13. The resulting figure, modified by AV5, is a big improvement to the previous classifications. The Ephemerovirus group, which has been previously placed as the furthest group of the dataset is now correctly grouped with the Vesiculoviruses. The groups of Novirhabdoviruses and Nucleoviruses are very distant from the rest of the groups, which corresponds to reality. Inside of the subgroups there are still a lot of transpositions, but overall implementing AV5 has been a big improvement.

The AV5 approach has been tested on the other datasets. It has similar results with the mitochondrial DNA, but as the results beforehand were already very good, the RF difference is usually only 2. For the datasets with short sequences the algorithm manages to classify the big families but makes some transpositions at the lowest taxonomic level. The results are overall acceptable, even placing the *Mus musculus* and *Rattus norvegicus* correctly, which were two problematic sequences in all of the datasets, but has worse results than the algorithm without AV5.

The only case where the AV5 clearly worsens the outcome is the Hepatitis A dataset. The phylogenetical tree of the Hepatitis A with LZ77&AV5 is presented next:

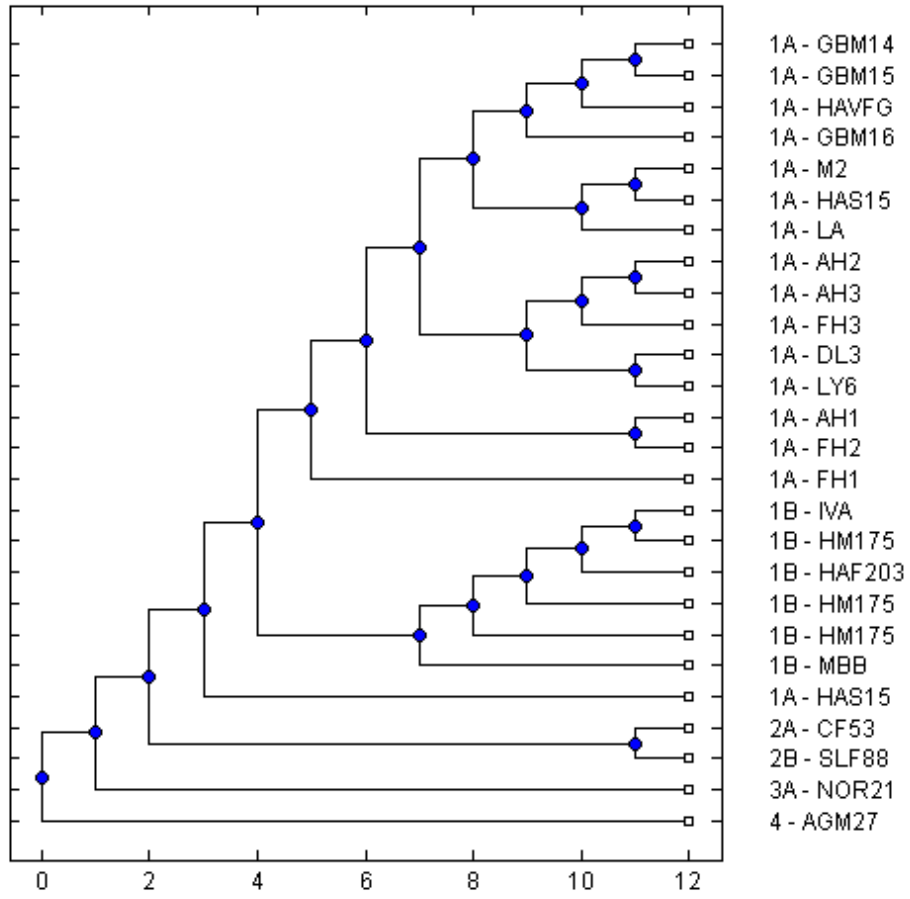

Figure 43: Cladogram of the hepatitis dataset computed by the modified LZ algorithm and AV5.

The main hepatitis groups are well recognized, even the two subgroups from the family denoted 1A have been correctly recognized, but inside of the groups, what has previously matched the reference is now in disorder. Almost every nod is wrong.

The difference between the Hepatitis A dataset to the others is that the sequences are very similar to each other. The other datasets sometimes also possess very similar LZ complexities, but their nominal value is larger. The hepatitis virus has not only very similar LZ complexities but also very small nominal values. Overall the AV5 approach works well to accentuate differences between groups, but is the reason of misclassifications inside of the groups. Therefore the AV5 weight should be used only on the right occasion.

The second weighting approach works with one of the important ideas behind the LZ algorithm for biological sequences. During the computation at each step, the LZ algorithm finds the longest common word for the two analyzed sequence and it adds one extra character to the string before continuing the building process. This mechanism is explained in the chapter 1.4.3. The original algorithm for data compression saves a lot of space by this mean, but it has a big meaning for the algorithm specialized for the classification of biological sequences as well The next example explains the value of this approach:

Let's consider the following two sequences:

Sequence 1): ACATAGTGACCCTCAGTAGGTAG

Sequence 2): ACATAGTGACCCACAGTAGGTAG

The Hamming distance between these two sequences is 1 and the difference is highlighted in red. Two scenarios will be considered onwards, the first scenario works with the classical LZ algorithm and the second scenario doesn't add the extra character to the sequence once the longest common word is found. The scenarios are presented in the following text:

Scenario 1:

Sequence 1): ACATAGTGACCCTCAGTAGGTAG

Sequence 2): ACATAGTGACCCACAGTAGGTAG

Step 1) The algorithm finds the longest common word: ACATAGTGACCC

Step 2) The algorithm adds the extra letter A to the string: ACATAGTGACCCA

Step 3) The algorithm finds the next longest common word: CAGTAGGTAG

The sequence 2) is encoded in two steps.

Scenario 2:

Sequence 1): ACATAGTGACCCTCAGTAGGTAG

Sequence 2): ACATAGTGACCCACAGTAGGTAG

Step 1) The algorithm finds the longest common word: ACATAGTGACCC

Step2) The algorithm doesn't add the extra letter A and continues searching Starting from A.

Step 3) The algorithm finds the next longest common word: ACA

Step 4) The algorithm finds the next longest word: GTAGGTAG

The sequence 2) is encoded in 3 steps.

This example does not only show that the number of steps (the LZ complexity) is influenced by adding one character to create a unique word but moreover in the first scenario the algorithm detects a mutation location. It can be expected, that if a word of more than 10 characters is found in both sequences, it is not a question of luck as the number of possible combination of nucleotides are 1048576, especially if the sequences are up to the length of mitochondrial DNA. This realization leads to the fact the nucleotide place following right after the long word has been found has a high probability to be a place where a mutation occurred.

Unfortunately as the algorithm cannot decide whether the mutation was an insertion, deletion or a substitution, all the mutations are considered as substitutions.

This is a weakness of the algorithm, but as LZ77 it is not an alignment based method it is not possible to decide which mutation occurred. For this reason this weight is an approximation and can be used only in specific situations.

Once the mutation point discovered, the algorithm saves the type of substitution in a variable. At the end of the algorithm a matrix of the frequencies of different substitution is at disposal. An example of such matrix can be seen below:

Table 18: An example of a mutation matrix

	A	**C**	**G**	**T**
A	0	24	64	14
C	23	0	23	12
G	5	12	0	12
T	25	54	24	0

The next step is inspired by the Kimura alignment method, which is based on the fact that amongst substitutional mutations transitions occur more often than transversions. This leads to the fact, that higher the ration of transversions/transitions is (the abbreviation TTr is going to be used), the more time there must have been for the transversions to occur and so the more distant the sequences are from each other.

The algorithm modified based on TTr simple multiplies the LZ complexity by this ratio. In this method the important factor is from which length of the longest common word should the algorithm consider the following nucleotide position to be a mutation (constant K). If the constant K is set too big, there won't be enough data and the result could be random. For this reason the constant K is chosen as follows: Length of the sequence ~= 4^K.

If the AV5 method and TTr are compared, the AV5 method works well for separating and correctly classifying big families of species that have a similar LZ complexity and the final outcome can be modified greatly. TTr is a decent method that changes the LZ value only slightly and is useful for classing sequences inside of the classes. Overall there is no big advantage to use the TTr modification as standalone. This approach shows good results with mitochondrial sequences from the taxonomic level of Primates and lower, but fails to classify the more distant mitochondrial sequences correctly, the standalone LZ algorithm is better. The TTr modification has slightly worse results with short sequences such as APOM and IL2. When tested on the Rhabdovirus dataset the outcome is not better than the simple LZ77 as can be seen on the figure below.

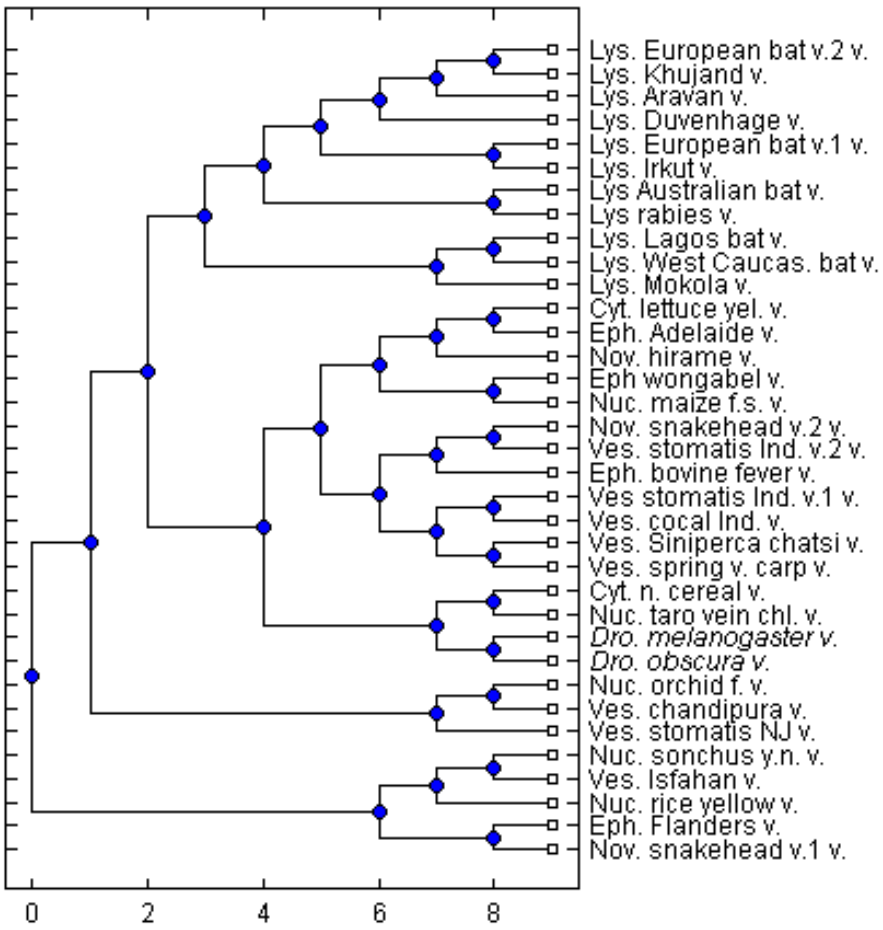

Figure 44: Cladogram of the rhabdovoris dataset computed by the modified LZ algorithm and TTr.

After the application of the TTr weight the algorithm mixes together some of the families. The TTr algorithm as a standalone doesn't bring good results.

The methods AV5 and TTr were designed in order to find additional information in the LZ77 algorithm that could describe the evolutionary distance for problematic sequences. The methods were designed so that one separates the main groups with a similar LZ complexity and the second one was designed as a light modification at the lowest taxonomical level. The methods can be used together in order to enhance these two aspects. The figure below displays the outcome with the most problematic - Rhabdovirus - dataset:

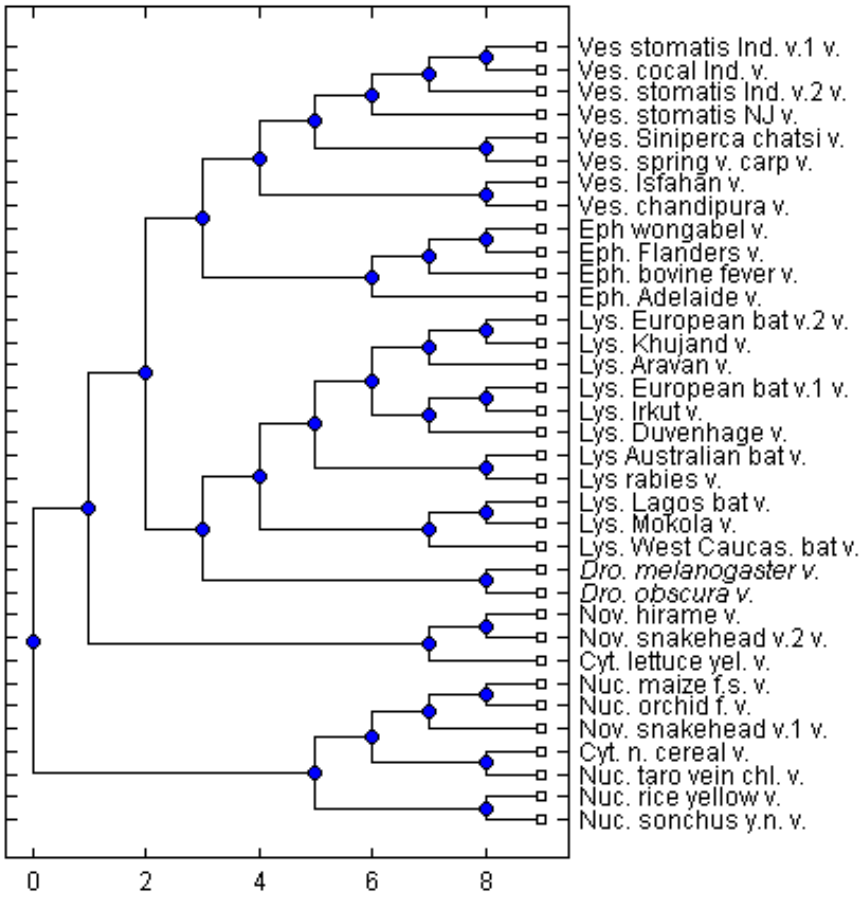

Figure 45: Cladogram of the rhabdovoris dataset computed by the modified LZ algorithm, TTr and AV5.

The result of the combination of AV5 and TTr on the Rhabdovirus dataset is a success. The family of the Vesiculoviruses is correctly associated with the Ephemeroviruses, as when AV5 was used, but this time even the classes of Drosophilia, Novirhbadobiruses and Nucleorhabdoviruses are grouped together correctly. There are still flaws remaining in the phylogenetical tree, such as the two sequences of Cytoviruses not being grouped together but the improvement compared to the first result is undeniable.

3.4.3 Conclusion of the new modifications

Three new approaches have been introduced in this chapter. Modification of the core of the LZ algorithm and two weighting methods.

The modification of the LZ algorithm works with a static rather than a dynamic dictionary, which fits better the biological classification methodology. This modification has proven to perform faster than the original algorithm and with same results. The LZ modified algorithm can be used universally.

Two weighting methods have been designed to work with problematic sequences. There is no reason to use them in normal cases, as the modified LZ algorithm performs

well as a standalone. The AV5 algorithm works with the maximum common words length and has successful results with the majority of sequences, improving the problematic Rhabdovirus sequence by a good margin. The strength of AV5 is accentuating differences between big groups of sequences. The TTr method is a lighter weight and is designed to improve the classification at a lower taxonomical level. As a standalone the algorithm doesn't perform well, but when combined with AV5, leads to good results for the Rhabdovirus sequence.

4 GRAPHICAL USER INTERFACE

In order to ease the usage of the developed methods a graphical user interface has been created. This chapter will describe the program.

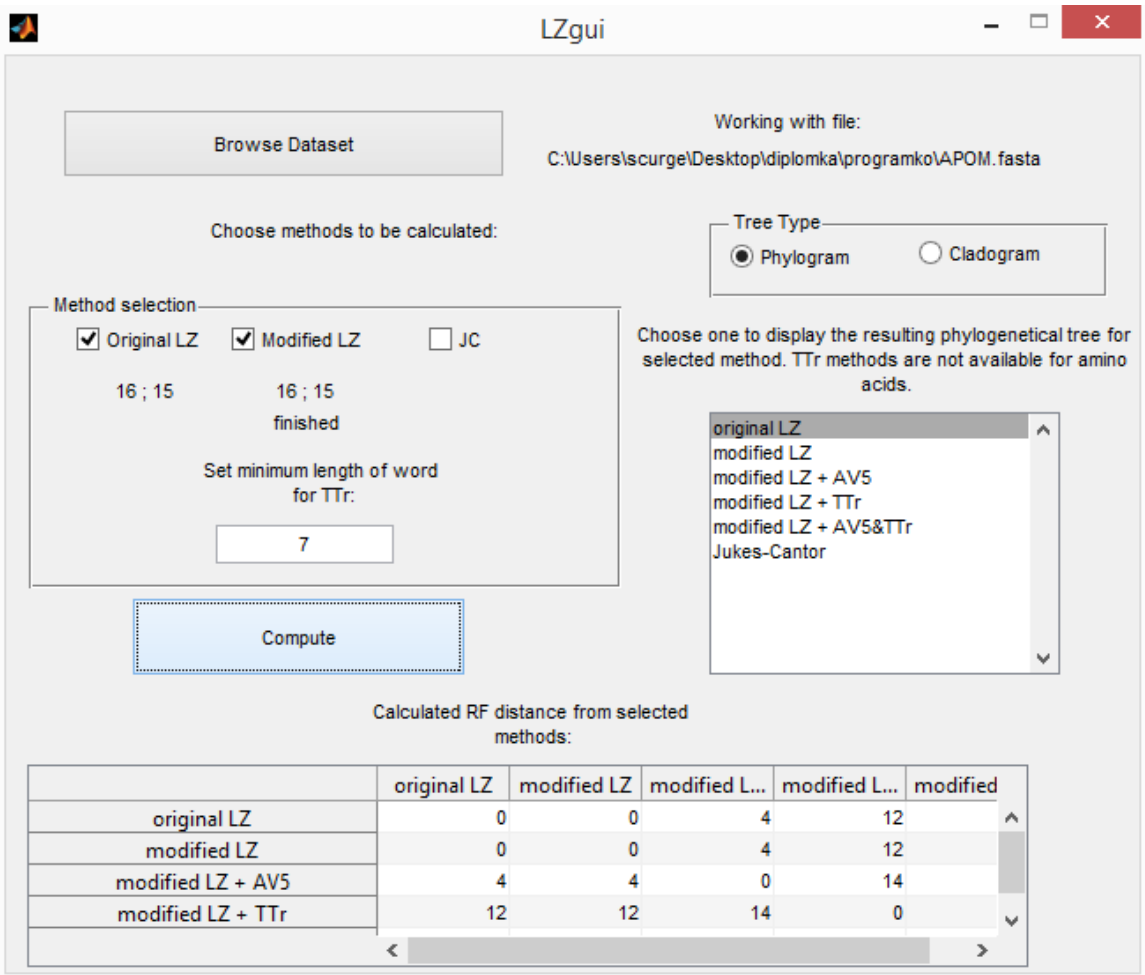

Figure 46: Grahical user interface for the usage of the LZ algorithm.

The figure 46 displays the whole GUI. At the start of the program the user is prompted to select a fasta file, which will be later processed. The path to the file and its file name is displayed on the top right side, so that when the files are changed or the user is multitasking, one does always know the working file. Once the file loaded the

user has the option to compute one of three methods, the original LZ algorithm, the modified LZ algorithm or the Jukes-Cantor reference. As throughout this paper the distance metric d_3 has always yielded the best results, the program is working only with the distance metric d_3 and there is no option to select a different metric. In the case of choice of the modified LZ algorithm, the AV5 and TTr weights are calculated as well. The user has the option to change the minimum common word length in the edit box.

Once the selection of methods is finished, the user should click on the compute button in order to calculate the distance matrixes. As the process of calculating can take several minutes, the text fields below the method checkboxes shows the current status of the algorithm and the last text field will display “finished” once the last selected method has been computed. The calculated data is used to compute the phylogenetical trees based on the neighbor-joining algorithm and the RF distance between all of the computed trees is displayed in the table at the very bottom of the GUI. Depending on the number of trees calculated the size of the table changes.

In the next step the user has the option to display the phylogenetical trees. Two choices are offered: either to display the phylogenetical tree as constructed by neighbor-joining or as cladograms. In order to visualize the tree, one has to click on one of the fields in the listbox on the right hand side. If the element the user clicks on hasn’t been computed, nothing will happen.

The GUI automatically recognizes if the input sequence is a nucleotide sequence or amino acid sequence and works for both. In the case of the input being an amino acid sequence the TTr methods are unavailable and therefore aren’t computed.

5 CONCLUSION

This master thesis deals with the problematic of classifying biological sequences utilizing the non-alignment based methods of lossless data compression. As numerous different methods of data compression exist a literal research has been made and the original Lempel-Ziv dictionary method has been chosen. This method allows estimating the entropy of two concatenated sequences by the LZ complexity, a measure based on the counts for one sequence to be built from another sequence.

The chapters following the literal research are testing the LZ approach thoroughly. 15 datasets haven been acquired from the NCBI database. The datasets were chosen in order to test a particular aspect of the LZ algorithm in each part of the testing.

A quick assessment of the LZ method has been made in the first part of the testing. 4 datasets of different length and different complexity were chosen. The outcome of the chapter showed that the LZ complexity is a measure that can describe evolutionary distances, but that in some cases a mismatch of equally distant groups of species can occur.

The second part of testing has been designed to inspect the ability of the LZ algorithm to classify sequences of different taxonomical levels, starting at the level of Euteleostomi and finishing at the level of Hominoidea. Out of the 6 datasets of mitochondrial DNA the LZ algorithm performed equally or better than its alignment based counterparts. In the first three very foreign datasets the LZ algorithm outclassed the Jukes-Cantor method by a large margin. The LZ algorithm had the success rate of classifying the species correctly, according to the RF distance compared to the NCBI reference, of 91%, 73% and 92% while the Jukes-Cantor algorithm only 55%, 64% and 66% respectively. The alignment methods of Kimura and Tamura performed similarly to the LZ algorithm. The three last datasets of Primates to Hominoidea were flawless for all of the methods.

As the mitochondrial sequences' length is of order of 16000bp, the LZ algorithm has been tested on shorter sequences. This has been accomplished in the third part of testing where 3 different genes of length between 400bp – 1800bp were chosen. The LZ algorithm once again performed as well as its alignment counterparts, except for the IL2 gene, where the result is slightly worse. This chapter also tested two proteomic sequences of the same genes. The results were in accordance to the NCBI reference. The Pearson correlation coefficient to the sequences computed from nucleotides was over 0.95 with a very high significance value. The fact that the LZ algorithm can work with amino acid sequences has been confirmed.

The fourth part of testing proposes innovations to the LZ algorithm. The algorithm has been modified in order to work with a static dictionary rather than with a dynamic one. This approach has been tested on all of the datasets with results equal to the original LZ algorithm. As the method is around two times faster than the original method and that the algorithm performs better with repetitive sequences, the modified algorithm has proven it can be utilized universally. Two weighting methods have been introduced to try to bring additional information into the algorithm, so that problematic sequences can be classified properly. The first method utilizes the average length of the 5 longest common words (AV5), while the second one utilizes the

transition/transversion ratio (TTr). The AV5 weight has been designed to separate groups of species while the TTr is a lighter weight, designed to specify the classification of the closest sequences. These weighting methods were designed to differentiate between sequences in problematic datasets and therefore change the outcome of the original LZ classification. A great improvement has been achieved while combining the AV5 and TTr method for the Rhabdovirus dataset, with the resulting tree's correctness outclassing all the other tested methods. On the other hand the TTr method yields very bad results when used alone and the AV5 method rarely improves the result of the LZ algorithm for well differentiated species. For this reason the AV5 and TTr methods cannot be used universally.

The last chapter of the paper presents the Graphical User Interface that has been implemented in order to ease the usage of the described methods.

This master thesis has proven that the compression technique based on the LZ algorithm can be used to classify both DNA and proteomic sequences. The algorithm is particularly strong when comparing very distant species, as the algorithm outclassed the alignment algorithms. The algorithm's speed at its current state is comparable to the alignment based techniques but the modification of the algorithm to use a static dictionary opens the option to implement advanced string matching techniques and further improve the speed of the algorithm.

REFERENCES

[1] RAMEZ MINA, DHUNDY BASTOLA, HESHAM H. ALI, Compression-based Alghorithms for Comparing Fragmented Genomic Sequences. College of Information Science and Technology, University of Nebraska at Omaha, Omaha, NE, USA, BIOTECHNO 2013.

[2] XIWU YANG, TIANMING WANG. A novel statistical measure for sequence comparison on the basis of k-word counts. Journal of Theoretical Biology 318, page 91-100, 2013

[3] KHALID SAYOOD, *Lossless Compression Handbook*, 2003 Elsevier Science (USA), Academic Press

[4] SALOMON D., MOTTA G., Handbook of Data Compression, fifth edition. ISBN 978-1-84882-902-2

[5] FERRAGINA P., GIANCARLO R., GRECO V., MANZINI G., VALIENTE G., Compression-based classification of biological sequences and structures via the Universal Similarity Metric: experimental assessment. 13. July 2007

[6] GIANCARLO R., SCATURRO D., UTRO F., Textual data compression in computational biology: a synopsis. Department of applied Mathematics, University of Palermo, Italy, February 2009.

[7] SNYDER TIM, Overview and Comparison of Genome Compression Algorithms. University of Minnesota. Morris, Departement of Computer Science.

[8] RANA J.M.S, *Bioinformatics – Tools and applications*, Uttarakhand State Biotechnology department, 2012

[9] NOUR S. BAKR, AMR A. SHARAWI. DNA Lossless compression algorithm: review. American Journal of Bioinformatics Research pages 72-81, 2013.

[10] XIN CHEN, SAM KWONG, MING LI, A Compression algorithm for DNA sequences. IEEE July/August 2001.

[11] KURUPPU S., BERESFORD-SMITH B., CONWAY T., ZOBEL J., Iterative Dictionary Construction for Compression of Large DNA Data Sets, IEEE/ACM transictions on computational biology and bioinformatics, vol 9. No. 1, January-Februray 2012.

[12] LI M., CHEN X., LI X., MA B. The Similarity Metric, IEEE T. Inform. Theory 2004 page 3250-3264.

[13] HASAN H. OTU, KHALID SAYOOD, A new sequence distance measure for phylogenetic tree construction, Departement of Electrical Engeneering, University of Nebraska-Lincoln, April 2003

[14] MARCIA L.B., MESSIAS S., de LIMA M.A, YOSHIDA C.FT., GASPAR A. M., GALLER R., Genetic variability of hepatitis A virus strain HAF-203 isolated in Brazil and expression of the VP1 gene in in Escherichia coli. Mem. Inst. Oswaldo Cruz, Rio de Janeiro, Vol. 101, pages 759-766, November 2006

[15] ZIV J., LEMPEL A., A Universal Algorithm for Sequential Data Compression. IEEE transactions of information theory, vol IT-23, NO. 3, May 1977.

[16] CAO M. D., DIX T., ALLISON L., MEARS C., A simple statistical algorithm for biological sequence compression. Data Compression Conference, 2007. DCC, pages 43-52, march 2007.

[17] LONGDON B., OBBARD D.J., JIGGINS F.M, Sigma viruses from three species of Drosophila form a major new clade in the rhabdovirus phylogeny. Institute of Evolutionary Biology, University of Edinburgh, Department of Genetics, University of Cambridge, Cambridge, UK 2009.

[18] BROCCHIERI L., KARLIN S., Protein length in eukaryotic and prokaryotic proteome. Nucleid acide research. Published online 2005.

[19] SOKOL R. MICHENER C. A statistical method for evaluationg systematic relationships. University of Kansas, science bulletin 38, 1958

[20] JUKES T.H., CANTOR C.R., Evolution of Protein Molecules, New Yor: Academic Press, 1969

[21] ROBINSON D.R., FOULDS L.R., Comparison of phylogenetic trees, Mathematical Bioscience 1981

[22] MAYR E., Cladistic analysis or cladistic classification?, Journal of Zoological Systematics and Ecolutionary Research, 2009

[23] SAITOU N. NEI M., The neighbor-joining method: a new method for reconstructing phylogenetic trees., Molecular Biology and Evolution, July 1987.

[24] KIMURA M., A simple method for estimating evolutionary rates of base substitutions through comparative studies of nucleotide sequences, Journal of Molecular Evolution 1980

[25] TAMURA K. Estimation of the number of nucleotide substitutions when there are strong transition-transversion and G+C content biases, Molecular Biology and Evolution, 1992

[26] PEARSON K. Notes on regression and inheritance in the case of two parents. Proceedings of the Royal Societly of London, June 1895

[27] NAVARRO G., MAKINEN V., Compressed Full-Text Indexes, University of Helsinky and University of Chile. ACM Comput. Surv. April 2007

[28] KIM D. K., KIM M., PARK H., Linearized Suffix Tree: an Efficient Index Data Structure with the Capabilities of Suffix Trees and Suffix Arrays, Algorithmica 2008, October 2007

[29] ABOUELHODA M. I., KURTZ S., OHLEBSUCH E., Replacing suffix trees with enhanced suffix arrays, Journal of Discrete Algorithms 2 (2004)

[30] GEIEGERICH R., KURTZ S., STOYE J., Efficient implementation of lazy suffix trees. Software - practice and experience, june 2003

[31] KARKKAINEN J, SANDERS P., BURKHARDT S., LinearWork Suffix Array Construction, Journal of the ACM vol. 53. November 2006

[32] VINGA S., ALMEIDA J., Alignment-free sequence comparison-a review.. Bioinformatics, Oxford, England . March 2003.

APPENDIX

Sequences used in this master thesis:

MT 16S RNA dataset:

gi|459485530:1089-2650	*Papio papio* mitochondrion, complete genome;
gi|238866918:1095-2669	*Eulemur fulvus mayottensis* mitochondrion, complete genome;
gi|49146236:1624-3181	*Macaca mulatta* mitochondrion, complete genome;
gi|5835834:1095-2654	*Pongo abelii* mitochondrion, complete genome;
gi|5835820:1089-2646	*Hylobates lar* mitochondrion, complete genome;
gi|408772040:1092-2649	*Nomascus gabriellae* mitochondrion, complete genome;
gi|529217390:1092-2648	*Nomascus leucogenys* mitochondrion, complete genome;
gi|5835163:1094-2651	*Pongo pygmaeus* mitochondrion, complete genome;
gi|195952353:1091-2648	*Gorilla gorilla gorilla* mitochondrion, complete genome;
gi|196123578:1667-3224	*Homo sapiens neanderthalensis* mitochondrion, complete genome;
gb|HQ260949.1|:1621-3179	*Homo sapiens* isolate S1 mitochondrion, complete genome;
gi|5835135:1091-2649	*Pan paniscus* mitochondrion, complete genome;
gi|5835121:1090-2647	*Pan troglodytes* mitochondrion, complete genome;

Sequences from the first part of testing mitochondrial dataset:

gi|643689|dbj|D38114.1|GORMTC	*Gorilla gorilla* mitochondrial DNA, complete genome
gi|12772|emb|X61145.1|	*Balaenoptera physalus* mitochondrial

	complete genome
gi|644494|dbj|D38115.1|ORAMTD	*Pongo pygmaeus* mitochondrial DNA, complete sequence
gi|2052151|emb|Y07726.1|	*Ceratotherium simum* complete mitochondrial DNA sequence
gi|414126|emb|X72204.1|	*Balaenoptera musculus* mitochondrial DNA complete genome
gi|13003|emb|V00662.1|	*H.sapiens* mitochondrial genome'
gi|1632801|emb|X99256.1|	*Hylobates lar* complete mitochondrial DNA sequence
gi|854269|emb|X14848.1|	*Rattus norvegicus* mitochondrial genome
gi|577571|emb|X79547.1|	*Equus caballus* mitochondrial DNA complete sequence
gi|13838|emb|V00711.1|	*Mus musculus* mitochondrial genome

Hepatitis A dataset:

gi|222597|dbj|D00924.1|SHVAGM27	Simian *hepatitis* A virus gene for polyprotein, complete cds;
gi|4001732|dbj|AB020564.1|	*Hepatitis* A virus genomic RNA, complete sequence, isolate AH1;
gi|4001734|dbj|AB020565.1|	*Hepatitis* A virus genomic RNA, complete sequence, isolate AH2;
gi|4001736|dbj|AB020566.1|	*Hepatitis* A virus genomic RNA, complete sequence, isolate AH3;
gi|52789965|gb|AY644676.1|	*Hepatitis* A virus isolate CF53/Berne, complete genome;
gi|33324701|gb|AF512536.1|	*Hepatitis* A virus isolate DL3, complete genome;
gi|603025|emb|X83302.1|	*Hepatitis* A virus complete genome;
gi|4001738|dbj|AB020567.1|	*Hepatitis* A virus genomic RNA, complete sequence, isolate FH1;
gi|4001740|dbj|AB020568.1|	*Hepatitis* A virus genomic RNA, complete sequence, isolate FH2;
gi|4001742|dbj|AB020569.1|	*Hepatitis* A virus genomic RNA, complete sequence, isolate FH3;
gi|443846|emb|X75215.1|	*Hepatitis* A virus GBM/WT RNA;
gi|443844|emb|X75214.1|	*Hepatitis* A virus GBM/FRhK RNA;
gi|443848|emb|X75216.1|	*Hepatitis* A virus GBM/HFS RNA;

gi|8810242|gb|AF268396.1|	*Hepatitis* A virus polyprotein precursor, gene, complete cds;
gi|329582|gb|M14707.1|HPA	*Hepatitis* A virus (wild-type) RNA, complete genome;
gi|9626732|ref|NC_001489.1|	*Hepatitis* A virus, complete genome;
gi|329594|gb|M16632.1|HPAA	*Hepatitis* A virus (attenuated) RNA, complete genome;
gi|62310|emb|X15464.1|	Human *hepatitis* A virus (HAV) strain HAS-15 mRNA for viral proteins VP1-4, 2A, 2B and 2C;
gi|109390447|gb|DQ646426.1|	*Hepatitis* A virus strain IVA, complete genome;
gi|329596|gb|K02990.1|HPAACG	Human *hepatitis* A virus, complete genome;
gi|19550900|gb|AF485328.1|	*Hepatitis* A virus isolate LY6, complete genome;
gi|62526564|gb|AY974170.1|	*Hepatitis* A virus strain M2 polyprotein mRNA, complete cds;
gi|329606|gb|M20273.1|HPACG	Human *hepatitis* virus type A RNA, complete genome;
gi|74381880|emb|AJ299464.3|	*Hepatitis* A virus polyprotein, genomic RNA, strain NOR-21;
gi|50295436|gb|AY644670.1|	*Hepatitis* A virus strain SLF88, complete genome;

Rhabdovirus dataset:

gi|9633477|ref|NC_000903.1|	Snakehead rhabdovirus complete genome
gi|948298106|ref|NC_028255.1|	Cocal virus Indiana 2, complete genome
gi|947834932|ref|NC_028246.1|	Adelaide River virus isolate DPP61, complete genome
gi|946699517|ref|NC_028235.1|	Flanders virus isolate BE AN 781455, complete genome
gi|761546856|ref|NC_009528.2|	European bat lyssavirus 2 isolate RV1333, complete genome
gi|55770806|ref|NC_006429.1|	Mokola virus, complete genome
gi|9635147|ref|NC_002251.1|	Northern cereal mosaic virus, complete genome
gi|701219253|ref|NC_025385.1|	Khujand lyssavirus, complete genome
gi|700075168|ref|NC_025377.1|	West Caucasian bat virus, complete genome
gi|700074710|ref|NC_025353.1|	Vesicular stomatitis Alagoas virus Indiana 3, complete genome

gi|667699573|ref|NC_024487.1|	*Drosophila subobscura* Nora virus, complete genome
gi|664651929|ref|NC_024473.1|	Vesicular stomatitis New Jersey virus isolate NJ1184HDB, complete genome
gi|149944272|ref|NC_009608.1|	Orchid fleck virus genomic RNA, segment RNA 1, complete sequence
gi|256535775|ref|NC_013135.1|	*Drosophila melanogaster* sigma virus AP30 N, P, X, M, G and L genes, genomic RNA, isolate AP30
gi|471237017|ref|NC_020810.1|	Duvenhage virus isolate 86132SA, complete genome
gi|471237011|ref|NC_020809.1|	Irkut virus, complete genome
gi|471237005|ref|NC_020808.1|	Aravan virus, complete genome
gi|471236999|ref|NC_020807.1|	Lagos bat virus isolate 0406SEN, complete genome
gi|471236993|ref|NC_020806.1|	Isfahan virus N gene, P gene, M gene, G gene and L gene, genomic
gi|471236987|ref|NC_020805.1|	Chandipura virus isolate CIN 0451, complete genome
gi|20428615|ref|NC_003746.1|	Rice yellow stunt virus, complete genome
gi|216967209|ref|NC_011639.1|	Wongabel virus, complete genome
gi|148724425|ref|NC_009527.1|	European bat lyssavirus 1, complete genome
gi|116536721|ref|NC_008514.1|	Siniperca chuatsi rhabdovirus, complete genome
gi|83659771|ref|NC_007642.1|	Lettuce necrotic yellows virus, complete genome
gi|134305391|ref|NC_001615.2|	Sonchus yellow net virus
gi|62327479|ref|NC_006942.1|	Taro vein chlorosis virus, complete genome
gi|50234098|ref|NC_005974.1|	Maize fine streak virus, complete genome
gi|34610114|ref|NC_005093.1|	Hirame rhabdovirus, complete genome
gi|17158068|ref|NC_003243.1|	Australian bat lyssavirus, complete genome
gi|14336454|ref|NC_002803.1|	Spring viraemia of carp virus, complete genome
gi|10086561|ref|NC_002526.1|	Bovine ephemeral fever virus, complete genome
gi|9628892|ref|NC_001724.1|	Snakehead retrovirus, complete genome
gi|9627229|ref|NC_001560.1|	Vesicular stomatitis Indiana virus, complete

gi|9627197|ref|NC_001542.1|	Rabies virus, complete genome genome

Sequences used for the second part of testing:

gi|1002164154|ref|NC_029423.1|	*Triplophysa dorsalis* mitochondrion, complete genome
gi|1011057294|ref|NC_029722.1|	*Chanodichthys ilishaeformis* mitochondrion, complete genome'
gi|107736076|ref|NC_008066.1|	*Chlorocebus sabaeus* mitochondrion, complete genome'
gi|148543101|ref|NC_009510.1|	*Ammotragus lervia* mitochondrion, complete genome
gi|194277529|ref|NC_011053.1|	*Propithecus coquereli* mitochondrion, complete genome
gi|240266584|ref|NC_012837.1|	*Limnonectes bannaensis* mitochondrion, complete genome
gi|281188575|gb|GU189676.1|	*Pan paniscus* isolate PP30 mitochondrion, complete genome
gi|304322880|ref|NC_014453.1|	*Lepilemur hubbardorum* mitochondrion, complete genome
gi|307777727|dbj|AP011544.1|	*Euphlyctis hexadactylus* mitochondrial DNA, complete genome'
gi|308746468|gb|HQ287897.1|	*Homo sapiens* isolate Ir4_10799_H mitochondrion, complete genome
gi|315142259|gb|HQ622775.1|	*Hylobates lar* isolate T11 mitochondrion, complete genome
gi|318039968|gb|HQ697277.1|	*Triturus karelinii* voucher 2360 mitochondrion, complete genome
gi|33438943|ref|NC_005055.1|	*Fejervarya limnocharis* mitochondrion, complete genome
gi|339906278|ref|NC_015792.1|	*Triturus karelinii* mitochondrion, complete genome
gi|3668119|emb|Y12025.1|	*Struthio camelus* complete mitochondrial genome
gi|394831045|ref|NC_018115.1|	*Aotus azarai azarai* mitochondrion, complete genome
gi|408772040|ref|NC_018753.1|	*Nomascus gabriellae* mitochondrion, complete genome
gi|41216035|gb|AY524977.1|	*Synodus variegatus* mitochondrion,

	complete genome
gi|435856991|ref|NC_020039.1|	*Cnemaspis limi* mitochondrion, complete genome
gi|457866490|dbj|AP013031.1|	*Mus musculus* mitochondrial DNA, complete genome, clone: P29mtC3H
gi|47156210|gb|AY612638.1|	*Macaca mulatta* mitochondrion, complete genome
gi|478432541|gb|KC603863.1|	*Homo sapiens* mitochondrion, complete genome
gi|507473161|gb|KC757404.1|	*Nomascus leucogenys* mitochondrion, complete genome
gi|507473259|gb|KC757411.1|	*Symphalangus syndactylus* mitochondrion, complete genome
gi|511347879|ref|NC_021391.1|	*Scomberomorus semifasciatus* strain GREY-SsPD211135 mitochondrion, complete genome
gi|511347893|ref|NC_021392.1|	*Scomberomorus munroi x Scomberomorus semifasciatus* strain Grey-SsCRC0703 mitochondrion, complete genome
gi|558479077|gb|KF680163.1|	*Trachypithecus pileatus* mitochondrion, complete genome
gi|568192363|ref|NC_023100.1|	*Homo heidelbergensis* mitochondrion, complete genome
gi|578003732|gb|KF914214.1|	*Gorilla gorilla gorilla* mitochondrion, complete genome
gi|5834995|ref|NC_001601.1|	*Balaenoptera musculus* mitochondrion, complete genome
gi|5835009|ref|NC_001602.1|	*Halichoerus grypus* mitochondrion, complete genome
gi|5835205|ref|NC_001700.1|	*Felis catus* mitochondrion, complete genome
gi|5835345|ref|NC_001788.1|	*Equus asinus* mitochondrion, complete genome
gi|5835568|ref|NC_001945.1|	*Dinodon semicarinatus* mitochondrion, complete genome
gi|5835820|ref|NC_002082.1|	*Hylobates lar* mitochondrion, complete genome
gi|604159100|gb|KJ179950.1|	*Dinodon rufozonatum* mitochondrion, complete genome
gi|619329278|gb|KJ631049.1|	*Jacana jacana* mitochondrion, complete

	genome
gi|619856195|gb|KF914213.1|	*Gorilla beringei graueri* mitochondrion, complete genome
gi|62184368|ref|NC_006915.1|	*Mus musculus molossinus* mitochondrion, complete genome
gi|628971407|ref|NC_024068.1|	*Jacana spinosa* voucher STRI:BC3332 mitochondrion, complete genome
gi|659104616|gb|KJ681495.1|	*Capreolus pygargus* isolate Cp8 mitochondrion, complete genome
gi|67082892|gb|DQ069713.1|	*Cercopithecus aethiops sabaeus* mitochondrion, complete genome
gi|683418040|gb|KM262190.1|	*Chlorocebus cynosuros* mitochondrion, complete genome
gi|699049576|ref|NC_025271.1|	*Capreolus pygargus* isolate Cp5 mitochondrion, complete genome
gi|71658036|ref|NC_007229.1|	*Cobitis sinensis* mitochondrion, complete genome
gi|722489592|ref|NC_025513.1|	*Macaca fuscata* mitochondrion, complete genome
gi|746000265|ref|NC_026120.1|	*Macaca nigra* mitochondrion, complete genome
gi|755573649|gb|KJ508413.2|	*Panthera tigris* isolate Malayan mitochondrion, complete genome
gi|757813536|gb|KP317203.1|	*Pan troglodytes troglodytes*, complete genome
gi|758374618|gb|KM679363.1|	*Macaca silenus* mitochondrion, complete genome
gi|769829586|ref|NC_026714.1|	*Triplophysa strauchii* mitochondrion, complete genome
gi|817526666|ref|NC_026976.1|	*Macaca nemestrina* mitochondrion, complete genome
gi|87299381|dbj|AB212225.1|	*Mantella madagascariensis* mitochondrial DNA, complete genome
gi|884997387|ref|NC_027449.1|	*Macaca cyclopis* isolate Mc-mitogm12060805 mitochondrion, complete genome
gi|906476668|ref|NC_027658.1|	*Callithrix kuhlii* mitochondrion, complete genome
gi|918020940|ref|NC_027740.1|	*Propithecus tattersalli* mitochondrion, complete genome

gi|944542639|ref|NC_028210.1|	*Propithecus verreauxi* mitochondrion, complete genome
gi|953245206|ref|NC_028442.1|	*Mandrillus leucophaeus* mitochondrion, complete genome
gi|955665322|gb|KR911720.1|	*Lycodon flavozonatus* mitochondrion, complete genome
gi|959125180|ref|NC_028592.1|	*Cercocebus atys* mitochondrion, complete genome
gi|966202078|ref|NC_028718.1|	*Microcebus murinus* isolate 920FAG mitochondrion, complete genome
gi|966202868|ref|NC_028730.1|	*Lycodon flavozonatus* mitochondrion, complete genome

Sequences used for the third part of testing:

CDS /gene="APOM	*Capra hircus* goat
CDS /gene="APOM	*Eptesicus fuscus* big brown bat
CDS /gene="APOM	*Physeter catodon* sperm whale
CDS /gene="APOM	*bos taurus*
CDS /gene="APOM	*camelus ferus*
CDS /gene="APOM	*gorilla gorilla gorilla*
CDS /gene="APOM	horse *equus caballus*
CDS /gene="APOM	human
CDS /gene="APOM	*macaca mulatta*
CDS /gene="APOM	*myotis lucifugus* little brown bat
CDS /gene="APOM	*ovis aries* sheep
CDS /gene="APOM	*pan troglodytes*
CDS /gene="APOM	*ursus maritimus* polar bear
CDS /gene="APOM	*orcinus orca* -killer
CDS /gene="APOM	*mus musculus*
CDS /gene="APOM	*rattus norvegicus*
CDS /gene="HSPA8	*Myotis lucifugus*
CDS /gene="HSPA8	*Physeter catodon*
CDS /gene="HSPA8	*bos taurus*
CDS /gene="HSPA8	*camelus ferus*
CDS /gene="HSPA8	*capra hiracus* goat
CDS /gene="HSPA8	*eptesicus fuscus*

CDS /gene="HSPA8	*equs caballus*
CDS /gene="HSPA8	*gorilla gorilla gorilla*
CDS /gene="HSPA8	human
CDS /gene="HSPA8	*macaca mulatta*
CDS /gene="HSPA8	*ovis aries* sheep
CDS /gene="HSPA8	*pan troglodytes*
CDS /gene="HSPA8	*ursus maritimus*
CDS /gene="Hspa8	*mus musculus*
'CDS /gene="Hspa8	*rattus norvegicus*
'CDS /gene="IL2	*Pan troglodytes*
'CDS /gene="IL2	*Physeter catodon*
CDS /gene="IL2	*bos taurus*
CDS /gene="IL2	*camelus ferus*
CDS /gene="IL2	*capra hircus*
CDS /gene="IL2	*eptesiscus fuscus*
CDS /gene="IL2	*equus caballus*
CDS /gene="IL2	*gorilla gorilla*
CDS /gene="IL2	human
CDS /gene="IL2	*macaca mulatta*
'CDS /gene="IL2	*myotis brandtii*
CDS /gene="IL2	*orcinus orca*
CDS /gene="IL2	*ovis aries*
CDS /gene="IL2	*ursus maritimus*
CDS /gene="Il2	*mus musculus*
CDS /gene="Il2	*rattus norvegicus*
gi|109733492|gb|AAI16846.1|	Il2 protein [*Mus musculus*]
gi|114052044|ref|NP_001040595.1|	interleukin-2 precursor [*Macaca mulatta*]
gi|117582508|gb|ABK41601.1|	interleukin-2 [*Ovis aries*]
gi|146198786|ref|NP_001078902.1	| interleukin-2 precursor [*Equus caballus*]
gi|149048754|gb|EDM01295.1|	interleukin 2 [*Rattus norvegicus*]
gi|28178861|ref|NP_000577.2|	interleukin-2 precursor [*Homo sapiens*]
gi|33330683|gb|AAQ10670.1|	interleukin-2 [*Bos taurus*]
gi|33330685|gb|AAQ10671.1|	interleukin-2 [*Capra hircus*]
gi|426345397|ref|XP_004040401.1|	PREDICTED: interleukin-2 [*Gorilla gorilla*

	gorilla]
gi|465981812|ref|XP_004265151.1|	PREDICTED: interleukin-2 [*Orcinus orca*]
gi|554541783|ref|XP_005865519.1|	PREDICTED: interleukin-2 [*Myotis brandtii*]
gi|114606419|ref|XP_518354.2|	PREDICTED: apolipoprotein M isoform X1 [*Pan troglodytes*]
gi|109070476|ref|XP_001112572.1|	PREDICTED: apolipoprotein M isoform X1 [*Macaca mulatta*]'
gi|148694705|gb|EDL26652.1|	apolipoprotein M [*Mus musculus*]
gi|149732042|ref|XP_001490472.1|	PREDICTED: apolipoprotein M [Equus caballus]
'gi|22091452|ref|NP_061974.2|	apolipoprotein M isoform 1 [*Homo sapiens*]
gi|426352431|ref|XP_004043716.1|	PREDICTED: apolipoprotein M isoform 1 [*Gorilla gorilla gorilla*]
gi|466089401|ref|XP_004286652.1|	PREDICTED: apolipoprotein M isoform X2 [*Orcinus orca*]
gi|548517981|ref|XP_005696653.1|	PREDICTED: apolipoprotein M isoform X2 [*Capra hircus*]