Saving our Planet with Compression

Clara Henzinger

Contents

Abstract

Data storage causes environmental pollution in many different ways. I investigate how to reduce storage requirements using file compression. Specifically, I implemented a widely-used text compression algorithm and study how well a variety of texts written in different languages, from different time epochs, and belonging to different text types compress. English is the most space-efficient, both before and after compression, while French and German use the most space, and German exhibits the worst compression ratio. Poems compress better than any other text type I analyzed. There is no clear relationship between the time period a text was written in and its compression ratio, but for all texts in my study compression almost halved their space requirement.

1 Introduction

Environmental pollution has become a huge problem that is threatening our future and, thus, we need to make every effort to protect our environment. This is not easy as it is our current way of life that is harming our planet, the things we do, and the tools we use. Specifically, many everyday electronic devices such as computers or cell phones contain materials with negative environmental impact. Their generation or extraction from the earth can lead to pollution, their usage in production can lead to pollution, and the improper disposal of devices containing them can lead to pollution. An example of such pollution is the extraction of rare earth metals which often requires large amounts of energy and resources.

Thus, it is important to study how we can reduce the number of electronic devices that we use. One such class of devices are storage devices for digital media. This might be the storage on our computer or secondary external storage devices such as portable memory devices (HDD) (see Figure 1), or solid-state drives (SDD), for example USB sticks. Storage devices are problematic for our planet as they contain rare earth metals such as cerium (Ce), dysprosium (Dy), neodymium (Nd), praseodym (Pr), and terbium (Tb) and precious and valuable metals such as silver (Ag), gold (Au), palladium (Pd), platin (Pt), and ruthenium (Ru), as was shown by a recent study by Kim, Lee, Kang, Sung Lee, and Lim [KLK+19]. This will become even more of an issue in the future, as the data storage market is expected to grow by more than 17% per year in the next 8 years [Ins23].

But data is not only stored on devices that we own. Nowadays, a large part of data globally is stored "in the cloud", which means in huge server farms that are accessible through the internet. A simple way of envisioning a server farm is a building that is packed with computers to store and process files (see Figure 2). Their environmental impact is large. It is estimated that in 2025 20% of the world's power supply will be used up by these server farms [Ray23], partially for computation, but 13% to 40% of their electricity will be needed for cooling [Car23]. The IT industry is increasingly becoming aware of this problem. For example, *Computer Weekly* published recently two articles on this topic, namely in September 2022 an article with the title *"Carbon copies: How to stop data retention from killing the planet"* [Mer22] and in February 2023 an article with the title *"Why we can no longer afford to overlook the environmental impact of the cloud"* [Czu23].

Figure 1: A portable memory device [OD23]

Figure 2: A data center at CERN [Hir23]

This makes the question of how to reduce storage needs impact very urgent. Note that if we could halve the size of the stored data, we could also significantly reduce the number of computers, storage devices, and servers needed, and, thus, also the negative environmental impact of their hardware and the carbon emissions from their electricity consumption.

My suggestion for dealing with this problem is to reduce storage needs by file compression. Hence, in this project, I want to study how we can reduce our storage needs so that we can reduce the number of storage devices we need and our data in the cloud. Specifically, I want to study how well files can be compressed to reduce our storage needs. File compression reduces e-waste as well as energy consumption, which in turn reduces pollution and carbon emissions. But how much space can actually be saved by the best file compression algorithm? Note that (unlike pictures and videos) text files are usually stored in their original, uncompressed form on our computer. Some operating systems allow the user to enable some form of file compression, but this requires a knowledgeable user, who knows how to turn compression on, by default it is disabled.

In all popular operating systems running on personal computers and laptops, a program is available that allows the user to compress and de-compress files, using the `zip` and `unzip` commands. It stores the data in the so-called `zip`-file format. While the program can be used for any kind of file, compression has the most effect on text files. The `zip`-file format and the corresponding algorithm are based on *Huffman coding*. Huffman coding is a way of encoding character-based data, invented by David A. Huffman in 1952 [Huf52].

Thus, the goal of my project is to study how much space could be saved if we compressed all our text files using Huffman coding. Note that each operating system stores additional overhead with each compressed file, how much overhead depends on the specific operating system. To measure the compression achieved by Huffman coding itself and to have results independent of a specific operating system, I implemented the Huffman coding algorithm myself. Then I performed a thorough analysis of the compression results of a variety of texts and languages. My goal is to study how well text compresses, but I also will investigate which factors influence whether a text compresses well. For example, I will investigate whether the language a text is written in influences the compression results and if so, which language is the most efficient. Over the centuries, languages have changed. Do old classic pieces of literature compress better or worse than modern ons? Do different types of text influence the compression ratio? These are all questions I will address in this project.

I will first give a general overview of compression in Section 2 and then describe the Huffman coding scheme in Section 3. Afterward I will present the results of my evaluation in Section 4.

2 Background on coding schemes

The idea behind many coding schemes is to represent each character in a text by a sequence of bits, called *character code*. Instead of storing the text using the standard ASCII representation, which uses 8 bits for every character, the concatenation of the character codes of all characters in the text is stored. This concatenation is simply one long bit sequence and it is called the *code of the text*. To decompress the text, the code is broken up into its individual character codes, and each of them is converted into its original character. In this way, no information is lost. Such coding schemes are called *lossless* coding schemes. There are also *lossy* coding schemes where it is not possible to restore the original text, but I will not describe them here.

There exist different lossless coding schemes. In some of them, called *fixed length coding schemes*, all character codes have the same length. This makes decompression very simple as it is easy to break up the code of the text into its character codes. If, for example, the length of each character code is 10, then counting from the beginning of the code, every 10-th bit is the end of a character code. However, codes generated with fixed-length coding schemes often need more space than codes that do not use the same length for all character codes. Such coding schemes are called *variable-length coding schemes*. As the goal of a coding scheme is to produce codes that need as little space as possible, the quality of a coding scheme is usually measured by its *compression ratio* r which is defined as follows:

$$r = \frac{\text{length of code}}{\text{length of original text}}$$

If r equals 1, the length of the code equals the length of the original text, i.e., the text was not compressed. Thus, if the compression scheme works correctly, r is a number between 0 and 1. The

goal of a compression scheme is to make the compression ratio of every text as small as possible. As mentioned above, the disadvantage of fixed-length coding schemes is that often their compression ratio is larger than the compression ratio of variable-length coding schemes. Thus, Huffman coding is a variable-length coding scheme.

2.1 Compression and decompression algorithms

Coding schemes that represent each character by a unique character code usually use the following generic compression and decompression algorithms. The compression algorithm works as follows.

Generic Compression Algorithm:

1. Build a structure that assigns to each character its character code.
2. Step through the whole text from the beginning to the end, advancing one character at a time, and replace each character with its character code. Output the text with all characters replaced by their character code.

To decompress the text the same structure is used as follows.

Generic Decompression Algorithm:

1. Step through the code of the text from beginning to end. Determine the unique character code at the beginning of the code, remove it from the code, and look up the character corresponding to the character code. Output that character and repeat this step until the code has length 0.

Note that the decoding step is required to identify the unique character code at the beginning of the code of the text. For fixed-length coding schemes this is easy: If the length of every character code is l then simply use the first l bits of the code.

For variable-length coding schemes, this is harder. One way to do this is to guarantee that there is a *unique* character code that matches the beginning of the code. To have this guarantee it is necessary that the character codes that are used need to fulfill a special property, called *prefix-freeness*, namely no character code can be a prefix of another character code. Here a *prefix* of a string s is any string that is equal to an initial segment of s. If a coding scheme uses a prefix-free set of character codes, then there can only be one character code that matches the beginning of the code: If there were two or more such codes, then all these character codes would need to match the beginning of the code and, thus, one of the character codes would need to be a prefix of the other. For example, if the code starts with '100111' and there is one character code that is '100' and another one that is '10011', then both of them match the beginning of '100111' and the first is a prefix of the second.

Step 2 of the compression algorithm is straightforward to implement in a programming language using a loop and string manipulation operations. The more challenging part is Step 1, as it needs to build a structure that can be used to construct a set of prefix-free character codes. To do so Huffman coding builds a special structure, called a *Huffman tree*. I describe it in detail in the next section.

3 Huffman coding

Huffman invented in 1952 a lossless, variable-length coding scheme that creates a prefix-free set of character codes. It is called *Huffman coding* and has found widespread usage as it achieves optimal compression ratio for all coding schemes that represent each character by its own character code, i.e., that compress each character individually. However, some coding schemes that were developed later and that represent sequences of characters instead of individual characters by a code achieve better compression ratios than Huffman coding. However, Huffman coding is still used today. As mentioned before, the very popular `zip` compression scheme is based on Huffman coding [Deu].

3.1 Compression and decompression algorithm

The basic idea in Huffman coding is to give shorter character codes to characters that are used more frequently in the text. In this way, the resulting code of the text is as short as possible. To achieve this goal, Huffman coding uses the following three steps to compress a text:

Huffman Compression Algorithm:

1. Determine the frequency of every character in the text.
2. Build a structure, called *Huffman tree*, that assigns to each character its character code based on its frequency.
3. Step through the whole text from the beginning to the end, advancing one character at a time, and replace each character with its character code. Output the text with all characters replaced by their character code.

For example, if the text is "VIENNA SCIENCE FAIR", then the letter frequencies are as follows:

```
'A' : 2, 'C' : 2, 'E' : 3, 'F' : 1, 'I' : 3, 'N' : 3, 'S' : 1, 'V' : 1, ' ': 2
```

where the last character is the empty space.

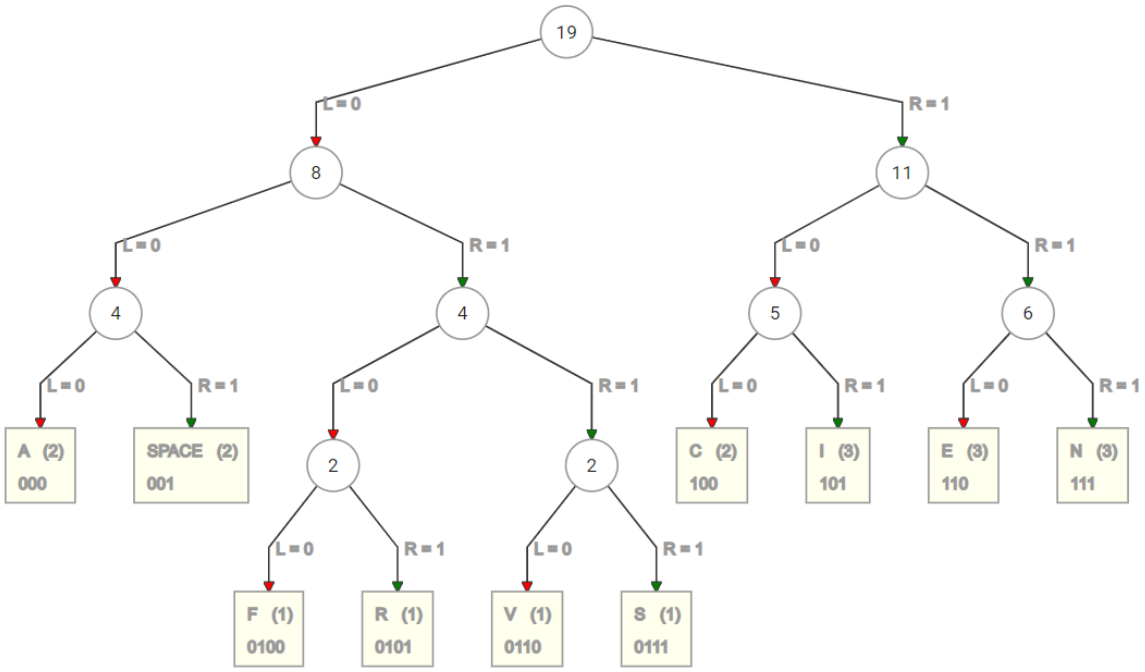

Figure 3: The Huffman tree for the string "Vienna Science Fair" generated by the Huffman tree generator available at `https://huffman.ooz.ie/`.

The corresponding Huffman tree is given in Figure 3. I generated it by entering the string "Vienna Science Fair" into an online Huffman tree generator available at `https://huffman.ooz.ie/`. Next, I will explain Huffman trees using the tree in this figure as an example. Each Huffman tree consists of a set of *nodes* and a set of *edges*. In the figure, the circles and squares are the *nodes*, and the lines with arrows are the *edges*. Each edge represents a connection between a pair of nodes. In the figure, these nodes are the two nodes where the edge starts and ends. They are called the *endpoints* of the edge. An edge is an *incoming edge* for the endpoint that is touched by its arrow, and it is an *outgoing edge* for the other endpoint.

A node without any incoming edge is called the *root* and a node without outgoing edges is called a *leaf*. In the figure, the root is the node with label 19, and every leaf is represented by a square and is labeled three pieces of information: (a) a character or the space sign, (b) the frequency of this character in parenthesis, and (c) the character code of this character. Note that all nodes, except for the root, have exactly one incoming edge, and all non-leaf nodes have exactly two outgoing edges, a *left* one, labeled with "L=0", and a *right* one, labeled with "R=1".

A *root-to-leaf path* consists of the root, one of its outgoing edges, the other endpoint v of this edge, one of the outgoing edges of v, etc., and this repeats until a leaf is reached. It is the *root-to-leaf path for that leaf.* For example, in Figure 3 the root-to-leaf path that consists of three left outgoing edges is the root-to-leaf path for the leaf labeled by the character 'A'. Now the character code of a character can be determined from the Huffman tree using the root-to-leaf path for the leaf labeled with this character: Start with an empty string at the root and traverse the root-to-leaf path to the leaf. Whenever a left outgoing edge is used on the root-to-leaf path, append a '0' to the string, whenever a right outgoing edge is used, append a '1'. For example, the root-to-leaf path to the leaf labeled 'A' consists of three left edges and, thus, the character code of 'A' is '000'. The root-to-leaf path to the leaf labeled 'S' consists of one left edge, followed by three right edges, and, thus, the character code of 'S' is '0111'.

To decompress a text, the Huffman tree produced by the compression algorithm is used as follows to convert the compressed text back into the original text.

Huffman Decompression Algorithm:

1. Process the code of the text from beginning to end as follows: Make the root the current node. As long as the code has a length larger than 0, repeat the following steps:
   (a) Traverse the left outgoing edge of the current node if the first bit of the code is 0 and the right outgoing edge if the first bit is 1 to reach a new node.
   (b) Remove the first bit from the code.
   (c) If the new node is a leaf, then output the character stored at the leaf and set the root as the new current node.

This will return exactly the original text.

3.2 Construction of the Huffman tree

I now describe how a Huffman tree is built. I assume that for each character in the text its frequency in the text has been computed by executing Step 1 of the compression algorithm. The basic idea of the construction is to have a set of *partial Huffman trees* that are combined repeatedly until only one tree is left, which will be the final Huffman tree.

Initially, the algorithm creates a node for every character in the original text and labels it with the character and its frequency in the text. Each such node is its own partial Huffman tree, being both the root and the unique leaf of the tree. This gives a set of partial Huffman trees and the root of each such tree has a frequency number, giving the sum of the frequency of all characters in the whole tree. Now as long as there are at least two trees in the set, the algorithm executes the following *merge step.* It determines a tree A from the set with minimum frequency at the root and removes it from the set. Then it finds a second tree B from the set with minimum frequency at the root and removes it from the set. Next, it creates a new node, gives it as frequency the sum of the frequencies stored at the roots of A and B, and creates two outgoing edges, namely a left outgoing edge that becomes an incoming edge of the root of A and a right outgoing edge that becomes an incoming edge of the roof of B. Note that now the trees A and B have been combined into one tree and neither the root of A nor the root of B is a root in the new tree. Then the algorithm adds this new tree to the set of partial Huffman trees and repeats the merge step. When the algorithm terminates, there is only one tree left, and this is the final Huffman tree.

For example, in Figure 3, the minimum frequency at the root of any node is 1 and the algorithm selects in the first merge step the tree of character 'F' and the tree of character 'R' as they both have minimum frequency and created a new node with frequency 2. In the second merge step it selected the tree of character 'V' and the tree of character 'S' as they both have minimum frequency and created a new node with frequency 2. Now the minimum frequency of any root has increased to 2. In the third merge step the algorithm selected the tree of character 'A' and the tree of the empty string as they both have minimum frequency 2 and created a new node with frequency 4. In the fourth merge step it selected the two trees with frequency 2 at the root, one containing the characters 'F' and 'R' and the other containing the character 'V' and 'S' and combined them, creating a new node with frequency 4. In the fifth merge step, it selected the tree of character 'C' and the tree of character 'I' as they both have minimum frequency 2, respectively, 3, and created a new node with frequency 5. After four more merge steps the final tree is complete.

European Parliament
2019-2024

TEXTS ADOPTED

P9_TA(2022)0428
Notification under the Carbon Offsetting and Reduction Scheme for International Aviation (CORSIA)
Committee on the Environment, Public Health and Food Safety
PE703.136
European Parliament legislative resolution of 13 December 2022 on the proposal for a decision of the European Parliament and of the Council amending Directive 2003/87/EC as regards the notification of offsetting in respect of a global market-based measure for aircraft operators based in the Union (COM(2021)0567 – C9-0323/2021 – 2021/0204(COD))

(Ordinary legislative procedure: first reading)
The European Parliament,
– having regard to the Commission proposal to Parliament and the Council (COM(2021)0567),
– having regard to Article 294(2) and Article 192(1) of the Treaty on the Functioning of the European Union, pursuant to which the Commission submitted the proposal to Parliament (C90323/2021),
– having regard to Article 294(3) of the Treaty on the Functioning of the European Union,
– having regard to the reasoned opinion submitted, within the framework of Protocol No 2 on the application of the principles of subsidiarity and proportionality, by Seanad Éireann, asserting that the draft legislative act does not comply with the principle of subsidiarity,
– having regard to the opinion of the European Economic and Social Committee of 20 October 20211,
– having regard to the opinion of the Committee of the Regions of 28 April 20222,
– having regard to the provisional agreement approved by the committee responsible under Rule 74(4) of its Rules of Procedure and the undertaking given by the Council representative by letter of 11 November 2022 to approve Parliament's position, in accordance with Article 294(4) of the Treaty on the Functioning of the European Union,
– having regard to Rule 59 of its Rules of Procedure,
– having regard to the opinion of the Committee on Transport and Tourism,
– having regard to the report of the Committee on the Environment, Public Health and Food Safety (A9-0145/2022),
1. Adopts its position at first reading hereinafter set out3;
2. Calls on the Commission to refer the matter to Parliament again if it replaces, substantially amends or intends to substantially amend its proposal;
3. Instructs its President to forward its position to the Council, the Commission and the national parliaments.

P9_TC1-COD(2021)0204
Position of the European Parliament adopted at first reading on 13 December 2022 with a view to the adoption of Decision (EU) 2023/… of the European Parliament and of the Council amending Directive 2003/87/EC as regards the notification of offsetting in respect of a global market-based measure for aircraft operators based in the Union
(As an agreement was reached between Parliament and Council, Parliament's position corresponds to the final legislative act, Decision (EU) 2023/136.)
1 OJ C 105, 4.3.2022, p. 140.
2 OJ C 301, 5.8.2022, p. 116.
3 This position replaces the amendments adopted on 8 June 2022 (Texts adopted, P9_TA(2022)0231).

Figure 4: Text TA-9-2022-0428 in English, adopted by the European Parliament on December 13, 2022

4 Evaluation setup and results

Our goal is to use Huffman coding to determine how well different texts can be compressed. Thus, I implemented the Huffman coding compression algorithm in Java and determined the size of the original file, the size of the compressed file, and the compression ratio for a variety of texts. Specifically, I studied

- identical texts in different languages,
- texts from different distinguished authors in different languages,
- texts of the same genre, namely poems, from different time epochs, and
- different types of text, such as newspaper articles, legal texts, and tweets.

In the following sections I describe my experiments and their results in detail.

4.1 Different languages

One goal is to analyze the space usage of different languages. Specifically, there are three concrete questions that I want to answer:

1. Are some languages more space-efficient than others, i.e., can the same information be expressed in fewer bytes by using a different language?
2. If there is a difference in the space efficiency of different languages, can compression make up for the space inefficiency of some languages? Said differently, does the same text after compression have roughly the same length in all languages?
3. Which languages have a small compression ratio, i.e., compress well?

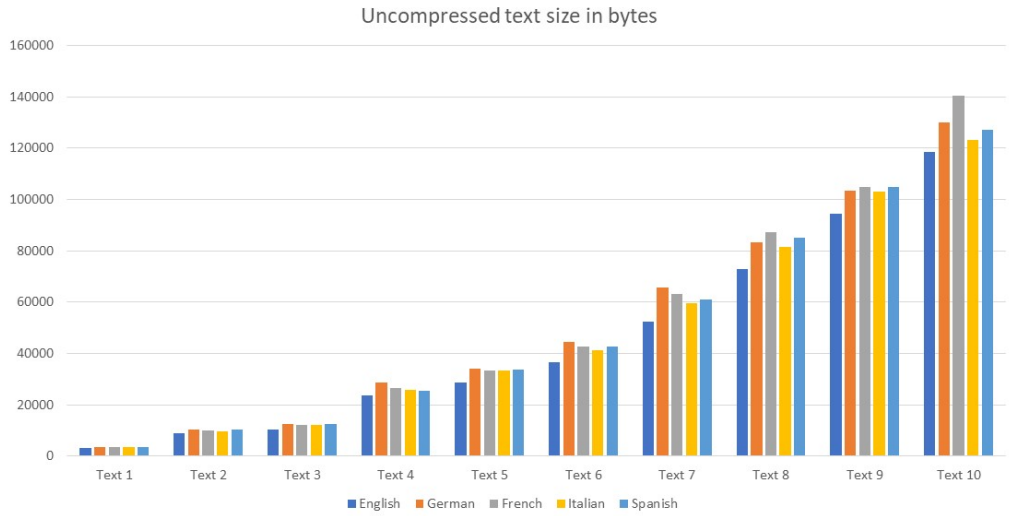

Figure 5: The sizes of the original, uncompressed texts of the European Parliament in bytes

My approach to answering these questions is to collect the same text in different languages. By analyzing the sizes of the text in different languages, one can compare the space efficiency of different languages. This answers the first question. To answer the second question, each version of the text is compressed and the sizes of the compressed files are compared. To better understand the result and to answer the third question, I will also compute the compression ratio for each language.

I do not have access to texts in all languages of the world, but some texts are freely available online in different languages. One such source of texts is the website of the EU parliament (`https://europarl.europa.eu/`). It provides the text of resolutions that were adopted by the EU parliament in its plenary meetings in all languages of the European Union. I collected 10 texts of different lengths for that website. All were adopted by the EU parliament in its plenary meeting from December 2022 to March 2023 from the website of the EU parliament. I collected each text in five languages, namely English, French, German, Italian, and Spanish. I chose these languages since a large part of the population of the European Union speaks at least one of these languages, and, thus, also many files stored on computers are likely to be in these languages. As an example, one of the texts in English is shown in Figure 4. In ASCII format the texts had different file sizes ranging between 3281 bytes and 130,112 bytes.

I first determined the length of these texts in uncompressed, i.e., original form. Figure 5 shows the sizes of the files in uncompressed form in bytes, ordered by size, and Table 1 shows the average file size of the uncompressed texts in bytes The percentage value in parenthesis behind each file size shows how much the file size is larger than the average English file size. As can be seen, in uncompressed form English is the most compact language. Italian uses on average 10% more space than English, Spanish uses 13% more, German uses 15% more, and French uses 17% more. Thus, Italian is the second most space-efficient language, German uses, for example, 5% more space than Italian. To summarize the answer to the first question is that English is by far the most space-efficient language of all five languages analyzed. When ordering the other four languages by space efficiency, the order is Italian, Spanish, German, and French. This shows that for these texts, French is the least space-efficient language.

Next, I will analyze how well different languages compress. Figure 6 shows the length of the compressed version of all texts. It is immediately clear that German does not compress as well as the other languages, as for each of the ten files, the compressed German text is the largest of all languages, even if this was not always the case for the uncompressed texts. Specifically, for all ten texts, the compressed French version is smaller than the compressed German version, even though this was not

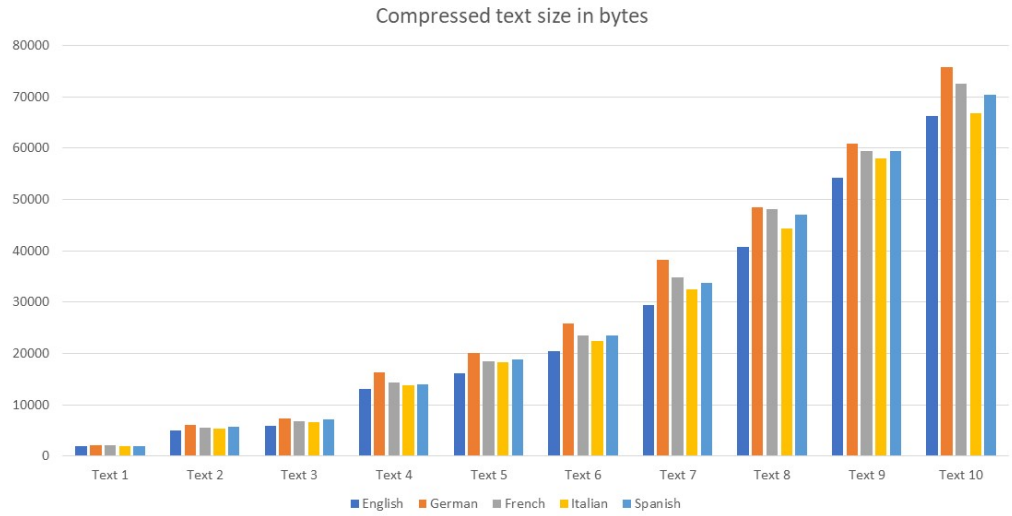

Figure 6: The sizes of the compressed texts of the European Parliament in bytes

always the case for the uncompressed texts.

	English	German	French	Italian	Spanish
Avg. original size	44,950	51,653 (14.9%)	52,431 (16.6%)	49,262 (9.6%)	50,627 (12.6%)
Avg. compressed size	25,343	30,118 (18.8%)	28,595 (12.8%)	27,025 (6.6%)	28,199 (11.3%)
Avg. compression ratio	0.569	0.588	0.566	0.551	0.560

Table 1: For each language the average uncompressed and compressed file sizes of the ten texts of the EU parliament in bytes, as well as the average compression ratios. The percentage value in parenthesis shows how much larger the file size is than the corresponding English file size.

To analyze this in more detail Table 1 contains also the average sizes of the compressed texts as well as the compression ratios of the different languages. In the same way as for the uncompressed text, the number in parenthesis gives the percentage value of how much larger the average file size of a given language is than the corresponding English file size. The table confirms the result of Figure 6: The German compressed files are by far the largest, they use 18.8% more space than the English ones, the French use 12.8% more space, the Spanish use 11.3% more space, and the Italian texts use 6.6% more space than the English. These percentages can be seen as a *size overhead* of a language when compared to English.

Recall that the smaller the compression ratio, the better texts in a language can be compressed. Ordered by increasing compression ratio the languages are Italian, Spanish, French, English, and German. Thus, one can draw make the following observations:

- Compression achieves a large space improvement for all languages. All compressed texts use much less space than the original texts, even the compressed texts of the least "compressible" language, which is German, use less space than the original texts of the most space-efficient language, which is English.

- German compresses worse than all other languages, even worse than the already very space-efficiently English language. This can be seen as the size overhead with respect to English increases from the uncompressed to the compressed setting. For all other languages the reverse

happens, i.e, they compress better than English. The same can be seen from the compression ratio. This is, however, not surprising: The size overhead computes

$$\frac{\text{length of German text}}{\text{length of English text}}$$

Thus the statement that the size overhead of the compressed text is larger than the size overhead of the uncompressed text simply states that

$$\frac{\text{length of compressed German text}}{\text{length of compressed English text}} > \frac{\text{length of uncompressed German text}}{\text{length of uncompressed English text}}.$$

On the other side, the statement that the compression ratio of German is larger than the compression ratio of English states that

$$\frac{\text{length of compressed German text}}{\text{length of uncompressed German text}} > \frac{\text{length of compressed English text}}{\text{length of uncompressed English text}}.$$

Thus, both statements are mathematically equivalent.

- Figure 5 and Figure 6 also show that the relative order of the top-3 space efficient languages remains the same *over all file sizes* and *in both the uncompressed and compressed setting*: English is the most space-efficient language, Italian is second, and Spanish is third.

  Only German and French show some influence of the compression. Even though French has the largest average *uncompressed* file size, German has the largest *compressed* file size. Thus, in the uncompressed setting, French is the least space-efficient language, and in the compressed setting, German is the least space-efficient language. However, Figure 5 shows that for short, original texts, French is more space-efficient, while for long, original texts, French is less space-efficient than German. However, this effect disappears for compressed texts: The sizes of the compressed French files are all smaller than the German ones.

- For each language I also computed the empirical standard deviation of the average compression ratios of all files using the formula

  $$\sqrt{\frac{\sum_{i=1}^{10}(r_i - \mu)^2}{9}},$$

  where μ is the average compression ratio for the language and r_i is the compression ratio of the i-th file for that language. The empirical standard deviation was between 0.010 and 0.011 for all languages, indicating that all texts for a given language compressed roughly equally well, independent of their length.

- The average compression ratios show that Italian compresses best, followed by Spanish, French, English, and German. However, since the English uncompressed files are so much smaller than the uncompressed files of the other languages, the English compressed files are still smaller than the compressed files of all the other languages.

Summary. The main conclusions of this analysis are the following:

- English is the most space-efficient language, both with as well as without compression.

- One can negatively answer our second question. German is one of the languages with the largest uncompressed file sizes and compression cannot compensate for this: It is also the language with the largest compressed file sizes. Furthermore, even though English has the second worst compression ratio, it has both the smallest compressed and the smallest uncompressed file sizes and the average file sizes of the compressed texts differ by up to 19%. Thus, even though all these texts contain the same information, their space usage is quite different.

- Italian compresses best, but even the good compression ratio cannot compensate for the fact that the uncompressed Italian texts are roughly 10% larger than the uncompressed English texts.

4.2 Different text types

Next, I want to investigate the question of whether different authors, text types, and time periods influence the compression ratio. Which one compresses better, classic authors or modern texts, poems, or plays? Ancient texts or more recent articles? This section tries to answer these questions.

4.2.1 Distinguished authors

First I want to understand how well different distinguished authors compress. To obtain full texts, I downloaded nine plays and novels from the website `https://www.gutenberg.org`. Specifically, I obtained three works of Shakespeare, Goethe, and Proust, all in their original language, namely

- from Shakespeare the plays "Macbeth", "Hamlet", and "Romeo and Juliet",
- from Goethe the novels "Die Leiden des jungen Werther", "Die Wahlverwandtschaften", and "Italienische Reise - Teil 1", and
- from Proust the novels "Sodome et Gomorrhe", "Un Amour de Swann", and " A l'Ombre des Jeunes Filles en Fleurs".

Note that these authors wrote in different languages and lived in different time periods:

- Shakespeare wrote in English and lived from 1564 to 1616.
- Goethe wrote in German and lived from 1782 to 1832.
- Proust wrote in French and lived from 1871 to 1922.

I want to use these texts to answer the following questions.

1. Do the compression ratio differ significantly between the different pieces of the same author?
2. Is there a large difference between the compression ratio of these authors?
3. Is the compression ratio for the texts written by these authors very similar to the compression ratio of the language (as determined in the previous section) they wrote in?

Table 2 shows the results, where file sizes are given in kilobytes. It can be seen that Shakespeare's plays in original format are rather short, in comparison to five of the novels of the other authors. This is also the case after the texts have been compressed. This is not surprising as plays are mainly intended to be seen in a theater, which limits their length. One of the novels of Goethe, "Die Leiden des jungen Werthers" is also rather short, which might be due to the fact that it was written in only six weeks (see `https://de.wikipedia.org/wiki/Die_Leiden_des_jungen_Werthers`).

Next I analyze the compression ratios presented in the table. Interestingly the compression ratios of both Goethe and Proust are very consistent over their three texts, while the compression ratio of Shakespeare's texts varies more. Thus, the first question is answered negatively for Goethe and Proust, but affirmatively for Shakespeare.

	original text size	compressed text size	compression ratio	average compression ratio of author
Macbeth	105.2	59.0	0.561	
Hamlet	198.8	113.6	0.572	0.553
Romeo and Juliet	175.2	92.1	0.526	
Leiden	122.1	68.1	0.558	
Wahlverwandtschaften	543.0	300.0	0.552	0.553
Reise	721.0	396.1	0.549	
Sodome & Gomorrhe	648.2	357.1	0.551	
Amour de Swann	482.4	263.7	0.547	0.549
A l'Ombre	434.8	239.1	0.550	

Table 2: The compression results for different classic authors. File sizes are given in kilobytes.

The third column in the table shows the average compression ratio of each author. All three authors achieve roughly the same average compression ratio, namely roughly 0.55. Proust has a somewhat smaller average compression ratio (0.549) than Goethe and Shakespeare (0.553 both). This is somewhat unexpected as they were born in different centuries and wrote in different languages. Additionally, Shakespeare's texts are plays, while I analyzed novels by Goethe and Proust. However, this could be the result of the fact that I was only able to analyze three texts from each of these authors in their original language.

Comparing these compression ratios to the ratios from the previous section shows that their averages are better than the averages achieved by the texts from the EU parliament in their respective languages. The texts of the French author compress slightly better than the other two, but the difference between the authors is much smaller than the differences for the texts in different languages studied in the previous section. Thus, I conclude that the compression ratio of these authors seems to be independent of the language they are using. This answers the third question.

Summary. Even though the time period and the language of the authors I studied differed largely, their average compression ratio was roughly the same, namely roughly 0.55.

4.2.2 Poems through time

Since I could not find a difference between the compression ratios for the three authors studied in the previous section even though they were born in different centuries, I decided to analyze literary texts of the same text type from different time epochs. I decided to analyze poems, as poems are relatively short, easy to find online, and have been used in many time epochs. Specifically, I want to answer the following questions:

1. Did the compression ratio of poems increase over time?
2. Do poems compress better or worse than the plays and novels studied in the previous section?
3. Does the compression ratio of poems show the same dependence on the language as the texts of the EU parliament?

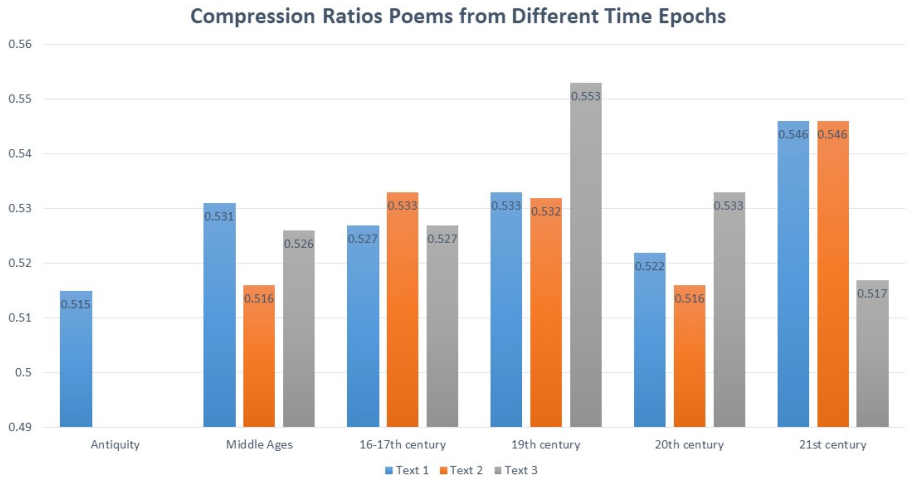

Figure 7: The compression ratio of various poems over different time epochs.

To answer these questions I collected the following poems:

- Homer's Odyssey in English from antiquity;
- three different German minnesong from the middle ages;

	original text size	compressed text size	compression ratio
Odyssey	732.5	377.3	0.515
Minnesong 1	1.4	0.7	0.531
Minnesong 2	0.8	0.4	0.516
Minnesong 3	1.1	0.6	0.526
Sonnet 3	0.61	0.32	0.527
Sonnet 37	0.59	0.31	0.533
Sonnet 81	0.63	0.33	0.527
Arthur Rimbaud	1.7	0.9	0.533
Paul Verlaine	1.6	0.9	0.532
Victor Hugo	2.6	1.5	0.553
Rilke	0.53	0.28	0.522
Kim Yideum	2.5	1.3	0.546
Bella Li	0.6	0.3	0.549
Eva Gerlach	0.8	0.4	0.517

Table 3: The compression results for various poems. File sizes are given in kilobytes.

- three sonnets by Shakespeare from the end of the 16th century;
- three French poems from the 19th century, namely one each by Arthur Rimbaud, Paul Verlaine, and Victor Hugo;
- three German poems by Rainer Maria Rilke from the 20th century; and
- one English poem each by Kim Yideum, Bella Li, and Eva Gerlach from the 21st century.

The results are given in Table 3. There is no clear pattern of dependence of the compression ratio on the time epoch. However, Figure 8 shows the average compression ratio over the time epochs and that indicates a slight increase in compression ratio, with the exception of the poems by Rilke. Thus, I cannot find a conclusive answer to the first question, more data would be needed.

However, I can give a clear answer to the second question: Poems compress in general well. Their average compression ratio is 0.531, while the average compression ratio of the literary texts studied in the previous section is 0.552. Thus, poems clearly compress better.

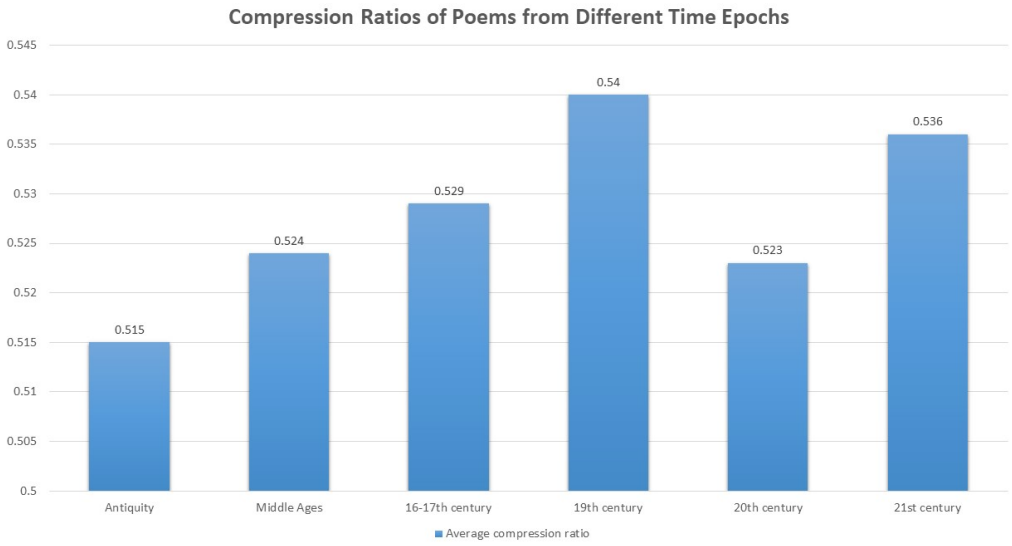

Figure 8: The average compression ratio of various poems over different time epochs

Note that both the minnesongs and the poems by Rilke are written in German. Still, they have a relatively small compression ratio in comparison to the other poems, while the French poems from

the 19th century exhibit the largest compression ratio. Thus, to answer the third question, I conclude that the compression ratio of poems does not show a clear increase over time and also does not seem to show the same dependent on the languages as the texts of the EU parliament. It seems to be more dependent on the authors.

Summary. The compression ratio of the poems I analyzed lies between 0.52 and 0.55 with an average of 0.531, which is better than the texts in both previous sections. Their compression ratio does not show a clear dependence on the time period and also not the same dependence on the language as the non-literary texts in this study.

4.2.3 Recent newspaper articles and tweets

My conclusion from the previous two sections is that the compression ratio of literary texts in general seems to depend more on the authors than on the language. This leads to the question of what happens in other non-literary texts. Specifically, my goal is to find an answer to the following question:

1. Does the compression ratio of other non-literary texts follow the same language pattern as the texts of the EU parliament?

To answer this question I collected both newspaper articles and tweets. I first present the study for newspaper articles and afterward the one for tweets. To collect the newspaper articles I downloaded them from the official websites of the newspaper. In this way I obtained articles from the New York Times, the French newspaper "Figaro", and the German newspaper "Süddeutsche Zeitung" from 2023. The three articles from each the New York Times and one of the German newspaper "Süddeutsche Zeitung" are full articles that I was able to download for free. The other two articles from the German newspaper and the three articles from the French newspaper are only beginnings or articles as the full articles were behind a paywall and I could only access their first part. Due to these restrictions, I cannot compare the sizes of the files.

The results are shown in Table 4. The compression ratio of the three articles of the New York Times is very consistent between 0.550 and 0.558, while there are larger differences for the texts from Figaro and the "Süddeutsche Zeitung". The reason might be that these articles are very short and, thus, relatively small changes in the size of the compressed text can lead to large changes in the compression ratio.

The last column of Table 4 shows the average compression ratio for each of the four newspapers. Interestingly, the same language pattern as for the texts from the EU parliament emerges: The average compression ratios of the French texts are slightly smaller than the compression ratio of the English texts, and both are much smaller than the compression ratio of the German texts.

	original text size	compressed text size	compression ratio	average comp ratio
NY Times 1	8.6	4.7	0.552	
NY Times 2	10.4	5.7	0.550	0.553
NY Times 3	16.4	9.2	0.558	
Figaro 1	2.9	1.6	0.560	
Figaro 2	3.7	2.0	0.543	0.550
Figaro 3	3.9	2.2	0.548	
Süddeutsche Zeitung 1	2.0	1.1	0.561	
Süddeutsche Zeitung 2	6.3	3.5	0.560	0.565
Süddeutsche Zeitung 3	3.1	1.8	0.573	

Table 4: The compression results for different newpaper articles. File sizes are given in kilobytes.

Next, I present the results for the tweets I analyzed. Twitter restricts the length of texts to only 280 characters. As a consequence users often use abbreviations, which can be seen as a sort of "human-readable" compression. As a result, tweets should compress less well than other texts. I collected and analyzed tweets in three different languages, namely English, French, and German. The average compression ratios are given in Table 5. Note that these ratios are slightly worse than the ones for newspaper, which might be due to the abbreviations used in tweets.

It is interesting that the results show the same language pattern observed for non-literary texts before: French compresses best and German worst. Here I do not present the sizes of the individual files as I downloaded roughly the same amount of data in each language, namely between 2 to 4 kilobytes of text.

English	French	German
0.566	0.556	0.573

Table 5: The average compression ratios of tweets in three different languages.

Summary. The compression ratio for newspapers and for tweets lies between 0.55 and 0.57 with an average of 0.556 for newspapers and 0.565 for tweets. They follow the same language pattern as the texts of the EU parliament: French compresses better than English and both compress better than German.

4.2.4 National constitutions

The results of the evaluations so far seem to indicate that the compression ratio of non-literary texts shows a clear pattern that depends on the language the text is written in, while literary texts do not follow this pattern. So far I analyzed three types of non-literary texts, namely texts published by the EU parliament, newspaper articles, and tweets. To analyze more non-literary texts, I collected the constitutions of three countries, namely,

- the US Constitution and its amendments,
- the French constitution, and
- the German constitution ("Grundgesetz").

As before, I want to understand the following question:

1. Does the compression ratio of the constitutions of different countries show the same dependence on the language as the other non-literary texts that I analyzed?

Table 6 shows the results of our study. First, it is interesting to note that different constitutions have different lengths, with the German one being the largest. Second, and more useful for answering the above question, is the result in the fourth column of the table: Indeed, the French constitution compresses best, the US Constitution as well as its amendments have the second smallest compression ratio, and the German Grundgesetz has the largest compression ratio.

	original text size	compressed text size	compression ratio
US Constitution	44.1	24.0	0.544
Amendments	21.6	11.9	0.551
French Constitution	89.5	48.4	0.541
German Constitution	180.9	101.8	0.563

Table 6: The compression results for the constitutions of three different countries. File sizes are given in kilobytes.

Summary. The compression ratios of the constitutions of three countries, written in English, French, and German, lie between 0.54 and 0.56 with an average compression ratio of 0.550. Their compression ratio follows the same language pattern as the other non-literary texts I studied.

5 Conclusion

One way to contribute to the reduction of environmental pollution is to trim the storage needs of text using text compression. This study shows that through text compression storage consumption can be

almost halved. I studied texts in five languages using a large variety of text types from antique poems to recent newspaper articles, from legal texts to tweets. All texts achieved a compression ratio between 0.52 and 0.59. Thus using compression can free up between 41% and 48% of the storage space.

For non-literary texts this study shows that when uncompressed, German and French need much more space than the same text in English, and also Spanish and Italian are less space efficient than English. These relationships continue to hold even after the text is compressed. To save disk space this means that it is especially important to compress German and French texts.

For literary texts, the compression ratio does not show any clear dependence on the language or the time epoch the text was written in. However, the poems in our study compressed clearly better than any other text type.

I did not discuss the running time of my implementation, but even though I am not a professional programmer, my program compressed all files—including entire novels—very quickly (in a matter of seconds or minutes). Thus, we all should compress our files, and operating systems should enable file compression by default. Furthermore, the next time when we want to buy a new computer or a new USB stick because their storage space is full, we should instead delete unnecessary files, compress the others, and use the freed-up space on the old devices. Let's protect our planet!

References

[Car23] Carbometrix. What consumes the most energy in a data center? `https://carbometrix.com/data-center-energy-consumption/`, 2023.

[Czu23] Amy Czuba. Why we can no longer afford to overlook the environmental impact of the cloud. `https://www.computerweekly.com/blog/Green-Tech/Why-we-can-no-longer-afford-to-overlook-the-environmental-impact-of-the-cloud`, 2023.

[Deu] L. Peter Deutsch. Deflate compressed data format specification version 1.3. page p. 1. sec. Abstract.

[Hir23] Copyright: Florian Hirzinger. `https://de.wikipedia.org/wiki/Rechenzentrum`, 2023.

[Huf52] David A. Huffman. A method for the construction of minimum-redundancy codes. In *Proceedings of the I.R.E., September 1952*, volume 40, pages 1098–1101, 1952.

[Ins23] Fortune Business Insights. `https://www.fortunebusinessinsights.com/infographics/data-storage-market-102991`, 2023.

[KLK+19] Se-Hee Kim, Su-Jin Lee, Sung-Eun Kang, Dae Sung Lee, and Seong-Rin Lim. Environmental effects of the technology transformation from hard-disk to solid-state drives from resource depletion and toxicity management perspectives. *Integrated Environmental Assessment and Management*, 15(2):292–298, 2019.

[Mer22] Chris Merriman. Carbon copies: How to stop data retention from killing the planet. `https://www.computerweekly.com/feature/Carbon-copies-How-to-stop-data-retention-from-killing-the-planet`, 2022.

[OD23] `https://www.office-discount.at/seagate-basic-1-tb-externe-hdd-festplatte-schwarz-241817`, 2023.

[Ray23] Aditya Rayaprolu. 15 crucial data center statistics to know in 2023. `https://techjury.net/blog/data-center-statistics/`, 2023.