Friedrich-Koenig-Gymnasium Würzburg Oberstufenjahrgang 2024/2026

SEMINARARBEIT

im W-Seminar

WM Kryptologie

im Fach

Mathematik

Thema:

Quellenkodierung I - Huffman-/Shannon-Fano-Kodierung

Verfasser / -in: Kai Rasmussen

Kursleiter / -in: Oliver Manger

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.

Ort, Datum

Unterschrift der Schülerin / des Schülers

Inhaltsverzeichnis

1 Einleitung

Abstrakt - Der Vergleich zwischen den ursprünglichen Datenkompressionsalgorithmen, den Huffman- und Shannon-Fano-Kodierungssystemen, sowie eine Betrachtung der verschiedenen Verbesserungen und Anwendungen verlustfreier Kompressionssalgorithmen.

Verlustfreie Datenkompressionssysteme sind für unser modernes Leben von enormer Bedeutung. Ohne diese Systeme hätten wir dieses technologische Niveau nicht erreichen können. Wie der Name schon sagt, handelt es sich dabei um Systeme, die den Speicherplatzbedarf von Daten reduzieren, ohne dabei Informationen zu verlieren. Die Grundlage verlustfreier Datenkomprimierungssysteme bilden die Codierungssysteme von Huffman, Shannon und Fano. Seit der Einführung dieser Systeme gab es jedoch zahlreiche Verbesserungen und unterschiedliche Ansätze für das Problem der verlustfreien Datenkomprimierung. 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]. In ähnlicher Weise wurden viele Versionen des optischen Semaphors verwendet, wobei vordefinierte Interpretationen bestimmter Positionen verwendet wurden, um eine vorab festgelegte Nachricht zu übermitteln. Einer der Nachteile beider Systeme besteht darin, dass die übertragbaren Informationen begrenzt sind, da nicht jede mögliche Nachricht in eine Chiffre passt. Die Übermittlung einzelner Buchstaben würde sehr viel Zeit in Anspruch nehmen. Um die zu sendende Datenmenge zu reduzieren und gleichzeitig alle für die Erstellung einer Nachricht erforderlichen Informationen zu erhalten, entstand die Idee der verlustfreien Datenkompression. Der erste wichtige Kompressionsalgorithmus war die nach ihm benannte Kodierungssystem von Claude Shannon, das auf seinem Theorem zur rauschfreien Kodierung basiert (Englisch: noiseless coding theorem). Diese Theorem und Kompressionsalgorithmus bilden die Grundlage dessen, was später zum Gebiet der Informationstheorie werden sollte[3]. Etwa zur gleichen Zeit entwickelte Robert Fano das später als Fano-Kodierung bekannte Verfahren. Obwohl sich die Codierungssysteme von Shannon und Fano grundlegend unterscheiden, auch wenn sie in etwa auf die gleiche Weise funktionieren, werden sie oft zur Shannon-Fano-Codierung kombiniert, wobei meist das Codierungssystem von Fano gemeint ist. Beide Systeme sind jedoch suboptimal, wenn es um die resultierende Kompressionsgröße geht[4]. Das änderte sich, als David A. Huffman einen Elektrotechnik-Aufbaustudiengang zum Thema Informationstheorie bei Robert Fano besuchte. In diesem Kurs hatten die Studenten die Wahl zwischen einer Semesterarbeit und einer Abschlussprüfung. Das Thema der Semesterarbeit bestand darin, das optimale verlustfreie Komprimierungssystem für ein binäres System zu finden – genau das, was Fano und Shannon selbst zu erreichen

[1] [17] Book X pp.42-43: 44. The Improvement introduced by Aeneas Tacticus
[2] [9] p.32: Cipher (American English)
[3] [14] p.36: 2.2.1 Introduction to Information Theory
[4] [11] p.1: Introduction

versuchten. Huffman entschied sich, die Herausforderung anzunehmen und die Semesterarbeit zu schreiben. Das Ergebnis war eine der einflussreichsten Arbeiten auf diesem Gebiet: „A method for the construction of minimum-redundancy codes". Er bewies, dass dies das optimale System zur Konstruktion eines solchen verlustfreien Komprimierungssystems war[5]. Seitdem gab es viele Verbesserungen auf diesem Gebiet, von denen wir einige im Abschnitt über higher-order Systeme und im Zusammenhang mit den adaptiven Versionen dieser ursprünglichen Systeme diskutieren werden. Diese Verbesserungen sind jedoch sehr unterschiedlich, sodass es unmöglich wäre, sie alle vorzustellen und die Wichtigkeit dieser Verbesserungen hängen stark vom Anwendungsbereich ab.

1.1 Zielsetzung

In der folgenden Seminararbeit werden zunächst die Funktionsweise der grundlegenden Kompressionssysteme Huffman, Shannon und Fano erklärt, und wie sie sich voneinander unterscheiden. Anschließend wird auf den Unterschied zwischen statische und adaptive Systeme eingegangen und wie die gesammten Systeme verbessert wurden und wie sie heutzutage eingesetzt werden. Danach werden die grundlegenden Kompressionssysteme anhand von Beispieldaten mit einem Java-Vergleichssystem verglichen, um herauszufinden, wie stark sich diese Systeme unterscheiden und ob es bestimmte Auffälligkeiten bei den Datenbeispielen gibt.

1.2 Grundlegende Prinzipien

1.2.1 Unterschied zwischen verlustfreie und verlustbehaftete Kompressionssysteme

Verlustfreie Kompressionssysteme sind eine Kompressionsmethode, bei der die originale unkomprimierte Form in ihrer originalen Qualität rekonstruiert werden kann. Im Gegensatz dazu stehen die verlustbehafteten Kompressionssysteme, bei dem weniger relevante Informationen verworfen werden, um die Größe zu komprimieren[6]. Hier befassen wir uns ausschließlich mit verlustfreien Kompressionssystemen.

1.2.2 Informationsentropie

Die Informationsentropie beschreibt die durchschnittliche Informationsgehalt pro Bit. Bei einem optimalen verlustfreien Kompressionssystem entspricht die durchschnittliche Codewortlänge dem Wert der Informationsentropie. Dieses Konzept war eines der wichtigsten in Shannons Grundlagen der Informationstheorie. In dem Moment, in dem die durchschnittliche Codewortlänge unter den Wert der Entropie H fällt, geht Information verloren, sodass das Kompressionssystem nicht mehr verlustfrei ist.

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

Diese Gleichung für die Entropie gilt, wenn die Eingabezeichen mit deren Auftrittwahrscheinlichkeit voneinander unabhängig sind[7].

[5] [12]: p.1: Introduction
[6] [2] p.40-41: 7 Lossy Compression Techniques
[7] [16] p.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 codiert, wobei Korrelationen zwischen den Zeichen in der Eingabesequenz nicht berücksichtigt werden. Ein higher-order System ist in der Lage, Muster über mehrere Zeichen hinweg zu erkennen und zu nutzen[8].

1.2.4 Unterschied zwischen statischen und adaptiven Kodierungssystemen

In einem statischen Kodierungssystem verfügen wir vor der Kodierung der Sequenz über Kenntnisse hinsichtlich der Wahrscheinlichkeiten der Quellsequenz. In einem adaptiven System ist die einzige Eingabe, über die der Kodierer verfügt, die Quellsequenz selbst. Das System passt dann das Kodierungsschema an die Änderungen in der Häufigkeit der Eingabezeichen an[9].

1.2.5 Optimalität

Die Optimalität eines Codes wird von Huffman beschreiben als eine Kodierungsschema, welche die kurzmöglichste durschnittliche Codewortlänge besitzt und so am wenigsten Redundanz besitzt (engl. minimum-redundancy codes)[10]. Diese kurzmöglichste Codewortlänge wird auch in der Prinzip der Informationsentropie von Shannon beschrieben. Die theoretisch optimale Kodierungsalgorithmus besitzt eine durchschnittliche Codewortlänge, welche gleich ist wie die Wert der Informationsentropie eines Eingabesequenz besitzt.

1.2.6 Präfixfreiheit

Die Präfixfreiheit oder Präfixeigenschaft besagt, dass ein Codewort kein Präfix einer anderen sein kann. So kann kein Codewort mit einem anderen verwechselt werden und die Ende eines Codeworts eindeutig gefunden werden. Dies macht die Dekodierung deutlich einfacher[11]. Ein Beispiel für einer Kodierungsschema welche diese Eigenschaft nicht besitzt, wäre wenn man zum Beispiel a mit 0 kodiert und b mit 00. a ist hier ein Präfix von b. So kann man bei der Codewort 00 nicht wissen ob es sich um einen b handelt oder zwei as.

[8] [14] p.40-44: 2.2.4 Higher-order Sources
[9] [14] p.60-61: 2.5 Adaptive Coding
[10] [8] p.1098: Introduction
[11] [18] p.22-23: 2.1 Symbol Codes

2 Grundlegende Kompressionssysteme

2.1 Shannon Codierung

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 diesem Bereich. 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:

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

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 stellt 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, wo:

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

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

Damit liefert Shannon zwar kein optimales, aber ein einfach berechenbares Codierungssystem. Shannon beschreibt auch die Grenzen der theoretischen Effizienz und bildet die Grundlagen für spätere Verfahren.

2.1.1 Beispiel: Statisches 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] p.52-53: 2.4.2 Shannon's Method; [16] p.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 [8] p.1101: Table III

2.1.2 Adaptive Shannon Kodierung

Travis Gagie schlägt eine Methode zur Erstellung eines adaptiven Shannon-Kodierungssystems vor. Bei dieser Methode werden die Aufgaben in Vordergrund- 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][6] p.3-4: III Dynamic Shannon Coding

2.2 Fano Codierung

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: Statisches 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 [8] p.1101: Table III

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

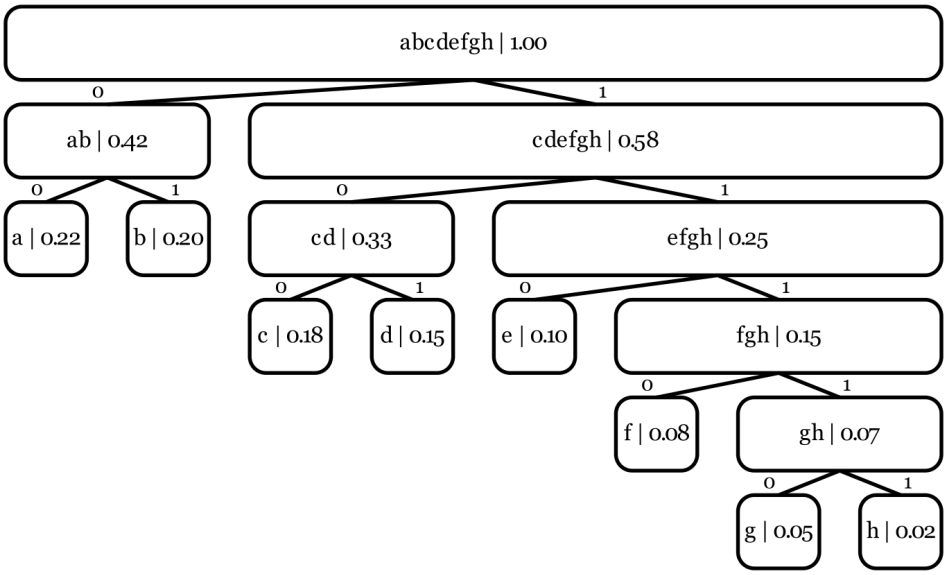

Abbildung 1: Fano Baum von Tabelle 2; Mit Manim Community v0.19.0 erstellt

2.2.2 Adaptive Fano Codierung

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 schlägt 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 codiert 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] p.1659-1660: 2.1 A brute-force method for adaptive Fano coding
[16] [13] p.1660: 2.2 The greedy encoding algorithm

2.3 Huffman Codierung

Die Huffman-Kodierung ist ein Verfahren zur Erstellung eines optimalen Präfixcodes für ein gegebenes Sequenzcode mit bekannten Wahrscheinlichkeiten. Das Grundprinzip besteht darin, dass häufig auftretende Symbole kürzere und seltene längere Codewörter erhalten. Zu Beginn wird die 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 nachverfolgen, bis man die Quellzeichen erreicht[17].

2.3.1 Beispiel: Statisches Huffman Codierung

Eingabe:

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

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 [8] p.1101: Table III

[17] [8] p.1011: Generalization of the Method; [14] p.50: Algorithm 1 Static Huffman Encoding

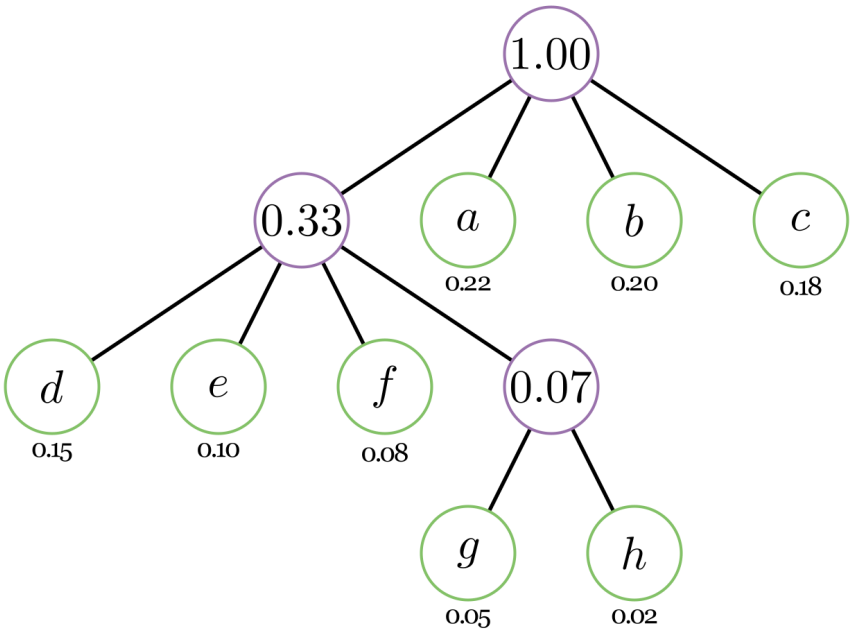

Abbildung 2: Huffman Baum von Tabelle 3; Mit Manim Community v0.19.0 erstellt

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.

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 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 statischen Kodierungssystemen optimaler, aber nur wenn die Wahrscheinlichkeiten der Eingabesequenz genau bekannt sind und nicht über Zeit varieren. So sind in der Realität adaptive Verfahren meist praktischer und effizienter, da

[18] [7] p.6: 2. The Sibling Property Definition
[19] [7] p.17–21: Adaptive Huffman Codes
[20] [10] p.163–165: Introduction

sie sich selbstständig anpassen an der zukommenden Datenmenge. Die Optimalität der adaptiven Kodierungssystemen wurde auch durch mehreren Beispiel gezeigt, dass sie nur wenig von ihren statischen Versionen abweichen[21].

[21] [4] p.314: VI Summary

3 Higher-order Systeme

eingabe/pr
-¿ funk-
tion
-¿ was
bringt es

3.1 Dictionary Based Kodierung

Storer und Szymanski legen in ihrer Arbeit „Data Compression via Textual Substitution" einen Grundstein für die dictionary-based Kompressionsverfahren. Ihr System füht mehrere Komponenten hinzu: externe Makroschemata, interne Makroschemata, ein Wörterbuch (dictionary) und ein Skelett. Das externe Makroschema ermöglicht die Kodierung einer Quellzeichenfolge unter Verwendung eines Wörterbuchs, eines Speichers für Referenzzeichenfolgen und eines Skeletts, einer Kombination aus Zeichen des Eingabealphabets und Zeigern auf das Wörterbuch. Die internen Makro-Schemata ermöglichen Zeigern auf wiederholte Abschnitte derselben Zeichenfolge[22]. Mit diesem System können wir den Grad der Redundanz in unserer Eingabesequenz verringern.

LZ77/Defla

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[23].

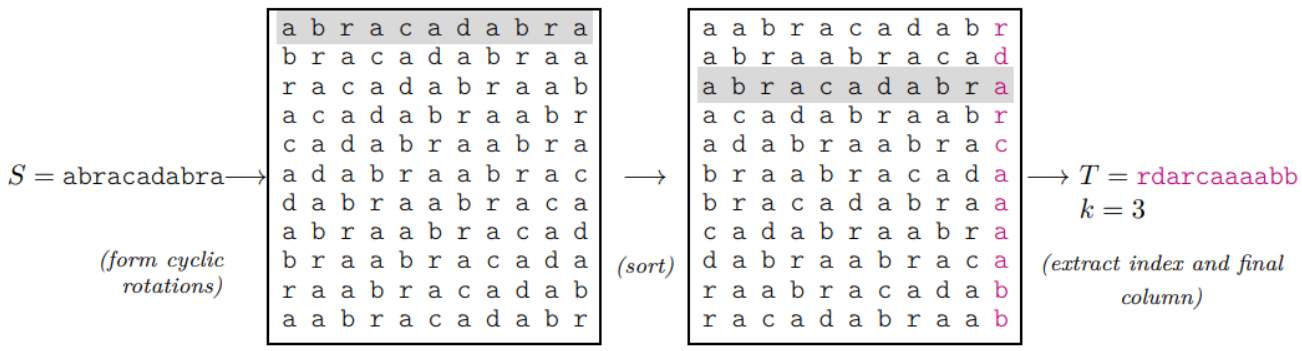

Abbildung 3: Beispiel aus [18]

Der eigentliche Vorteil dieser Methode liegt darin, dass die transformierte Sequenz T häufig viele gleiche oder ähnliche Zeichenfolgen enthält. Dadurch wird sie besonders gut für nachgelagerte Verfahren wie Move-to-Front oder Huffman-Codierung geeignet. Die BWT selbst komprimiert die Daten also nicht direkt, sondern ordnet sie so um, dass andere Kodierungsverfahren ihre Redundanz wesentlich besser ausnutzen können.

[22] [19] p.929-932: 2 The Model and Basic Definitions
[23] [14] p.79-80: 2.6.5 Block Sorting Compression; [18] p.33-34: Block Compression

3.3 Grammatikbasierte Codierung

Bei der grammatikbasierten Codierung (englisch: *Grammar-Based Coding*) wird die Eingabesequenz nicht direkt codiert, sondern über eine Grammatik beschrieben, die genau diese Sequenz erzeugt. Dadurch lassen sich Wiederholungen und strukturelle Muster in der Zeichenkette effizient darstellen[24].

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 codiert, 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 Codierung 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[25].

Ein Beispiel aus [3] 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). 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. citation needed

3.4 Prediction with Partial Matching

Die Methode der Prediction with Partial Matching (PPM) ist ein kontextbasiertes Verfahren zur Wahrscheinlichkeitsvorhersage und Codierung 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.

[24] [14] p.80–81: 2.6.6 Grammar-Based Coding
[25] [3] p.1–2: Introduction

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[26]:

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:

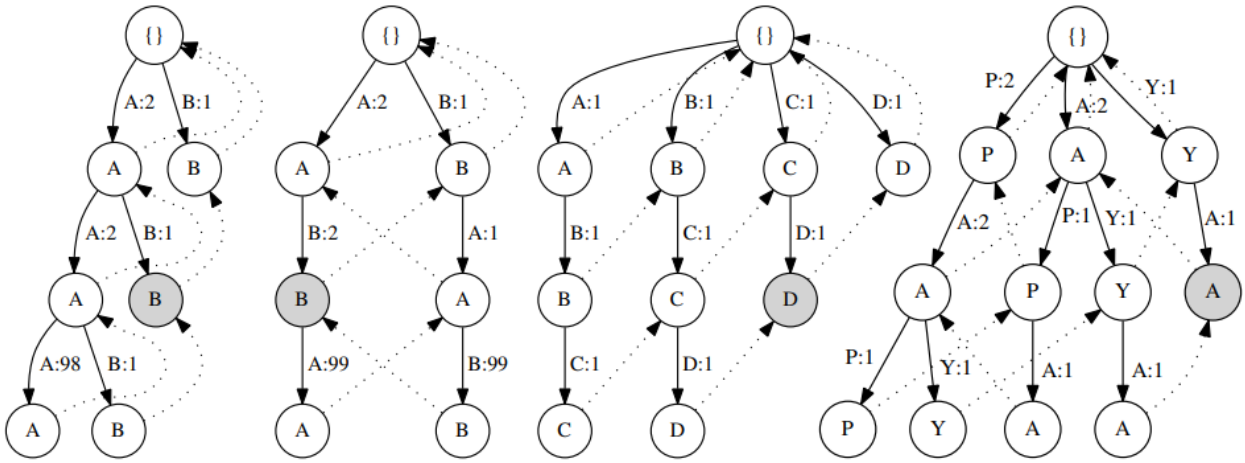

Abbildung 4: Beispiel aus [18]

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 Codierungen erzeugen. Insbesondere verbessert sich die Kompressionsleistung gegenüber einfacheren Modellen (wie Null- oder Erstordnungsmodellen), 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

[26][18], p.111–113: 6.3.1 Basic Operation

abschätzen – allerdings auf Kosten höherer Rechenzeit und Speichernutzung. Insgesamt zählt PPM daher zu den effektivsten statistischen Verfahren für verlustfreie Datenkompression.

4 Vergleich der ursprünglichen Kompressionsalgorithmen

Beispiel	**Speicherbedarf**[27]	**Entropie** H
Wordnik[28]	20.5	4.25
Shakespeare[29]	44.6	4.84

Tabelle 4: Liste mit Beispielen zum Vergleich der verschiedenen Kompressionssysteme mit ihrer entsprechenden Speichernutzung in Mb.

Kompressionssystem	**Prozesse**[30]	**Größe**[31]	$\overline{L}$ **Avg. L**[32]	**Prozentsatz**[33]
Huffman	2565462	11.0	4.29	53.6%
Shannon	2565500	12.0	4.70	58.7%
Fano	2565349	11.4	4.44	55.5%

Tabelle 5: Vergleich der verschiedenen Kompressionssysteme mit Wordnik

Kompressionssystem	**Prozesse**	**Größe**	$\overline{L}$ **Avg. L**	**Prozentsatz**
Huffman	5575693	27.2	4.88	61.0%
Shannon	5576446	29.6	5.31	66.4%
Fano	5575268	27.4	4.91	61.4%

Tabelle 6: Vergleich der verschiedenen Kompressionssysteme mit Shakespeare

[27] Berechnet Anzahl der Eingabezeichen multipliziert mit 8

[28] [1]: Wordnik Liste öffentlich zugänglicher englischer Wörter

[29] [15]: Die gesammelten Werke von William Shakespeare

[30] Berechnungsprozesse sind relativ, verschiedene Arten von Prozessen dauern unterschiedlich lange, daher sollte diese Zahl eher vage interpretiert werden.

[31] Speichernutzung nach Komprimierung ohne Berücksichtigung des Speicherplatzes, der vom Komprimierungsbaum oder Kodierungsschema belegt wird. Der Speicherplatz, den diese im Verhältnis zu einer größeren Datenmenge beanspruchen, ist so gering, dass wir diesen geringen zusätzlichen Speicherbedarf im Grunde ignorieren können.

[32] Durchschnittscodewortlänge

[33] Prozentualer Anteil der komprimierten Speichergröße im Vergleich zum Zeichensatz in Tabelle 4.

5 Schluss

Auch wenn die ursprünglichen Komprimierungssysteme von Huffman, Shannon und Fano heute nicht mehr so häufig 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, dass der Unterschied zwischen ihren Kompressionsgrößen zunächst recht gering ist, aber mit zunehmender Dateigröße zunimmt. In diesem Beispiel können wir auch sehen, wie Shannons Theorie funktioniert: Je größer die Datei wird, desto mehr Speicherplatz wird eingespart. Wenn wir uns higher-order Modelle ansehen, können die Dinge viel komplizierter und speichereffizienter werden, 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] URL: https://raw.githubusercontent.com/wordnik/wordlist/refs/heads/main/wordlist-20210729.txt (besucht am 23. 04. 2025).

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

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

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

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

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

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

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

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

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

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

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

[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.