Friedrich-Koenig-Gymnasium Würzburg

W-Seminar 2024 — 2026

Mathematik

On The Optimality of Huffman and Shannon-Fano Coding and Their Subtypes

---

Lehrkraft:
Oliver Manger

Schüler:
Kai Rasmussen

Bewertung	**Note**	**Notenstufe in Worten**	**Punkte**		**Punkte**
Schriftliche Arbeit				× 3	
Präsentation				× 1	
				Summe:	
		Gesamtleistung nach § 61 (7) GSO = Summe : 2 (gerundet):			

Abgabe im Sekretariat

Unterschrift des Kursleiters

Contents

Abstract - The comparison between the original basis of data compression algorithms, Huffman and Shannon-Fano Coding systems, as well as a deeper look into the different improvements to and applications of lossless compression algorithms.

Keywords: Huffman coding, Shannon coding, Fano coding, lossless data compression, static compression systems, adaptive compression systems, and higher-order models.

1 Introduction

Lossless data compression systems have a massive importance to our modern lives. Without these systems we couldn't have managed to reach this level of the technological era. As the name says, these are systems to reduce the amount of storage data takes up without loosing any information in the process, i.e. lossless. The foundation of lossless data compression systems are Huffman, Shannon, and Fano coding systems but since these systems were introduced there have been many improvements and different approaches to this problem of lossless data compression.

1.1 Aims

In this paper we will firstly discuss the history of data compression leading up to the invention of the fundamental lossless compression systems and we will then go through these systems, going through how they work and how they differ from each other. We will then go into the difference between their static and adaptive counterparts before briefly looking at how these systems were improved upon and how they are used in the modern day. Afterwards we will compare the fundamental lossless compression systems using example data with a java comparison system to find out by how much these systems differ and if there are certain biases with the data examples.

1.2 Theoretical Basis

The purpose of this section is as a reference during the reading of the rest of the paper for any base concepts.

1.2.1 Lossless vs. Lossy Compression Systems

Lossless compression systems are a method of compression in which the original uncompressed form can be reconstructed to it's original quality. In comparison lossy compression is a system where less relevant information is disregarded to compress the size[1]. In this paper we will only be looking at lossless compression systems.

1.2.2 Variable vs. Fixed Length Coding

Fixed length coding systems assign a certain amount of bits per character, each character being the same length. This makes it easier to search for a specific position of a character through simple multiplication, i.e. if you were looking for the 10th character in an 8-bit fixed length coding system you would look from bit 81 to 88. Variable length coding

[1] [2] p.40-41: 7 Lossy Compression Techniques

systems however, as the name tells us, the character length vary. This is then helpful when we want to compress data so we can assign shorter codes to more common characters[2].

1.2.3 Information Entropy

Information entropy describes the relation between the probabilities of the input sequence and the length of the code word. For the most optimal lossless compression system, the average code word length is equal to the value of the information entropy. This concept was one of the most important in Shannon's foundation of information theory[3]. The moment when the average code word length goes below the value of the entropy H, information must be lost, so the compression system is no longer lossless.

$$H = -\sum p_i \log(p_i) \tag{1}$$

This equation is valid when the characters are independent of each other.

1.2.4 Higher-order vs. Zeroth-order Systems

In a zeroth-order system, each character is encoded independently, any correlation between the characters in the input sequence aren't taken into account. A higher-order system has the ability to create and observe patterns over multiple characters.

citation needed

1.2.5 Dictionary Based Coding

1.2.6 Dynamic Markov Compression

1.2.7 Block Sorting Compression

1.2.8 Grammar Based Coding

In grammar based coding, we use the concept of languages and grammars in computer science, a grammar G being made out of a list of terminals T or an alphabet Σ, a list of non-terminals N, a list of production rules P, and a start node s. With this higher-order system instead of encoding the string x itself, we encode the grammar G_x. Due to the fact that the grammar only constructs the string x, it is called a context-free grammar[4].

$$L(G_x) = \{x\} \tag{2}$$

1.2.9 Prediction with Partial Matching

1.2.10 Burrows-Wheeler Transform

1.2.11 Context Tree Weighting

1.2.12 Multilevel Pattern Matching

Mention java code of comparison system

[2] [17] p.825-827: 1 Introduction
[3] [14] p.396-399: 7 The Entropy of an Information Source
[4] [8] p.2-4: Introduction

2 History of Data Compression

In general, a data compression system is used when we want to compact a larger amount of data into a smaller amount of information and as a result also decrease the amount of time required to transfer messages. This principle can already be seen in the 4th century BCE in Greece where Polybius tells us of an improved communication system introduced by Aeneas Tacticus, in which two people in viewing distance would use a torch and two identical inscribed vessels to communicate at faster speeds in times of war[5]. Here they used water and engravings in the vessel, a certain volume of water being a specific message. This is an early example of a cryptographic cypher, where a certain input, here the volume of water, can be decrypted to a certain plain text message[6]. In the same vane, many versions of the optical semaphore were used in a similar manner, using predefined interpretations of certain positions to communicate a predetermined message. However, one of the flaws of both of these systems is that there is a limit to what information you can transmit as you cannot fit every possible message into a cypher. Communicating a single letter at a time would take a significant amount of time so to reduce the quantity of data being sent but retain all the information required to produce any message, the idea of lossless data compression was born. The first main compression algorithm to come up was Claude Shannon's coding system, named after himself (See Section 3.2: Shannon Coding) based on his noiseless coding theorem. This theorem and compression algorithm make the foundations of what would become the field of information theory[7].

Image of graphic of hydraulic telegraph

3 Compression Systems

In this paper, when explaining and comparing these systems, we will follow a specific notation and terminology in the creation of our compression tree. As the input for any compression system, we need an input sequence $\mathcal{S}$ which as an array of m unique characters. Each of these characters will then, through the compression system, receive their own code.

$$\mathcal{S} = \{s_1 \to s_m\} \tag{3}$$

To history of shannon and fano systems

Each of these characters in the input sequence has an attached probability. This probability can be calculated by dividing the amount of appearances of this character by the total amount of characters. These probabilities are also kept in an array $\mathcal{P}$.

$$\mathcal{P} = \{p_1 \to p_m\} \tag{4}$$

$$p = \frac{\textbf{Amount of appearances}}{\textbf{Total amount of characters}} \tag{5}$$

To Huffman

To modern extensions and versions (Out of scope)

For the compression system, we also need a code alphabet with which we want to compress our code, the simplest of which being the binary system with the code alphabet $\mathcal{A} = \{0, 1\}$ where $r = 2$. This tells us that each internal node in the tree has two children and only one sibling. For most of the examples, we will use this code alphabet for simplicity, but you should keep in mind that all of these systems can also work with larger code alphabets.

$$\mathcal{A} = \{a_1 \to a_r\} \tag{6}$$

[5] [15] Book X pp.42-43: 44. The Improvement introduced by Aeneas Tacticus
[6] [7] p.32: Cipher (American English)
[7] [12] p.36: 2.2.1 Introduction to Information Theory

The compression tree with which the message code is encoded and decoded is depicted with $\mathcal{C}$. Each of the nodes in the compression tree is listed in the array $\mathcal{Q}$ with n being the total amount of nodes in the compression tree.

$$\mathcal{Q} = \{q_1 \to q_n\} \tag{7}$$

Each node q_i in the compression tree has a corresponding weight τ_i, which can be calculated as the sum of the weights of the children j of that node and is kept in an array $\mathcal{T}$.

$$\mathcal{T} = \{\tau_1 \to \tau_i\} \tag{8}$$

$$\tau_i = \sum_{o=1}^{j} \tau_o \tag{9}$$

After the encoding process using the compression tree $\mathcal{C}$, the result is an output sequence $\mathcal{Y}$ that is a concatenation of all the resulting code words with a length $L_{\mathcal{Y}}$. In addition, we will use the array $\mathcal{X}$ to list the resulting code words.

$$\mathcal{X} = \{x_1 \to x_m\} \tag{10}$$

The resulting encoding scheme that acts like a translation table between encoded and decoded codes is listed in ϕ.

$$\phi : s_1 \to x_1, \ldots, s_i \to x_i \tag{11}$$

The average length of the code words in $\mathcal{X}$ is represented by $\overline{L}$.

$$\overline{L}(\mathcal{X}) = \sum_{i=1}^{m} L(x_i) \tag{12}$$

The main objective of our compression systems is to assign the most frequent input sequence character s_1 to the shortest code word x_1, the ideal being that the average code word length $\overline{L}$ is as small as possible. This smallest possible average code word length is referred to as a minimum redundancy code, coined by David A. Huffman in his paper on the construction of just such a code under the conditions that: No two messages shall have the same resulting sequence $\mathcal{Y}$ and each code word shall be distinguishable without needing to specify the start or end of a code word.[8]

3.1 Huffman Coding

In Huffman's system, we start with a sequence code $\mathcal{S}$ that is sorted by descending probability and assign each of these characters a node in $\mathcal{Q}$.

$$p_i \geq p_{i+1} \geq \cdots \geq p_m \tag{13}$$

$$q_i \leftarrow s_i, (\tau_i = p_i) \tag{14}$$

We then take a maximum of r but at least 2 characters with the lowest probability and connect them together with a new node and put this new node back in the array $\mathcal{Q}$ in the right position corresponding to its weight.

$$q_{new} \leftarrow q_{new\,j}, \tau_{new} = \sum_{i=1}^{j} \tau_i \tag{15}$$

[8] [5] p.1098: Introduction

You can repeat this process until $\tau_{new} = 1$. At the end, we can visualise the compression tree by tracing out the children of each node until we reach the source characters. [9]

3.1.1 Example: Static Huffman Coding

Input:

- $\mathcal{S} = \{a, b, c, d, e, f, g, h\}$
- $\mathcal{P} = \{0.22, 0.20, 0.18, 0.15, 0.10, 0.08, 0.05, 0.02\}$
- $\mathcal{A} = \{0, 1, 2, 3\}$

s_i	p_i	L_i	Code
a	0.22	1	1
b	0.20	1	2
c	0.18	1	3
d	0.15	2	00
e	0.10	2	01
f	0.08	2	02
g	0.05	3	030
h	0.02	3	031

Table 1: Adapted from [5] p.1101: Table III

In Figure 1, the green nodes represent the single characters with their corresponding probabilities p and the purple nodes are the connecting nodes q with their corresponding weight τ. While encoding and decoding, we use this tree as a translation guide, the furthest left being 0, then 1, and so on.

3.1.2 Adaptive Huffman Coding

Adaptive Huffman Coding or Dynamic Huffman Coding, in comparison to the static version, is a system that adapts the construction of the Huffman tree to changes in the count of characters, rather than having a presorted list with given probabilities. This theory suggested by Robert G. Gallager, is based on the sibling property, where if the nodes can be listed in decreasing counts and each node is adjacent to its sibling, then the code is optimal for the current character count[10]. In a binary system each node, or sibling pair, contains 5 different components. Two are the current counters of the children nodes, two are links to the children nodes, and the last is a link to parent node. Each of the links also having an extra property indicating whether the node is the 0th or the 1st child node. As an input to our adaptive system, we have the Huffman tree of one less character read and the next character. The output then being the Huffman tree with the updated count. With the incrementing of the character count, if it is found that the count exceeds the count of the next higher sibling pair, then these two nodes are interchanged, changing the code and the resulting Huffman tree[11]. This algorithm was then further expanded upon

[9] [5] p.1011: Generalization of the Method; [12] p.50: Algorithm 1 Static Huffman Encoding
[10] [4] p.6: 2. The Sibling Property Definition
[11] [4] p.17-21: Adaptive Huffman Codes

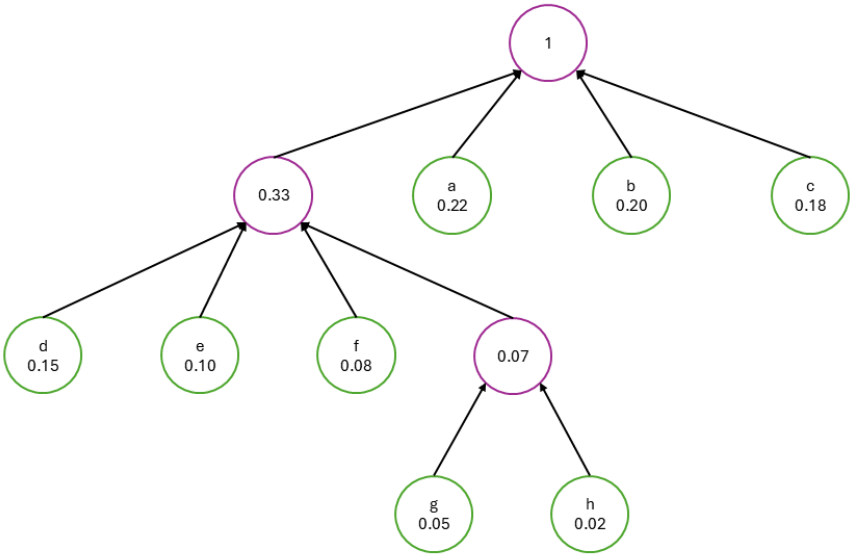

Figure 1: Huffman Tree of Table 1; Made using Microsoft PowerPoint

by Donald E. Knuth with his paper on Dynamic Huffman Coding. Here he fills in some gaps left by Gallager's adaptive system like the decrementing of the character counts[12].

3.2 Shannon Coding

Even though the Shannon Method suggested by Claude E. Shannon is not commonly used due to its inefficiency when it comes to the resulting compression size, it still has a relevance in the field. Compared to the Huffman Coding system, this one is based on a top-down construction system making it more intuitive and faster to calculate. This system gives a unique string of numbers by using a cumulative probability function P_i and then taking the first l_i digits of the rth decimal representation of the cumulative probability function to create the corresponding code word.[13]

$$P_i = \begin{cases} 0 & \textbf{for} \quad i = 1 \\ P_{i-1} + p_{i-1} & \textbf{for} \quad 2 \leq i \leq m \end{cases} \tag{16}$$

$$l_i = \lceil log_r(p_i^{-1}) \rceil \tag{17}$$

3.2.1 Example: Static Shannon Coding

Make example

3.2.2 Adaptive Shannon Coding

Make example encoding scheme using example

Explanatio of dif-

[12] [9] p.163-165: Introduction

[13] [12] p.52-53: 2.4.2 Shannon's Method; [14] p.58-61: 9. The Fundamental Theorem for a Noiseless Channel

3.3 Fano Coding

The Fano coding method is, like the Shannon coding method, based on a top-down construction system. The Fano method starts with a sequence code $\mathcal{S}$ that is sorted by descending probability in $\mathcal{P}$ (See Equation 11). In the binary system, we then split this list into two subgroups of equal probability $\mathcal{S}_0$ and $\mathcal{S}_1$, and as a result also $\mathcal{P}_0$ and $\mathcal{P}_1$. There codes then starting with either 0 or 1. This process is then repeated until the subdivisions are equal to the single characters. Here the code word length is equal to $\log_2 m$ if m is a power of two. If m is not a power of two, the length of the code word is one of the two closest integers to $\log_2 m$.[14]

Exact explanation of process

3.3.1 Adaptive Fano Coding

The adaptive Fano coding system functions using the same concept as the adaptive Huffman coding system in the sense that the source character weights are not calculated beforehand. Luis Rueda suggests two different ways to make an adaptive Fano coding system, one they call the brute-force method and the other the greedy method. In the brute-force method for each change in the character counter, the system recomputes the corresponding compression tree. This is of course rather inefficient[15].

Make Example

Make example tree using example

3.4 Further Improvements

3.4.1 Lempel-Ziv Algorithm

3.4.2 Deflate Algorithm

Explain difference to static

The Deflate Algorithm is a higher-order system based on both the LZ77 system and the Huffman coding system.

Make example

3.4.3 Lempel-Ziv-Welch Algorithm

Rough Explanation of LZ77 and LZ78

3.5 Comparison System

The way that we have compared the different compression systems is a program in which we compare the amount of calculations used and the size of the resulting compressed file relative to the original file size.

3.5.1 Surrounding System

Explanation of combination of Huffman and LZ

Processes that do not vary between the different compression systems will not be included in the resulting calculation time. The metric of calculation time is supposed to be a comparison between the construction of the compression tree. Therefore, the static compression system that are based on an already constructed compression tree, do not show the metric of the number of calculation processes.

Use cases, improvements

General explanation reference to introduction

Data	ASCII-1965[16]	USASCII-8[17]	UTF-32[18]
Wordnik[19]	16.57	18.93	75.74
Shakespeare[20]	-[21]	43.03	172.12

Table 2: List of Examples used for comparing the different compression systems with their corresponding storage usage in Mb.

3.6 Results

Compression System	Processes[22]	Size[23]	$\overline{L}$ Avg. L[24]	Percentage Saved[25]
Static Huffman	–	–	–	–
Static Shannon	–	–	–	–
Static Fano	–	–	–	–

Table 3: Comparison of the different compression systems with Wordnik

Compression System	Processes	Size	$\overline{L}$ Avg. L	Percentage Saved
Static Huffman	–	–	–	–
Static Shannon	–	–	–	–
Static Fano	–	–	–	–

Table 4: Comparison of the different compression systems with Shakespeare

4 Conclusion

In general, even though the original compression systems of Huffman, Shannon, and Fano are practically not used anymore, their influence can still definitely be felt in the modern compression systems of today.

Comparison which algorithm for what use case

[14] [3] p.5-6: Selection from N Equally Likely Choices; [12] p.55: Algorithm 3 Static Fano Encoding

[15] [11] p.1659-1660: 2.1 A brute-force method for adaptive Fano coding

[16] [10] p.423-425: 23.2 ASCII-1965; also ISO/IEC 646: [6] p.6-7: Table 1 (Calculated: Number of Characters multiplied by 7)

[17] [10] p.431-433: 23.11 USASCII-8 (Calculated: Number of Characters multiplied by 8)

[18] [16] p.77: 2.5.1 UTF-32 (Calculated: Number of Characters multiplied by 32)

[19] [1]: Wordnik Public Domain List of English Words

[20] [13]: The Complete Works of William Shakespeare

[21] File from Project Gutenberg contains some formatting characters that aren't included in the standard ASCII-1965

[22] Calculation processes are relative, different types of processes take a different amount of time so this number should be interpreted rather vaguely.

[23] Storage usage after compression not including the storage used by the compression tree or encoding scheme. The storage taken by these in proportion to a larger amount of data makes it so that we can basically ignore this small extra storage usage.

[24] Average Code Word Length (See Section 3: Equation 10)

[25] Percentage of the compressed storage size compared to the smallest working coded character set in Table 2.

4.1 Acknowledgements

This paper was written and researched without the use of generative AI. This paper was also originally written in English and translated into the German with the use of the DeepL translator for the first draft translation and then corrected and modified by me.

Useful sources

Dev Environment

References

[1] URL: https://raw.githubusercontent.com/wordnik/wordlist/refs/heads/main/wordlist-20210729.txt (visited on 04/23/2025).

[2] Guy E Blelloch et al. "Introduction to data compression". In: *Computer Science Department, Carnegie Mellon University* 54 (2001).

[3] Robert M Fano. *The transmission of information*. Vol. 65. Massachusetts Institute of Technology, Research Laboratory of Electronics ..., 1949.

[4] Robert Gallager. "Variations on a theme by Huffman". In: *IEEE Transactions on Information Theory* 24.6 (2003), pp. 668–674.

[5] David A Huffman. "A method for the construction of minimum-redundancy codes". In: *Proceedings of the IRE* 40.9 (1952), pp. 1098–1101.

[6] *ISO/IEC 646:1991 Information technology - ISO 7-bit coded character set for Information Interchange*. The International Organization for Standardization and The International Electrotechnical Commission. 1991.

[7] Robert E Joyce. *Committee on National Security Systems Glossary*. National Security Agency. Fort Meade, MD, United States of America, 2022.

[8] John C Kieffer and En-Hui Yang. "Grammar-based codes: A new class of universal lossless source codes". In: *IEEE Transactions on Information Theory* 46.3 (2000), pp. 737–754.

[9] Donald E Knuth. "Dynamic Huffman Coding". In: *Journal of algorithms* 6.2 (1985), pp. 163–180.

[10] Charles E Mackenzie. *Coded-Character Sets: History and Development*. Addison-Wesley Longman Publishing Co., Inc., 1980.

[11] Luis Rueda and B John Oommen. "A fast and efficient nearly-optimal adaptive Fano coding scheme". In: *Information Sciences* 176.12 (2006), pp. 1656–1683.

[12] Luis Gabriel Rueda. "Advances in data compression and pattern recognition". PhD thesis. Carleton University, 2002.

[13] William Shakespeare. *The Complete Works of William Shakespeare*. Project Gutenberg, 1994.

[14] Claude Elwood Shannon. "A mathematical theory of communication". In: *The Bell system technical journal* 27.3 (1948), pp. 379–423.

[15] Evelyn S Shuckburgh. *The Histories of Polybius*. Vol. 2. MacMillan and Co., 1889.

[16] *The Unicode Standard, Version 17.0*. The Unicode Consortium. San Francisco, CA, United States of America, 2025.

[17] Jeffrey Scott Vitter. "Design and analysis of dynamic Huffman codes". In: *Journal of the ACM (JACM)* 34.4 (1987), pp. 825–845.