Friedrich-Koenig-Gymnasium Würzburg Oberstufenjahrgang 2024/2026

SEMINARARBEIT

im W-Seminar

Kryptologie

im Fach

Mathematik

Thema:

Quellenkodierung I - Huffman-/Shannon-Fano-Kodierung

Verfasser / -in: Kai Rasmussen

Kursleiter / -in: Oliver Manger, OStR

spät. Abgabetermin: Di., 11.11.2025

Tag der Abgabe im OS-Büro: _____

Bewertung: ☐ Punkte für die W-Seminararbeit

(einfache Wertung, Ergebnis liegt zwischen 0 und 15 Punkten)

☐ Punkte für die Präsentation

Gesamtnote: ☐ Punkte

(Arbeit dreifach gewertet, Präsentation einfach gewertet, Summe geteilt durch 2, Ergebnis gerundet; Gesamtergebnis liegt zwischen 0 und 30 Punkten)

Besprochen am: _____

_____

Unterschrift der Kursleiterin / des Kursleiters

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Seminararbeit selbstständig und ohne fremde Hilfe verfasst und keine anderen als die im Literaturverzeichnis angeführten Quellen und Hilfsmittel verwendet habe. Insbesondere versichere ich, dass alle Stellen der Seminararbeit, die anderen Quellen im Wortlaut oder dem Sinn nach entnommen sind, durch Angabe der Herkunft kenntlich gemacht wurden. Dies gilt auch für Zeichnungen, Skizzen, bildliche Darstellungen sowie Quellen aus dem Internet.

Bei der Anfertigung der vorliegenden Arbeit habe ich KI-Tools/Chatbots (vgl. beiliegende Übersichtsliste) verwendet. Nach dem Einsatz habe ich die generierten Ergebnisse jeweils vollumfänglich geprüft und überarbeitet. Ich versichere die lückenlose Einhaltung der erlernten wissenschaftlichen Standards und übernehme die volle Verantwortung für die gesamte vorliegende Arbeit. Mir ist bekannt, dass die vorliegende Arbeit mit 0 Punkten bewertet werden kann, wenn gegen diese Grundsätze verstoßen wurde, und dass in diesem Fall eine Zulassung zum Abitur nicht möglich ist.

Ort, Datum	Unterschrift der Schülerin / des Schülers

Benutzung von KI-Tools und Anhang

- ChatGPT

  Zur Verbesserung der sprachlichen Verständlichkeit mit Prompts wie:

  Wie könnte man diesen Textausschnitt verständlicher schreiben: (...)

  Formuliere diesen Textausschnitt flüssiger und verständlicher: (...)

- ChatGPT Codex

  Zum Programmieren der Vorlage des Vergleichssystems

  `https://github.com/SxGSimulationxG/compressionsystems/pull/1`

  `https://github.com/SxGSimulationxG/compressionsystems/pull/2`

  `https://github.com/SxGSimulationxG/compressionsystems/pull/3`

Inhaltlich wurde nichts mit KI-Tools generiert oder für die Suche nach Quellen verwendet. Alle für die Arbeit verwendeten Dateien inklusive Quellen sind unter diesem Link sichtbar: https://github.com/SxGSimulationxG/compressionsystems.
Alle Informationen zu den Dateien stehen in der README.md unter demselben Link.

Inhaltsverzeichnis

1 Einleitung

Abstrakt - In dieser Arbeit werden drei grundlegende Datenkompressionsalgorithmen, namentlich Huffman- und Shannon-Fano-Kodierungssystemen verglichen, sowie verschiedene Verbesserungen und Anwendungen verlustfreier Kompressionssalgorithmen betrachtet.

Verlustfreie Datenkompressionssysteme sind für unser modernes digitales Leben von zentraler Bedeutung. Ohne sie wäre der heutige technologische Fortschritt kaum denkbar. Wie der Name schon andeutet, handelt es sich dabei um Verfahren, die den Speicherbedarf von Daten reduzieren, ohne dabei Informationen zu verlieren. Die Grundlage dieser Systeme bilden die Kodierungsverfahren von Claude E. Shannon, Robert Fano und David A. Huffman. Seit ihrer Einführung wurden zahlreiche Varianten und Verbesserungen entwickelt, die unterschiedliche Ansätze zur Optimierung der Kompressionsleistung verfolgen. Dieses Prinzip ist eng mit der Geschichte der Kryptologie verbunden.

Bereits im 4. Jahrhundert v. Chr. können wir in Griechenland beobachten, wie Polybius von einem verbesserten Kommunikationssystem berichtet, das von Aeneas Tacticus eingeführt wurde. Dabei verwendeten zwei Personen, die sich in Sichtweite befanden, eine Fackel und zwei identische, beschriftete Gefäße, um in Kriegszeiten schneller zu kommunizieren[1]. Hier verwendeten sie Wasser und Gravuren in den Gefäßen, wobei eine bestimmte Wassermenge eine bestimmte Nachricht darstellte. Dies ist ein frühes Beispiel für eine kryptographische Chiffre, bei der eine bestimmte Eingabe, hier die Wassermenge, zu einer bestimmten Klartextnachricht entschlüsselt werden kann[2]. Spätere optische Telegraphiesysteme (Semaphoren) folgten einem ähnlichen Prinzip: vordefinierte Zeichenstellungen standen für festgelegte Botschaften. Der Nachteil solcher Systeme lag in der begrenzten Informationsmenge und der zeitaufwendigen Übermittlung einzelner Zeichen.

Daraus entstand die Idee, Nachrichten möglichst effizient zu kodieren, also Daten zu komprimieren, ohne Informationen zu verlieren. Den ersten theoretisch fundierten Ansatz entwickelte Claude Shannon in seinem „noiseless coding theorem“, das die Grundlage der modernen Informationstheorie bildet[3]. Etwa zur gleichen Zeit entwickelte Robert Fano das später als Fano-Kodierung bekannte Verfahren. Obwohl sich die Kodierungssysteme von Shannon und Fano grundlegend unterscheiden, werden sie oft zur Shannon-Fano-Kodierung kombiniert, wobei meist das Kodierungssystem von Fano gemeint ist. Beide Systeme sind jedoch suboptimal, wenn es um die resultierende Kompressionsgröße geht[4].

Eine entscheidende Verbesserung gelang David A. Huffman im Rahmen eines Seminars zur Informationstheorie bei Robert Fano. Antelle einer Abschlussprüfung entschied er sich für eine Projektarbeit mit dem Ziel, das optimale verlustfreie Kompressionsverfahren für ein binäres System zu finden – eine Aufgabe, an der sich selbst Shannon und Fano versucht hatten. Das Ergebnis war einer

[1] [17] Book X S.42f.: 44. The Improvement introduced by Aeneas Tacticus
[2] [8] S.32: Cipher (American English)
[3] [14] S.36: 2.2.1 Introduction to Information Theory
[4] [10] S.1: Introduction

der einflussreichsten Arbeiten auf diesem Gebiet: „A method for the construction of minimum-redundancy codes" und bewies damit, dass dies das optimale System zur Konstruktion eines solchen verlustfreien Komprimierungssystems war[5]. Seitdem wurde das Feld der verlustfreien Datenkompression erheblich weiterentwickelt. Moderne Verfahren wie adaptive, kontextbasierte oder grammatikbasierte Systeme bauen direkt auf den ursprünglichen Ideen von Shannon, Fano und Huffman auf und passen sie an neue Anwendungen und Datenstrukturen an. Die Bedeutung der jeweiligen Verbesserungen hängt jedoch stark vom Anwendungsbereich ab, weshalb eine vollständige Darstellung aller Varianten im Rahmen dieser Arbeit nicht möglich ist.

1.1 Zielsetzung

Diese Seminararbeit erläutert zunächst die Funktionsweise der grundlegenden Kompressionssysteme von Huffman, Shannon und Fano und zeigt, worin sich diese Verfahren unterscheiden. Anschließend werden die Unterschiede zwischen statischen und adaptiven Systemen beschrieben sowie deren Weiterentwicklungen und heutige Einsatzgebiete untersucht. Schließlich werden die drei Basissysteme mithilfe eines in Java implementierten Vergleichssystems anhand von Beispieldaten praktisch gegenübergestellt, um die Unterschiede in ihrer Effizienz zu analysieren und mögliche Auffälligkeiten in den Ergebnissen zu identifizieren.

1.2 Grundlegende Prinzipien

1.2.1 Unterschied zwischen verlustfreien und verlustbehafteten Kompressionssystemen

Verlustfreie Kompressionssysteme sind Verfahren, bei denen die ursprünglichen Daten nach der Dekompression vollständig und in unveränderter Qualität wiederhergestellt werden können. Im Gegensatz dazu entfernen verlustbehaftete Kompressionssysteme gezielt weniger relevante Informationen, um eine höhere Kompressionsrate zu erzielen[6]. Diese Arbeit befasst sich ausschließlich mit verlustfreien Kompressionssystemen.

1.2.2 Informationsentropie

Die Informationsentropie beschreibt den durchschnittlichen Informationsgehalt pro Symbol in einer Datenquelle. Bei einem optimalen verlustfreien Kompressionssystem entspricht die mittlere Codewortlänge genau dem Wert der Entropie. Dieses Konzept wurde von Claude E. Shannon als zentrales Element seiner Informationstheorie eingeführt. Sinkt die durchschnittliche Codewortlänge unter den Entropiewert H, geht Information verloren und das System ist dann nicht mehr verlustfrei.

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

Hier beschreibt p_i die Auftrittswahrscheinlichkeit eines Zeichens in der Eingabesequenz. Diese Gleichung gilt außerdem unter der Annahme, dass die Eingabezeichen unabhängig voneinander auftreten[7].

[5] [12]: S.1: Introduction
[6] [1] S.40f.: 7 Lossy Compression Techniques
[7] [16] S.396–399: 7 The Entropy of an Information Source

1.2.3 Unterschied zwischen Higher-order und Zeroth-order Systemen

In einem zeroth-order System wird jedes Zeichen unabhängig voneinander kodiert. Korrelationen zwischen aufeinanderfolgenden Zeichen in der Eingabesequenz werden also nicht berücksichtigt. Ein higher-order System hingegen erkennt und nutzt Abhängigkeiten über mehrere Zeichen hinweg, wodurch sich die Kompressionseffizienz oft deutlich verbessert[8].

1.2.4 Unterschied zwischen statischen und adaptiven Kodierungssystemen

In einem statischen Kodierungssystem sind die Wahrscheinlichkeiten der Eingabezeichen bereits vor der Kodierung bekannt und bleiben während des gesamten Prozesses unverändert. Ein adaptives System hingegen kennt diese Wahrscheinlichkeiten nicht im Voraus. Es analysiert die Daten während der Kodierung und passt das Kodierungsschema dynamisch an die Häufigkeit der bisher aufgetretenen Zeichen an[9].

1.2.5 Optimalität

Die Optimalität eines Codes wird von Huffman als ein Kodierungsschema definiert, das die kürzestmögliche durchschnittliche Codewortlänge bei minimaler Redundanz besitzt (engl. minimum-redundancy codes)[10]. Ein theoretisch optimaler Kodierungsalgorithmus erreicht eine mittlere Codewortlänge, die exakt dem Entropiewert H der Eingabesequenz entspricht.

1.2.6 Präfixfreiheit

Die Präfixfreiheit (oder Präfixeigenschaft) bedeutet, dass kein Codewort der Präfix eines anderen sein darf. Dadurch kann jedes Codewort eindeutig identifiziert werden, was die Dekodierung erheblich vereinfacht[11].

Ein Beispiel für ein nicht präfixfreies Kodierungsschema wäre: $a \rightarrow 0$ und $b \rightarrow 00$. Hier ist das Codewort von a ein Präfix von b, sodass beim Lesen von 00 unklar bleibt, ob es sich um ein einzelnes b oder zwei aufeinanderfolgende a handelt.

[8] [14] S.40–44: 2.2.4 Higher-order Sources
[9] [14] S.60f.: 2.5 Adaptive Coding
[10] [7] S.1098: Introduction
[11] [18] S.22f.: 2.1 Symbol Codes

2 Grundlegende Kompressionssysteme

2.1 Shannon Kodierung

Auch wenn die von Claude E. Shannon vorgeschlagene Shannon-Methode aufgrund ihrer Ineffizienz hinsichtlich der resultierenden Kompressionsgröße nicht häufig verwendet wird, hat sie dennoch eine gewisse Relevanz in dem Bereich der Informationstheorie. Dieses Verfahren ordnet jedem Symbol über eine kumulative Wahrscheinlichkeitsfunktion P_i einen eindeutigen Abschnitt im Intervall $[0;1[$ zu und erzeugt daraus eine Zahlenfolge, die als Grundlage für die Codewörter dient[12]. Die kumulative Wahrscheinlichkeitsfunktion wird rekursiv definiert als:

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

Die Länge jedes Codeworts ergibt sich aus der Wahrscheinlichkeit des jeweiligen Symbols nach:

$$l_i = \lceil log_r(p_i^{-1}) \rceil \tag{3}$$

wobei seltener auftretende Symbole längere Codewörter erhalten.

Shannon stellte außerdem fest, dass mit zunehmender Länge N, sich die durchschnittliche Codewortlänge H' der Entropie H nähert, was zeigt, dass Shannons Methode zwar nicht optimal, aber asymptotisch effizient ist.

$$H' = \frac{1}{N} \sum (m_s p_s) \tag{4}$$

Hier ist p_s der Wert für die kumulative Wahrscheinlichkeitsfunktion P_i. m_s ist eine ganze Zahl, wobei:

$$\log_2(\frac{1}{p_s}) \leq m_s < \log_2(\frac{1}{p_s}) + 1 \tag{5}$$

$$H \leq H' < H + \frac{1}{N} \tag{6}$$

Damit lieferte Shannon zwar kein optimales, aber ein einfach berechenbares Kodierungssystem. Shannon beschrieb auch die Grenzen der theoretischen Effizienz und bildete die Grundlagen für spätere Verfahren.

2.1.1 Beispiel: Statische Shannon Kodierung

Eingabe:

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

[12] [14] S.52f.: 2.4.2 Shannon's Method; [16] S.401-403: 9. The Fundamental Theorem for a Noiseless Channel

s_i	p_i	P_i	l_i	r-Darstellung von P_i	Code
a	0.22	0	3	$0.000\ldots$	000
b	0.20	0.22	3	$0.001\ldots$	001
c	0.18	0.42	3	$0.011\ldots$	011
d	0.15	0.60	3	$0.100\ldots$	100
e	0.10	0.75	4	0.11	1100
f	0.08	0.85	4	$0.1101\ldots$	1101
g	0.05	0.93	5	$0.11101\ldots$	11101
h	0.02	0.98	6	$0.111110\ldots$	111110

Tabelle 1: Angepasst aus Beispiel von [7] S.1101: Table III

2.1.2 Adaptive Shannon Kodierung

Travis Gagie schlug eine Methode zur Erstellung eines adaptiven Shannon-Kodierungssystems vor. Bei dieser Methode werden die Aufgaben in Vorder- und Hintergrundprozesse unterteilt. Die Vordergrundaufgabe berücksichtigt das Gewicht der vorherigen Zeichenanzahl und aktualisiert die Gewichte entsprechend dem neu eingegebenen Zeichen. Befindet sich das Zeichen noch nicht im Kodierungsschema, wird der Knoten zur Zeichenanzahl hinzugefügt. Im Hintergrund wird parallel für jedes im Vordergrund verarbeitete Zeichen ein Zeichen nachgeführt, sodass die Gewichtungen kontinuierlich an die tatsächliche Symbolverteilung angepasst werden. Die Gewichtung eines bestimmten Zeichens a wird also jedes Mal aktualisiert, wenn es auftritt, oder neu mit einem Gewicht eingefügt, das seiner geschätzten Wahrscheinlichkeit entspricht. Diese Trennung ermöglicht es, die Wahrscheinlichkeitsverteilung laufend zu aktualisieren, ohne den gesamten Kodierungsbaum neu zu berechnen[13].

[13] [5] S.3f.: III Dynamic Shannon Coding

2.2 Fano Kodierung

Die Fano-Methode beginnt mit einem Sequenzcode $\mathcal{S}$, der nach absteigender Wahrscheinlichkeit in $\mathcal{P}$ sortiert ist. Im binären System teilen wir diese Liste dann in zwei Untergruppen mit ungefähr gleicher Wahrscheinlichkeit $\mathcal{S}_0$ und $\mathcal{S}_1$ und damit auch $\mathcal{P}_0$ und $\mathcal{P}_1$. Ihre Codes beginnen dann entweder mit 0 oder 1. Dieser Vorgang wird dann so lange wiederholt, bis die Unterteilungen den einzelnen Zeichen entsprechen. Hier ist die Codewortlänge gleich $\log_2 m$, wenn m eine Potenz von zwei ist. Wenn m keine Zweierpotenz ist, können die Codewortlängen nicht alle gleich sein. In diesem Fall erhalten einige Symbole ein Bit mehr als andere, um die Präfixfreiheit zu gewährleisten[14].

2.2.1 Beispiel: Statische Fano Kodierung

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

s_i	p_i	L_i	Code
a	0.22	2	00
b	0.20	2	01
c	0.18	3	100
d	0.15	3	101
e	0.10	3	110
f	0.08	4	1110
g	0.05	5	11110
h	0.02	5	11111

Tabelle 2: Angepasst aus Beispiel von [7] S.1101: Table III

[14] [4] S.5f.: Selection from N Equally Likely Choices; [14] S.55: Algorithm 3 Static Fano Encoding

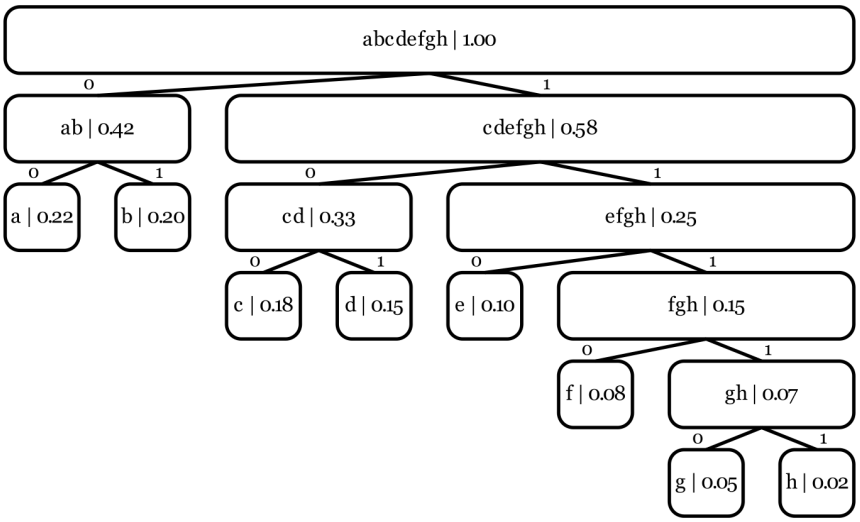

Abbildung 1: Fano Baum von Tabelle 2; Mit Manim Community v0.19.0 erstellt

2.2.2 Adaptive Fano Kodierung

Im Gegensatz zur statischen Variante, bei der die Wahrscheinlichkeiten vorher bekannt sind, muss die adaptive Fano-Kodierung ihre Struktur während der Kodierung ständig anpassen. Rueda schlug dazu zwei Ansätze vor: eine exakte, aber rechenintensive Methode (Brute-Force) und eine effizientere, heuristische Methode (Greedy). Bei der Brute-Force-Methode berechnet das System bei jeder Änderung des Zeichenzählers den entsprechenden Kompressionsbaum neu[15]. Dies ist natürlich ziemlich ineffizient, aber funktioniert immer, um einen statisches in ein adaptives Kompressionssystem zu wandeln. Bei der Greedy-Methode beginnen wir mit einer Liste des Eingabealphabets und initialisieren die Wahrscheinlichkeiten jedes Zeichens als gleich. Immer wenn ein Zeichen kodiert wird, wird ein bestimmtes Partitionierungsverfahren aufgerufen, das das Codewort ausgibt. Je nach verwendetem Codealphabet können verschiedene Partitionierungsverfahren zum Einsatz kommen. Der Einfachheit halber wird angenommen, dass dieses System die Wahrscheinlichkeiten auf die eine oder andere Weise optimal partitioniert, aber weitere Informationen zu verschiedenen Partitionierungsverfahren sind in [13] zu finden. Das Ergebnis ist, dass wir bei der Codierung eines Zeichens in unserer Eingabesequenz nicht mehr unseren Komprimierungsbaum für jedes neue Eingabezeichen aktualisieren müssen. Es werden nur die Änderungen im Gewicht eines bestimmten Zeichens berücksichtigt und alle erforderlichen Änderungen im aktuellen Kodierungsschema[16].

[15] [13] S.1659f.: 2.1 A brute-force method for adaptive Fano coding
[16] [13] S.1660: 2.2 The greedy encoding algorithm

2.3 Huffman Kodierung

Die Huffman-Kodierung ist ein Verfahren zur Erstellung eines optimalen Präfixcodes für einen gegebenen Sequenzcode mit bekannten Wahrscheinlichkeiten. Das Grundprinzip besteht darin, dass häufig auftretende Symbole kürzere und seltene längere Codewörter erhalten. Zu Beginn wird der Sequenzcode $\mathcal{S}$ nach absteigender Wahrscheinlichkeit sortiert. Jedes Symbol s_i wird durch einen Knoten q_i mit Gewicht $\tau_i = p_i$ in der Liste $\mathcal{Q}$ dargestellt.

$$p_i \geq p_{i+1} \geq \cdots \geq p_m \tag{7}$$

$$q_i \leftarrow s_i, (\tau_i = p_i) \tag{8}$$

Wir nehmen dann maximal r, mindestens jedoch 2 Zeichen mit der geringsten Wahrscheinlichkeit, verbinden sie mit einem neuen Knoten und fügen diesen neuen Knoten wieder in die richtige Position entsprechend seinem Gewicht in das Array $\mathcal{Q}$ ein. Durch das wiederholte Zusammenfassen der unwahrscheinlichsten Knoten entstehen in der Baumstruktur kurze Wege für häufige Symbole und lange Wege für seltene. Dadurch wird die Gesamtlänge aller Codewörter minimiert.

$$q_{new} \leftarrow q_{new\,j}, \tau_{new} = \sum \tau_i \tag{9}$$

Dieser Vorgang wird so lange wiederholt, bis der gesamte Wahrscheinlichkeitsraum durch einen einzigen Wurzelknoten mit $\tau_{new} = 1$ dargestellt wird. Am Ende kann man den Kompressionsbaum visualisieren, indem man die Kinder jedes Knotens nachverfolgt, bis man die Quellzeichen erreicht[17].

2.3.1 Beispiel: Statische Huffman Kodierung

Eingabe:

- $\mathcal{S} = \{a, b, c, d, e, f, g, h\}$
- $\mathcal{P} = \{0.22, 0.20, 0.18, 0.15, 0.10, 0.08, 0.05, 0.02\}$
- $\mathcal{A} = \{0, 1, 2, 3\} \rightarrow r = 4$

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

Tabelle 3: Beispiel von [7] S.1101: Table III

In Abbildung 2 stellen die grünen Knoten die einzelnen Zeichen mit ihren entsprechenden Wahrscheinlichkeiten p dar, während die violetten Knoten die Verbindungsknoten q mit ihrem entsprechenden Gewicht τ darstellen. Bei der Kodierung und Dekodierung verwenden wir diesen Baum als Übersetzungshilfe, wobei der am weitesten links stehende Knoten 0 ist, dann 1 usw.

[17] [7] S.1011: Generalization of the Method; [14] S.50: Algorithm 1 Static Huffman Encoding

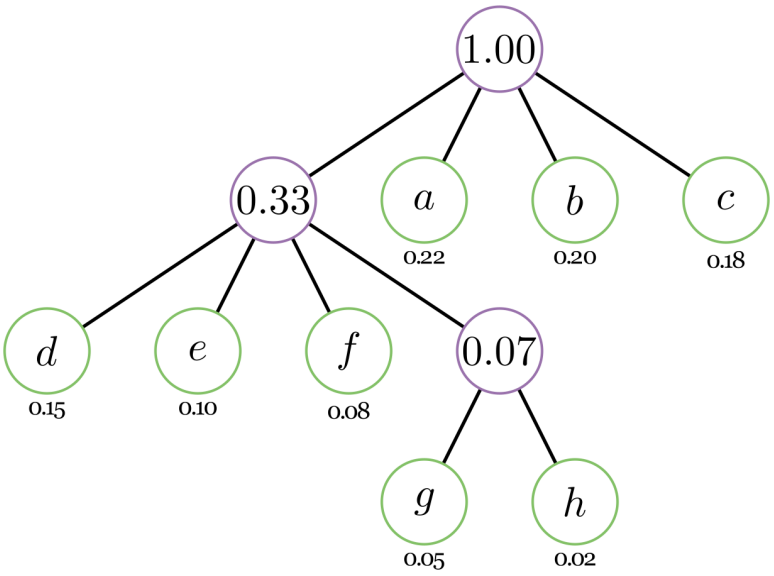

Abbildung 2: Huffman Baum von Tabelle 3; Mit Manim Community v0.19.0 erstellt

2.3.2 Adaptive Huffman Codierung

Die adaptive oder dynamische Huffman-Kodierung erweitert das klassische statische Verfahren um die Fähigkeit, sich während der Kodierung automatisch an Veränderungen der Symbolhäufigkeiten anzupassen. Im Gegensatz zur statischen Variante, bei der die Wahrscheinlichkeiten der Symbole im Voraus bekannt sein müssen, wird der Huffman-Baum hier fortlaufend angepasst, während die Eingabedaten gelesen werden. Dadurch kann die Kompression in einem einzigen Durchlauf erfolgen, ohne dass ein vorheriger Statistikaufbau erforderlich ist. Die Grundidee besteht darin, dass der Baum nach jedem neu eingelesenen Zeichen so umstrukturiert wird, dass er weiterhin die optimalen Eigenschaften der Huffman-Kodierung erfüllt. Diese Eigenschaft wird durch die sogenannte Geschwister-Eigenschaft (engl. sibling property) definiert. Sie besagt, dass für eine gegebene Kodierung der Baum genau dann optimal ist, wenn alle Knoten in absteigender Reihenfolge ihrer Gewichtungen angeordnet werden können und jedes Knotenpaar von Geschwistern aufeinanderfolgende Positionen in dieser Ordnung einnimmt[18]. In einem binären System besteht jeder Knoten (oder jedes Geschwisterpaar) aus fünf Komponenten: zwei Zählern für die untergeordneten Knoten, zwei Zeigern auf diese untergeordneten Knoten sowie einem Zeiger auf den übergeordneten Knoten. Jeder dieser Zeiger enthält zusätzlich die Information, ob es sich um den linken (0) oder rechten (1) Kindknoten handelt. Das Verfahren arbeitet fortlaufend: Als Eingabe dient der aktuelle Huffman-Baum sowie das nächste zu kodierende Zeichen. Nach jedem Schritt wird das Gewicht des entsprechenden Knotens – also die Häufigkeit des Symbols – um eins erhöht. Dadurch kann sich die Reihenfolge der Knoten ändern: Sobald ein Knoten ein höheres Gewicht erreicht als das nächstgrößere Geschwisterpaar, werden die beiden Knoten vertauscht. Auf diese Weise bleibt der Baum stets in einer Form, die der optimalen Huffman-Struktur

[18] [6] S.6: 2. The Sibling Property Definition

entspricht. Wird ein bisher unbekanntes Symbol eingelesen, so wird ein neuer Knoten für dieses Symbol eingefügt und in die bestehende Baumstruktur integriert. Die Kodierung erfolgt dann unmittelbar mit dem aktualisierten Baum, sodass die Methode immer konsistent bleibt. Das Dekodieren kann parallel erfolgen, da der Empfänger dieselbe Abfolge von Baumaktualisierungen nachvollzieht. Diese adaptive Methode wurde von Robert G. Gallager vorgeschlagen, der die theoretischen Grundlagen der Geschwister-Eigenschaft entwickelte und zeigte, dass sie eine eindeutige und optimale Baumstruktur garantiert, solange die Aktualisierungsregeln eingehalten werden[19]. Donald E. Knuth verbesserte später Gallagers System, indem er effiziente Verfahren zur Verwaltung der Knoten und zum Umgang mit der Verringerung von Symbolzählern einführte, falls Zeichen aus dem Datenstrom entfernt werden[20]. Das adaptive Huffman-System stellt somit eine effiziente Lösung für Echtzeit-Kompression dar, insbesondere in Szenarien, in denen die Symbolwahrscheinlichkeiten nicht im Voraus bekannt oder nicht konstant sind.

2.4 Unterschied in der Effizienz zwischen statischen und adaptiven Kodierungssystemen

Theoretisch gesehen sind statische Kodierungssysteme optimaler, aber nur wenn die Wahrscheinlichkeiten der Eingabesequenz genau bekannt sind und nicht über Zeit variieren. So sind in der Realität adaptive Verfahren meist praktischer und effizienter, da sie sich selbstständig gemäß der nachfolgenden Datenmenge anpassen. Bezüglich der Optimalität adaptiver Kodierungssysteme wurde durch mehrere Beispiele gezeigt, dass sie nur wenig von ihren statischen Versionen abweichen[21]. So lohnt es sich in der Realität fast immer einen adaptiven statt einen statischen Kodierungssystem zu implementieren.

[19] [6] S.17–21: Adaptive Huffman Codes
[20] [9] S.163–165: Introduction
[21] [3] S.314: VI Summary

3 Higher-order Systeme

3.1 Dictionary Based Kodierung

Es gibt viele Beispiele für die Dictionary Based Kodierung, eines davon stammt von Storer und Szymanski in ihrer Arbeit „Data Compression via Textual Substitution“[19], in der sie ein allgemeines Modell für Datenkompression vorstellen. Ihr System erweitert die herkömmliche Kompression um mehrere Komponenten: externe Makroschemata, interne Makroschemata, ein Wörterbuch (dictionary) und ein Skelett. Das externe Makroschema ermöglicht die Kodierung einer Quellzeichenfolge mithilfe eines Wörterbuchs, eines Speichers für Referenzzeichenfolgen und eines Skeletts – einer Kombination aus Zeichen des Eingabealphabets und Zeigern auf das Wörterbuch. Das interne Makroschema hingegen erlaubt Zeiger auf wiederholte Abschnitte innerhalb derselben Zeichenfolge[22]. Durch diese Technik wird die Redundanz in der Eingabesequenz verringert. Das in [19] vorgestellte Makromodell bildet die theoretische Grundlage für Verfahren wie LZ77 und dessen Varianten (z. B. LZSS), die später in Programmen wie ARJ oder RAR eingesetzt wurden. Weitere Beispiele für Dictionary Based Kodierungssysteme sind LZ77, LZW, DEFLATE und LZMA[23].

3.2 Block-Sorting Kompressionsalgorithmus

Der Block-Sorting-Kompressionsalgorithmus, auch bekannt als Burrows-Wheeler-Transformation (BWT), ist eine Methode zum Sortieren einer Eingabesequenz in eine Form, die für die Komprimierung mit anderen Methoden besser geeignet ist. Wir beginnen mit einer Eingabesequenz $\mathcal{X} = \{x_1 \to x_M\}$ und erstellen eine $M \times M$-Matrix. Jede Zeile der Matrix ist die Eingabesequenz, verschoben um eine Spalte im Gegensatz zur Zeile davor. Anschließend sortieren wir die Matrix alphabetisch und bezeichnen sie als A. Die Ausgabe der Transformation besteht aus die letzte Spalte von A, die wir als T bezeichnen, und die Index k der angibt, in welcher Zeile die ursprüngliche Sequenz $\mathcal{X}$ steht. Zur Rücktransformation ordnet man die Zeichen in T alphabetisch neu, wodurch eine Liste L entsteht. Mithilfe einer Abbildung F, die die Position jedes Zeichens in L zu seiner Position in T speichert, kann man die ursprüngliche Sequenz $\mathcal{X}$ eindeutig rekonstruieren[24].

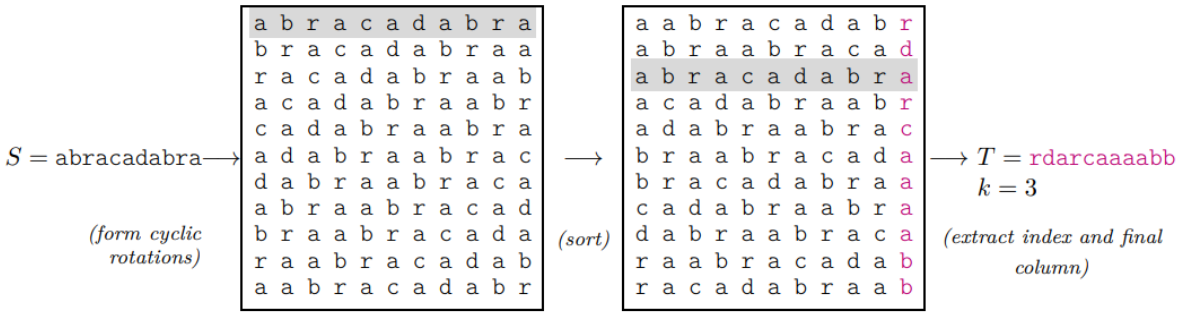

Abbildung 3: Beispiel aus [18]

Der eigentliche Vorteil dieser Methode liegt darin, dass die transformierte Sequenz T häufig viele gleiche oder ähnliche Zeichenfolgen enthält. Dadurch wird

[22] [19] S.929-932: 2 The Model and Basic Definitions
[23] [18] S.30f.: 2.3.2 History and Significance
[24] [14] S.79f.: 2.6.5 Block Sorting Compression; [18] S.33f.: Block Compression

sie besonders gut für nachgelagerte Verfahren wie Move-to-Front oder Huffman-Codierung geeignet, wie zum Beispiel beim bzip2 Format[25]. Die BWT selbst komprimiert die Daten also nicht direkt, sondern ordnet sie so um, dass andere Kodierungsverfahren ihre Redundanz wesentlich besser ausnutzen können.

3.3 Grammatikbasierte Kodierung

Bei der grammatikbasierten Kodierung (englisch: *Grammar-Based Coding*) wird die Eingabesequenz nicht direkt kodiert, sondern über eine Grammatik beschrieben, die genau diese Sequenz erzeugt. Dadurch lassen sich Wiederholungen und strukturelle Muster in der Zeichenkette effizient darstellen[26].

Eine Grammatik G besteht aus:

- einer Menge von Terminalsymbolen T (oder Alphabet A oder Σ),
- einer Menge von Nichtterminalen N,
- einer Menge von Produktionsregeln P,
- und einem Startsymbol s.

Mit diesem höheren Abstraktionsgrad wird nicht die Zeichenkette x selbst kodiert, sondern die Grammatik G_x, die x eindeutig erzeugt. Da G_x genau eine Zeichenkette erzeugt, handelt es sich um eine kontextfreie Grammatik, für die gilt:

$$L(G_x) = \{x\} \tag{10}$$

Das Ziel der grammatikbasierten Kodierung besteht darin, die kleinstmögliche Grammatik zu finden, die eine gegebene Zeichenfolge x beschreibt. Die Größe einer Grammatik $|G_x|$ entspricht der Anzahl der Symbole auf der rechten Seite der Produktionsregeln P. Dieses Optimierungsproblem wird als Smallest Grammar Problem (SGP) bezeichnet[27].

Ein Beispiel aus [2] verdeutlicht das Prinzip:

$$P = \{\langle S\rangle \to \langle B\rangle\langle B\rangle\langle A\rangle, \quad \langle A\rangle \to \text{a rose}, \quad \langle B\rangle \to \langle A\rangle \text{ is}\} \tag{11}$$

Diese Grammatik erzeugt die Zeichenkette „a rose is a rose is a rose“ und besitzt die minimale Größe $|G_x| = 14$. Damit beschreibt sie die Struktur der Wiederholungen sehr effizient, da gleiche Teilsequenzen nur einmal definiert werden müssen.

Das Finden der kleinsten Grammatik ist ein NP-schweres Problem. Ein Problem wird als NP-schwer bezeichnet, wenn es mindestens so schwierig ist wie die schwierigsten Probleme in der Komplexitätsklasse NP (nichtdeterministisch polynomiell)[28]. Das bedeutet, dass bisher kein Algorithmus bekannt ist, der das Problem in polynomieller Zeit für alle Eingaben lösen kann. Stattdessen wächst der Rechenaufwand mit zunehmender Eingabelänge exponentiell. In der Praxis greift man daher auf Heuristiken oder Approximationsverfahren zurück, um eine hinreichend kleine, aber nicht unbedingt optimale Grammatik effizient zu finden.

[25] [15] S.2: 1 Introduction
[26] [14] S.80f.: 2.6.6 Grammar-Based Coding
[27] [2] S.1f.: Introduction
[28] [2] S.4-7: V Hardness

3.4 Prediction with Partial Matching

Die Methode der Prediction with Partial Matching (PPM) ist ein kontextbasiertes Verfahren zur Wahrscheinlichkeitsvorhersage und Kodierung von Symbolen. Sie verwendet dabei eine spezielle Baumstruktur, einen sogenannten Trie, mit einer maximalen Tiefe D. Jeder Knoten im Baum repräsentiert einen Kontext, also eine bestimmte Folge vorheriger Zeichen, und enthält eine eigene Wahrscheinlichkeitsverteilung der möglichen Folgesymbole. Zusätzlich besitzt jeder Knoten sogenannte Vine-Zeiger, die auf übergeordnete Knoten verweisen und so den Übergang zu kürzeren Kontexten ermöglichen.

Der Algorithmus arbeitet in mehreren Schritten[29]:

1. Initialisieren Sie einen leeren Trie.
2. Wiederholen Sie die folgenden Schritte, bis das Ende der Eingabe erreicht ist:
   - Lesen Sie das nächste Eingabesymbol.
   - Verwenden Sie die Wahrscheinlichkeitsverteilung des aktuellen Knotens, um dieses Symbol zu kodieren.
   - Aktualisieren Sie anschließend die Wahrscheinlichkeitsverteilung und den entsprechenden Vine-Knoten.
   - Suchen Sie den Knoten, der dem aktuellen Symbol entspricht. Falls die maximale Tiefe D erreicht ist, folgen Sie dem Vine-Zeiger zur nächsthöheren Ebene. Existiert der benötigte Knoten nicht, wird er erstellt und der Baum entsprechend erweitert.
   - Setzen Sie den Zeiger des aktuellen Knotens auf den neuen Kindknoten und wechseln Sie zu diesem.

Dieses Verfahren lässt sich grafisch darstellen: Die durchgezogenen Linien zeigen den Pfad, den der Algorithmus durchläuft, während die gestrichelten Linien die Vine-Zeiger darstellen. Der schattierte Knoten markiert die Position nach Abschluss der Kodierung:

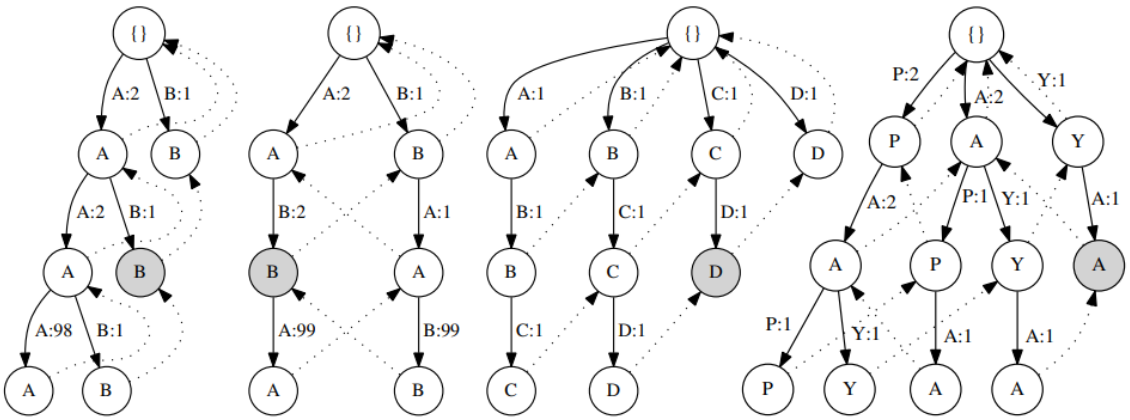

Abbildung 4: Beispiel aus [18]

[29] [18] S.111–113: 6.3.1 Basic Operation

Der Vorteil dieser Methode liegt darin, dass sie die Wahrscheinlichkeitsverteilung jedes Symbols dynamisch an den tatsächlichen Kontext der Daten anpasst. Dadurch kann PPM sehr präzise Vorhersagen über das nächste Zeichen treffen und so extrem effiziente Kodierungen erzeugen. Insbesondere verbessert sich die Kompressionsleistung gegenüber einfacheren Modellen (wie Zeroth- oder First-order-Modellen), weil PPM Redundanzen über längere Zeichenfolgen hinweg erkennt und nutzt. Je tiefer der Kontext (also je größer die Tiefe D des Tries), desto genauer kann der Algorithmus die Symbolwahrscheinlichkeiten abschätzen – allerdings auf Kosten höherer Rechenzeit und Speichernutzung. Insgesamt zählt PPM daher zu den effektivsten statistischen Verfahren für verlustfreie Datenkompression.

Dieses Verfahren wird meistens für die Kompression und Verarbeitung von menschlich geschriebenen Texten verwendet[30].

3.5 Weitere Higher-order Systeme

Die Anwendungsbereich solcher Systeme sind sehr vielfältig. Eine wichtige System noch für die verlustfreie Kompression ist das DEFLATE, welche in Formaten wie ZIP, PNG oder PDF verwendet wird[31]. Dies besteht aus eine Kombination der LZ77- und Huffman Kodierungsverfahren, wo zuerst Redundanzen der Quellensequenz durch das LZ77-Verfahren entfernt werden und dann durch das Huffman Kodierungsverfahren gesetzt werden[32]. Es gibt natürlich viele andere Higher-order Systeme und Algorithmen mit vielen verschiedenen Anwendungsbereiche und auch verlustbehaftete Kompressionssysteme, aber dies wird hier nicht weiter ausgeführt. Eine breitere Übersicht sind auch in [14], [1] und [18] zu finden.

[30] [18] S.139: Applications of PPM-like Algorithms
[31] [18] S.30f.: 2.3.2 History and Significance
[32] [11] S.432: Compression Algorithm (Deflate)

4 Vergleich der drei grundlegenden Kompressionsalgorithmen

Um die ursprünglichen Kompressionsalgorithmen zu vergleichen wurden zehn verschiedene Beispieldaten in ein Vergleichsprogramm gegeben[33]. Aus den Ausgabewerten wurden zwei Abbildungen erstellt:

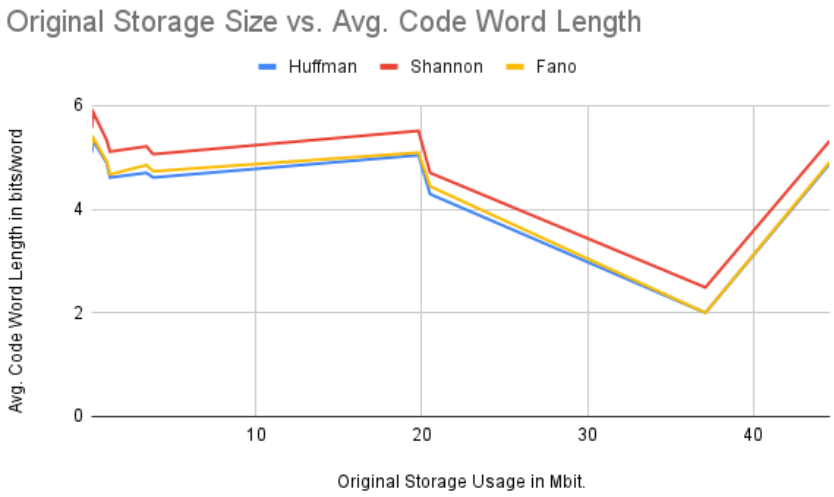

Abbildung 5: Mit Google Sheets erstellt

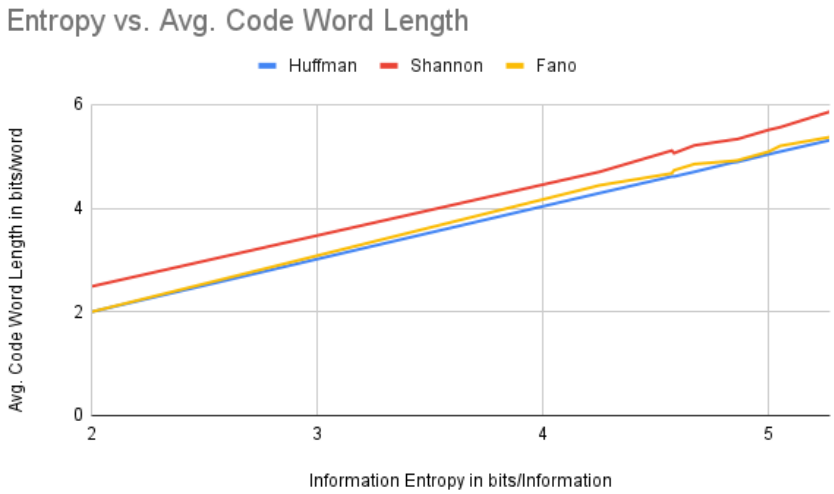

Abbildung 6: Mit Google Sheets erstellt

In Abbildung 5 lässt sich kein klarer Zusammenhang zwischen dem anfänglichen Speicherbedarf und der endgültigen Kompressionsgröße erkennen. In Abbildung 6 hingegen zeigt sich ein deutlicher Zusammenhang zwischen der Informationsentropie und der Kompressionsgröße beziehungsweise der durchschnittlichen Codewortlänge. In beiden Abbildungen wird deutlich, dass das Shannon-Verfahren deutlich ineffizienter ist als die Verfahren von Huffman und Fano, die wiederum eine sehr ähnliche Effizienz aufweisen. Dabei liegt Huffman meist knapp unter Fano und ist somit etwas effizienter.

[33] Siehe https://github.com/SxGSimulationxG/compressionsystems

5 Schlusswort

Auch wenn die ursprünglichen Komprimierungssysteme von Huffman, Shannon und Fano heute nicht mehr allzu frequent in ihrer ursprünglichen Form verwendet werden, bilden sie doch die Grundlage für moderne verlustfreie Komprimierungssysteme. Huffman zeigt uns, dass es in einem zeroth-order Komprimierungssystem einen optimalen Komprimierungsalgorithmus gibt, der mit einem Bottom-up-Konstruktionssystem funktioniert. Die Kodierungssysteme von Fano und Shannon sind zwar nicht so effektiv, wenn es darum geht, sich dem Wert der Entropie anzunähern, aber sie bieten einen intuitiveren Überblick und eine intuitivere Methode zur Implementierung eines verlustfreien Kompressionsalgorithmus. Anhand der verwendeten Beispiele können wir jedoch sehen, wie stark sich diese Verfahren unterscheiden. Wenn wir uns higher-order Modelle ansehen, werden die Algorithmen viel komplizierter und speichereffizienter, aber am Ende kommt es fast immer zu einem zeroth-order Komprimierungssystem, nur dass die Eingabe effektiver komprimiert werden kann als die ursprüngliche Eingabe.

Literatur

[1] Guy E Blelloch u. a. “Introduction to data compression”. In: *Computer Science Department, Carnegie Mellon University* 54 (2013).

[2] Moses Charikar u. a. “The smallest grammar problem”. In: *IEEE Transactions on Information Theory* 51.7 (2005), S. 2554–2576.

[3] J Cleary und I Witten. “A comparison of enumerative and adaptive codes”. In: *IEEE Transactions on Information Theory* 30.2 (1984), S. 306–315.

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

[5] Travis Gagie. “Dynamic shannon coding”. In: *European Symposium on Algorithms*. Springer. 2004, S. 359–370.

[6] Robert Gallager. “Variations on a theme by Huffman”. In: *IEEE Transactions on Information Theory* 24.6 (2003), S. 668–674.

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

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

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

[10] Stanislav Krajči u. a. “Performance analysis of Fano coding”. In: *2015 IEEE International Symposium on Information Theory (ISIT)*. IEEE. 2015, S. 1746–1750.

[11] Savan Oswal, Anjali Singh und Kirthi Kumari. “Deflate compression algorithm”. In: *International Journal of Engineering Research and General Science* 4.1 (2016), S. 430–436.

[12] Inna Pivkina. “Discovery of Huffman codes”. In: *Mathematical Association of America* (2014).

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

[14] Luis Gabriel Rueda. “Advances in data compression and pattern recognition”. Diss. Carleton University, 2002.

[15] Julian Seward. “bzip2 and libbzip2”. In: *http://www. bzip. org* (1996), S. 8–18.

[16] Claude Elwood Shannon. “A mathematical theory of communication”. In: *The Bell system technical journal* 27.3 (1948), S. 379–423.

[17] Evelyn S Shuckburgh. *The Histories of Polybius*. Bd. 2. MacMillan und Co., 1889.

[18] Christian Steinruecken. “Lossless data compression”. Diss. University of Cambridge, 2015.

[19] James A Storer und Thomas G Szymanski. “Data compression via textual substitution”. In: *Journal of the ACM (JACM)* 29.4 (1982), S. 928–951.