ELEN0060 - Information and coding theory
Project 2 - Source coding, data compression and channel coding

March 2021

The goal of this second project is to apply some of the principles seen in the lectures about source coding, data compression and channel coding. We ask you to write a *brief* report (*pdf* format) collecting your answers to the different questions.

The assignment must be carried out by group[1] of *two students* and the report and the scripts should be submitted as a *tar.gz* or *zip* file on Montefiore's Submission platform (`http://submit.montefiore.ulg.ac.be`) before **April 25 23:59**. Note that attention will be paid to how you present your results and your analyses. By submitting the project, each member of a group shares the responsibility for what has been submitted (*e.g.*, in case of plagiarism).

From a practical point of view, every student should have registered on the Submission platform and have joined a group **before** the deadline. Group, archive and report should be named by the concatenation of your student ID (sXXXXXX) (*e.g.*, s000007s123456).

In what follows, you will need to write a few pieces of code that will reproduce several algorithms seen in the theoretical course. It is advised to make them in *python* or *julia*.

Source coding and reversible (lossless) data compression

High-throughput sequencing technologies simplify the genome sequencing process, leading to an astonishingly rapid accumulation of genomic data. The storage and transfer of such tremendous amount of data is a common problem and encourage the use of data compression techniques. In what follows, we will use and compare several data compression techniques on a provided genome sequence.

The DNA is made of four different symbols $\{A, C, G, T\}$, also known as the four DNA bases, and is typically divided in codons made of three DNA bases. Each codon corresponds to either an amino acid or a stop signal (see Figure 1).

1. Implement a function that returns a *binary* Huffman code for a given probability distribution. Give the main steps of your implementation. Explain how to extend your function to generate a Huffman code of *any* alphabet size. Verify your code on Exercise 7 of the second list of exercises, and report the output of your code for this example.

2. Given a sequence of symbols, implement a function that returns a dictionary and the encoded sequence using the *on-line Lempel-Ziv algorithm* (see *State of the art in data compression*, slide 50/53). Reproduce and report the example given in the course.

3. Compare (without implementing the basic version) the two versions of the Lempel-Ziv algorithm seen in the theoretical course (*i.e.*, the basic and the on-line versions). Discuss what are the (practical) advantages and drawbacks of each version.

4. The *LZ77* algorithm is another dictionary algorithm. It consists in replacing repeated sequences of symbols with references to a previous occurrence of the same sequence of symbols. Typically, the encoder considers a sliding window search buffer of size l and a look-ahead buffer[2], and searches for past occurrences of the beginning of the look-ahead buffer (*i.e.*, the prefix) within the search buffer. Each codeword is made of three elements: the offset (*i.e.*, distance) of the longest prefix in

[1] See instructions on `https://people.montefiore.uliege.be/asutera/ICT.php`.
[2] You can constrain the size of the look-ahead buffer if you wish, but pay attention to choose a relevant size and to specify it in your report.

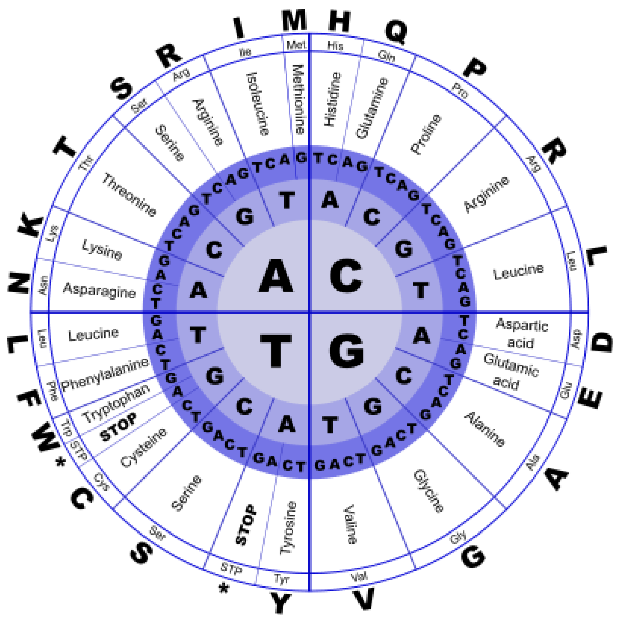

Figure 1: Standard DNA codon table organized as a wheel. Taken from this source.

the search buffer, the length of the prefix, and the symbol following the prefix. Note that the offset and the length are in practice rewritten in binary.

Implement a function that returns the encoded sequence using the *LZ77* algorithm as described by Algorithm 1 given an input string and a sliding window size l. Reproduce the example given in Figure 2 with $l = 7$.

Algorithm 1: LZ77 compression algorithm sliding window

```
Result: Write here the result
A sliding window size $l$;
An input string;
while input is not empty do
    prefix := longest prefix of input that begins in window;
    if prefix exists in window then
        $d :=$ distance to the start of the prefix;
        $l :=$ length of prefix;
        $c :=$ char following the prefix in input;
    else
        $d := 0$;
        $l := 0$;
        $c :=$ first symbol of input;
    end
    append $(d, l, c)$ to encoded input;
    shift the sliding window by $l + 1$ symbols (i.e., discard $l + 1$ symbols from the beginning of
      window and add the $l + 1$ first symbols of the input at the end of the window).
end
```

5. Estimate the marginal probability distribution of all codons from the given genome, and determine the corresponding binary Huffman code and the encoded genome. Give the total length of the encoded genome and the compression rate.

6. Give the expected average length for your Huffman code. Compare this value with (a) the empirical average length, and (b) theoretical bound(s). Justify.

7	6	5	4	3	2	1							output
							a	b	r	a	c	ada...	(0,0,a)
						a	b	r	a	c	a	dab...	(0,0,b)
					a	b	r	a	c	a	d	abr...	(0,0,r)
				a	b	r	a	c	a	d	a	bra...	(3,1,c)
		a	b	r	a	c	a	d	a	b	r	ad...	(2,1,d)
a	b	r	a	c	a	d	a	b	r	a	d		(7,4,d)
a	d	a	b	r	a	d							
sliding window							look-ahead buffer						

Figure 2: Example of the encoding of the string `abracadabrad` using the *LZ77* algorithm. Taken from this source.

7. Plot the evolution of the empirical average length of the encoded genome using your Huffman code for increasing input genome lengths. Discuss your result.

8. How could you increase the compression rate of the Huffman code? Discuss your ideas, and in particular those that change the source model and take into account the characteristics of the input message (*i.e.*, the genome).

9. Encode the genome using the *on-line Lempel-Ziv algorithm*. Give the total length of the encoded genome and the compression rate.

10. Encode the genome using the *LZ77 algorithm*. Give the total length of the encoded genome and the compression rate.

11. Famous data compression algorithms combine the *LZ77* algorithm and the Huffman algorithm. Explain how these algorithms can be combined and discuss the interest of the possible combinations.

12. Encode the genome using the *best* (according to your answer in the previous question) combination of *LZ77* and Huffman algorithms. Give the total length of the encoded genome and the compression rate.

13. Report the total lengths and compression rates using (a) *LZ77* and (b) the combination of *LZ77* and Huffman, to encode the genome for different values of the sliding window size l. Compare your result with the total length and compression rate obtained using the *on-line Lempel-Ziv algorithm*. Discuss your results.

14. It is typically assumed that repetitions occur at long distance in a genome. Based on your results in the previous question(s), discuss what could be the best data compression algorithm(s) and/or how to adapt the algorithms used in this project.

Channel coding

Let us consider a sound signal that is sent through a noisy channel. Let us take a .wav file "sound.wav" as sound signal. Its quantisation is such that possible values are between 0 and 255, and its sampling rate is $11025Hz$. The channel is a binary symmetric channel with a probability of error equal to 0.01.

In order to send the sound signal through the channel, the signal is first encoded in a binary alphabet and then each binary symbol is sent through the channel.

15. Give the plot of the sound signal and listen to it.

16. Encode the sound signal using a fixed-length binary code. What is the appropriate number of bits? Justify.

17. Simulate the channel effect on the binary sound signal. Then decode the sound signal. Plot and listen to the decoded sound signal. What do you notice?

18. Instead of sending directly through the channel the binary sound signal, you will first introduce some redundancy. To do that, implement a function that returns the Hamming (7,4) code for a given sequence of binary symbols. Then, using your function, encode the binary sound signal (from question 16).

19. Simulate the channel effect on the binary sound signal with redundancy. Then decode the binary sound signal. Plot and listen to the decoded sound signal. What do you notice? Explain your decoding procedure.

20. How would you proceed to reduce the loss of information and/or to improve the communication rate? Justify.