Universität Paderborn
Fakultät Elektrotechnik, Informatik und Mathematik
Institute für Mathematik und Informatik
Warburger Str. 100
33102 Paderborn

Diplomarbeit

Tabellenbasierte arithmetische Kodierung bei allgemeinen Alphabeten

Marcel R. Ackermann
mra@uni-paderborn.de

vorgelegt bei

Prof. Dr. Johannes Blömer
Universität Paderborn
Institut für Informatik

25. Dezember 2003

Korrigierte Version vom 4. Februar 2004

Erklärung:

Hiermit versichere ich, dass ich die folgende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen als Hilfsmittel benutzt sowie Zitate kenntlich gemacht habe.[1]

Paderborn, 25. Dezember 2003

[1] Um diese Erklärung nicht ad absurdum zu führen, möchte ich hiermit auch direkt angeben, dass ich deren Formulierung sinngemäß und nahezu wörtlich aus der Diplomprüfungsordnung Mathematik der Universität Paderborn vom 1. Oktober 1996 übernommen habe.

Für meine Lehrer.

Vielen Dank an meinen Betreuer Prof. Dr. Johannes Blömer sowie an Dr. Alexander May, Birgitta Grimm, Martin Otto und Mirko Hessel für Literatur, Rat, Kritik und die vielfache, äusserst hilfreiche Diskussion.

Die untersuchten Testdaten entstammen dem Projekt „Online Codekompression für Smartcards“ mit freundlicher Unterstützung der Infineon Technologies AG München.

Diese Arbeit entstand mittels LaTeX 2_ε unter RedHat Linux 7.3. Mein Dank gilt Donald Knuth, Leslie Lamport und Linus Torvalds für diese hervorragenden Produkte.

Besonderen Dank an Martin Otto für die LaTeXnische Unterstützung sowie Elisabeth Kappius, Gabriele Kappius und Mirko Hessel für die Hilfe im Kampf gegen die deutsche Sprache.

Inhaltsverzeichnis

1 Einleitung **1**

2 Grundlagen **5**
2.1 Wahrscheinlichkeitstheorie 5
2.2 Kodierungstheorie 6
2.3 Informationstheorie 8

3 Tabellenbasierte arithmetische Kodierung für binäre Alphabete **11**
3.1 Einführung in die arithmetische Kodierung 12
3.2 Modell und Kontext 18
3.3 Ganzzahlige Arithmetik mit endlicher Präzision 20
3.4 Quasi-arithmetische Kodierung 26
3.5 Konstruktion der Dekodiertabellen 32

4 Kodierung über beliebigen Alphabeten durch Verteilungsbäume **35**
4.1 Grundsätzliches über Bäume 36
4.2 Konstruktion von Verteilungsbäumen 38
4.3 Binäre arithmetische Kodierung mit Verteilungsbäumen 40
4.4 Komprimierte Verteilungsbäume 45
4.5 Effiziente Datenstruktur komprimierter Verteilungsbäume 55
4.6 Fazit 59

5 Konzept eines Dekodierschaltkreises mit Tabellen-Arithmetik **63**
5.1 Idee des Schaltkreises 64
5.2 Geordnete Verteilungen 66
5.3 Dyadische Verteilungen 70
5.4 Baugruppen und deren Komplexität 78
5.5 Approximation durch dyadische Verteilungen 86
5.6 Empirische Ergebnisse 92
5.7 Fazit 99

Symbolverzeichnis

$\emptyset$	Leere Menge
$\mathbb{N}$	Menge der natürlichen Zahlen $1, 2, 3, \ldots$
$\mathbb{N}_0$	Menge der natürlichen Zahlen inklusive der Zahl 0
$\mathbb{R}$	Menge der reellen Zahlen
$[a, b]$	Abgeschlossenes Intervall der Zahlen x mit $a \leq x \leq b$
$[a, b)$	Rechts-offenes Intervall der Zahlen x mit $a \leq x < b$
$(a, b]$	Links-offenes Intervall der Zahlen x mit $a < x \leq b$
(a, b)	Offenes Intervall der Zahlen x mit $a < x < b$
$\# M$	Kardinalität der Menge M
$M \times N$	Karthesisches Produkt der Mengen M und N
M^n	n-faches karthesisches Produkt der Menge M
$M \cap N$	Schnitt der Mengen M und N
$M \cup N$	Vereinigung der Mengen M und N
$M \uplus N$	Disjunkte Vereinigung der Mengen M und N
$M \setminus N$	Differenzmenge von M bezüglich N
$\mathrm{Pow}(M)$	Potenzmenge von M
$\mathbb{B}$	Binäres Symbolalphabet $\mathbb{B} = \{0, 1\}$
Σ	Allgemeines Symbolalphabet $\Sigma = \{a_1, a_2, \ldots, a_m\}$
Σ^*	Menge aller endlichen Wörter über dem Alphabet Σ
Σ^n	Menge aller Wörter der Länge n über dem Alphabet Σ
$w_1 w_2 \ldots w_n$	Konkatenation der Symbole oder Teilworte $w_1, w_2, \ldots, w_n$
$\|w\|$	Länge des Wortes w
$\mathrm{bin}(x)$	Binärdarstellung der Zahl x
$\log(x)$	Binärer Logarithmus von x zur Basis 2
$\ln(x)$	Natürlicher Logarithmus von x zur Basis $e \approx 2.71828\ldots$
$\lceil x \rceil$	Kleinste ganze Zahl n mit $x \leq n$
$\lfloor x \rfloor$	Größte ganze Zahl n mit $x \geq n$
$n!$	Fakultät von n
$\binom{n}{k}$	Binomialkoeffizient von n über k
$g \in \mathcal{O}(f)$	g wächst asymptotisch schwächer oder gleich stark wie f
$g \in o(f)$	g wächst asymptotisch strikt schwächer als f

1 Einleitung

„A common mistake that people make when trying to design something completely foolproof is to underestimate the ingenuity of complete fools.“

— Douglas Adams

Stellt die Größe einer Datenmenge ein Problem dar, so ist Datenkompression eine nahe liegende Lösung. Der Ansatz eines verlustfreien Verfahrens ist dabei stets der Gleiche: Gesucht wird eine Möglichkeit, dieselbe Information in so wenig Informationsträgern — in aller Regel Bits — wie möglich auszudrücken. Ausgehend von SHANNONS wegweisenden Theorien der späten 40er Jahre gibt es heute bereits eine ganze Reihe wohl bekannter Algorithmen. Texte in deutscher Sprache können heutzutage zur Speicherung problemlos auf etwa die Hälfte ihrer ursprünglichen Größe reduziert werden.

Durch die Entwicklung der letzten Jahrzehnte wurden Speichermodule immer preiswerter und deren Kapazitäten stiegen geradezu expotentiell an. Die Bedeutung der Kompressionstechniken für die Datenspeicherung scheint somit gegenüber deren Notwendigkeit etwa bei der Datenübertragung zurückzufallen. Doch das genaue Gegenteil ist der Fall. Auf Grund der stark zunehmenden Verbreitung von intelligenten Speicher- oder Mikroprozessorkarten, so genannter *Smartcards*, werden hier völlig neue Problem- und Anwendungsbereiche eröffnet. Entwickler dieser Systeme sind mit hochgradig platzbeschränkten Szenarien konfrontiert. Verfügt ein herkömmlicher Heim-PC heutzutage spielend über RAM-Kapazitäten von mehr als 512 Megabyte, so müssen Smartcards noch mit wenigen hundert Kilobyte auskommen.

Es liegt daher die Idee nahe, den auf derart eingeschränkten Mikrochips benötigten Quellcode — seien es die Software des Betriebssystems oder Anwenderprogramme — in komprimierter Form im Hauptspeicher zu halten. Bei einer möglichen Rate von 50 Prozent käme dies de facto einer Verdopplung der Speicherkapazität gleich. Da die zentrale Prozes-

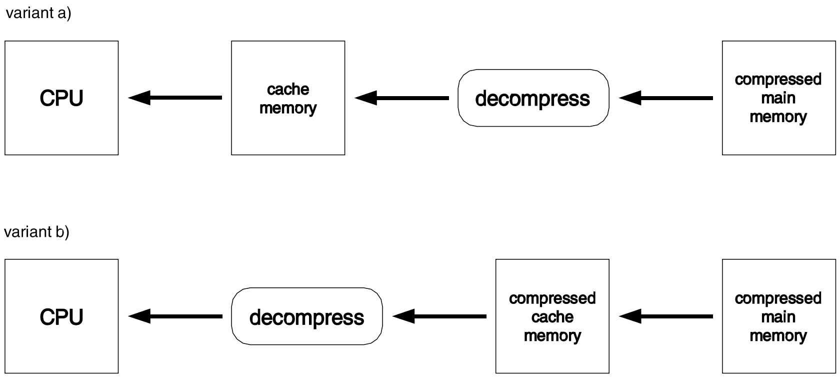


Abbildung 1.1: Varianten der Online Codekompression

soreinheit (CPU) aber selbstverständlich das unkomprimierte Programm zur Ausführung benötigt, muss zwischen CPU und Hauptspeicher ein spezieller Schaltkreis zur Dekompression existieren. Im Prinzip kann hier einer von zwei Ansätzen angestrebt werden. Zum einen können die komprimierten Daten im Speicher beim Einlesen in den schnellen, der CPU vorgelagerten Cache dekodiert werden. Diese Variante wäre vergleichsweise einfach zu realisieren. Des Weiteren könnte die Dekompression auch erst zwischen CPU und Cache erfolgen. Dies stellt wesentlich höhere Leistungsansprüche an die Dekodiereinheit, würde aber durch die zusätzliche kapazitive Vergrößerung des Cache und der damit einhergehenden Reduzierung der Cache-Lesefehlschläge eine weitere Beschleunigung der Rechnungen bewirken. Die Ansätze sind in Abbildung 1.1 schematisiert. Beiden ist gemein, dass der benötigte Dekodierschaltkreis strengen Auflagen an Größe, Latenz sowie gerade auf Smartcards an Fläche und Energiebedarf genügen muss.

Die in dieser Arbeit vorliegenden Untersuchungen sind aus den Forschungsergebnissen des Projektes „Online Codekompression für Smartcards" der Arbeitsgruppe von Prof. Dr. Blömer an der Universität Paderborn in Zusammenarbeit mit der Infineon Technologies AG München motiviert.[1] Hierbei soll die erste Variante des oben skizzierten Konzeptes auf den Chipsätzen der Infineon SL88-Familie realisiert werden. Wir wollen uns jedoch insgesamt mehr mit den mathematischen, algorithmischen und informationstheoretischen Eigenschaften des Dekodierschaltkreises und weniger mit den notwendigen elektrotechnischen und implementierungsspezifischen Details beschäftigen.

Die fast schon kanonische Wahl eines statistischen Verfahrens zur verlustfreien Datenkompression ist die Huffman-Kodierung. Diese zeichnet sich durch einen einfachen Algorithmus, eine effiziente Laufzeit und eine asymptotisch optimale Kompressionsrate aus. Allerdings weist die Huffman-Kodierung auch einige Schwachstellen auf, welche vor allem eine Umsetzung bei speicherplatzbeschränkten Szenarien in ein zumindest kritisches Licht rücken:

- Die Huffman-Kodierung ist lediglich für Blockkodierungen, welche die Codewörter der einzelnen Blöcke mit gegebener Länge einfach hintereinander hängt, beweisbar optimal. Jedes Codewort, und sei es noch so wahrscheinlich, erhält bei dieser Art

[1] Siehe auch `http://www.uni-paderborn.de/cs/ag-bloemer/forschung/kompression/` .

der Kodierung stets eine minimale Länge von einem Bit. Dies führt zu Verlusten bei stark unbalancierten Quellverteilungen.

- Zur Kompression ist die Speicherung des Huffman-Baumes oder zumindest eines Codewort-Verzeichnisses notwendig, welche sich über Alphabeten mit großer Symbolanzahl oder bei großen Blöcken als äusserst speicherineffizient erweisen. Deren Größe wächst expotentiell in der Blocklänge. Das Verfahren weist also auch eine schlechte Skalierbarkeit auf.

- Änderungen des Huffman-Baumes sind algorithmisch recht schwierig. Das bedeutet, dass eine adaptive Kodierung nur durch aufwendige Operationen auf dem Huffman-Baum zu erreichen ist.

Wir wollen uns aus diesem Grund einem anderen statistischen Ansatz zuwenden, welcher in den obigen Punkten der Huffman-Kodierung überlegen ist.

Arithmetische Kodierung ist ein schon seit den 70er Jahren bekanntes, asymptotisch optimales Kompressionsverfahren, das seit einiger Zeit durch die effiziente Implementierung von Witten, Neal und Cleary eine größere Aufmerksamkeit in der Forschungsgemeinschaft erhalten hat (vgl. [WNC87]). Es zeichnet sich neben der Speichereffizienz vor allem durch seine große Flexibilität und die strikte Trennung zwischen Modellbildung und Kompressionsalgorithmus aus. Der große Nachteil der arithmetischen Kodierung sind jedoch die komplexen arithmetischen Operationen, die zur Kodierung benötigt werden. Um diesbezüglich Abhilfe zu schaffen, haben Howard und Vitter in [HV92] einen tabellenbasierten Ansatz entwickelt, welcher die Arithmetik vollständig ersetzt. Dieser wurde allerdings nur für Kodierungen über dem binären Alphabet $\mathbb{B}$ formuliert.

Lekatsas und Wolf haben bereits ein Konzept für einen tabellenbasierten arithmetischen Dekodierschaltkreis zur Codekompression vorgeschlagen (vgl. [LW99]). Allerdings arbeitet auch dieser so genannte SAMC-Kodierer lediglich auf binären Alphabeten. Die Aufgabe dieser Diplomarbeit ist es, den tabellenbasierten Ansatz systematisch für allgemeine Alphabete zu untersuchen. Dabei wollen wir insbesondere auf die Hardwaretauglichkeit der tabellenbasierten arithmetischen Kodierung über allgemeinen Alphabeten eingehen. Die Arbeit gliedert sich in die folgenden Kapitel.

Kapitel 2 — In diesem Kapitel geben wir eine Zusammenfassung der grundlegenden stochastischen und informationstheoretischen Definitionen und Ergebnisse, die zum Verständnis der weiteren Abschnitte benötigt werden.

Kapitel 3 — Im dritten Kapitel beschreiben wir das Grundprinzip der arithmetischen Kodierung. Wir stellen darüber hinaus die Implementierung von Witten, Neal und Cleary vor, welche die arithmetische Kodierung mit endlicher Präzision und ganzzahliger Arithmetik realisiert. Anschließend präsentieren wir den tabellenbasierten arithmetischen Kodierer nach Howard und Vitter. Hier stellen wir dar, wie bei binären Alphabeten mit Hilfe von Tabellenzugriffen kostspielige arithmetische Operationen vermieden werden können.

Kapitel 4 — In dem darauf folgenden Kapitel präsentieren wir einen Ansatz, welcher einen binären, tabellenbasierten Kodierer auf beliebige Alphabete verallgemeinert. Wir konkretisieren dazu die Vorschläge von Howard und Vitter bezüglich einer Datenstruktur für komprimierte Verteilungsbäume, beweisen deren Verlustfreiheit gegenüber der Kompressionsrate und analysieren sie auf ihre Einsatztauglichkeit.

Kapitel 5 — Hier präsentieren wir ein Konzept zur Implementierung eines tabellenbasierten Dekodierschaltkreises in Hardware, welches den hochgradig iterativen Charakter der Kodierung nach Howard und Vitter vermeidet. Wir zeigen empirisch, dass sich unter realistischen Einsatzkriterien somit eine arithmetische Dekodierung zur Codekompression auf Smartcards umsetzen lässt und stellen darüber hinaus noch einige interessante Beobachtungen zur Approximation durch so genannte dyadische Verteilungen an.

Dabei sind Kapitel 2 und 3 auf einander aufbauend formuliert. Kapitel 4 und 5 berufen sich beide auf Kapitel 3, können jedoch unabhängig von einander betrachtet werden.

2 Grundlagen

„Die ganzen Zahlen hat der liebe Gott geschaffen,
alles andere ist Menschenwerk.“
— Leopold Kronecker

Die mathematischen Grundlagen dieser Arbeit liegen vor allem im Bereich der Stochastik sowie einiger teilweise schon seit SHANNON (vgl. [Sha48]) wohl bekannten kodierungs- und informationstheoretischen Erkenntnisse. Wir wollen in diesem Kapitel einen kurzen Überblick geben. Dabei wird weniger Wert auf eine vollständige und allgemeine Einführung der Thematik als auf ein bekannt machen mit der verwendeten Terminologie für die zu betrachtenden Szenarien gelegt.

2.1 Wahrscheinlichkeitstheorie

Wir gehen davon aus, dass der Leser mit dem Konzept von diskreten Ereignisräumen, Wahrscheinlichkeiten und Zufallsvariablen vertraut ist. Eine kompakte Übersicht liefert unter anderem [BSMM00]. Im Folgenden werden dennoch die Sprechweisen und Notationen eingeführt, welche innerhalb der Arbeit verwendet werden.

Definition 2.1 a) *Unter einer* (diskreten) Wahrscheinlichkeitsverteilung *zu m Ereignissen verstehen wir einen geordneten Vektor $p = (p_1, p_2, \dots, p_m) \in \mathbb{R}^m$ wobei stets $p_i \geq 0$ und $\sum_{i=1}^{m} p_i = 1$ gilt. Dabei beschreibt p_i die Wahrscheinlichkeit des i-ten Ereignisses.*

b) *Wir sagen, eine* Zufallsvariable X *verhält sich wie p, kurz $X \sim p$, wenn stets* $\mathrm{prob}(X$ *beschreibt das i-te Ereignis a_i*$) := \mathrm{prob}(X = a_i) = p_i$ *gilt.*

c) *Wir bezeichnen Zufallsvariablen X und Y als* stochastisch unabhängig, *wenn für Ereignisse a und b stets* $\mathrm{prob}(X = a \wedge Y = b) = \mathrm{prob}(X = a) \cdot \mathrm{prob}(Y = b)$ *gilt.*

d) Sei für alle i mit $w_i \in \mathbb{R}$ der Wert bezeichnet, welchen wir mit dem i-ten Ereignis assoziieren. Dann definiert $\mathrm{E}(X) := \sum_{i=1}^{m} p_i w_i$ *den Erwartungswert von $X \sim p$.*

In den von uns betrachteten Szenarien entspricht die Wahrscheinlichkeit stets der relativen Häufigkeit eines Ereignisses im Bezug zur Gesamthäufigkeit aller Ereignisse. In der Kodierungstheorie gehen diese zumeist aus Häufigkeitsanalysen der zu kodierenden Daten hervor. Wir wollen daher auch den folgenden Begriff verwenden.

Definition 2.2 *a) Unter einer* Häufigkeitsverteilung *zu m Ereignissen verstehen wir einen geordneten Vektor $f = (f_1, f_2, \ldots, f_m) \in \mathbb{N}_0^m$. Es entspricht dabei f_i der Häufigkeit des i-ten Ereignisses.*

b) Bezeichne $F_m = \sum_{i=1}^{m} f_i$. Dann beschreibt $p = (p_1, p_2, \ldots, p_m)$, wobei $p_i := \frac{f_i}{F_m}$, die durch f induzierte Wahrscheinlichkeitsverteilung.

Wir werden in dieser Arbeit die Begriffe Wahrscheinlichkeitsverteilung und Häufigkeitsverteilung weitgehend synonym verwenden. Wichtige Definitionen werden stets für beide Verteilungsformen angegeben, wobei sich die Begriffe zu Häufigkeiten in der Regel durch die Eigenschaften der induzierten Wahrscheinlichkeitsverteilung ergeben.

Gerade bei der arithmetischen Kodierung, welche wir in Kapitel 3 vorstellen wollen, wird von Zeit zu Zeit die *kumulative Verteilung* eine Rolle spielen.

Definition 2.3 *a) Bezeichnet $p = (p_1, p_2, \ldots, p_m)$ eine Wahrscheinlichkeitsverteilung, so gibt $P = (P_0, P_1, P_2, \ldots, P_m)$ mit $P_i := \sum_{j=1}^{i} p_j$ die dazugehörige* kumulative Wahrscheinlichkeitsverteilung *an.*

b) Analog beschreibt zu der Häufigkeitsverteilung $f = (f_1, f_2, \ldots, f_m)$ das $(m+1)$-Tupel $F = (F_0, F_1, F_2, \ldots, F_m)$ mit $F_i := \sum_{j=1}^{i} f_j$ die kumulative Häufigkeitsverteilung.

2.2 Kodierungstheorie

Wir wollen zunächst einführen, was wir allgemein unter einer Kodierung — wozu dann selbstverständlich auch die verlustfreie Datenkompression gehört — verstehen. Unsere Sprechweisen orientieren sich dabei an der Einführung dieser Thematik in [Mac03].

Eine *Quelle* bezeichnet dabei einen Strom von Symbolen über einem endlichen *Alphabet* $\Sigma = \{a_1, a_2, \ldots, a_m\}$. Zu jeder Stelle j im Quellstrom gehört eine Verteilung p_j, welche die Auftrittswahrscheinlichkeit der Symbole aus Σ beschreibt. Dieses Verhalten lässt sich an Hand von Zufallsvariablen modellieren. Je Position im Strom $X_1, X_2, X_3, \ldots$ entsprechen dabei die Ereignisse dem Auftreten des jeweiligen Alphabetsymbols an dieser Stelle mit $X_j \sim p_j$.

Oft werden die Symbole einer Quelle zu gleich langen Blöcken endlicher Länge n zusammengefasst. Diese Block-Kodierung fasst dann $\Gamma := \Sigma^n$ als das zu Grunde liegende Alphabet auf. Man bezeichnet diese Klartextwörter als *Nachricht*. Eine binäre Kodierung ordnet nun jeder Nachricht eine Zeichenkette über dem binären Alphabet $\mathbb{B}$ zu. Wir bezeichnen $C(w)$ als das *Codewort* von w unter dem Code C.

Definition 2.4 *a) Eine* (binäre) Kodierung *ist eine nicht-singuläre Abbildung* $C : \Gamma \to \mathbb{B}^*$ *über der endlichen Nachrichtenmenge* Γ*; d.h. für Nachrichten* $v, w \in \Gamma$ *mit* $v \neq w$ *gilt stets* $C(v) \neq C(w)$.

b) Eine Erweiterung *einer Kodierung* C *ist die Erweiterung ihres Definitionsbereiches auf* $C^* : \Gamma^* \to \mathbb{B}^*$ *mittels* $C^*(w_1 w_2 \dots w_k) := C(w_1)C(w_2)\dots C(w_k)$.

Wenn wir nun über die Eindeutigkeit der Dekodierung eines Codes sprechen, so genügt es in aller Regel nicht, nur einzelne Nachrichten zu betrachten. Viel mehr müssen wir von einer Konkatenation mehrerer Codewörter ausgehen, also einer Erweiterung von C^{-1} auf ganz $C^*(\Gamma^*)$. Zur Beschreibung werden daher die zwei folgenden, äußerst wichtigen Eigenschaften für Kodierungen verwendet.

Definition 2.5 *a) Eine Kodierung* C *heißt* eindeutig entschlüsselbar*, falls zu jedem* $x \in \mathbb{B}^*$ *höchstens ein Wort* $w \in \Gamma^*$ *mit* $C^*(w) = x$ *existiert.*

b) Eine Kodierung C *heißt* präfixfrei *(oder* selbst-punktierend*), falls keine* $v, w \in \Gamma$ *mit* $v \neq w$ *existieren, so dass* $C(v)$ *ein Präfix von* $C(w)$ *ist.*

Mit anderen Worten bedeutet dies, dass ein Code präfixfrei ist, wenn man das Ende eines Codewortes eindeutig erkennen kann. Es ist damit leicht einzusehen, dass ein präfixfreier Code auch eindeutig entschlüsselbar sein muss. Die Umkehrung gilt im Allgemeinen nicht.

Von besonderer Bedeutung ist der Satz von KRAFT (vgl. [Kra49]), welcher die Existenz präfixfreier Kodierungen anhand der einzelnen Codewort-Längen charakterisiert.

Satz 2.6 *(Satz von Kraft) Seien die Codewort-Längen* $l_1, l_2, \dots, l_k$ *zu den Nachrichten aus* $\Gamma = \{w_1, w_2, \dots, w_k\}$ *gegeben. Es existiert genau dann eine präfixfreie Kodierung* C *mit* $|C(w_i)| = l_i$*, falls gilt:*

$$\sum_{i=1}^{k} 2^{-l_i} \leq 1 \ .$$

Wir sagen im obigen Fall auch, dass die Codewort-Längen $l_1, l_2, \dots, l_k$ die *Kraft-Ungleichung* erfüllen. Interessant ist dabei außerdem die direkte Folgerung über eine triviale, präfixfreie Kodierung.

Folgerung 2.7 *Zu jeder endlichen Nachrichtenmenge* Γ *mit* $\#\,\Gamma = k$ *existiert eine präfixfreie Kodierung* $C : \Gamma \to \mathbb{B}^*$*, so dass für alle* $w \in \Gamma$ *stets* $|C(w)| = \lceil \log k \rceil$ *gilt.*

Wir können uns diese Kodierung derart vorstellen, dass alle Elemente aus Γ von 0 bis $k-1$ durchnummeriert sind. Als Kodierung des i-ten Elements nehmen wir dann stets die Binärdarstellungen von i, welche mit führenden Nullen alle auf die gleiche Länge gebracht werden. Die maximal benötigte Länge ist dabei die zur Darstellung des größten Wertes, also von $k-1$, und somit $\lceil \log k \rceil$.

Bei der Datenkompression interessieren wir uns selbstverständlich für Kodierungen, welche in der Regel besonders kurze Codewörter erzeugen. Ziehen wir nun die Wahrscheinlichkeit des Auftretens einer Nachricht mit in Betracht, so führt dies zu der erwarteten Codewort-Länge eines Codes.

Definition 2.8 *a) Sei $C : \Gamma \to \mathbb{B}^*$ eine Kodierung mit Verteilung p der Nachrichten $\Gamma = \{w_1, w_2, \ldots, w_k\}$. Dann beschreibt*

$$\mathrm{ECL}(C, p) := \sum_{i=1}^{k} p_i \, |C(w_i)|$$

die erwartete Codewort-Länge *von Nachrichten aus Γ. Dabei bezeichnet $|\cdot|$ die Bitlänge eines Wortes aus $\mathbb{B}^*$.*

b) Eine präfixfreie Kodierung heißt kompakt*, falls deren erwartete Codewort-Länge bezüglich p minimal ist.*

McMillan formuliert in [McM56] das interessante Ergebnis, dass eindeutig entschlüsselbare sowie präfixfrei Codes bezüglich der Codewort-Längen die gleichen Optima erwarten lassen, also denselben Kompaktheitsbegriff definieren. Dies rechtfertigt die Konzentration auf präfixfreie Kodierungen.

2.3 Informationstheorie

Die Informationstheorie hat sich damit beschäftigt, den Informationsgehalt eines Ereignisses durch die relative Häufigkeit seines Auftretens zu beschreiben. Dabei bildet die *Entropie* einen zentralen Begriff als das Maß für die Zufälligkeit einer Wahrscheinlichkeitsverteilung: Schiefe Verteilungen weisen eine höhere Entropie auf als annähernd gleichverteilte.

Definition 2.9 *Sei eine Verteilung $p = (p_1, p_2, \ldots, p_m)$ zu m Ereignissen gegeben.*

a) Zu dem i-ten Ereignis definiert $\log\left(\frac{1}{p_i}\right)$ dessen (binäre) Selbstinformation *oder auch das* Informationsgewicht.

b) Die (binäre) Entropie *einer Verteilung bezeichnet die erwartete Selbstinformation*

$$\mathrm{H}(p) := \sum_{i=1}^{m} p_i \log\left(\frac{1}{p_i}\right) .$$

Wir haben hier die Definitionen bezüglich des Logarithmus zur Basis 2 formuliert. Selbstverständlich hätten die Begriffe bezüglich einer anderen Basis die gleichen Eigenschaften; nur die numerischen Werte würden sich unterscheiden. Der binäre Logarithmus stellt jedoch die kanonische Wahl dar, wenn wir binäre Kodierungen betrachten wollen. Offensichtlich gilt nach den Logarithmusgesetzen der folgende Zusammenhang.

Beobachtung 2.10 *Sei H_b die b-äre* Entropie*, also wie oben mit Logarithmus zur Basis b definiert. Dann gilt für alle Verteilungen p:*

$$\mathrm{H}_b(p) = \log(b)^{-1} \cdot \mathrm{H}(p) .$$

Die relative Entropie gibt uns ein Maß an die Hand, die „Ähnlichkeit“ zweier Verteilungen p und q abzuschätzen.

Definition 2.11 *Zu einer weiteren Verteilung $q = (q_1, q_2, \ldots, q_m)$ beschreibt*

$$\mathrm{D}\,(p\|q) := \sum_{i=1}^{m} p_i \log\big(\frac{1}{q_i}\big) - \mathrm{H}(p) = \sum_{i=1}^{m} p_i \log\big(\frac{p_i}{q_i}\big)$$

die relative Entropie *(oder auch* Kullback-Leibler-Distanz*) von p bezüglich q.*

Allerdings handelt es sich dabei um keine Distanz im mathematischen Sinne, wie die folgende Beobachtung zeigt.

Beobachtung 2.12 *Es gilt:*

a) $\mathrm{D}\,(p\|q) \geq 0$ *für alle Verteilungen p und q.*

b) $\mathrm{D}\,(p\|q) = 0$ *genau dann, wenn $p = q$.*

c) Es existieren jedoch Verteilungen p und q mit $\mathrm{D}\,(p\|q) \neq \mathrm{D}\,(q\|p)$.

Teil a) der Beobachtung 2.12 ist auch als *Gibbs-Ungleichung* bekannt. Die Beweise können [CT91] oder [Mac03] entnommen werden.

Das besondere Interesse an der Entropie im Bereich der Datenkompression rührt von der Erkenntnis her, dass wir durch sie eine untere Schranke für die erwartete Komprimierbarkeit einer Quelle, deren Nachrichten $X \sim p$ verteilt sind, erhalten.

Satz 2.13 *(Quellkodierungs-Theorem) Die erwartete Codewort-Länge einer präfixfreien Kodierung C mit der Verteilung p ist stets größer oder gleich der Entropie von p, d.h.:*

$$\mathrm{H}(p) \leq \mathrm{ECL}(C, p) \ .$$

Auch der Beweis dieses wichtigen Ergebnisses kann in [CT91] oder [Mac03] gefunden werden.

3 Tabellenbasierte arithmetische Kodierung für binäre Alphabete

„There are only 10 kinds of people, those
who understand binary numbers
and those who don't.“

— Internet-Sprichwort

Die Wurzeln der arithmetischen Kodierung liegen in den Arbeiten von Shannon (vgl. [Sha48]) und Fano (vgl. [Fan49]). Darin wird — unabhängig von einander — das Verfahren beschrieben, welches wir heute als die so genannte Shannon-Fano-Kodierung kennen. Arithmetische Kodierung ist in der Tat weniger eine eigene Kodierung als eine Methode zur Berechnung eines Shannon-Fano-Codes.

Entwickelt wurde die arithmetische Kodierung in einem nicht publizierten Text von Elias (vgl. [Abr63] oder [Jel68]) sowie in den Arbeiten von Pasco und Rissanen (siehe [Pas76] und [Ris76]). Alle Ansätze teilen im Kern die Idee, zur Kodierung die kumulative Verteilung der Nachrichten bezüglich einer lexikographischen Sortierung heranzuziehen und diese rekursiv auszurechnen. Die namensgebende Arithmetik, welche dazu notwendig ist, stand mit ihren rechenzeitintensiven Operationen einer Realisierung jedoch meistens im Wege.

Stets im Schatten der kompakten und dennoch simplen Huffman-Kodierung (vgl. [Huf52]) stehend, wurde die arithmetische Kodierung in den vergangenen Jahren populär, als Witten, Neal und Cleary in [WNC87] eine praxistaugliche Implementierung vorstellten. Auf diesem Verfahren basierend entwickelten Howard und Vitter einen Ansatz für binäre Alphabete, welcher die gesamte Arithmetik durch Tabellenauswertungen ersetzt (vgl. [HV92]), die so genannte *tabellenbasierte arithmetische Kodierung*.

Im ersten Abschnitt wollen wir einen einführenden Überblick über das mathematische Konzept der arithmetischen Kodierung geben. Anschließend werden wir in Abschnitt zwei

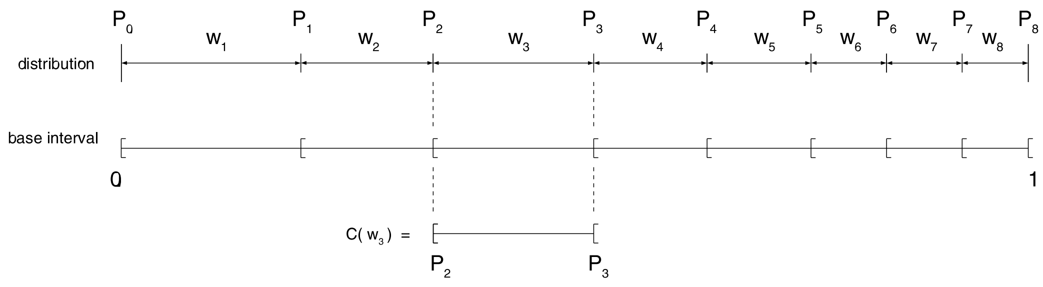


Abbildung 3.1: Prinzip der arithmetischen Kodierung bei $\# \Gamma = 8$

die Auswirkungen der Modellbildung auf das Verfahren diskutieren. Der dritte Abschnitt stellt schließlich den Ansatz und die Vorteile der Implementierung nach Witten, Neal und Cleary vor. Abschnitt vier präsentiert dann die so genannte quasi-arithmetische Kodierung nach Howard und Vitter, welche als Grundlage für die weiteren Kapitel diesen soll. Abschließend wollen wir noch gesondert und an Hand von Beispielen auf die Konstruktion der Dekodiertabellen für binäre Alphabete eingehen.

3.1 Einführung in die arithmetische Kodierung

Bevor wir nun die arithmetische Kodierung an Hand unserer Begriffe aus Abschnitt 2.2 beschreiben, wollen wir zunächst einen etwas intuitiveren Blick auf das Kodierverfahren richten, um anschließend zu einer mathematisch exakten Beschreibung zurück zu kehren. Wir werden dazu nur eine vereinfachte Form der arithmetischen Kodierung auf endlichen Nachrichtenmengen Γ betrachten. Dies deckt sich mit den Anwendugsfällen, welche wir in den folgenden Kapiteln betrachten werden.

In Missachtung von Definition 2.4 läßt sich die arithmetische Kodierung wie folgt angeben.

Definition 3.1 *Unter* arithmetischer Kodierung *verstehen wir die nicht-singuläre Abbildung $C : \Gamma \to \mathrm{Pow}\big([0,1)\big)$ über der endlichen Nachrichtenmenge $\Gamma = \{w_1, w_2, \ldots, w_k\}$ mit kumulativer Verteilung $P = (P_0, P_1, P_2, \ldots, P_k)$, bei welcher die „Codewörter“*

$$C(w_i) := [P_{i-1}, P_i)$$

als Intervalle aus dem Basisintervall $[0,1)$ definiert sind.

Abbildung 3.1 skizziert diese Kodierung. Dabei sollte klar sein, dass die definierten Code-Intervalle eine disjunkte Partition $\biguplus_{i=1}^{m} [P_{i-1}, P_i) = [0,1)$ des Basisintervalls angeben. Offensichtlich ist die Abbildung also nicht-singulär.

Wir können uns mit dieser Beschreibung sicherlich nicht zufrieden geben. An Hand der Definition lassen sich unsere Kriterien wie eindeutige Entschlüsselbarkeit und Präfixfreiheit nicht anwenden. Auch und gerade im Hinblick auf eine Hardware-Realisierung ist eine zudem reellwertige Beschreibung der Codewörter ungeeignet. Gesucht wird daher eine geeignete Beschreibung der Intervalle in $\mathbb{B}^*$.

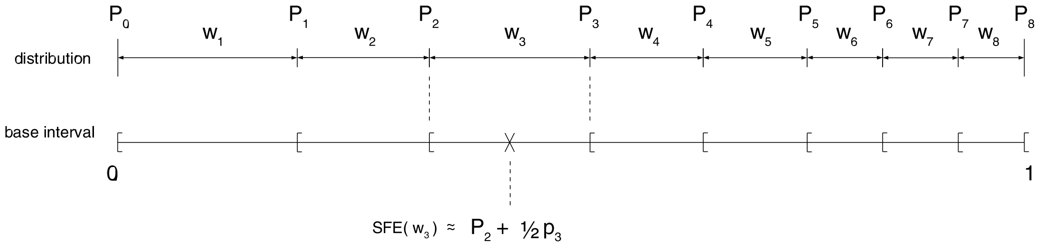


Abbildung 3.2: Prinzip der *SFE*-Kodierung bei $\#\Gamma = 8$

Die Idee ist nun, statt dem Intervall zu a_i nur einen Wert $x \in [P_{i-1}, P_i)$ anzugeben. Diese Kodierung ist damit sicherlich noch immer nicht-singulär. Zudem können wir Werte aus $[0, 1)$ als Zeichenkette von Nachkommastellen interpretieren. Betrachten wir dabei die Binärdarstellung $\mathrm{bin}(x)$, so sind alle Ziffern aus dem binären Alphabet $\mathbb{B}$.

Allerdings besitzt nicht jedes Element von $[0, 1)$ eine endliche Darstellung. Außerdem wollen wir die Stellvertreter für die Intervalle derart wählen, dass sie eine präfixfreie Kodierung angeben. Die Berücksichtigung dieser Faktoren führt zu einer ersten, konkreten Form der arithmetischen Kodierung.

Definition 3.2 *Sei die Abbildung* $\mathrm{SFE} : \Gamma \to \mathbb{B}^*$ *über Nachrichten* $\Gamma = \{w_1, w_2, \ldots, w_k\}$ *mit Verteilung* $p = (p_1, p_2, \ldots, p_k)$ *und kumulativer Verteilung* $P = (P_0, P_1, P_2, \ldots, P_k)$. *Zu* $l_i := \left\lceil \log \frac{1}{p_i} \right\rceil + 1$ *gibt*

$$\mathrm{SFE}(w_i) := \left\lfloor \mathrm{bin}\left(P_{i-1} + \frac{p_i}{2}\right)\right\rfloor_{l_i}$$

das Codewort zu w_i *an. Dabei beschreibt* $\lfloor \mathrm{bin}(\cdot) \rfloor_{l_i} \in \mathbb{B}^*$ *die ersten* l_i *Nachkommastellen der Binärdarstellung. Wir bezeichnen* SFE *als die* Shannon-Fano-Elias-Kodierung.

Die *SFE*-Kodierung wählt also als Repräsentanten des Intervalls $[P_{i-1}, P_i)$ dessen Mittelpunkt und rundet nach l_i Stellen einfach nach unten ab. Abbildung 3.2 illustriert diesen Vorgang. Wir wollen uns nun vergewissern, dass diese Kodierung unserem Anspruch an Präfixfreiheit genügt. Wir folgen dabei der Beweisführung aus der gängigen Grundlagenliteratur (vgl. [CT91], [Say00] oder [Mac03]).

Beobachtung 3.3 *a) Bezeichnet* $l = \left\lceil \log \frac{1}{p} \right\rceil + 1$, *so gilt* $2^{-l} \le \frac{p}{2}$.

b) Stimmen die Binärdarstellungen von $x, y \in \mathbb{R}$ *auf den ersten* l *Nachkommastellen überein, also* $\lfloor \mathrm{bin}(x) \rfloor_l = \lfloor \mathrm{bin}(y) \rfloor_l$, *so gilt* $|x - y| < 2^{-l}$.

Beweis: Die Behauptung a) folgt durch einfaches Umstellen aus der Definition von l:

$$l \ge \log \frac{1}{p} + 1 \;\Leftrightarrow\; l - 1 \ge \log \frac{1}{p} \;\Leftrightarrow\; 2^{l-1} \ge \frac{1}{p} \;\Leftrightarrow\; 2^{-l+1} \le p \;\Leftrightarrow\; 2^{-l} \le \frac{p}{2}\,.$$

Für die zweite Behauptung zu o.E. $x \geq y$ sei die Binärdarstellung der Differenz $\text{bin}(x-y)$ betrachtet. Da die Darstellungen von x und y auf den ersten l Stellen übereinstimmen, müssen die dazugehörigen Positionen von $\text{bin}(x-y)$ gleich 0 sein. Also gilt:

$$x - y \leq \sum_{i=l+1}^{\infty} 2^{-i} = 2^{-(l+1)} \cdot \sum_{i=l+1}^{\infty} 2^{-i+l+1} \overset{j:=i-(l+1)}{=} 2^{-(l+1)} \cdot \sum_{j=0}^{\infty} 2^{-j} < 2^{-(l+1)} \cdot 2 = 2^{-l}$$

und es folgt die Behauptung. □

Satz 3.4 *Die* SFE-*Kodierung ist präfixfrei.*

Beweis: Angenommen, es existieren zwei Nachrichten $w_i, w_j \in \Gamma$ mit $w_i \neq w_j$ und $SFE(w_i)$ ist ein Präfix von $SFE(w_j)$. Seien $M_i := P_{i-1} + \frac{p_i}{2}$ und $M_j := P_{j-1} + \frac{p_j}{2}$ die beiden Mittelpunkte der dazu gehörenden Intervalle. Dann muss $\lfloor \text{bin}(M_i) \rfloor_{l_i} = \lfloor \text{bin}(M_j) \rfloor_{l_i}$ gelten. Nach Beobachtung 3.3 folgt somit

$$|M_i - M_j| \overset{\text{b)}}{<} 2^{-l_i} \overset{\text{a)}}{\leq} \frac{p_i}{2} \; .$$

Falls nun $i < j$, so gilt auch

$$\begin{aligned} & M_i + \frac{p_i}{2} = P_{i-1} + p_i = P_i \leq P_{j-1} < M_j \\ \Rightarrow \quad & M_j - M_i - \frac{p_i}{2} > 0 \\ \Rightarrow \quad & |M_i - M_j| = M_j - M_i > \frac{p_i}{2}, \end{aligned}$$

was zum Widerspruch führt. Gilt andererseit $i > j$, so folgt

$$\begin{aligned} & M_j = P_{j-1} + p_j < P_j \leq P_{i-1} = M_i - \frac{p_i}{2} \\ \Rightarrow \quad & M_i - M_j - \frac{p_i}{2} > 0 \\ \Rightarrow \quad & |M_i - M_j| = M_i - M_j > \frac{p_i}{2} \end{aligned}$$

und damit abermals der Widerspruch. □

Damit ist insgesamt gezeigt, dass kein Codewort $SFE(w_i)$ Präfix eines weiteren Codewortes $SFE(w_j)$ ist. Da dies auch den trivialen Präfix $SFE(w_i) = SFE(w_j)$ mit einschließt, haben wir gleichzeitig auch die Nicht-Singularität formal bewiesen. Werfen wir nun einen Blick auf die zu erwartende Codewort-Länge, so sehen wir, dass die *SFE*-Kodierung lediglich einen Verlust von weniger als zwei Bit gegenüber der theoretisch maximalen Kompression erreicht.

Satz 3.5 *Für alle Wahrscheinlichkeitsverteilungen p gilt:*

$$\mathrm{H}(p) \leq \mathrm{ECL}(\mathrm{SFE}, p) \leq \mathrm{H}(p) + 2 \; .$$

Beweis: Der linke Teil der Ungleichung gilt bereits nach Satz 2.13. Da nun für die obere Schranke

$$\begin{aligned} ECL(SFE,p) &= \sum_{i=1}^{k} p_i\,|SFE(w_i)| = \sum_{i=1}^{k} p_i \left(\left\lceil \log \frac{1}{p_i} \right\rceil + 1 \right) \\ &\le \sum_{i=1}^{k} p_i \left(\log \frac{1}{p_i} + 2 \right) = \sum_{i=1}^{k} p_i \log \frac{1}{p_i} + 2 \sum_{i=1}^{k} p_i = H(p) + 2 \end{aligned}$$

gilt, folgt somit der Satz. □

Wir können also zu gegebener kumulativer Verteilung und Nachricht das entsprechende Codewort direkt bestimmen. In der Praxis wäre es jedoch eher ungünstig, die Verteilung der ganzen Nachrichtenmenge Γ zu speichern. Betrachten wir etwa $\Gamma = \Sigma^n$, und besitzt Σ genau m Elemente, so sind in Γ genau $k = m^n$ Elemente enthalten. Dem entsprechend besitzt der für Γ resultierende Verteilungsvektor bei wachsender Nachrichtenlänge auch sehr viele Einträge. Aus diesem Grund wendet man bei der arithmetische Kodierung normalerweise ein rekursives Berechnungsverfahren an.

Wir wollen dazu im Folgenden annehmen, dass wir zur Kodierung eine so genannte *erinnerungslose Quelle* vorliegen haben. Das heißt, dass die Wahrscheinlichkeit eines Alphabetsymbols w_i aus der Nachricht $w = w_1 w_2 \dots w_n \in \Sigma^n$ unabhängig vom Auftreten der anderen Symbole w_j mit $j \neq i$ ist; kurzum:

$$\mathrm{prob}(X_1 = w_1 \wedge X_2 = w_2 \wedge \cdots \wedge X_n = w_n) = \prod_{i=1}^{n} \mathrm{prob}(X_i = w_i)\ .$$

Bezeichnet p die Symbolwahrscheinlichkeit der Quelle mit $X_i \sim p$, so bezeichnen wir die resultierende Produktverteilung auf Nachrichten der Länge n mit p^n. Die Reihenfolge der Einträge des Vektors p^n entspricht dabei der durch die Reihenfolge auf p gegebenen lexikographischen Ordnung der Nachrichten aus Σ^n.

Entsprechend dieser Annahmen lässt sich die *SFE*-Kodierung somit auch rekursiv definieren. Dabei ist in jedem Rekursionsschritt lediglich die Quellverteilung p interessant.

Notation 3.6 *Bezeichne $[a,b) \subset \mathbb{R}$ mit $b > a$ ein Intervall. Dann entsprechen die Operationen $(+)$ und $(\cdot)$, für $\gamma, \lambda \in \mathbb{R}$, $\lambda > 0$ vermöge*

$$\gamma + [a,b) \;:=\; [\gamma + a, \gamma + b) \qquad \text{und} \qquad \lambda \cdot [a,b) \;:=\; [\lambda a, \lambda b)$$

mit $\gamma + \lambda \cdot [a,b) = \gamma + \big(\lambda \cdot [a,b)\big)$, der Translation *bzw. der* Skalierung *des Intervalls.*

Definition 3.7 *Sei die Abbildung* $\mathrm{SFE} : \Sigma^n \to \mathbb{B}^*$ *über dem Alphabet* $\Sigma = \{a_1, a_2, \dots, a_m\}$ *mit kumulativer Quellverteilung $P = (P_0, P_1, P_2, \dots, P_m)$ gegeben. Zur Codewort-Länge*

$$l(w_1 w_2 \dots w_n) := \left\lceil \log \left(\prod_{i=1}^{n} \mathrm{prob}(X_i = w_i)^{-1} \right) \right\rceil + 1$$

und mit Hilfe der rekursiven Abbildung $\sigma : \Sigma^n \to \Sigma^* \times \mathrm{Pow}(\mathbb{R}) \times \mathbb{N}$, *welche für* $a_i \in \Sigma$ *und für* $w' \in \Sigma^*$ *als*

$$\sigma\big(\varepsilon, [a,b), l\big) := \left\lfloor \mathrm{bin}\left(\frac{a+b}{2}\right)\right\rfloor_l \quad \textit{und}$$
$$\sigma\big(a_i w', [a,b), l\big) := \sigma\big(w', a + (b-a)\cdot[P_{i-1}, P_i), l\big)$$

definiert ist, gibt

$$\mathrm{SFE}(w) := \sigma\big(w, [0,1), l(w)\big)$$

das Codewort zu $w \in \Sigma^n$ *an. Dabei bezeichnet* $\varepsilon \in \Sigma^0$ *das leere Wort.*

Das rekursive Verfahren ist in Abbildung 3.3 skizziert. Im Prinzip wird einfach für jedes Symbol des Wortes $w_1 w_2 \ldots w_n \in \Sigma^n$, der aktuelle Teilabschnitt — beginnend mit dem Basisintervall $[0,1)$ — proportional zur Quellverteilung p partitioniert. Das Teilintervall, welches zum aktuellen Symbol w_i gehört, ist anschließend der Ausgangspunkt für die nächste Runde. Sind alle Symbole abgearbeitet, so wird der Mittelpunkt des letzten Abschnitts binär kodiert.

Es ist relativ leicht einzusehen, wenn auch technisch etwas mühsam, dass sowohl Definition 3.2 als auch 3.7 die gleiche Kodierung beschreiben. Wir müssen dazu nur induktiv aufzeigen, dass die folgenden beiden Invarianten gelten. Zum einen gilt nach k Schritten, dass die linke Intervallgrenze $a^{(k)}$ der kumulativen Wahrscheinlichkeit P^k_{i-1} der k-ten Produktverteilung entspricht. Ferner gilt für die Intervallgröße $b^{(k)} - a^{(k)}$ nach Runde k, dass diese der Wahrscheinlichkeit $\prod_{i=1}^k \mathrm{prob}(X_i = w_i)$ der ersten k Symbole von $w_1 w_2 \ldots w_n$ entspricht.

Ist dies gezeigt, so folgt

$$P^n_{i-1} + \frac{1}{2}\prod_{i=1}^n \mathrm{prob}(X_i = w_i) = a^{(n)} + \frac{1}{2}(b^{(n)} - a^{(n)}) = \frac{2a^{(n)} + b^{(n)} - a^{(n)}}{2} = \frac{a^{(n)} + b^{(n)}}{2}\ .$$

Somit wird also in beiden Verfahren durch die Verteilung $P_{i-1} + \frac{1}{2}\,\mathrm{prob}(X = w)$, bzw. den Mittelpunkt der rekursiv unterteilten Intervalle $\frac{a^{(n)}+b^{(n)}}{2}$ die gleiche Zahl bestimmt. Da die Werte l_i einer Nachricht w_i aus Definition 3.2 mit dem Wert von $l(w_i)$ übereinstimmen, wird diese Zahl auch auf die gleiche Anzahl von Stellen gerundet. Also bestimmen beide Ansätze das gleiche Codewort.

Bezüglich der Voraussetzung einer erinnerungslosen Quelle wollen wir die Kompressionsrate je Nachrichtensymbol der *SFE*-Kodierung noch einmal genauer betrachten.

Satz 3.8 *Betrachte eine* SFE-*Kodierung* $\Sigma^n \to \mathbb{B}^*$ *über einer erinnerungslosen Quelle mit Verteilung p. Dann gilt:*

$$\lim_{n\to\infty} \frac{\mathrm{ECL}(\mathrm{SFE}, p^n)}{n} = \mathrm{H}(p),$$

d.h. für Nachrichten aus Σ^n *ist die Kodierung asymptotisch optimal.*

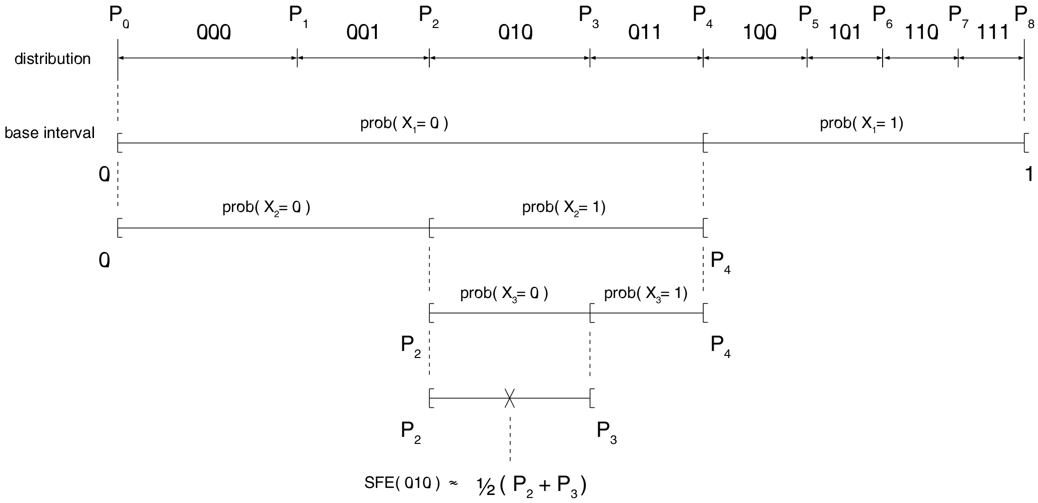


Abbildung 3.3: Prinzip der rekursiven *SFE*-Kodierung bei $\Gamma = \mathbb{B}^3$, also $\#\,\Gamma = 8$

Beweis: Betrachten wir zunächst die Entropie $H(pq)$ der Produktverteilung zu zwei stochastisch unabhängigen Verteilungen $p = (p_1, p_2, \ldots, p_m)$ und $q = (q_1, q_2, \ldots, q_k)$. Dann gilt:

$$
\begin{aligned}
H(pq) &= \sum_{i=1}^{m}\sum_{j=1}^{k} p_i q_j \log \frac{1}{p_i q_j} \\
&= \sum_{i=1}^{m}\sum_{j=1}^{k} p_i q_j \big(\log \frac{1}{p_i} + \log \frac{1}{q_j}\big) \\
&= \sum_{i=1}^{m}\sum_{j=1}^{k} \big(p_i q_j \log \frac{1}{p_i}\big) + \big(p_i q_j \log \frac{1}{q_j}\big) \\
&= \sum_{i=1}^{m}\sum_{j=1}^{k} p_i q_j \log \frac{1}{p_i} + \sum_{i=1}^{m}\sum_{j=1}^{k} p_i q_j \log \frac{1}{q_j} \\
&= \left(\sum_{i=1}^{m} p_i \log \frac{1}{p_i}\right) \cdot \underbrace{\left(\sum_{j=1}^{k} q_j\right)}_{=1} + \underbrace{\left(\sum_{i=1}^{m} p_i\right)}_{=1} \cdot \left(\sum_{j=1}^{k} q_j \log \frac{1}{q_j}\right) \\
&= \sum_{i=1}^{m} p_i \log \frac{1}{p_i} + \sum_{j=1}^{k} q_j \log \frac{1}{q_j} \\
&= H(p) + H(q)\ .
\end{aligned}
$$

Induktiv folgt somit auch $H(p^n) = n \cdot H(p)$ für unsere erinnerungslose Quelle.

Setzen wir dies nun in das Ergebnis aus Satz 3.5 ein, so folgt:

$$
\begin{aligned}
& H(p^n) \leq ECL(SFE, p^n) \leq H(p^n) + 2 \\
\Rightarrow \quad & n \cdot H(p) \leq ECL(SFE, p^n) \leq n \cdot H(p) + 2 \\
\Rightarrow \quad & H(p) \leq \frac{ECL(SFE, p^n)}{n} \leq H(p) + \frac{2}{n} \, .
\end{aligned}
$$

Demnach muss die erwartete Codewort-Länge je Symbol für $n \to \infty$ gegen $H(p)$ streben.

□

Wir sehen also, dass die arithmetische Kodierung bei großen Nachrichtenlängen gute Ergebnisse liefert. Gerade hier zeigt sich eine Überlegenheit gegenüber der Huffman-Kodierung. Während in dem oben betrachteten Modell alleine die Quellverteilung p benötigt wurde, müsste anderenfalls der gesamt Huffman-Baum zu der Verteilung p^n erzeugt und gespeichert werden Dieser hat jedoch eine expotentiell Anzahl an Blättern.

Für Anwendungsfälle, bei welchen der für die Berechnung zur Verfügung stehende Speicherplatz begrenzt ist, zeigt sich ein klarer Vorteil für die arithmetische Kodierung. Berechnungen auf Smartcards unterliegen in der Regel einer solchen Beschränkung. Auf der Negativseite stehen allerdings noch immer die aufwendigen arithmetischen Operationen des Verfahrens, weswegen wir uns in Abschnitt 3.3 und 3.4 noch für eine cleverere Implementierung interessieren werden.

3.2 Modell und Kontext

Wir haben uns zur Definition der rekursiven Berechnung einer arithmetischen Kodierung im vorangegangenen Abschnitt auf eine erinnerungslose Quelle beschränkt. Im Prinzip ist diese Einschränkung jedoch unnötig. Wichtig war dabei lediglich, dass eine eindeutige Wahrscheinlichkeitsverteilung vorliegt, welche — auf die Größe $b - a$ des aktuellen Intervalls $[a, b)$ skaliert — zur disjunkten Unterteilung herangezogen werden kann. Diese Verteilung könnte sich ohne weiteres von denen der vorausgegangenen Runden unterscheiden, sofern dies für eine bessere Kompression von Nutzen ist. Die verwendeten Verteilungen müssen später dem Dekodierer nur ebenfalls bekannt sein. Man bezeichnet das systematische Auswahlverfahren einer geeigneten Verteilung als *Modellbildung*.

Das Geheimnis eines guten Kompressionsverfahrens ist das eines guten Modells. Als Kriterium, welches über die Auswahl der Verteilung bestimmt, wird üblicherweise der Kontext des aktuellen Symbols herangezogen. Wir sprechen in diesem Fall von einem *kontextgesteuerten Modell*. Das bereits betrachtete Szenario der erinnerungslosen Quelle hingegen steht für eine *kontextunabhängige Modellierung*.

Unter dem Begriff *Kontext* versteht man dabei in der Regel eine Auswahl oder gar alle bisher gelesenen Symbole der Nachricht. Die unmittelbar folgenden Zeichen werden vom Kodierer nicht mit betrachtet, da sonst auch der Dekodierer dieses Wissen besitzen müsste, obwohl er diese Symbole noch gar nicht dekodieren konnte.

Zwei prominente Vertreter der kontextgesteuerten Modellierung sind *Markov-Modelle* und *vermischende Modelle* wie etwa PPM (*prediction by partial matching*). Markov-

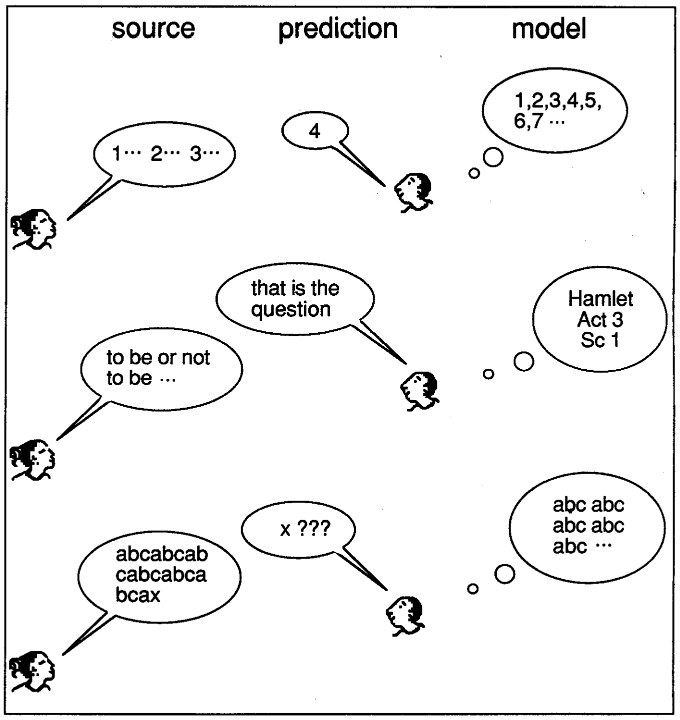


Abbildung 3.4: Intuition zur kontextgesteuerten Modellbildung (Quelle: [BCW90])

Modelle der Ordnung k wählen dabei eine Verteilung in Abhängigkeit der letzten k gelesenen Symbole. Es sollte klar sein, dass bei steigendem k zwar Verteilungen mit einer geringeren Entropie, also besserem Kompressionsverhalten zu erwarten sind, deren Anzahl aber auch expotentiell in k steigt. Vermischende Modelle hingegen erlauben innerhalb desselben Modells verschiedene Ordnungen. In der Regel wird dazu das Symbolalphabet um spezielle *Flucht-Symbole* erweitert, welche den Übergang zu einem Kontext niederer Ordnung signalisieren.

Eine weitere Ebene der Unterteilung ist nicht die Frage nach der Auswahl sondern nach der Generierung der gesuchten Verteilung. Da Verteilungen zumeist aus Häufigkeitsanalysen hervorgehen, spricht man in diesem Fall auch von *Statistiken*. Die einfachste Gruppe bilden dabei die *statischen Modelle*. Hier werden auf Erfahrungen oder Durchschnittswerten beruhende Verteilungen für jede zu kodierende Nachrichtenmenge verwendet.

Eine etwas einfühlsamere Abwandlung davon ist die *semi-adaptive Modellierung*. In einem ersten Durchgang werden dabei die benötigten Statistiken für alle Nachrichten analysiert. Anschließend werden diese als Grundlage für eine statische Modellierung verwendet. Der Vorteil besteht darin, dass die analysierten Statistiken das Verhalten der Nachrichten wesentlich genauer ausdrücken als geschätzte Werte.

Als dritten Ansatz gibt es die sehr weit verbreiteten *adaptiven Modelle*. Dabei wird in nur einem Durchgang gleichzeitig die Statistik für die bisher beobachteten Symbole generiert und — auf dieser Grundlage — das neuste Symbol kodiert. Man kann die adaptive Modellierung in sofernauch als ein probabilistisches „raten“ des nächsten Symbols an Hand der bisher beobachteten Symbole verstehen. Abbildung 3.4 karikiert diesen

Zusammenhang auf eingängige Art und Weise.

Sowohl bei statischen als auch adaptiven Modellen muss gesondert Rücksicht darauf genommen werden, dass Symbole zum ersten Mal in einem betrachteten Kontext auftreten können. Dies ist als das *Null-Häufigkeits-Problem* bekannt. Es existieren mehrere Ansätze, dieses Problem zu lösen, wir wollen hier aber nicht weiter darauf eingehen.

Ein großer Vorteil der arithmetischen Kodierung ist deren Flexibilität. Die Formulierungen des rekursiven Verfahrens gehen stets von einer vorliegenden Verteilung aus und nutzen diese einfach, um das nächste Teilintervall zu bestimmen. Wie diese Verteilung zustande kam, spielt für die Kodierung somit keine Rolle, wirkt sich aber selbstverständlich auf die zu erwartende Codewort-Länge aus. Es ist daher algorithmisch sehr einfach, jede beliebige Modellbildung in die arithmetische Kodierung zu integrieren. Eine gängige Wahl in der Praxis der Textkompression ist eine kontextgesteuerte Modellierung mit adaptiv generierten Statistiken wie etwa den PPM-Verfahren.

Wir werden in dieser Arbeit weitestgehend die Möglichkeiten und Methoden der Modellbildung und Statistikgenerierung außen vor lassen und uns stattdessen auf die Realisierung der aktuellen Kodierung konzentrieren. Dazu werden wir annehmen, dass die passende Verteilung stets zur Verfügung steht. Die bei der arithmetischen Kodierung vorherrschende starke Trennung dieser beiden wichtigen Bereiche kommt uns dabei sehr entgegen. Ein detaillierter Überblick zum Thema Modellbildung kann [BCW90] entnommen werden.

3.3 Ganzzahlige Arithmetik mit endlicher Präzision

Die arithmetische Kodierung nach Witten, Neal und Cleary gilt als die erste praxistaugliche Implementierung dieses Verfahrens. Nach der Veröffentlichung in [WNC87] entstanden auf dieser Basis eine ganze Reihe von Publikationen und Weiterentwicklungen. Einige Einschätzungen waren sogar derart euphorisch darin, die arithmetische Kodierung als state-of-the-art zu verkünden, dass sich Bookstein und Klein genötigt sahen, eine Lanze für die Huffman-Kodierung zu brechen (vgl. [BK93]). Letzten Endes scheinen jedoch von Situation zu Situation die Vorteile des einen oder anderen Verfahrens zu überwiegen.

Nichtsdestotrotz stellt die arithmetische Kodierung in der vorliegenden Umsetzung einen interessanten Ansatz für einen Code mit minimaler Redundanz dar. Dabei unterscheidet sich die Kodierung nach witten, Neal und Cleary von der bereits betrachteten *SFE*-Kodierung nur in der Bestimmung des zu kodierenden Punktes. Man könnte dies formal wie folgt definieren.

Definition 3.9 *Sei Abbildung* $\mathrm{WNC} : \Sigma^n \to \mathbb{B}^*$ *über den durch eine Anordnung auf* Σ *lexikographisch geordneten Nachrichten* $\Sigma^n = \{w_1, w_2, \ldots, w_k\}$ *mit kumulativer Verteilung* $P = (P_0, P_1, P_2, \ldots, P_k)$*. Zur Codewort-Länge* $l_i := \lceil \log (\mathrm{prob}(X = w_i)^{-1}) \rceil + 1$ *gibt*

$$\mathrm{WNC}(w_i) := \left\lfloor \mathrm{bin} \left(P_{i-1} + 2^{-l_i} \right) \right\rfloor_{l_i}$$

das Codewort zu w_i *an. Wir bezeichnen* WNC *als die* Witten-Neal-Cleary-Kodierung.

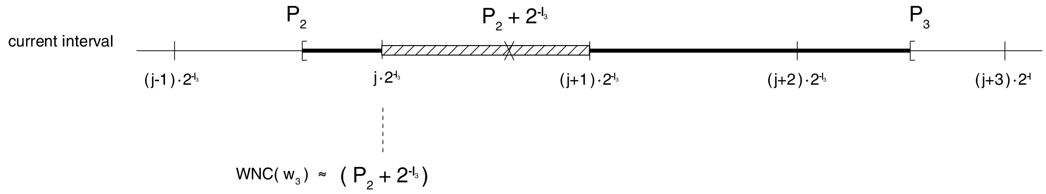


Abbildung 3.5: Prinzip der Codewort-Bestimmung bei *WNC*-Kodierung

Die einzige Abweichung liegt bei der Wahl des zu kodierenden Repräsentanten. An Stelle einer Approximation des Mittelpunktes von $[P_{i-1}, P_i)$ wird das kleinste Vielfache von 2^{-l_i} kodiert, welches echt größer als die linke Intervallgrenze P_{i-1} ist. Der additive Term von 2^{-l_i} auf den linken Rand dient im Wesentlichen dazu, zusammen mit der Rundung der Binärdarstellung auf l Stellen nach unten dieses „echt größer" zu garantieren.[1] Abbildung 3.5 illustriert die Wahl von $WNC(w_i)$. Trotz dieser Änderungen bleibt die Präfixfreiheit aber erhalten.

Beobachtung 3.10 *Bezeichne $P = (P_0, P_1, P_2, \ldots, P_k)$ die kumulative Verteilung über Σ^n und seien die l_i wie in Definition 3.9.*

a) Seien natürliche Zahlen l und $j < 2^l$ gegeben. Dann liegen alle $x \in [0,1)$, zu welchen ein $\lambda \geq l$ existiert, so dass $\left\lfloor \mathrm{bin}\left(\frac{j}{2^l}\right)\right\rfloor_l$ ein Präfix von $\lfloor \mathrm{bin}(x)\rfloor_\lambda$ ist, in $\left[\frac{j}{2^l}, \frac{j+1}{2^l}\right)$.

b) Sei j bezüglich $P_{i-1} < \frac{j}{2^{l_i}}$ minimal gewählt. Dann gilt: $\left[\frac{j}{2^{l_i}}, \frac{j+1}{2^{l_i}}\right) \subseteq [P_{i-1}, P_i)$.

Beweis: Sei zunächst zu j und l ein $x \in [0,1)$ und ein $\lambda \in \mathbb{N}$ gegeben, so dass $\left\lfloor \mathrm{bin}\left(\frac{j}{2^l}\right)\right\rfloor_l$ ein Präfix von $\lfloor \mathrm{bin}(x)\rfloor_\lambda$ ist. Dann gilt $\left\lfloor \mathrm{bin}\left(\frac{j}{2^l}\right)\right\rfloor_l = \lfloor \mathrm{bin}(x)\rfloor_l$; d.h. beide Zahlen stimmen auf den ersten l Nachkommastellen überein. Da $j \cdot 2^{-l}$ ein ganzzahliges Vielfaches von 2^{-l} ist, gilt insbesondere, dass $\mathrm{bin}\left(\frac{j}{2^l}\right)$ ab der $(l+1)$-ten Nachkommastelle ohnehin nur aus Nullen besteht. Wäre x nun kleiner als $j \cdot 2^{-l}$, so würden beide sich wenigstens an Position l unterscheiden, was der obigen Einsicht widerspräche. Gilt andererseits, dass $x \geq (j+1) \cdot 2^{-l} = j \cdot 2^{-l} + 2^{-l}$ ist, so würde abermals mindestens ein Unterschied in den ersten l Nachkommastellen auftreten. Also gilt die erste Beobachtung. In Abbildung 3.5 ist das in Frage kommende Intervall schraffiert angedeutet.

Für Teil b) betrachte zunächst die Wahl der l_i. Da $l_i \geq \log \frac{1}{p_i} + 1$, folgt durch Umstellen der Formel $2 \cdot 2^{-l_i} = 2^{-l_i+1} \leq p_i$. Nach Voraussetzung gilt bereits $P_{i-1} < j \cdot 2^{-l_i}$. Da nach Wahl von j aber $(j-1) \cdot 2^{-l_i} \leq P_{i-1}$, gilt auch

$$(j+1) \cdot 2^{-l_i} = (j-1) \cdot 2^{-l_i} + 2 \cdot 2^{-l_i} \leq P_{i-1} + 2 \cdot 2^{-l_i} \leq P_{i-1} + p_i = P_i \ ,$$

und somit folgt auch diese Behauptung. $\square$

[1] Dass ein $j \cdot 2^{-l_i} > P_{i-1}$ und nicht $j \cdot 2^{-l_i} \geq P_{i-1}$ verlangt wird, hat eigentlich keine mathematische Begründung. Viel mehr resultiert dies aus der eleganten Formulierung einer `if`-Anweisung im ANSI-C Quellcode der Originalarbeit.

Satz 3.11 *Die* WNC*-Kodierung ist präfixfrei.*

Beweis: Sei $w_i \in \Sigma^n$ und j bezüglich $P_{i-1} < \frac{j}{2^{l_i}}$ minimal gewählt. Somit gilt als Codewort $WNC(w_i) = \left\lfloor \mathrm{bin}\left(\frac{j}{2^{l_i}}\right)\right\rfloor_{l_i}$. Angenommen, $WNC(w_i)$ wäre nun der Präfix eines anderen Codewortes. Dann muss dieses nach Beobachtung 3.10 durch das Runden einer reellen Zahl $x \in \left[\frac{j}{2^{l_i}}, \frac{j+1}{2^{l_i}}\right)$, also aus $[P_{i-1}, P_i)$, entstanden sein. Da jedoch die $P_0, P_1, \ldots, P_k$ eine disjunkte Partition von $[0, 1)$ beschreiben, kann darin keine weitere Nachricht aus $\Sigma \setminus \{w_i\}$ einen binär zu kodierenden Wert x definieren. □

Darüber hinaus ändern die Anpassungen von Witten, Neal und Cleary auch nichts an der asymptotischen Optimalität der Kodierung, wie sich analog zu den Beweisen aus Satz 3.5 und 3.8 zur *SFE*-Kodierung folgern lässt.

Auch die *WNC*-Kodierung kann, ähnlich der Definition 3.7, rekursiv definiert werden. Wir wollen auf deren mathematische Formulierung jedoch hier verzichten, da es sich wirklich um dieselbe Abbildung mit leicht angepasster Endrekursion handelt. Stattdessen werden wir deren Arbeitsweise an Hand der algorithmischen Beschreibung aus [WNC87] angeben, da dies für das Verständnis des folgenden Abschnittes sehr hilfreich ist.

Wir wollen zunächst diesen Ansatz intuitiv motivieren. Eine der wichtigsten Änderungen bei der Implementierung nach Witten, Neal und Cleary ist die Abkehr von der Arithmetik in den reellen Zahlen. Da die zu Grunde liegenden Statistiken in aller Regel aus Häufigkeitsanalysen entstanden sind, lag normalerweise sowieso eine rationale Arithmetik vor. Die Gesamthäufigkeit F_m der kumulativen Häufigkeitsverteilung $F = (F_0, F_1, F_2, \ldots, F_m)$ spielt bei der durch sie induzierten Wahrscheinlichkeitsverteilung die Rolle eines Hauptnenners. Es liegt daher nahe, quasi durch Multiplikation mit diesem Hauptnenner, nicht das reelle Intervall $[0, 1)$ zu unterteilen, sondern stattdessen mit ganzzahligen Unterteilungen des ganzzahligen Intervalls $[0, F_m)$ zu arbeiten.

Es treten jedoch vor allem zwei Probleme auf, diesen Ansatz auf einem endlich präzisen Rechner umzusetzen. Zum Einen muss bei der Implementierung auf eine hardwarebedingte Obergrenze für integrale Werte Rücksicht genommen werden. Zum Anderen kann — und wird in aller Regel auch — nach mehreren rekursiven Schritten die Größe des Intervalls derart schrumpfen, dass eine ganzzahlige Unterteilung durch die aktuelle Verteilung F nicht mehr möglich ist.

Das erste Problem wird in [WNC87] dadurch gelöst, dass eine feste Obergrenze N eingeführt wird. Wir wollen N als die *Feinheit* des Kodierers bezeichnen. Alle zu betrachtenden Verteilungen werden als kumulative Häufigkeiten $F = (F_0, F_1, F_2, \ldots, F_m)$ mit $F_m < \frac{1}{4}N$ angegeben. N muss dabei derart gewählt sein, dass ein Produkt von $N \cdot F_m$ den darstellbaren, ganzzahligen Bereich nicht verlässt. Ferner soll angenommen werden, dass N ein Vielfaches von 4 ist, um ganzzahlige Werte $\frac{1}{4}N$, $\frac{1}{2}N$ und $\frac{3}{4}N$ zu ermöglichen.

Für die Lösung des zweiten Problems führen Witten, Neal und Cleary das wichtige Konzept der *Renormalisierung* ein. Beobachtet man die binären Nachkommastellen der rekursiv berechneten Intervallgrenzen aufmerksam, so stellt man fest, dass die führenden Stellen der unteren und oberen Schranken recht schnell stationär werden, das heißt, sich im weiteren Verlauf der Berechnung nicht wieder verändern. Mit diesen stationären Stellen ist somit aber auch schon ein Präfix der Darstellung jedes Wertes aus dem Intervall, also

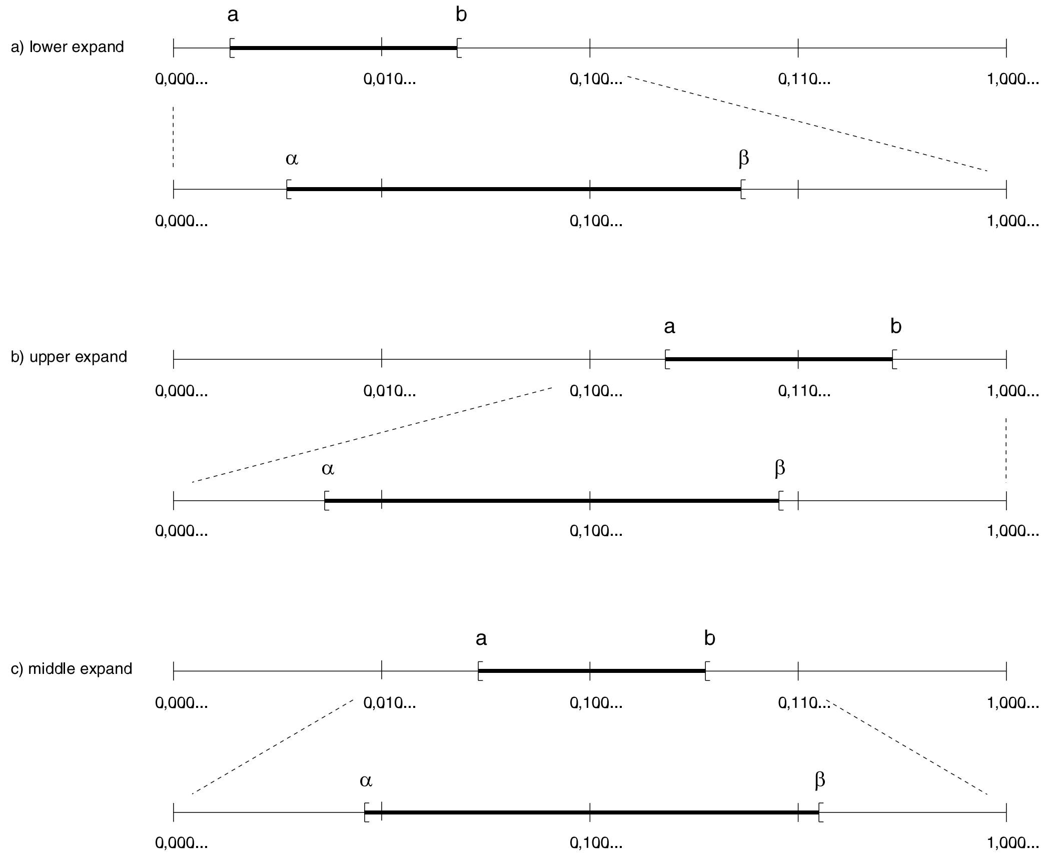


Abbildung 3.6: Renormalisierung mittels *(a)* unterem, *(b)* oberem und *(c)* mittlerem Expansionsschritt

auch des Codewortes, bestimmt. Die Idee ist es nun, die bereits bekannten Bits auszugeben und durch Skalierung um den Faktor 2 je Bit eine Mindestgröße für das aktuelle Intervall zu garantieren.

Dabei sind lediglich die drei in Abbildung 3.6 dargestellten Fälle von Interesse. Die Skizze 3.6a zeigt eine Situation, in welcher sowohl die untere als auch die obere Intervallgrenze in der linken Hälfte des Basisintervalls $[0, N)$ liegen. Dadurch ist bereits klar, dass die führende Nachkommastelle und damit das erste Bit des Codewortes eine 0 ist. Da sich diese Unterteilung rekursiv fortsetzt, genügt es nun, diese 0 an die Ausgabe weiterzureichen und die Werte der Intervallgrenzen zu verdoppeln. Wir sprechen von einem *unteren Expansionsschritt*. Da nach Voraussetzung beide Intervallgrenzen kleiner oder gleich $\frac{1}{2}N$ sind, befindet sich das resultierende Intervall somit wieder im Basisintervall.

Völlig analog verhält sich die Situation, wenn das Intervall komplett in der rechten Hälfte von $[0, N)$ liegt. In diesem Fall ist lediglich eine 1 statt einer 0 als führendes Bit bereits bestimmt. Da die Expansion der Intervallgrenzen nun in der oberen Hälfte stattfindet, muss vor der Verdopplung noch der Wert $\frac{1}{2}N$ von beiden Grenzen abgezogen werden. Wir bezeichnen dies als eine *obere Expansion* des aktuellen Intervalls. Abbildung 3.6b skizziert dieses Szenario. Insgesamt erfüllen die beiden Unterroutinen 3.12 und 3.13 die geforderte Aufgabe.

Unterprogramm 3.12 *(untere Expansion)*
Eingabe: *Aktuelles Intervall* $[a, b)$ *mit* $0 \leq a < b \leq \frac{1}{2}N$.
1. *Bestimme neues Intervall als* $[\alpha, \beta) \leftarrow 2 \cdot [a, b)$.
2. *Schreibe eine* 0 *in den Ausgabepuffer.*
3. *Liefere* $[\alpha, \beta)$ *zurück.*

Unterprogramm 3.13 *(obere Expansion)*
Eingabe: *Aktuelles Intervall* $[a, b)$ *mit* $\frac{1}{2}N \leq a < b \leq N$.
1. *Bestimme neues Intervall als* $[\alpha, \beta) \leftarrow 2 \cdot \big([a, b) - \frac{1}{2}N\big)$.
2. *Schreibe eine* 1 *in den Ausgabepuffer.*
3. *Liefere* $[\alpha, \beta)$ *zurück.*

Für den dritten Fall einer *mittleren Expansion* sieht die Lage jedoch etwas anders aus. Wenn ein Intervall um den Mittelpunkt $\frac{1}{2}N$ sattelt, so ist die erste Nachkommastelle noch nicht bestimmt. Liegen die Intervallgrenzen jedoch wie in 3.6c dargestellt zwischen $\frac{1}{4}N$ und $\frac{3}{4}N$, so lässt sich dennoch eine Aussage treffen. Offensichtlich können die ersten beiden Bits des Codewortes nicht 00 oder 11 sein, also nicht übereinstimmen. Wenn im weiteren Verlauf der Kodierung jedoch das nächste Mal ein führendes Bit über eine untere oder obere Expansion bestimmt wird, so ist damit auch die folgende Stelle sofort als dessen Inverses bestimmt.

Dies bedeutet aber, dass wir durchaus auch in dieser Situation renormalisieren können. Es muss lediglich darauf geachtet werden, dass nach der nächsten Ausgabe eines Bits dessen Inverses anschließend mit ausgegeben wird. Wir wollen dies derart vorstellen, dass dazu ein spezielles Symbol f an den Ausgabepuffer weiter gegeben werden kann. Die Folge $(f, 1)$ würde beispielsweise durch den Puffer als $(1, 0)$ an den Ausgabestrom weitergereicht; eine längere Folge $(f, f, \ldots, f, 1)$ würde zu $(0, 1, 1, \ldots, 1)$ werden. Wir bezeichnen dies als eine *follow-on Sequenz*. Unterprogramm 3.14 realisiert diesen Expansionsschritt.

Unterprogramm 3.14 *(mittlere Expansion)*
Eingabe: *Aktuelles Intervall* $[a, b)$ *mit* $\frac{1}{4}N \leq a < b \leq \frac{3}{4}N$.
1. *Bestimme neues Intervall als* $[\alpha, \beta) \leftarrow 2 \cdot \big([a, b) - \frac{1}{4}N\big)$.
2. *Schreibe ein* f *in den Ausgabepuffer.*
3. *Liefere* $[\alpha, \beta)$ *zurück.*

Wir wollen, in Anlehnung an die vorausgegangenen Beobachtungen, für die Teilintervalle von $[0, N)$ die folgende Sprechweise verwenden. Wir werden auf diese Bezeichnungen insbesondere noch in Kapitel 5 zurückgreifen.

Definition 3.15 *Ein ganzzahliges Intervall* $[a, b) \subseteq [0, N)$ *eines Kodierers mit Feinheit* N *heißt* expandierbar, *wenn eine der folgenden Bedingungen zutrifft:*

(i) Eine untere Expansion ist möglich, d.h. es gilt $0 \leq a < b \leq \frac{1}{2}N$.

(ii) Eine obere Expansion ist möglich, d.h. es gilt $\frac{1}{2}N \leq a < b \leq N$.

(iii) Eine mittlere Expansion ist möglich, d.h. es gilt $\frac{1}{4}N \leq a < b \leq \frac{3}{4}N$.

Anderenfalls nennen wir es nicht-expandierbar.

Man kann leicht die folgenden Beobachtungen über die Expandierbarkeit beziehungsweise Nicht-Expandierbarkeit eines Intervalls anstellen.

Beobachtung 3.16 *Es gilt:*

a) Jedes Intervall $[a, b)$ *mit Größe* $b - a \leq \frac{1}{4}N$ *ist expandierbar.*

b) Jedes Intervall $[a, b)$ *mit Größe* $b - a > \frac{1}{2}N$ *ist nicht-expandierbar.*

Der Algorithmus von WITTEN, NEAL und CLEARY wird nun je Symbol der Nachricht immer zuerst das aktuelle Intervall unterteilen und es anschließend so oft wie möglich expandieren. Da nach Beobachtung 3.16 ein nicht-expandierbares Intervall $[a, b)$ mindestens Größe $b - a > \frac{1}{4}N$ besitzen muss und wir eingangs den maximalen Wert einer Häufigkeitsverteilung F durch $\frac{1}{4}N$ begrenzt haben, ist somit stets eine Unterteilung von $[a, b)$ mittels F in ganzzahlige Teilintervalle gewährleistet. Schliesslich kann so jedem Symbol wenigstens ein Teilstück der Länge 1 zugeordnet werden.

Ein weiterer Vorteil der Renormalisierung besteht darin, dass bereits während der Kodierung Bit für Bit das Codewort entsteht. Das entstehende Teilwort kann somit im Sinne eines Pipelining-Verfahrens bereits weiter verarbeitet werden, ehe die komplette Kompression abgeschlossen ist. Ein solches Verhalten erweist sich bei Hardwareimplementierungen in aller Regel als sehr effizienzsteigernd. Das einzige Problem liegt im möglichen, wenn auch unwahrscheinlichen, Auftreten längerer follow-on Sequenzen, welche dieses Verfahren ausbremsen könnten.

Insgesamt kann die rekursive *WNC*-Kodierung wie folgt mit ganzzahliger Arithmetik und endlicher Präzision implementiert werden.

Algorithmus 3.17 *(rekursive* WNC*-Kodierung)*

Eingabe: *Nachricht* $w_1 w_2 \dots w_n \in \Sigma^n$

1. *Initialisiere als Startintervall* $[a, b) \leftarrow [0, N)$.
2. *Für alle Symbole* w_i *von* w_1 *bis* w_n *wiederhole 3–8.*
3.   *Bestimme gemäß Modell aktuelle Verteilung* F.
4. $[a, b) \leftarrow$ *Intervall-Unterteilung*$\big(F, [a, b)\,, w_i\big)$
5.   *Solange* $[a, b)$ *expandierbar, führe einen der drei Fälle 6–8 aus.*
6.   *Falls untere Expansion möglich,* $[a, b) \leftarrow$ *untere_Expansion*$\big([a, b)\big)$.
7.   *Falls obere Expansion möglich,* $[a, b) \leftarrow$ *obere_Expansion*$\big([a, b)\big)$.
8.   *Falls mittlere Expansion möglich,* $[a, b) \leftarrow$ *mittlere_Expansion*$\big([a, b)\big)$.
9. *Falls* $a < \frac{1}{4}N$, *schreibe* $(\mathrm{f}, 0)$, *anderenfalls* $(\mathrm{f}, 1)$ *in den Ausgabepuffer.*

Die notwendigen Beweise zur Korrektheit des Algorithmus können [WNC87] entnommen werden. Aus Gründen der Vollständigkeit fehlt noch die Beschreibung der Routine, welche das ganzzahlige Teilintervall zum Symbol a_i und der kumulativen Verteilung F bestimmt. Dazu wird von WITTEN, NEAL und CLEARY der folgenden Algorithmus zur Intervall-Unterteilung vorgeschlagen.

Unterprogramm 3.18 *(Intervall-Unterteilung)*
Eingabe: *Kumulative Verteilung* $F = (F_0, F_1, F_2, \ldots, F_m) \in \mathbb{N}^m$,
aktuelles Intervall $[a, b)$ *und* $a_i \in \Sigma$.

1. *Finde* F_{i-1} *und* F_i *in* F, *so dass* $f_i \leftarrow F_i - F_{i-1}$ *die Häufigkeit von* a_i *angibt.*
2. *Bestimme neues Intervall als* $[\alpha, \beta) \leftarrow a + \big(f_i \cdot [a, b) \operatorname{div} F_m\big)$.
3. *Gib* $[\alpha, \beta)$ *zurück.*

In obigem Algorithmus stellt die Div-Operation die ganzzahlige Division unter Wegfall des Restes dar. Diese sind jedoch leider recht ineffizient. In einer überarbeiteten Version führen Moffat, Neal und Witten daher eine alternative Berechnung ein, in welcher Multiplikationen und Divisionen durch Additionen und Shift-Operationen ersetzt werden. Die genaue Beschreibung kann [MNW98] entnommen werden.

3.4 Quasi-arithmetische Kodierung

Die Kodierung von Witten, Neal und Cleary bietet einen praxistauglichen Ansatz, arithmetische Kodierung in Hardware umzusetzen. Allerdings stellen die zur Arithmetik benötigten Rechenoperationen selbst mit den Beschleunigungen aus [MNW98] einen signifikant großen Ressourcenverbrauch dar. Howard und Vitter präsentieren aus diesem Grund einen Ansatz, die arithmetischen Operationen aus der Kodierung komplett zu entfernen und durch Tabellenauswertungen zu ersetzen.

Kern ihrer Arbeit in [HV92] ist eine alternative Sichtweise auf die sich wiederholenden Unterteilungs- und Expansionsschritte. Wenn zu einem gegebenen Startintervall dieses unterteilt und anschließend schrittweise zu einem nicht-expandierbaren Intervall vergrößert wird, so interessieren die vielen, aufwendig berechneten Zwischenschritte im Grunde überhaupt nicht. Lediglich die Tatsache, dass ausgehend von $[a, b)$ bei dem zu kodierenden Symbol a am Ende der Berechnung das Intervall $[\alpha, \beta)$ aktuell ist und währenddessen die Sequenz $b_1 b_2 \ldots b_l \in \{0, 1, f\}^*$ ausgegeben wurde, spielt eine Rolle. Dieser Sachverhalt lässt sich aber auch ohne Arithmetik an Hand der Zustandsübergänge eines Automaten modellieren. Wir werden im Folgenden die nicht-expandierbaren Teilintervalle von $[0, N)$ auch als *Zustände* eines Tabellenkodierers mit Feinheit N bezeichnen.

Howard und Vitter konzentrieren sich in ihren Ausführungen auf Kodierungen über dem binären Symbolalphabet $\mathbb{B}$. Für den Übergang auf beliebige Alphabete verweisen sie auf eine eigene Datenstruktur, welcher wir das gesamte Kapitel 4 gewidmet haben. Wir wollen zudem annehmen, dass für die Quellsymbole mit $p_0 := \operatorname{prob}(X = 0)$ und $p_1 := \operatorname{prob}(X = 1)$ stets $0 < p_0, p_1 < 1$ gilt. Im trivialen Fall $p_0 = 1$ oder $p_1 = 1$ müsste ohnehin kein Bit kodiert werden, da das einzige in Frage kommende Symbol ja schon eindeutig bestimmt ist.

Zu einer gegebenen Quellverteilung $p = (p_0, p_1)$ kann man sich die Bearbeitung eines Symbols also wie in Abbildung 3.7 skizziert vorstellen. Die jeweils zum Automaten gehörende Übergangsfunktion $\delta_p : \operatorname{Pow}\big([0, N)\big) \times \mathbb{B} \to \mathbb{B} \times \{0, 1, f\}^*$ kann durch eine Tabelle angegeben und evaluiert werden. Die δ_p enthalten somit alle möglichen, schon im Vorhinein durchgerechneten arithmetischen Operationen der Kodierung.

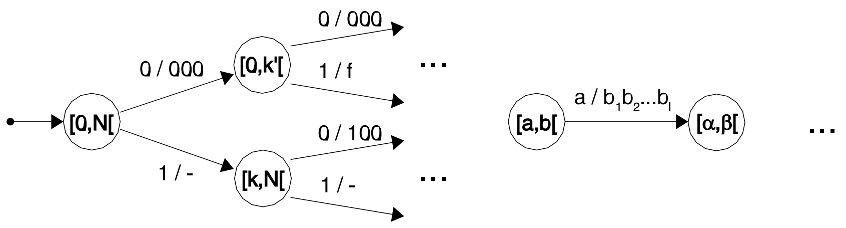


Abbildung 3.7: Arithmetische Kodierung als endlicher Automat

Dabei wird jedoch die Verteilung p lediglich zur ganzzahligen Unterteilung des Ausgangsintervalls genutzt. Je Zustand interessieren also eigentlich nicht wirklich deren exakte Werte, sondern nur, welches Teilintervall eine Verteilung liefert. Aus diesem Grund lassen sich die δ_p zu einer großen Kodiertabelle zusammenfassen, in welcher Ausgabe und neuer Zustand in Abhängigkeit von aktuellem Zustand, Verteilung und Symbol ausgewertet werden. Abbildung 3.8 zeigt einen Auszug aus einer solchen Tabelle. Die diskrete Zuordnung $\operatorname{prob}(X = 0) \approx \frac{l}{k}$ der kontinuierlich vielen, möglichen Wahrscheinlichkeiten findet dabei mit Hilfe gewisser Schwellenwerte statt. Die exakte Bestimmung dieser Werte ist in [HV92] beschrieben; uns soll an dieser Stelle genügen, dass bei einem Intervall $[a, b)$ der Größe $k := b - a$ stets ein „nahes" Vielfaches von $\frac{1}{k}$ gewählt wird.

Die algorithmische Geschwindigkeit einer Tabellenauswertung ist im Wesentlichen davon abhängig, wie lange es dauert, den gesuchten Eintrag der Tabelle zu finden. Wir wollen daher die Anzahl der Tabellenzeilen als ein Maß für die Komplexität ansehen. Diese ergibt sich ihrerseits aus der Anzahl der möglichen Zustände sowie der Anzahl der zu betrachtenden Unterteilungen je Zustand.[2] Dabei hilft das folgende Lemma. Teil a) entstammt [HV92]. Auf einen Beweis wurde dort jedoch verzichtet.

Lemma 3.19 *Sei ein binärer Tabellenkodierer mit Feinheit N gegeben.*

a) Dann existieren genau $\frac{3}{16}N^2$ nicht-expandierbare Zustände.

b) Sei $[a, b)$ ein nicht-expandierbares Intervall der Größe $k := b - a$. Dann existieren genau $k - 1$ (nicht-triviale) ganzzahlige Unterteilungen $[\alpha, \beta) \subset [a, b)$.

Beweis: Bezeichne zunächst $\mathcal{N}$ die Menge aller nicht-expandierbaren, ganzzahligen Teilintervalle von $[0, N)$. Betrachten wir deren Partition bezüglich der Wahl der linken Intervallgrenze a in $\mathcal{N} = \mathcal{N}_{a\in\left[0,\frac{1}{4}N\right)} \uplus \mathcal{N}_{a\in\left[\frac{1}{4}N,\frac{1}{2}N\right)} \uplus \mathcal{N}_{a\in\left[\frac{1}{2}N,N\right)}$.

Sei also $[a, b)$ mit $a \in \left[0, \frac{1}{4}N\right)$ gegeben. Dann ist das Intervall genau dann expandierbar, wenn $b \leq \frac{1}{2}N$ gilt. Somit gibt es wegen $\frac{1}{4}N$ Möglichkeiten für a und $\frac{1}{2}N$ für b genau $\frac{1}{4}N \cdot \frac{1}{2}N = \frac{1}{8}N^2$ nicht-expandierbare Intervalle in $\mathcal{N}_{a\in\left[0,\frac{1}{4}N\right)}$.

Betrachte nun $a \in \left[\frac{1}{4}N, \frac{1}{2}N\right)$. Wäre $b \leq \frac{3}{4}N$, so wäre $[a, b)$ in der Mitte expandierbar. Also muss $b \in \left(\frac{3}{4}N, N\right]$ gelten. In diesem Fall kann das Intervall weder oben noch unten expandierbar sein und es folgt $\#\mathcal{N}_{a\in\left[\frac{1}{4}N,\frac{1}{2}N\right)} = \frac{1}{4}N \cdot \frac{1}{4}N = \frac{1}{16}N^2$.

[2] Da die Anzahl der Alphabetsymbole von $\mathbb{B}$ mit $\#\,\mathbb{B} = 2$ konstant und zudem angenehm klein ist, wollen wir diese in unsere Betrachtungen nicht mit einbeziehen.

			case $X=0$		case $X=1$	
state	prob($X=0$)		next state	output	next state	output
$[0,8)$	$\approx \frac{1}{8}$	$\mapsto$	($[0,8)$,	'000')	($[1,8)$,	-)
	$\approx \frac{2}{8}$	$\mapsto$	($[0,8)$,	'00')	($[2,8)$,	-)
	$\approx \frac{3}{8}$	$\mapsto$	($[0,6)$,	'0')	($[3,8)$,	-)
	$\approx \frac{4}{8}$	$\mapsto$	($[0,8)$,	'0')	($[0,8)$,	'1')
	$\approx \frac{5}{8}$	$\mapsto$	($[0,5)$,	-)	($[2,8)$,	'1')
	$\approx \frac{6}{8}$	$\mapsto$	($[0,6)$,	-)	($[0,8)$,	'11')
	$\approx \frac{7}{8}$	$\mapsto$	($[0,7)$,	-)	($[0,8)$,	'111')
	$\vdots$		$\vdots$	$\vdots$	$\vdots$	$\vdots$
$[0,6)$	$\approx \frac{1}{6}$	$\mapsto$	($[0,8)$,	'000')	($[1,6)$,	-)
	$\approx \frac{2}{6}$	$\mapsto$	($[0,8)$,	'00')	($[0,8)$,	'f')
	$\approx \frac{3}{6}$	$\mapsto$	($[0,6)$,	'0')	($[2,8)$,	'f')
	$\approx \frac{4}{6}$	$\mapsto$	($[0,8)$,	'0')	($[0,8)$,	'10')
	$\approx \frac{5}{6}$	$\mapsto$	($[0,5)$,	-)	($[0,8)$,	'101')
	$\vdots$		$\vdots$	$\vdots$	$\vdots$	$\vdots$
$[3,8)$	$\approx \frac{1}{5}$	$\mapsto$	($[0,8)$,	'011')	($[0,8)$,	'1')
	$\approx \frac{2}{5}$	$\mapsto$	($[0,8)$,	'ff')	($[2,8)$,	'1')
	$\approx \frac{3}{5}$	$\mapsto$	($[2,8)$,	'f')	($[0,8)$,	'11')
	$\approx \frac{4}{5}$	$\mapsto$	($[3,7)$,	-)	($[0,8)$,	'111')
	$\vdots$		$\vdots$	$\vdots$	$\vdots$	$\vdots$

Abbildung 3.8: Auszug aus einer Kodiertabelle zur Feinheit $N=8$

Gilt nun für den letzten Fall $a \geq \frac{1}{2}N$, so ist $[a, b)$ auf jeden Fall oben expandierbar, und es folgt $\mathcal{N}_{a \in [\frac{1}{2}N, N)} = \emptyset$. Somit gilt

$$\#\mathcal{N} = \frac{1}{8}N^2 + \frac{1}{16}N^2 + 0 = \frac{3}{16}N^2 \ .$$

Aussage b) folgt sofort durch Abzählen aller nicht-trivialen Trennstellen zwischen a und b. Selbstverständlich gibt es davon $b - a - 1$ viele. □

Wir können nun die Anzahl der Tabellenzeilen recht genau abschätzen. Da nach Beobachtung 3.16 alle nicht-expandierbaren Intervalle größer als $\frac{1}{4}N$, jedoch natürlich kleiner oder gleich N sind, hat die Tabelle somit zwischen $\frac{3}{64}N^3$ und $\frac{3}{16}N^3$ Einträge. Dieses kubische Wachstum in der Feinheit N sorgt somit leider noch für recht große Verzögerungszeiten bei der Auswertung. Aus diesem Grund geben HOWARD und VITTER noch verschiedene Methoden der Größenreduzierung auf $\mathcal{O}(N)$ Zeilen an.

Der erste offensichtliche Schritt, welcher die Tabelle verkleinert, ist eine Reduzierung der Feinheit N. Deren theoretische Wahl wäre groß genug, um jede verwendete Statistik p mit dem Hauptnenner N exakt darstellen zu können. Wenn wir nun N geringer wählen, so können die Quellverteilungen nur noch annähernd angegeben werden. Dies war zwar auch schon bei dem endlichen präzisen Algorithmus von WITTEN, NEAL und CLEARY der Fall, jedoch werden dort in der Praxis Werte im Bereich von $N = 65536$ oder gar noch höher eingesetzt.[3] Bei der vorliegenden Tabellenkodierung wird jedoch eher an $N = 128$, $N = 32$ oder gar $N = 8$ gedacht. Die Autoren sprechen in diesem Fall von *quasi-arithmetischer Kodierung*.

HOWARD und VITTER zeigen daher auf, dass sogar relativ grobe Näherungen der echten Quellverteilung zu guten Kompressionsergebnissen führen. Das folgende Ergebnis stammt ebenfalls aus [HV92], wurde dort aber nicht bewiesen. Der dazu gehörige Beweis folgt den Methoden aus [BCW90].

Lemma 3.20 *Sei $p = (p_1, p_2, \ldots, p_k)$ die Wahrscheinlichkeitsverteilung einer Nachrichtenmenge Σ^n. Wird zur* WNC*-Kodierung statt dessen eine Verteilung $q = (q_1, q_2, \ldots, q_k)$ verwendet, so gilt für den erwarteten Kompressionsverlust ε in Bits:*

a) $\varepsilon \ \leq \ \mathrm{D}(p\|q) + 1$ *und*

b) $\mathrm{D}(p\|q) \ = \ \frac{1}{2 \cdot \ln 2} \cdot \sum_{i=1}^{k} \frac{d_i^2}{p_i} + \mathcal{O}\left(\frac{d_i^3}{p_i^2}\right)$,

wobei die $d_i := q_i - p_i$ den Differenzvektor der Verteilungen p und q bezeichnen.

Beweis: Da das Codewort $C(w_i)$ nun die Länge $\left\lceil \log \frac{1}{q_i} \right\rceil + 1$ statt $\left\lceil \log \frac{1}{p_i} \right\rceil + 1$ erhält, jedoch

[3] Die exakte Wahl hängt von der maximal darstellbaren Integerzahl ab.

weiterhin mit Wahrscheinlichkeit p_i auftritt, gilt für den erwarten Kompressionsverlust

$$\begin{aligned}\varepsilon &= \sum_{i=1}^{k} p_i \cdot \left(\left\lceil \log \frac{1}{q_i} \right\rceil + 1 - \left\lceil \log \frac{1}{p_i} \right\rceil - 1 \right) \\ &\leq \sum_{i=1}^{k} p_i \cdot \left(\log \frac{p_i}{q_i} + 1 \right) = \sum_{i=1}^{k} p_i \log \frac{p_i}{q_i} + \sum_{i=1}^{k} p_i = D(p\|q) + 1 \ .\end{aligned}$$

Betrachten wir daher $D(p\|q)$. Es folgt mit Substitution der q_i durch $p_i + d_i$:

$$D(p\|q) = \sum_{i=1}^{k} p_i \log \frac{p_i}{p_i + d_i} = -\sum_{i=1}^{k} p_i \log \frac{p_i + d_i}{p_i} = -\sum_{i=1}^{k} p_i \log \left(1 + \frac{d_i}{p_i}\right) \ .$$

Nun gilt laut der Taylor-Reihenentwicklung bei $|x| < 1$ für $\ln(1 + x) = x - \frac{1}{2}x^2 + \mathcal{O}(x^3)$ mit kubischem Fehlerterm. Da $\left|\frac{d_i}{p_i}\right| < 1$ für $q_i \approx p_i$, können wir dies auf den Logarithmus in der Summe anwenden und es folgt

$$\begin{aligned}D(p\|q) &= -\frac{1}{\ln 2} \cdot \sum_{i=1}^{k} p_i \ln \left(1 + \frac{d_i}{p_i}\right) \\ &= -\frac{1}{\ln 2} \cdot \sum_{i=1}^{k} p_i \left(\frac{d_i}{p_i} - \frac{d_i^2}{2p_i^2} + \mathcal{O}\left(\frac{d_i^3}{p_i^3}\right)\right) \\ &= -\frac{1}{\ln 2} \cdot \sum_{i=1}^{k} \left(d_i - \frac{d_i^2}{2p_i} + \mathcal{O}\left(\frac{d_i^3}{p_i^2}\right)\right) \\ &= -\frac{1}{\ln 2} \cdot \underbrace{\sum_{i=1}^{k} d_i}_{=0} + \frac{1}{2\ln 2} \cdot \sum_{i=1}^{k} \frac{d_i^2}{p_i} + \mathcal{O}\left(\frac{d_i^3}{p_i^2}\right) \\ &= \frac{1}{2\ln 2} \cdot \sum_{i=1}^{k} \frac{d_i^2}{p_i} + \mathcal{O}\left(\frac{d_i^3}{p_i^2}\right),\end{aligned}$$

da wegen $d_i = q_i - p_i$ für die Summe $\sum d_i = \sum(q_i - p_i) = \sum q_i - \sum p_i = 0$ gilt. □

Die Tatsache, dass der präzisionsbedingte Fehler d_i einer Wahrscheinlichkeit lediglich quadratisch in den Fehlerterm eingeht, zeigt diesbezüglich eine gute Robustheit der Kompressionsrate. Es kann also davon ausgegangen werden, dass jede geeignete Approximation der echten Nachrichtenverteilung nur zu geringen Abweichungen führt.

Abgesehen davon gibt es noch weitere Möglichkeiten, die Größe einer Kodiertabelle bei gegebener Feinheit zu reduzieren. Ein Ansatz, welcher die Anzahl der Zeilen in etwa halbiert, basiert auf der Beobachtung, dass die Tabellen wie in Abbildung 3.8 ein großes Maß an Symmetrie aufweisen. So verhalten sich z.B. die Codewortlängen bei wachsender Wahrscheinlichkeit des Symbols 0 und fallender Wahrscheinlichkeit der 1 völlig analog.

			case $X = MPS$		case $X = LPS$	
state	prob($X = MPS$)		next state	output	next state	output
$[0, 8)$	$\approx \frac{4}{8}$	$\mapsto$	($[0, 8)$,	'0')	($[0, 8)$,	'1')
	$\approx \frac{5}{8}$	$\mapsto$	($[0, 5)$,	-)	($[2, 8)$,	'1')
	$\approx \frac{6}{8}$	$\mapsto$	($[0, 6)$,	-)	($[0, 8)$,	'11')
	$\approx \frac{7}{8}$	$\mapsto$	($[0, 7)$,	-)	($[0, 8)$,	'111')
	$\vdots$		$\vdots$	$\vdots$	$\vdots$	$\vdots$
$[0, 6)$	$\approx \frac{3}{6}$	$\mapsto$	($[0, 6)$,	'0')	($[2, 8)$,	'f')
	$\approx \frac{4}{6}$	$\mapsto$	($[0, 8)$,	'0')	($[0, 8)$,	'10')
	$\approx \frac{5}{6}$	$\mapsto$	($[0, 5)$,	-)	($[0, 8)$,	'101')
	$\vdots$		$\vdots$	$\vdots$	$\vdots$	$\vdots$

Abbildung 3.9: Auszug aus einer verkleinerten Kodiertabelle zur Feinheit $N = 8$

Dank der Flexibilität der arithmetischen Kodierung kann jede Runde nicht nur die verwendete Verteilung, sondern auch die Anordnung der Unterteilungen wechseln. Daher kann bei binären Alphabeten auch in jeder Runde das häufigere Symbol den unteren und das weniger wahrscheinliche Symbol den oberen Abschnitt erhalten.

Wir können also als Symbolalphabet auch die Menge $\{MPS, LPS\}$ betrachten, wobei MPS das wahrscheinlichere Symbol (*most propable symbol*) und LPS das unwahrscheinlichere Symbol (*least propable symbol*) bezeichnet. Die Zuordnung von 0 und 1 auf diese beiden Symbole kann aus gegebener Verteilung p direkt abgeleitet werden. Offensichtlich muss dann stets prob$(X = MPS) \geq \frac{1}{2}$ gelten, was uns knapp die Hälfte aller Zeilen einspart. Dieses Verfahren geht auf Langendon und Rissanen zurück (vgl. [LR86]) und bildet die Grundlage vieler unserer Betrachtungen aus Kapitel 5.

Abbildung 3.9 zeigt ein Beispiel einer solchen verkleinerten Kodiertabelle. Es fällt außerdem auf, dass gewisse Zustände — im Beispiel etwa $[3, 8)$ — gar nicht mehr auftreten können. Dieses Intervall hat eine ungerade Intervallgrenze, kann also nicht aus den Verdopplungen der Expansionsschritte entstanden sein. Ebenso wenig ist es aber die Unterteilung durch eine wie oben beschrieben angeordnete Verteilung. Somit können also weitere Zustände ignoriert werden.

Howard und Vitter geben weitere Verbesserungen an, welche wir in dieser Arbeit jedoch nicht weiter betrachten wollen, wie etwa einem systematischen Ansatz zur Entwicklung eines Tabellenkodierers mit nur linear in N vielen Tabellen oder die optimalen Wahl der Wahrscheinlichkeits-Schwellenwerte mittels so genannter ε/ρ-Partitionen. Diese Konzepte spielen jedoch für unsere folgenden Betrachtungen keine Rolle.

Eine ausführlichere Beschreibung des binären Tabellenkodierers als in [HV92] kann bei Interesse in [Kun03] gefunden werden.

3.5 Konstruktion der Dekodiertabellen

Wir haben uns bisher nur auf die Kodierung der Nachrichten aus $\mathbb{B}^n$ mittels Tabellenarithmetik konzentriert. Es ist daher noch notwendig, die Dekodierung von Codeworten zu betrachten. Satz 3.11 garantiert uns die eindeutige Entschlüsselbarkeit der *WNC*-Kodierung. Wie diese Dekodierung unter Vermeidung arithmetischer Operationen durch Tabellen realisiert werden kann, soll in diesem Abschnitt an Hand von Beispielen aufgezeigt werden.

Betrachten wir dazu die Kodiertabelle 3.9. Auch der Dekodierer besitzt dieselben Zustände, die zur Kodierung verwandt wurden. Somit kommt deren reduzierte Anzahl auch der Dekodiertabelle zu Gute. Darüber hinaus nehmen wir an, dass stets — entsprechend des Modells — die richtige Verteilung zur Verfügung steht.

Es lassen sich nun auch die Dekodierschritte durch eine Übergangstabelle δ definieren. Dabei werden in Abhängigkeit von aktuellem Zustand, Verteilung und darüber hinaus auch noch dem Codewort das neue Intervall sowie das ursprüngliche Symbol MPS bzw. LPS bestimmt. Da wir dabei auch das Codewort x Bit für Bit abarbeiten wollen, um die Pipelining-Eigenschaft der arithmetischen Kodierung zu erhalten, interessieren wir uns in der Tat nur für einen kleinen Ausschnitt von x. Dieses Fenster wird dann Stück für Stück über das Codewort hinweg geschoben. Zu diesem Zweck werden wir die Tabelle nicht in Abhängigkeit des gesamten Wortes, sondern nur eines Präfixes des Teilwortes ab der aktuellen Fensterposition formulieren und zudem als Ausgabe noch einen Wert zurückliefern, um welchen dieses Fenster „nach rechts“ verschoben werden soll.

Wir wollen zum besseren Verständnis eine konkrete Konstruktion angeben. Betrachte dazu als aktuellen Zustand $[0,8)$ und als Verteilung $X \sim p$ mit $\text{prob}(X = MPS) \approx \frac{4}{8}$, also die erste Zeile aus der Kodiertabelle 3.9. Betrachtet man diese nun aufmerksam, so sieht man sofort, dass genau dann ein MPS kodiert wurde, falls das Codewort-Fenster mit einer 0 beginnt und genau dann ein LPS an der Reihe war, falls dort eine 1 steht. Der Folgezustand ist in beiden Fällen $[0,8)$. Man kann für δ in der Tabelle also

$$\begin{aligned}\delta: \quad & \big(\,[0,8)\;,\;\text{prob}(X=MPS)\approx\tfrac{4}{8}\;,\;\text{prefix } 0\,\big) &\mapsto\quad & \big(\,[0,8)\;,\;\text{shift } 1\;,\;\text{'}MPS\text{'}\,\big)\\ & \big(\,[0,8)\;,\;\text{prob}(X=MPS)\approx\tfrac{4}{8}\;,\;\text{prefix } 1\,\big) &\mapsto\quad & \big(\,[0,8)\;,\;\text{shift } 1\;,\;\text{'}LPS\text{'}\,\big)\end{aligned}$$

formulieren.

Untersuchen wir nun zum gleichen Zustand die Verteilung mit $\text{prob}(X = MPS) \approx \frac{5}{8}$, so ist die Dekodierung nicht ganz so offensichtlich. Betrachten wir also zunächst einmal den Fall, dass das Codewort-Fenster mit einer 0 beginnt. Offensichtlich musste also beim Kodieren der nächste Expansionsschritt, der ausgeführt wurde, eine untere oder eine mittlere Expansion gewesen sein. Lag eine follow-on Sequenz vor, so musste zumindest der erste darauf folgende Expansionsschritt wieder ein unterer gewesen sein. Eine obere Expansion hätte schließlich eine 1 ausgegeben. Anders ausgedrückt bedeutet dies, dass der *WNC*-kodierte Punkt in der linken Hälfte des Ausgangsintervalls liegen muss. Dieser Bereich ist aber komplett dem Symbol MPS zugeordnet worden. Abbildung 3.10 illustriert diesen Zusammenhang. Also ist das Symbol in diesem Fall bestimmt. Da laut Kodiertabelle aber für MPS kein Code-Bit ausgegeben wurde, wird das Fenster auch nicht weiter geschoben, und der neue Zustand ist $[0,5)$.

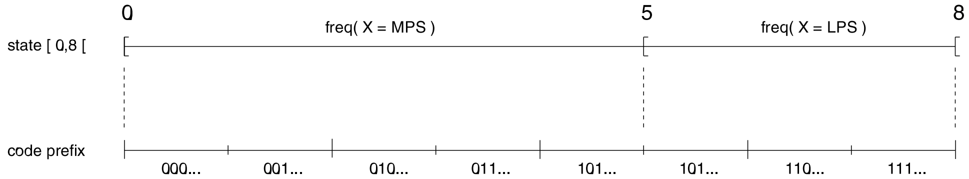


Abbildung 3.10: Binärer Unterteilungsschritt von $[0,8)$ bei prob$(X = MPS) \approx \frac{5}{8}$

Was aber, wenn das Codewort mit einer 1 beginnt? Abbildung 3.10 lässt erkennen, dass dann sowohl MPS als auch LPS möglich wären. Jedoch kann nur ein Codewort, dessen Präfix 100 ist, zum Symbol MPS korrespondieren. Alle anderen möglichen Codewörter müssen demnach zu LPS gehören. In diesem Fall wurde laut Kodiertabelle ein Bit ausgegeben und in den Zustand $[5,8)$ gewechselt. Wir können also in der δ-Tabelle die folgenden Einträge angeben.

$$\begin{array}{llcl}
\delta: & \big(\ [0,8)\ ,\ \text{prob}(X = MPS) \approx \frac{5}{8}\ ,\ \text{prefix } 0\ \big) & \mapsto & \big(\ [0,5)\ ,\ \text{shift } 0\ ,\ \text{'}MPS\text{'}\ \big) \\
& \big(\ [0,8)\ ,\ \text{prob}(X = MPS) \approx \frac{5}{8}\ ,\ \text{prefix } 100\ \big) & \mapsto & \big(\ [0,5)\ ,\ \text{shift } 0\ ,\ \text{'}MPS\text{'}\ \big) \\
& \big(\ [0,8)\ ,\ \text{prob}(X = MPS) \approx \frac{5}{8}\ ,\ \text{else}\ \big) & \mapsto & \big(\ [5,8)\ ,\ \text{shift } 1\ ,\ \text{'}LPS\text{'}\ \big)
\end{array}$$

Mit Sorgfalt und etwas „Handarbeit“ kann also die Dekodiertabelle aus der Kodiertabelle abgeleitet werden. Abbildung 5.6 zeigt dazu ein Beispiel. Wir werden in Kapitel 5 eine Verallgemeinerung der Konstruktion dieser Tabellen auf beliebige Alphabete betrachten.

LEKATSAS und WOLF verwenden diese Technik auf binären Alphabeten zur Erläuterung der Dekodierung bei ihrem *SAMC*-Kodierer (vgl. [LW99]). Als generelles Problem dieses Ansatzes wird jedoch angemerkt, dass jeweils nur ein Nachrichtensymbol auf einmal bestimmt wird. Die Autoren schlagen daher vor, mehrere Schritte des Dekodierers in einer großen Tabelle zusammen zu fassen. Da dies in einigen Punkten unseren späteren Betrachtungen im Wege steht, wollen wir diese Verbesserung nicht weiter betrachten.

state	prob($X = MPS$)	prefix		next state	shift	output
$[0,8)$	$\approx \frac{4}{8}$	0	$\mapsto$	($[0,8)$,	1 ,	'MPS')
		1	$\mapsto$	($[0,8)$,	1 ,	'LPS')
	$\approx \frac{5}{8}$	0	$\mapsto$	($[0,5)$,	0 ,	'MPS')
		100	$\mapsto$	($[0,5)$,	0 ,	'MPS')
		else	$\mapsto$	($[2,8)$,	1 ,	'LPS')
	$\approx \frac{6}{8}$	0	$\mapsto$	($[0,6)$,	0 ,	'MPS')
		10	$\mapsto$	($[0,6)$,	0 ,	'MPS')
		11	$\mapsto$	($[0,8)$,	2 ,	'LPS')
	$\approx \frac{7}{8}$	111	$\mapsto$	($[0,8)$,	3 ,	'LPS')
		else	$\mapsto$	($[0,7)$,	0 ,	'MPS')
$\vdots$	$\vdots$	$\vdots$		$\vdots$	$\vdots$	$\vdots$
$[0,6)$	$\approx \frac{3}{6}$	00	$\mapsto$	($[0,6)$,	1 ,	'MPS')
		010	$\mapsto$	($[0,6)$,	1 ,	'MPS')
		011	$\mapsto$	($[2,8)$,	1 ,	'LPS')
		1	$\mapsto$	($[2,8)$,	1 ,	'LPS')
	$\approx \frac{4}{6}$	0	$\mapsto$	($[0,8)$,	1 ,	'MPS')
		1	$\mapsto$	($[0,8)$,	2 ,	'LPS')
	$\approx \frac{5}{6}$	0	$\mapsto$	($[0,5)$,	0 ,	'MPS')
		100	$\mapsto$	($[0,5)$,	0 ,	'MPS')
		else	$\mapsto$	($[0,8)$,	3 ,	'LPS')
$\vdots$	$\vdots$	$\vdots$		$\vdots$	$\vdots$	$\vdots$

Abbildung 3.11: Auszug aus einer Dekodiertabelle zur Feinheit $N = 8$

4 Kodierung über beliebigen Alphabeten durch Verteilungsbäume

„Habt Ehrfurcht vor dem Baum.
Er ist ein einziges großes Wunder,
und euren Vorfahren war er heilig.“
— Alexander Freiherr von Humboldt

Howard und Vitter schlagen in [HV92] die Verwendung einer eigenen Datenstruktur vor, um mit einem binären arithmetischen Kodierer Texte über Alphabeten beliebiger Größe zu komprimieren. Die Rede ist von den sogenannten *komprimierten Verteilungsbäumen* (*compressed trees*).

Leider verzichten sie bei ihren Ausführungen auf eine nähere Beschreibung. Wir wollen daher im Folgenden eine konkrete Arbeitsweise für diese Datenstruktur vorschlagen und sie auf ihre Nützlichkeit hin analysieren. Dazu werden wir eine Strategie zur platzeffizienten Speicherung von Bäumen heranziehen, die auf der Arbeit von Jacobson basiert (vgl. [Jac89b]).

Im ersten Abschnitt dieses Kapitels wollen wir einige wohl bekannte Definitionen und Ergebnisse über bestimmte Bäume wiederholen. Der nächste Abschnitt beschäftigt sich mit der Konstruktion unserer Baumstruktur. Danach werden wir beweisen, dass die binäre arithmetische Kodierung mit Hilfe von Verteilungsbäumen die selben Ergebnisse erzielt, wie die bereits betrachtete arithmetische Kodierung für beliebige Alphabete. Anschließend werden wir im vierten Abschnitt einige Platzschranken für die komprimierte Variante unserer Verteilungsbäume beweisen, um im darauf folgenden fünften Abschnitt die konkrete Datenstruktur anzugeben. Dem schliesst sich in Abschnitt sechs ein Fazit an, welches die Szenarien umreißt, für welche komprimierte Verteilungsbäume eine lohnende Option sind.

4.1 Grundsätzliches über Bäume

Um die komprimierten Verteilungsbäume näher zu untersuchen, müssen wir einige grundlegende Erkenntnisse über Bäume anführen. Wir wollen jedoch auf eine vollständige Einführung in diese Thematik verzichten. Eine sehr gute Übersicht dazu liefern [Sed88] oder [CLR90].

Wir werden uns vielmehr auf die für uns notwendigen Definitionen und Ergebnisse konzentrieren. Dazu gehören auch die folgenden Sprechweisen.

Definition 4.1 *a) Unter einem* Baum $T = (V, E)$ *verstehen wir einen gerichteten, zyklenfreien Graphen, der einen* Wurzelknoten $v \in V$ *besitzt, so dass zu jedem anderen Knoten ein eindeutiger Pfad existiert. Knoten* $u, w \in V$ *heißen* Vorfahre *bzw.* Nachfahre *von einander, wenn sie als Vorgänger bzw. Nachfolger auf einem solchen Pfad auftauchen.*

b) Die Länge des Pfades von der Wurzel zu einem Knoten w heißt die Tiefe *von w. Sprechen wir von der Tiefe eines Baumes T, so meinen wir die maximale Tiefe seiner Knoten.*

c) Die Anzahl der direkten Nachfahren eines Knotens w nennen wir den Grad *von w. Knoten mit Grad 0 heißen* Blätter*, alle anderen nennen wir* innere Knoten. *Ein Baum heißt t-*är*, wenn jeder Knoten höchstens Grad t hat.*

d) Ein t-ärer Baum heißt regulär*, falls jeder innere Knoten genau Grad t besitzt. Existiert darüber hinaus ein $d \in \mathbb{N}_0$, so daß jedes Blatt die Tiefe d hat, so nennen wir den Baum* vollständig. *Wir nennen einen regulären Baum* fast vollständig*, falls jedes Blatt entweder Tiefe d oder $d - 1$ hat.*

Im Folgenden werden wir uns nun mit einigen Eigenschaften von vollständigen beziehungsweise fast vollständigen Bäumen befassen. Diese Ergebnisse werden uns in den Beweisen und Konstruktionen der kommenden Abschnitte helfen.

Lemma 4.2 *Ein fast vollständiger, t-ärer Baum mit m Blättern ist genau dann vollständig, wenn $m = t^d$ gilt, wobei d die Tiefe von T ist.*

Beweis: Wir zeigen die Behauptung durch vollständige Induktion nach der Tiefe des Baumes d. Für $d = 0$ ist die Aussage dabei mit $m = 1$ trivialerweise klar.

Es gelte nun die Aussage für Bäume mit Tiefe d, und sei T ein vollständiger Baum mit Tiefe $d + 1$. Man konstruiere nun den ebenfalls vollständigen Baum $\hat{T}$, indem aus T alle Blätter mit deren adjazenten Kanten entfernt werden.

Nun sind die Blätter von $\hat{T}$ gerade die direkten Eltern-Knoten der Blätter von T, und alle Blätter befinden sich auf Tiefe d. Da T nach Voraussetzung regulär ist, also jeder innere Knoten genau t Kinder hat, besitzt T nun t-mal so viele Blätter wie $\hat{T}$. Mit der Induktionsvoraussetzung hat $\hat{T}$ genau t^d Blätter, und die Anzahl der Blätter von T beträgt $t \cdot t^d = t^{d+1}$.

Damit ist die eine Richtung der Aussage gezeigt. Sei für die Rückrichtung nun T ein fast vollständiger, t-ärer Baum mit t^d Blättern, und sei $\hat{T}$ wie oben konstruiert. Da $\hat{T}$ die Tiefe $d-1$ und t^{d-1} Blätter hat, muss der Baum nach Voraussetzung auch vollständig sein, d.h. alle Blätter befinden sich auf Tiefe $d-1$. Entsprechend der Konstruktion von $\hat{T}$ folgt somit auch die Behauptung für T. □

Wir werden sehen, dass komprimierte Verteilungsbäume spezielle Teilbäume von fast vollständigen, binären Bäumen sind. Hilfreich bei unseren Betrachtungen ist auch das folgende Lemma.

Lemma 4.3 *Sei T ein regulärer, binärer Baum mit m Blättern. Dann besitzt T genau $m-1$ innere Knoten.*

Beweis: Zum Beweis führen wir eine vollständige Induktion nach der Anzahl der Blätter m. Sei also zunächst $m = 1$. Dann existieren trivialerweise keine innereren Knoten, da auf Grund des regulären Grades der inneren Knoten von T sonst noch mindestens ein weiteres Blatt existieren müsste.

Sei nun angenommen, die Behauptung gilt für ein $m > 1$. Betrachte einen regulären Baum T mit $m+1$ Blättern. Dann gibt es auch ein Blatt v mit maximaler Tiefe d. Sei u der Vater von v und, da u als innerer Knoten Grad 2 haben muss, w das andere Kind von u. Nun muss u ebenfalls ein Blatt sein, denn sonst würden Blätter mit einer Tiefe größer als d exsitieren, was der Wahl von v widerspräche.

Wir konstruieren aus T nun einen Baum $\hat{T}$, indem wir sowohl das Blatt v als auch den inneren Knoten u entfernen und das Blatt w an die Stelle von u setzen. $\hat{T}$ hat nun ein Blatt und einen inneren Knoten weniger T. Somit besitzt $\hat{T}$ nun m Blätter und nach Induktionsvoraussetzung $m-1$ innere Knoten. Demzufolge hat T genau $m-1+1=m$ innere Knoten und es folgt die Behauptung. □

Folgerung 4.4 *Sei T ein regulärer, binärer Baum mit m Blättern. Dann besitzt T insgesamt $2m-1$ Knoten. Insbesondere ist die Anzahl der Knoten also immer ungerade.*

Um die Bezeichnung „Verteilungsbaum“ begrifflich zu rechtfertigen, benötigen wir noch die Erkenntnis über eine einfache Konstruktion eines fast vollständigen Binärbaumes.

Lemma 4.5 *Sei $m \in \mathbb{N}$. Der ungerichtete Graph $T_m = (V, E)$ mit*

$$V_m := \{1, 2, \ldots, 2m-1\} \quad \textit{und}$$
$$E_m := \Big\{ (v, 2v), (v, 2v+1) \,\Big|\, v \in \{1, 2, \ldots, m-1\} \Big\}$$

ist ein fast vollständiger, binärer Baum mit m Blättern.

Beweis: Auch diesen Beweis werden wir mittels vollständiger Induktion führen. Betrachte also zunächst $m = 1$. Dann erfüllt der Graph $T_1 = \big(\{1\}, \emptyset\big)$ trivialerweise die Behauptung.

Sei also im Folgenden angenommen, dass T_{m-1} ein fast vollständiger, binärer Baum mit $m-1$ Blättern ist. Wir betrachten dann T_m. Auf Grund seiner Konstruktion ist dann

T_{m-1} ein echter Teilgraph von T_m mit $V_m \backslash V_{m-1} = \{2m-2, 2m-1\}$ und $E_m \backslash E_{m-1} = \{(m, 2m-2), (m, 2m-1)\}$.

Da T_{m-1} bereits ein Baum war, bleibt T_m auch ein Baum, da nur zwei Kanten zu zwei vorher nicht verbundenen Knoten eingefügt wurden. Dabei war in T_{m-1} der Knoten m ein Blatt; durch das Anhängen von nur zwei neuen Kindern bleibt somit die binäre Eigenschaft erhalten. Mit diesen beiden Kindern hat der Knoten m Grad 2, und auf Grund der Regularität von T_{m-1} haben somit auch alle inneren Knoten von T_m genau Grad 2.

Es bleibt somit nur zu zeigen, dass darüber hinaus noch ein $d \in \mathbb{N}_0$ existiert, so dass alle Blätter von T_m die Tiefe d oder $d-1$ haben. Aus der Konstruktion geht hervor, dass $\{m, m+1, \ldots, 2m-1\}$ die Blätter von T_m sind. Nun ist der Pfad von der Wurzel 1 zu dem Knoten v durch $\left(1, \ldots, \left\lfloor \frac{v}{4} \right\rfloor, \left\lfloor \frac{v}{2} \right\rfloor, v\right)$ bestimmt. Also beträgt die Tiefe eines Knotens gerade $d_v := \lfloor \log v \rfloor$. Auf Grund der Monotonie des Logarithmus gilt daher für jedes Blatt v

$$d_m = \lfloor \log m \rfloor \;\leq\; d_v$$

und

$$d_v \leq \lfloor \log(2m-1) \rfloor \leq \lfloor \log(2m) \rfloor = \lfloor \log(m) \rfloor + 1 \;=\; d_m + 1.$$

□

Über diese Bäume lassen sich auch eine Reihe weiterer, sehr einfacher Beobachtungen anstellen, auf deren Beweise wir hier verzichten wollen.

Beobachtung 4.6 *Sei $m \in \mathbb{N}$ und T_m wie in Lemma 4.5 definiert. Bezeichne d die Tiefe von T_m. Dann gilt:*

a) Die Tiefe von T_m beträgt genau $d = \lceil \log(m) \rceil$.

b) Ein beliebiger Knoten v von T_m ist genau dann ein innerer Knoten, wenn $v \leq m-1$.

c) Ein innerer Knoten v von T_m hat genau dann mindestens einen inneren Knoten als Kind, wenn $v \leq \left\lfloor \frac{m-1}{2} \right\rfloor$.

d) Ein innerer Knoten v von T_m hat genau dann exakt zwei innere Knoten als Kinder, wenn $v \leq \left\lfloor \frac{m-2}{2} \right\rfloor$.

e) Ein Blatt v von T_m hat genau dann Tiefe $d-1$, wenn $v < 2^d$.

f) Ein Blatt v von T_m hat genau dann Tiefe d, wenn $v \geq 2^d$.

4.2 Konstruktion von Verteilungsbäumen

Wir wollen nun zunächst einführen, was unter einem Verteilungsbaum zu einer gegebenen Wahrscheinlichkeitsverteilung zu verstehen ist. Dabei werden wir zunächst die einfache, unkomprimierte Variante betrachten, bevor wir in den nächsten Abschnitten unseren Blick auf die Möglichkeiten einer platzeffizienten Speicherung richten wollen.

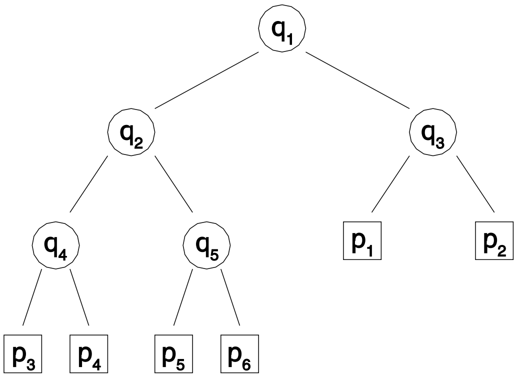


Abbildung 4.1: Struktur eines Verteilungsbaumes mit $m = 6$

Sei ein Symbolalphabet Σ der Größe m, und dessen Wahrscheinlichkeitsverteilung p gegeben. Unter einem Verteilungsbaum für Σ und p wollen wir einen fast vollständigen Baum verstehen, dessen Knoten reellen Wert zwischen 0 und 1 zugeordnet sind. Stellt $\{1, 2, .., 2m-1\}$ die Knotenmenge dar, so konstruieren wir den Verteilungsbaum derart, dass die Kinder des Knotens v die Knoten $2v$ und $2v+1$ sind.

Dabei korrespondieren die Blätter zu je genau einem Symbol von Σ; die Werte der Blätter entsprechen gerade der jeweiligen Symbolwahrscheinlichkeit. Die Werte der inneren Knoten hingegen stellen den Anteil der Summe der Werte der Blätter des linken Teilbaums im Verhältnis zu der Summe über die Blätter beider Teilbäume dar[1].

Insgesamt werden Verteilungsbäume — entsprechend den obigen Ausführungen — durch die folgende formale Definition beschrieben.

Definition 4.7 *Sei $p = (p_1, p_2, \ldots, p_m)$ eine Wahrscheinlichkeitsverteilung der Größe m. Sei die Sequenz $q = (q_1, q_2, \ldots, q_{m-1})$, $q_i \in [0,1] \cup \{-1\}$, mit den rekursiv definierten Hilfssummen s_i für alle $1 \le i \le 2m-1$ wie folgt:*

$$s_i := \begin{cases} s_{2i} + s_{2i+1} & \text{falls } 1 \le i \le m-1 \\ p_{i-m+1} & \text{falls } m \le i \le 2m-1 \end{cases}$$

und für alle $1 \le i \le m-1$

$$q_i := \begin{cases} \frac{s_{2i}}{s_i} & \text{falls } s_i \neq 0 \\ -1 & \text{falls } s_i = 0 \end{cases} .$$

Der Baum $T = (V, E)$, wobei

$$V := \{1, 2, .., 2m-1\} \text{ und}$$
$$E := \left\{ (v, 2v), (v, 2v+1) \,\middle|\, v \in \{1, 2, .., m-1\} \right\} ,$$

[1]Selbstverständlich hätte man hier auch völlig analog eine Charakterisierung über die Summe der Blätter des rechten Teilbaumes heranziehen können.

mit der Wertabbildung $w : V \to [0,1] \cup \{-1\}$, *so dass*

$$w(v) := \begin{cases} q_v & \textit{falls } 1 \leq v \leq m-1 \\ p_{v-m+1} & \textit{falls } m \leq v \leq 2m-1 \end{cases}$$

heißt der Verteilungsbaum *zu* p.

Diese zugegeben etwas undurchsichtige Konstruktion entspricht den vorausgegangenen verbalen Beschreibungen der Verteilungsbäume. Dies veranschaulicht unter anderem auch die folgende Beobachtung.

Beobachtung 4.8 *Seien* T *und* s_i *wie in Definition 4.7. Dann gibt für jeden Knoten* v *von* T *die Hilfsumme* s_v *die Summe über die Werte der Blätter des in* v *wurzelnden Teilbaumes an.*

Beweis: Bezeichne zu v der Wert t_v die Anzahl der Kanten des längsten Pfades von v zu einem seiner Blätter. Die Aussage folgt sofort aus einer einfachen Induktion nach t_v. Ist nämlich $t_v = 0$, so ist v selbst ein Blatt. Gilt hingegen die Behauptung für alle Knoten w mit Pfadlänge $0 \leq t_w < t_v$, so gilt nach Konstruktion $s_v = s_{2v} + s_{2v+1}$. Da nach Induktionsvorrausetzung die Kinder die Aussage erfüllen, folgt somit auch die Aussage für den Knoten v. □

Der Baum T und die Abbildung w sind demnach so wie angegeben wohldefiniert. Es wird jedoch erst in Lemma 4.13 genauer darauf eingegangen, warum es sinnvoll ist, einen Knoten v mit $w(v) = -1$ zu markieren, falls $s_v = 0$ gilt.

Verteilungsbäume wie oben beschrieben können effizient konstruiert werden. Dazu kann einfach der Baum T in $\mathcal{O}(m)$ Schritten direkt angegeben werden. Anschliessend werden die Hilfssummen s_i und damit die Abbildung w entsprechend der Konstruktionsvorschrift in einer Schleife beginnend beim Knoten $2m-1$ bis zur Wurzel 1 berechnet, also in ebenfalls $\mathcal{O}(m)$ Schritten. Da die Größe des Alphabets in der Regel konstant ist, ergibt sich somit insgesamt eine konstante Laufzeit.

4.3 Binäre arithmetische Kodierung mit Verteilungsbäumen

Um nun ein Wort w über einem beliebigen Alphabet Σ mit Hilfe eines binären arithmetischen Kodierers zu komprimieren, wollen wir jedes einzelne Symbol von w durch eine Sequenz über dem binären Alphabet $\mathbb{B}$ ausdrücken. Diese wollen wir als die *Spur* von w bezeichnen. Zu deren eindeutiger Definition werden wir den Verteilungsbaum zu der Quellwahrscheinlichkeitsverteilung von Σ heranziehen.

Definition 4.9 *Sei* Σ *ein Alphabet mit* m *Symbolen und* p *deren Wahrscheinlichkeitsverteilung. Bezeichne* $T = (V, E)$ *den Verteilungsbaum von* p. *Dazu sei* $P_a := (v_0, v_1, \ldots, v_k)$

der Pfad in T von der Wurzel $v_0 = 1$ bis zu dem Blatt v_k, welches zu a korrespondiert, und es sei

$$b_i := \begin{cases} 0 & \textit{falls } v_i = 2v_{i-1} \\ 1 & \textit{falls } v_i = 2v_{i-1} + 1 \end{cases} .$$

Dann sei die Abbildung $\tau : \Sigma \to \mathbb{B}^*$ definiert durch

$$\tau(a) := b_1 b_2 \ldots b_k.$$

Für die Erweiterung $\tau : \Sigma^* \to \mathbb{B}^*$ definiere

$$\tau(w) := \tau(w_1)\tau(w_2) \ldots \tau(w_n)$$

wobei $w = w_1 w_2 \ldots w_n$ mit $w_i \in \Sigma$. Wir bezeichen $\tau(w)$ als die Spur *von w.*

Zur Ermittlung der Spur eines Wortes $w \in \Sigma^*$ können wir also den Verteilungsbaum von der Wurzel zu dem Blatt, welches dem zuletzt gelesenen Symbol w_i entspricht, traversieren, und geben jeweils eine 0 aus, falls ein Knoten links verlassen wurde, bzw. geben eine 1 aus, falls wir uns für den rechten Teilbaum entschieden haben. Die $b_i \in \mathbb{B}$ sind dabei wohldefiniert, da nach Konstruktion von T jede Kante $(v, w) \in E$ entweder von der Form $(v, 2v)$ oder $(v, 2v + 1)$ sein muss. Wir sehen also, dass wir für jedes mittels τ zu kodierende Symbol einmal einen Pfad von der Wurzel bis zu einem Blatt durchlaufen müssen. Bei einem Nachrichtenblock der Länge n entspräche dies bei einem Alphabet der Größe m in etwa $\mathcal{O}(n \cdot \log m)$ vielen Schritten. Da die Alphabetgröße in der Regel als konstant angenommen werden kann, ergibt sich somit eine lineare Laufzeit.

Wir können die Spur eines Symbols aber auch direkt bestimmen. Ist unser Alphabet $\Sigma = \{a_1, a_2, \ldots, a_m\}$, so entsprechen die Symbole nach Konstruktion in eben dieser Reihenfolge den Blättern $\{m, m+1, \ldots, 2m-1\} \subset V$, also a_i entspricht dem Knoten $m + i - 1$. Es ist leicht ersichtlich, dass die Spur von a_i gerade den Ziffern der Binärdarstellung der Zahl $m + i - 1$ hinter der führenden 1 entspricht.

Dass wir ebenso aus der Spur eines Wortes die ursprüngliche Nachricht eindeutig rekonstruieren können, ist offensichtlich. Noch allgemeiner ist hingegen das folgende Lemma.

Lemma 4.10 *Betrachte die Abbildung $\tau : \Sigma^n \to \mathbb{B}^*$ auf den Nachrichtenwörtern der Blocklänge n. Dann stellt τ einen präfixfreien Code dar.*

Beweis: Würde ein echter Präfix $\tau(a_i)$ von $\tau(a_j)$ für Symbole $a_i, a_j \in \Sigma$ existieren, so entspräche nach den vorausgegangenen Beobachtungen a_i einem echten Vorgängerknoten auf dem Pfad von der Wurzel bis zu dem Knoten zu a_j. Symbol a_i gehört aber genau wie a_j zu einem Blatt, was zum Widerspruch führt. Betrachtet man nun einen angenommenen Präfix $\tau(w)$ von $\tau(v)$ für Nachrichten aus Σ^n, so lässt sich induktiv zeigen, dass diese Symbol für Symbol übereinstimmen müssen. □

Zur Komprimierung der Spur $\tau(w) = b_1 b_2 \ldots b_r$, $b_i \in \mathbb{B}$, mit einem binären Kodierer benötigen wir aber natürlich noch ein passendes Wahrscheinlichkeitsmodell. Hier kommt wieder der Verteilungsbaum T von p ins Spiel. Zu dem aktuellen Knoten v definiert $w(v)$ eine Verteilung mit $\mathrm{prob}(Y = 0) = w(v)$ und $\mathrm{prob}(Y = 1) = 1 - w(v)$. Diese liegt zu jedem b_i dem binären arithmetischen Kodierer zu Grunde.

Wir haben also gezeigt, dass wir das Komprimieren eines Nachrichtenblockes über einem beliebigen Alphabet auf das Komprimieren von Nachrichten über dem binären Alphabet $\mathbb{B}$ in Echtzeit zurückführen können. Dabei haben wir aber bisher außer Acht gelassen, wie sich dieser Übergang auf die Kompressionsrate auswirkt. Wir werden nun sehen, dass sich bei binärer, arithmetischer Kodierung mit Verteilungsbäumen dieselbe Codewort-Länge erwarten lässt, wie wir sie auch bei Kodierung über einem beliebigen Alphabet erwartet hätten. Zu diesem Zweck werden wir zunächst das folgende, zentrale Lemma beweisen.

Lemma 4.11 *Sei Σ ein Alphabet mit m Symbolen, p deren Wahrscheinlichkeitsverteilung und bezeichne $T = (V, E)$ den Verteilungsbaum von p mit der Abbildung w. Ferner sei zu $a \in \Sigma$ mit $P_a = (v_0, v_1, \ldots, v_k)$ der Pfad von dem Wurzelknoten $v_0 = 1$ zu dem mit a korrespondierenden Blatt v_k bezeichnet. Mit der Partitionierung der inneren Knoten von P_a in*

$$L_a := \left\{ v_i \in \{v_0, \ldots, v_{k-1}\} \,\middle|\, v_{i+1} = 2v_i \right\} \text{ und}$$
$$R_a := \left\{ v_i \in \{v_0, \ldots, v_{k-1}\} \,\middle|\, v_{i+1} = 2v_i + 1 \right\} ,$$

also die Knoten, welche links bzw. rechts verlassen werden, gilt dann für $X \sim p$:

$$\mathrm{prob}(X = a) = \prod_{v \in L_a} w(v) \cdot \prod_{v \in R_a} \bigl(1 - w(v)\bigr) .$$

Mit anderen Worten: Das Produkt der binären Wahrscheinlichkeiten der inneren Knoten entlang des Pfades P_a entspricht der Symbolwahrscheinlichkeit von a in Σ.

Beweis: Bezeichne im Folgenden — analog zur Definition 4.7 — $q_v = w(v)$ den Wert des inneren Knotens v und bezeichne $p_a = \mathrm{prob}(X = a)$ die Quellwahrscheinlichkeit des Symbols a und damit den Wert des zu a korrespondierenden Blattes. Wir wollen hier zwei Fälle unterscheiden.

Angenommen, es existiert ein innerer Knoten v_i auf dem Pfad P_a mit $s_{v_i} = 0$. Sei o.E. v_i der erste solche Knoten auf dem Pfad. Dann kann v_i nicht der Wurzelknoten $v_0 = 1$ sein, denn nach Beobachtung 4.8 wäre somit die Summe über die Blätter gleich 0, was der Tatsache widerspräche, dass p eine Wahrscheinlichkeitsverteilung ist.

Demnach hat v_i einen Vorgänger v_{i-1} auf dem Pfad P_a. Da v_i der erste Knoten mit $s_{v_i} = 0$ ist, gilt $s_{v_{i-1}} > 0$. Ist v_i nun das linke Kind von v_{i-1}, so folgt nach Konstruktion $q_{v-1} = 0$, anderenfalls muss $q_{v-1} = 1$ gelten. In jedem Fall kommt durch q_{v-1} oder $1 - q_{v-1}$ ein Faktor 0 in das zu zeigende Produkt.

Da nun auch $p_a = 0$ gelten muss, weil dies sonst nach Beobachtung 4.8 $s_{v_i} = 0$ widerspricht, folgt sofort die Gleichheit.

Betrachten wir also nun den interessanteren Fall, d.h. es gelte für alle $i = 0 \ldots k - 1$, dass $s_{v_i} > 0$. Hierbei lassen sich einige einfache Beobachtungen über die Hilfssummen machen. Entlang des Pfades P_a gilt für alle $i = 1, \ldots, k-1$:

$$s_{v_{i+1}} = \begin{cases} s_{2v_i} & \text{falls } v_i \in L_a \\ s_{2v_i+1} & \text{falls } v_i \in R_a \end{cases} .$$

Dies folgt direkt aus der Definition 4.7 des Verteilungsbaumes T. Demnach hat der innere Knoten v_i lediglich entweder das linke Kind $2v_i$ oder das rechte Kind $2v_i + 1$.

Betrachtet man nun die q_{v_i} entsprechend ihrer Konstruktion, so gilt nach der vorausgegangenen Überlegung:

$$\begin{aligned} q_{v_i} &= \frac{s_{2v_i}}{s_{v_i}} = \frac{s_{v_i+1}}{s_{v_i}} && \text{falls } v_i \in L_a \\ (1-q_{v_i}) &= \frac{s_{2v_i+1}}{s_{v_i}} = \frac{s_{v_i+1}}{s_{v_i}} && \text{falls } v_i \in R_a \end{aligned}$$

Dies bedeutet also, dass sich in der zu zeigenden Gleichung der Faktor zu v_i — ungeachtet dessen, ob es nun q_{v_i} oder $1 - q_{v_i}$ ist — stets als der Bruch $\frac{s_{v_i+1}}{s_{v_i}}$ ausdrücken lässt. Damit gilt also, da $L_a \cup R_a = P_a \setminus \{v_k\}$:

$$\prod_{v \in L_a} q_v \cdot \prod_{v \in R_a} (1 - q_v) = \frac{s_{v_1}}{s_{v_0}} \cdot \frac{s_{v_2}}{s_{v_1}} \cdot \ldots \cdot \frac{s_{v_i}}{s_{v_{i-1}}} \cdot \frac{s_{v_{i+1}}}{s_{v_i}} \cdot \ldots \cdot \frac{s_{v_k}}{s_{v_{k-1}}} .$$

Wie man nun leicht sieht, lässt sich dieser Ausdruck als Teleskop-Produkt sehr stark kürzen. Da nun, zusätzlich zu unseren bisherigen Überlegungen, nach Beobachtung 4.8

$$s_{v_0} = \sum_{u=m}^{2m-1} w(u) = \sum_{\alpha \in \Sigma} p_\alpha = 1$$

gilt, folgt somit

$$\prod_{v \in L_a} q_v \cdot \prod_{v \in R_a} (1 - q_v) = \frac{1}{s_{v_0}} \cdot s_{v_k} = s_{v_k} = p_a$$

nach der Definition von s_i für die Blätter von T, und die Behauptung ist bewiesen. □

Mit dieser Erkenntnis gelingt es uns nun, die Äquivalenz beider Kodieransätze zu zeigen.

Satz 4.12 *Die Kompressionsrate eines arithmetischen Kodierers über einem beliebigen Alphabet entspricht der eines binären arithmetischen Kodierers mit Verteilungsbäumen.*

Beweis: Wir zeigen, dass Wort für Wort die Kompressionsraten der beiden Kodierer übereinstimmen. Da die Wörter aus der gleichen Quelle stammen, muss dann auch die erwartete Codewort-Länge gleich sein.

Sei Σ das beliebige Alphabet mit m Symbolen und p die dazugehörige Wahrscheinlichkeitsverteilung. Da wir die einzelnen Quellsymbole als von einander stochastisch unabhängig ansehen, gilt insbesondere für $w = w_1 w_w \dots w_n$:

$$\operatorname{prob}(X = w) \;=\; \prod_{i=1}^{n} \operatorname{prob}(X_i = w_i) \;.$$

Es genügt also, das Verhalten auf den Symbolen von Σ zu untersuchen. Betrachte nun zu einem $a \in \Sigma$ die Wahrscheinlichkeit $\operatorname{prob}(X = a)$ mit $X \sim p$. Dann definiert X eine weitere Zufallsvariable Y mit $X \sim Y$ durch

$$\operatorname{prob}\bigl(Y = \tau(a)\bigr) := \operatorname{prob}(X = a) \;.$$

Sei nun zu jedem inneren Knoten v aus dem Verteilungsbaum T von p eine weitere Zufallsvariable Y_v vermöge

$$\operatorname{prob}(Y_v = 0) := w(v) \text{ und } \operatorname{prob}(Y_v = 1) := 1 - w(v)$$

definiert. Dann folgt mit Lemma 4.11 für den Pfad $P_a = (v_0, v_1, \dots, v_k)$ und dessen Partitionierung in L_a und R_a:

$$\begin{aligned}
\operatorname{prob}\bigl(Y = \tau(a)\bigr) &= \operatorname{prob}(X = a) \\
&= \prod_{v \in L_a} w(v) \cdot \prod_{v \in R_a} \bigl(1 - w(v)\bigr) \\
&= \prod_{v \in L_a} \operatorname{prob}(Y_v = 0) \cdot \prod_{v \in R_a} \operatorname{prob}(Y_v = 1) \\
&= \prod_{i=0}^{k-1} \operatorname{prob}(Y_{v_i} = b_i) \;.
\end{aligned}$$

wobei $\tau(a) = b_1 b_2 \dots b_{k-1}$ mit $b_i \in \mathbb{B}$ gilt.

Demnach ist die Wahrscheinlichkeit eines Wortes $w = w_1 w_2 \dots w_n \in \Sigma^*$ die einfache Produktwahrscheinlichkeit über die einzelnen Bits der Spur von w. Bezeichnet nun

$$l(w) \;=\; \left\lceil \log\bigl(\operatorname{prob}(X = w)^{-1}\bigr) \right\rceil + 1$$

wie in Definition 3.9 die Codewort-Länge von w auf einem arithmetischen Kodierer für das Alphabet Σ, so entspricht sie der Codewort-Länge

$$l'\bigl(\tau(w)\bigr) \;=\; \left\lceil \log\left(\prod_{i=1}^{k} \operatorname{prob}(Y_{v_i} = b_i)^{-1} \right) \right\rceil + 1$$

von $\tau(w) = b_1 b_2 \dots b_{k-1}$ auf einem binären, arithmetischen Kodierer mit Verteilungsbäumen. $\square$

4.4 Komprimierte Verteilungsbäume

Wir haben in den vorausgegangenen Abschnitten gesehen, wie man Verteilungsbäume effizient konstruiert. Wir haben ausserdem gezeigt, dass sich die Schrittanzahlen eines arithmetischen Kodierers für binäre und für beliebige Alphabete nur um einen konstanten Faktor unterscheiden und dabei die Kompressionsrate unangetastet bleibt. Gerade aber bei Implementierungen in Hardware oder auf speicherarmen Medien wie etwa Smartcards spielt der Platzbedarf der Verteilungsbäume eine wesentliche Rolle. Wir wollen daher diesen Bedarf nun auch noch analysieren und einen konkreten Verbesserungsvorschlag beschreiben, welcher unsere bisherigen Ergebnisse nicht gefährdet.

Asymptotisch benötigt ein Verteilungsbaum sicherlich $\mathcal{O}(m)$ Einträge, um seine $2m-1$ Knoten und deren Werte abzuspeichern. Wir werden hier diesbezüglich auch keinen Durchbruch präsentieren können. Vielmehr werden wir eine Reduzierung des absoluten Speicherbedarfes anstreben. Dabei folgen wir dem Ansatz von Howard und Vitter, indem wir zu unserem Verteilungsbaum nur die Werte abspeichern wollen, die auch wirklich benötigt werden (vgl. [HV92]).

Als erste Einschränkung können wir uns sicher aller Blätter des Verteilungsbaumes entledigen. Wie bei den Traversierungen des letzten Abschnittes beschrieben, sind für die binäre arithmetische Codierung nur die Werte der inneren Knoten von Bedeutung[2].

Einen inneren Knoten werden wir nur dann benötigen, falls sein Wert zumindest einmal bei der Kodierung eines Symbols genutzt wird. Dazu muss er auf einen Pfad von der Wurzel zu dem Blatt eines Symbols liegen, dessen Symbolwahrscheinlichkeit echt größer 0 ist. Das Kriterium zur Unterscheidung danach, welche inneren Knoten nun notwendig sind und welche nicht, haben wir bereits in unsere Definition 4.7 eingebaut.

Lemma 4.13 *Sei Σ ein Alphabet, p dessen Wahrscheinlichkeitsverteilung und T der Verteilungsbaum zu p mit der Werteabbildung w. Ein innerer Knoten v von T liegt dann, und nur dann, auf einem Pfad $P_a = (v_0, v_1, \ldots, v_k)$ von der Wurzel $v_0 = 1$ zu einem Blatt v_k, welches mit einem $a \in \Sigma$ mit $\mathrm{prob}(X = a) > 0$ korrespondiert, wenn $w(v) \neq -1$.*

Beweis: Seien im Folgenden die Hilfssummen s_i wie in Definition 4.7 beschrieben. Betrachte zunächst einen inneren Knoten v, welcher auf einem solchen Pfad liegt. Dann hat zumindest ein Symbol, welches zu einem Blatt des in v wurzelnden Teilbaumes korrespondiert, einen Wert $\mathrm{prob}(X = a) > 0$. Nach Beobachtung 4.8 entspricht s_v der Summe über die Werte aller Blätter des Teilbaumes von v, also folgt auch $s_v > 0$. Somit wird in der Konstruktion von w der Knoten auch nicht mit $w(v) = -1$ markiert.

Sei nun umgekehrt v ein innerer Knoten mit $w(v) \neq -1$. Dann muss nach der Definition von w insbesondere $s_v > 0$ gelten. Wiederum nach Beobachtung 4.8 muss also mindestens ein Blatt des Teilbaumes in v einen positiven Wert haben. Es folgt die Behauptung. □

Demnach können nicht notwendige Knoten bereits bei der Konstruktion des Verteilungsbaumes en passant identifiziert werden. An Hand dieser Charakterisierung wollen wir nun auch die *komprimierten Verteilungsbäume* einführen.

[2] Und selbst wenn wir doch einmal den Wert eines Blattes benötigen würden, so gibt uns Lemma 4.11 an, wie wir diesen aus den Werten der inneren Knoten berechnen können.

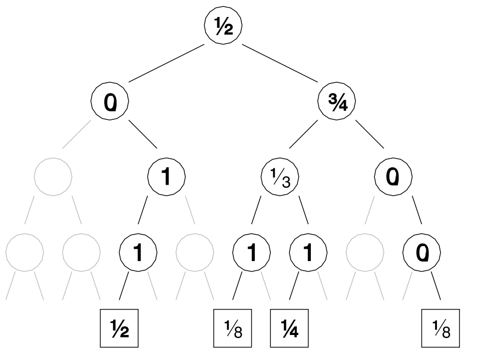


Abbildung 4.2: Beispiel eines komprimierten Verteilungsbaumes zu der Wahrscheinlichkeitsverteilung $p = \left(0, 0, 0, 0, \frac{1}{2}, 0, 0, 0, \frac{1}{8}, 0, \frac{1}{4}, 0, 0, 0, 0, \frac{1}{8}\right)$

Definition 4.14 *Sei $T = (V, E)$ ein Verteilungsbaum zu der Wahrscheinlichkeitsverteilung p mit der Werteabbildung w. Dann sei $T^* = (V^*, E^*)$ wie folgt definiert:*

$$V^* := \left\{v \in \{1, \dots, m-1\} \,\middle|\, w(v) \neq -1\right\} \cup \left\{v \in \{m, \dots, 2m-1\} \,\middle|\, w(v) \neq 0\right\}$$

und

$$E^* := \left\{(u, v) \in E \,\middle|\, u, v \in V^*\right\} .$$

Wir bezeichnen T^* zusammen mit der Abbildung $w^* := w|_{V^*}$ als den komprimierten Verteilungsbaum *zu p.*

Nach obiger Definition bleiben alle Blätter, welche ein Symbol mit Wahrscheinlichkeit größer 0 repräsentieren, formal als Blätter unseres komprimierten Verteilungsbaumes bestehen. Deren Speicherung ist zwar nach unseren einleitenden Überlegungen nicht notwendig, sie in die Definition mit einzubeziehen erleichtert uns aber die folgenden Beweise.

Die Bezeichnung von T^* als binärer Baum ist gerechtfertigt. Da er nach Konstruktion Teilgraph von T ist, erbt er dessen Zyklenfreiheit. Aus Lemma 4.13 folgt direkt, dass für jeden Knoten in T^*, sei es ein innerer Knoten oder ein Blatt, ein Pfad von der Wurzel bis zu diesem Knoten in T^* existiert. Dieser Pfad ist eindeutig, da er es in T bereits war, und somit sind alle Baumeigenschaften erfüllt. Jedoch ist T^* im Allgemeinen nicht regulär.

Wir werden im folgenden Satz Schranken für die Anzahl der inneren Knoten eines komprimierten Verteilungsbaumes angeben. Die obere Schranke stammt von Howard und Vitter. Leider verzichten sie in [HV92] auf einen Beweis, weswegen wir ihn hier nachholen werden. Darüber hinaus können wir auch noch eine untere Schranke beweisen, was uns sehr hilft, die Nützlichkeit von komprimierten Verteilungsbäumen abzuschätzen.

Satz 4.15 *Sei $T^* = (V^*, E^*)$ der komprimierte Verteilungsbaum zu einer Wahrscheinlichkeitsverteilung $p = (p_1, p_2, \dots, p_m)$ der Größe $m = 2^d$, $d \in \mathbb{N}$, und sei*

$$U := V^* \setminus \{m, \dots, 2m-1\}$$

die Menge der inneren Knoten von T^*. Bezeichnet dann k die Anzahl der Wahrscheinlichkeiten aus p mit $p_i > 0$, so gilt:

$$\log \frac{m}{2k} + k \;\leq\; \#U \;\leq\; k \cdot \log \frac{2m}{k} - 1 \; .$$

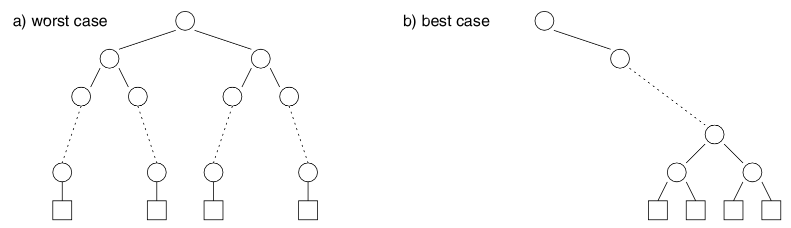


Abbildung 4.3: Skizze von komprimierten Verteilungsbäumen mit $k = 4$. *(a)* beschreibt den ungünstigsten Fall für die obere Schranke aus Satz 4.15, *(b)* den günstigsten Fall für die untere Schranke.

Bevor wir die Aussage beweisen können, müssen wir noch untersuchen, wie wir bezüglich Alphabet Σ und zu gegebenem k einen „größtmöglichen" bzw. „kleinstmöglichen" komprimierten Verteilungsbaum beschreiben können.

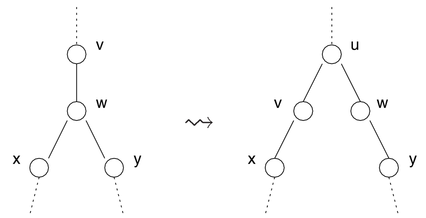
Wir beginnen mit den Betrachtungen zur oberen Schranke. Besonders interessant sind hierbei diejenigen Bäume, welche die folgende Eigenschaft besitzen.

Definition 4.16 *Ist T^* ein komprimierter Verteilungsbaum, so dass keiner seiner inneren Knoten mit nur einem Kind eine geringere Tiefe als ein innerer Knoten mit zwei Kindern hat, so nennen wir T^** maximal-konform.

Lemma 4.17 *Sei T^* ein komprimierter Verteilungsbaum mit k von m ursprünglichen Blättern, und habe dieser bezüglich k die maximale Anzahl innerer Knoten. Dann ist T^* maximal-konform.*

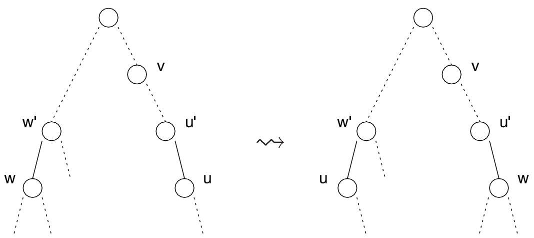
Beweis: Bezeichne im Folgenden zu jedem Knoten $v \in V^*$ mit d_v die Tiefe von v. Nehmen wir nun an, dass die Aussage nicht gilt. Dann existieren also innere Knoten v mit nur einem Kind und w mit zwei Kindern, für die $d_v < d_w$ gilt. Wir unterscheiden nun zwei Szenarien.

Betrachten wir zunächst den Fall, dass v und w auf einem gemeinsamen Pfad von der Wurzel bis zu einem Blatt liegen. Seien o.B.d.A v und w derart gewählt, dass v der direkte Vorgänger von w ist. Ein solches Paar muss existieren, da sonst kein Verstoss gegen die Behauptung vorläge. Bezeichne ferner x und y die beiden Kinder von w. Dann läßt sich aus T^* ein weiterer komprimierter Verteilungsbaum T' konstruieren, indem wir einen weiteren Knoten u einfügen und statt der Kanten (v, w) und (w, x) die Kanten (u, v), (u, w) und (v, x) einfügen.



Dann hat T' ebenfalls k Blätter aber $\# V^* + 1$ Knoten, was der Maximalität von T^* widerspräche.

Betrachten wir also nun den Fall, dass v und w nicht auf einem gemeinsamen Pfad von der Wurzel bis zu einem Blatt liegen. Nach Lemma 4.13 muss v aber dennoch auf einem Pfad von der Wurzel zu einem Blatt liegen. Also existiert auch ein Nachfolger u von v mit $d_u = d_w$. Bezeichne u' den direkten Vorgänger von u und w' den direkten Vorgänger von w. Nun konstruieren wir aus T^* einen weiteren komprimierten Verteilungsbaum $T' = (V', E')$, indem wir die Knotenmengen gleich lassen, aber die Kanten (u', u) und (w', w) durch (u', w) und (w', u) ersetzen; wir tauschen quasi die Teilbäume in u und w aus.



Nach unserem ersten Fall kann man dann aber einen komprimierten Verteilungsbaum T'' mit $\# V' + 1 = \# V^* + 1$ konstruieren, was zum Widerspruch führt. Insgesamt folgt also, dass T^* maximal-konform sein muss. □

Es ist leicht ersichtlich, dass wir mit den Transformationen aus dem obigen Beweis jeden komprimierten Verteilungsbaum T^* mit k Blättern in einen maximal-konformen, komprimierten Verteilungsbaum T' gleicher Blattanzahl überführen können. Es gilt dann stets $\# V^* \leq \# V'$. Dabei erhält die Transformation lediglich die Anzahl der Blätter des Baumes; die zu Grunde liegende Verteilung p und die Wertefunktion w kann (und wird im Allgemeinen auch) auf den neuen Baum nicht mehr zutreffend sein.

Wir haben im Lemma 4.17 gesehen, weswegen wir den Begriff maximal-konform gewählt haben. Alle komprimierten Verteilungsbäume maximaler Knotenzahl erfüllen diese Eigenschaft. Die Umkehrung gilt aber im Allgemeinen nicht, wenn der ursprüngliche Verteilungsbaum T nur fast vollständig war. Wir werden uns daher in den Beweisen auf Bäume zu Verteilungen mit $m = 2^d$, $d \in \mathbb{N}$, Wahrscheinlichkeiten beschränken.

Lemma 4.18 *Zu festen Parametern $m = 2^d$ und k haben alle maximal-konformen, komprimierten Verteilungsbäume mit k Blättern die gleiche Anzahl an Knoten.*

Beweis: Seien T^* und T' zwei solche Bäume. Da für beide Bäume wegen $m = 2^d$ der ursprüngliche Verteilungsbaum nach Lemma 4.2 vollständig war, haben somit alle Blätter die gleiche Tiefe d. Auf Grund der Maximal-Konformität und der gleichen Anzahl an Blättern existiert auch eine gemeinsame Tiefe $d' < d$, so dass alle inneren Knoten mit Tiefe kleiner d' zwei Kinder, und alle mit Tiefe größer d' ein Kind haben.

Betrachtet man nun sowohl von T^* als auch T' die Knoten der Ebenen 0 bis $d' - 1$, so haben diese alle den regulären Grad 2 und deren Anzahl ist für beide Bäume $2^{d'} - 1$. Auf Tiefe genau d' befinden sich, als Nachfolger der vollständigen Ebenen darüber, somit bei beiden Bäumen genau $2^{d'}$ Knoten.

Da die Anzahl der Blätter der Anzahl der Pfade von der Wurzel zu den Blättern entspricht, müssen auch alle k Pfade einen Knoten der Tiefe d' enthalten. Die Pfade müssen nun aber ab der Tiefe $d' + 1$ bis zu den Blättern auf Tiefe d paarweise disjunkt sein, da jeder innere Knoten ab Tiefe größer d' nur ein Kind hat. Demnach haben auch beide Bäume ab Tiefe $d' + 1$ genau $k \cdot (d - d')$, und somit insgesamt gleich viele Knoten. □

Folgerung 4.19 *Sei $T^*_{max} = (V^*, E^*)$ ein komprimierter Verteilungsbaum mit k der ursprünglichen $m = 2^d$ Blätter, so dass für alle inneren Knoten $v \in V^*$ mit nur einem Kind und alle inneren Knoten $w \in V^*$ mit zwei Kindern stets $v \geq w$ gilt. Dann hat T^*_{max} bezüglich k die maximale Anzahl innerer Knoten.*

Beweis: Da für alle inneren Knoten v von T^*_{max} mit nur einem Kind und alle inneren Knoten w mit zwei Kindern $v \geq w$ gilt, folgt aus der Konstruktion für komprimierte Verteilungsbäume, dass auch v die gleiche oder eine größere Tiefe als w haben muss. Also ist T^*_{max} maximal-konform.

Angenommen, es gibt einen komprimierter Verteilungsbaum $T' = (V', E')$ mit $\# V^* < \# V'$. Falls nun T' maximal-konform ist, so würde dies dem vorausgegangenen Lemma 4.18 widersprechen.

Also muss T' gegen diese Eigenschaft verstossen. Mit der Konstruktion aus dem Beweis von Lemma 4.17 lässt sich dann ein maximal-konformer, komprimierter Verteilungsbaum $T'' = (V'', E'')$ mit $\# V'' \geq \# V'$ angeben. Damit gilt dann

$$\# V^* < \# V' \leq \# V''$$

und es folgt abermals der Widerspruch mit Lemma 4.18. □

Wir benötigen für den Beweis der oberen Schranke von Satz 4.15 noch eine spezielle Ungleichung. Diese wird im folgenden Lemma gezeigt.

Lemma 4.20 *Für alle $x \in [0, 1]$ gilt $2^{1-x} \leq 2 - x$.*

Beweis: Sei $f(x) = 2 - x - 2^{1-x}$ die Differenz beider Seiten der Ungleichung. Wir werden zeigen, dass $f(x)$ für alle $x \in [0, 1]$ größer oder gleich 0 ist. Da sowohl $f(0) = 0$ als auch $f(1) = 0$ gilt, genügt es zu zeigen, dass f auf $(0, 1)$ keinen negativen Wert annehmen kann. Dazu werden wir ausnutzen, dass f zweimal stetig differenzierbar ist und führen eine einfache Kurvendiskussion durch.

Betrachte die erste Ableitung $f'(x) = \ln 2 \cdot 2^{1-x} - 1$. Ist nun x_0 eine Nullstelle von f', so gilt:

$$\begin{aligned}
& & 0 &= \ln 2 \cdot 2^{1-x_0} - 1 \\
&\Leftrightarrow & \frac{1}{\ln 2} &= 2^{1-x_0} \\
&\Leftrightarrow & -\log(\ln 2) &= 1 - x_0 \\
&\Leftrightarrow & x_0 &= 1 + \log(\ln 2) \approx 0,4712\ldots \;.
\end{aligned}$$

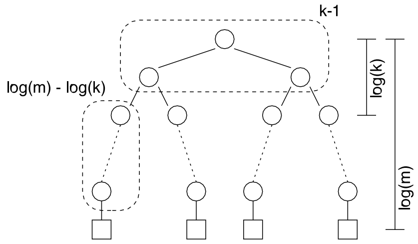
Demnach ist x_0 das einzige Extremum von f. Da für alle x stets $2^{1-x} > 0$ ist, gilt für die zweite Ableitung stets $f''(x) = -(\ln 2)^2 \cdot 2^{1-x} < 0$. Also ist x_0 ein Maximum. Da für die Intervallgrenzen $f(0) = f(1) = 0$ gilt, kann somit f auf $(0, 1)$ keinen negativen Wert annehmen. □

Beweis: (obere Schranke von Satz 4.15) Wir beweisen zunächst die obere Schranke

$$\# U \leq k \cdot \log \frac{2m}{k} - 1 \ .$$

Da m eine Zweierpotenz ist, ist nach Lemma 4.2 der Baum T vollständig, d.h. alle Pfade von der Wurzel zu einem Blatt sind gleich lang.

Sei ferner zunächst k, also die Anzahl der Blätter von T^*, ebenfalls eine Potenz von 2. Bezeichne T^*_{max} einen komprimierten Verteilungsbaum mit bezüglich k maximaler Knotenzahl wie in 4.19 beschrieben. Da T^* und T^*_{max} die gleiche Anzahl Blätter haben, besitzt T^*_{max} auch die maximale Anzahl innerer Knoten $\# U_{max}$. Die Situation folgt demnach der Skizze in Abbildung 4.3a.

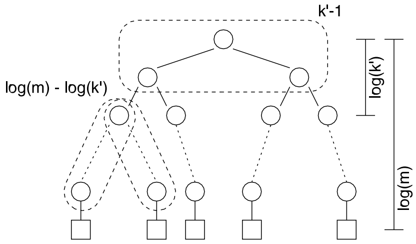


Einfaches Abzählen der inneren Knoten liefert nun schon $\# U_{max}$. Betrachtet man in T^*_{max} die Knoten vom Grad 2, so stellen diese einen perfekt balancierten, vollständigen Binärbaum dar, welche als „Blätter“ die k Pfade bis zu den wirklichen Blättern besitzen. Nach Lemma 4.3 gibt es also $k - 1$ solche Knoten. Da diese Pfade alle auf Tiefe $\log(k)$ beginnen, aber bei den Blättern auf Tiefe $\log(m)$ enden, besteht jeder Pfad demnach aus $\log(m) - \log(k)$ inneren Knoten. Da es nun aber genau k solche Pfade gibt folgt sofort

$$\begin{aligned} \# U_{max} &= k \cdot (\log m - \log k) + k - 1 \\ &= k \cdot \log \frac{m}{k} + k \log 2 - 1 \\ &= k \cdot \log \frac{2m}{k} - 1 \ . \end{aligned}$$

Betrachten wir nun ein k, welches keine Zweierpotenz ist, so existiert eine Zweierpotenz k' so dass $k' < k < 2k'$. Damit bezeichnet k' also die größte Zweierpotenz kleiner k. Insbesondere gilt dann $\lfloor \log(k) \rfloor = \log(k')$.

Zählt man nun die inneren Knoten von T^*_{max} ab, so sieht man, dass hier ebenfalls $k-1$ Knoten den Grad 2 haben. Diese teilen sich aber in $k' - 1$ Knoten auf, welche maximal die Tiefe $\log(k')$ haben und $k - k'$ Knoten, welche in der Ebene darunter liegen. Diese Knoten liegen auf genau zwei Pfaden zu zwei verschiedenen Blättern. Wenn man nun, wie im vorausgegangenen Fall, die Pfade ab Tiefe $\log(k')$ aufsummiert, so wird man diese



$k - k'$ Knoten jeweils doppelt zählen. Daher folgt

$$\begin{aligned}
\# U_{max} &= k \cdot (\log m - \log k') - (k - k') + k' - 1 \\
&= k \cdot \log \frac{m}{k'} - k + 2k' - 1 \\
&= k \cdot \left(\log \frac{m}{2k'} + \frac{2k'}{k} \right) - 1 \ .
\end{aligned}$$

Wir wollen zeigen, dass auch nun $\# U_{max} \leq k \cdot \log \frac{2m}{k} - 1$ gilt. Seien dazu im Folgenden $d, b \in \mathbb{N}_0$ und $\alpha \in (0,1)$ die Exponenten, so dass $m = 2^d$, $k' = 2^b$ und $k = 2^{b+\alpha}$. Dann gilt:

$$\begin{aligned}
& k \cdot \left(\log \frac{m}{2k'} + \frac{2k'}{k} \right) - 1 \leq k \cdot \log \frac{m}{2k} - 1 \\
\Leftrightarrow \quad & \log \frac{m}{2k'} + \frac{2k'}{k} \leq \log \frac{2m}{k} \\
\Leftrightarrow \quad & d - (b+1) + 2^{(b+1)-(b+\alpha)} \leq (d+1) - (b+\alpha) \\
\Leftrightarrow \quad & 2^{1-\alpha} \leq 2 - \alpha
\end{aligned}$$

Dies aber gilt offensichtlich nach Lemma 4.20 für alle $\alpha \in (0,1)$ und somit ist die obere Schranke bewiesen. □

Wenden wir uns nun der unteren Schranke zu. Auch hier wollen wir zunächst eine Charakterisierung von komprimierten Verteilungsbäumen minimaler Größe angeben.

Definition 4.21 *Ist T^* ein komprimierter Verteilungsbaum, so dass auf jeder Ebene maximal ein Knoten existiert, dessen induzierter Teilbaum nicht vollständig ist, so nennen wir T^** minimal-konform*.*

Im Gegensatz zur Maximal-Konformität kann es durchaus komprimierte Verteilungsbäume minimaler Größe geben, welche nicht minimal-konform sind. Auch die Umkehrung gilt im Allgemeinen nicht. Beschränken wir uns jedoch auf den Fall, dass der ursprüngliche Verteilungsbaum vollständig war, so impliziert Minimal-Konformität auch eine minimale Knotenzahl, wie wir mit Hilfe der folgenden Lemmata zeigen werden.

Lemma 4.22 *Sei $T^* = (V^*, E^*)$ ein komprimierter Verteilungsbaum mit k von $m = 2^d$ ursprünglichen Blättern. Dann existiert ein komprimierter Verteilungsbaum $T' = (V', E')$ mit k Blättern und $\# V^* \geq \# V'$, so dass T' minimal-konform ist.*

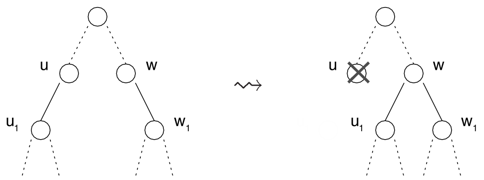
Beweis: Gibt d_v die Tiefe des Knotens v an, so bezeichnet $h_v := d - d_v$ die Höhe von v. Wir wollen mittels vollständiger Induktion nach der Höhe der Knoten zeigen, dass sich T' derart konstruieren lässt, so dass auf jeder Ebene maximal ein Knoten keinen vollständigen Teilbaum induziert.

Betrachte zunächst die Knoten mit Höhe 0, also Tiefe d. Auf dieser Ebene befinden sich keine inneren Knoten, sondern lediglich die Blätter, und somit folgt die Behauptung trivialerweise.

Gelte also nun die Aussage für alle inneren Knoten von T^* mit einer Höhe kleiner h. Das bedeutet insbesondere, dass höchstens ein Knoten mit Höhe $h-1$ einen Teilbaum induziert, welcher nicht vollständig ist. Wir wollen zeigen, dass wir dann auch einen Baum T' konstruieren können, so dass die Eigenschaft dann auch für alle Knoten auf Höhe h gilt. Dazu unterscheiden wir zwei Fälle.

Angenommen, die induzierten Teilbäume der Knoten mit Höhe $h-1$ sind alle vollständig. Dies bedeutet für alle Knoten mit Höhe h, dass ihre Teilbäume genau dann vollständig sind, wenn sie zwei Kinder haben. Dagegen ist der durch einen Knoten aufgespannte Teilbaum genau dann nicht vollständig, wenn er nur ein Kind hat. Schließlich muss jeder innere Knoten eines komprimierten Verteilungsbaumes wenigstens ein Kind haben, um auf einem Pfad von der Wurzel zu einem Blatt zu liegen, wie es Lemma 4.13 verlangt.

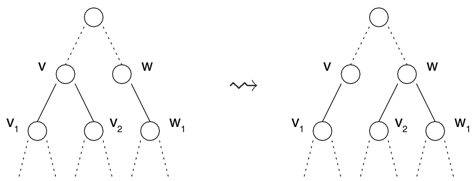
Nun läßt sich aus T^* schrittweise ein weiterer komprimierter Verteilungsbaum T' konstruieren. Dazu wähle, solange noch mindestens zwei Knoten u und w existieren, die nur ein Kind haben, diese beiden Knoten aus. Bezeichne mit u_1 und w_1 die jeweiligen Kinder. Nun ersetze die Kante (u, u_1) durch (w, u_1). Da sowohl u_1 als auch w_1 vollständige Teilbäume induzieren, ist nun auch der Teilbaum von w vollständig. Dafür hat u keinen Sinn mehr und wird, inklusive aller seiner Vorgänger bis zum ersten Knoten mit zwei Kindern, aus dem Baum entfernt.



Dieser Vorgang wird wiederholt bis höchstens noch ein innerer Knoten mit Höhe h existiert, dessen induzierter Teilbaum nicht vollständig ist. Der neue komprimierte Verteilungsbaum hat dann ebenfalls genau k Blätter. Da lediglich Knoten entfernt, aber keine hinzugefügt werden, folgt auch $\# V^* \geq \# V'$.

Betrachten wir also nun den Fall, dass genau ein Knoten auf Höhe $h-1$ existiert, welcher keinen vollständigen Teilbaum induziert. Bezeichne v dessen direkten Vorgänger. v induziert somit auch keinen vollständigen Teilbaum. Durch die oben vorgestellte Methode kann sichergestellt werden, dass höchstens ein weiter Knoten existiert, der ebenfalls gegen diese Eigenschaft verstößt. Falls kein solcher Knoten vorliegt, ist die Aussage ohnehin bereits gezeigt; sei also w dieser eine weitere Knoten. Mit der Argumentation von oben folgt, dass w genau ein Kind w_1 hat, und dass dieses Kind nach Induktionsvorraussetzung einen vollständigen Teilbaum induzieren muss.

Hat v seinerseits ebenfalls nur ein Kind, so können wir die Knoten einfach nach der obigen Methode zusammenlegen und wären bereits fertig. Habe v daher zwei Kinder v_1 und v_2 und sei o.B.d.A. v_1 der einzige Knoten mit Höhe $h-1$, in dem kein vollständiger Teilbaum wurzelt. Nun können wir T' konstruieren, indem wir die Kante (v, v_2) gegen die Kante (w, v_2) austauschen und somit auch den Teilbaum von w vervollständigen.



T' hat dann ebenfalls k Blätter, und mit v bleibt nur noch ein Knoten auf Höhe h übrig, der keinen vollständigen Teilbaum aufspannt. Natürlich gilt auch $\# V^* \geq \# V'$ und wir haben somit die Aussage bewiesen. □

Lemma 4.23 *Zu festen Parametern $m = 2^d$ und k haben alle minimal-konformen, komprimierten Verteilungsbäume mit k Blättern die gleiche Anzahl an Knoten.*

Beweis: Seien T^* und T' zwei solche Bäume. Nach Vorausetzung haben dann beide Bäume die Tiefe d. Hat der Knoten v die Tiefe d_v, so bezeichnet $h_v := d - d_v$ die Höhe von v. Wir werden zeigen, dass stets die Anzahl der Knoten mit Höhe h in T^* und T' übereinstimmen. Dazu führen wir eine vollständige Induktion nach h.

Betrachten wir zunächst die Knoten mit Höhe $h = 0$. Da wegen $m = 2^d$ auf dieser Ebene gerade alle Blätter liegen, haben sowohl T^* als auch T' insgesamt k Knoten mit dieser Höhe. Damit folgt also unser Induktionsanfang.

Gelte im Folgenden, dass T^* und T' gleich viele Knoten mit Höhe $h-1$ haben. Sei dazu N_{h-1} diese Anzahl. Da nach Voraussetzung unter allen Knoten mit Höhe h höchstens je Baum ein v in T^* und ein w in T' existiert, dessen induzierter Teilbaum nicht vollständig sind, haben auch alle anderen Knoten zwei Kinder. Die Knoten v und w haben demnach jeweils beide ein Kind, falls N_{h-1} ungerade, und beide ebenfalls zwei Kinder, falls N_{h-1} gerade ist.

Also gilt für die Anzahl der Knoten mit Höhe h beide Male $N_h = \left\lceil \frac{N_{h-1}}{2} \right\rceil$, und es folgt die Behauptung. □

Folgerung 4.24 *Besitzt der komprimierte Verteilungsbaum $T^*_{min} = (V^*, E^*)$ genau die k rechten der $m = 2^d$ Blätter des ursprünglichen Verteilungsbaum, so hat er bezüglich k die minimale Anzahl innerer Knoten.*

Beweis: Da T^*_{min} alle k rechten Blätter seines ursprünglichen Verteilungsbaumes besitzt, liegen in T^*_{min} auch alle dazugehörigen, rechten inneren Knoten. Demnach kann stets — Ebene für Ebene — höchstens der am weitesten linke innere Knoten keinen vollständigen Teilbaum induzieren; also stets maximal ein Knoten. Demnach ist T^*_{min} minimal-konform.

Angenommen, es existiert ein weiterer komprimierter Verteilungsbaum $T' = (V', E')$ mit k Blättern, so dass $\# V^* > \# V'$. Falls dann T' ebenfalls minimal-konform wäre, so folgt nach Lemma 4.23 der Widerspruch.

Demnach darf diese Eigenschaft also für T' nicht gelten. Dann existiert jedoch nach Lemma 4.22 ein minimal-konformer, komprimierter Verteilungsbaum $T'' = (V'', E'')$, so dass darüber hinaus $\# V' \geq \# V''$ gilt. Also gilt insgesamt

$$\# V^* > \# V' \geq \# V'' ,$$

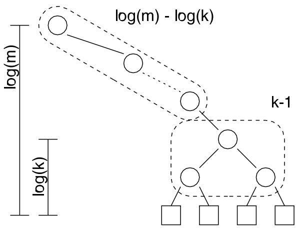
was jedoch wiederum im Widerspruch zu Lemma 4.23 steht. □

Beweis: (untere Schranke von Satz 4.15) Wir beweisen nun die untere Schranke

$$\log \frac{m}{2k} + k \leq \# U .$$

Da m eine Zweierpotenz ist, folgt aus Lemma 4.2, dass in T alle Pfade von der Wurzel zu einem Blatt gleich lang sind.

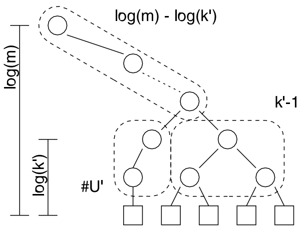
Wir betrachten zunächst den Fall, dass k, die Anzahl der Blätter von T^*, ebenfalls eine Potenz von 2 ist. Bezeichne dazu T^*_{min} einen komprimierten Verteilungsbaum mit k Blättern wie in 4.24 beschrieben. Demnach hat T^*_{min} bezüglich k die minimale Anzahl an Knoten. Da T^* und T^*_{min} die gleiche Anzahl Blätter haben, besitzt T^*_{min} auch die minimale Anzahl innerer Knoten $\# U_{min}$. Es liegt also ein Fall vor, wie er als Skizze in Abbildung 4.3b beschrieben ist.



Bestimmen wir nun die Anzahl der Knoten $\# U_{min}$, so existiert ein innerer Knoten auf Tiefe $\log m - \log k$, welcher einen vollständigen Teilbaum mit k Blättern induziert. Dieser hat nach Lemma 4.3 genau $k-1$ innere Knoten. Dazu kommen lediglich die $\log m - \log k$ Knoten bis zur Wurzel. Also folgt sofort

$$\begin{aligned} \# U_{min} &= \log m - \log k + (k-1) \\ &= \log \frac{m}{k} - \log 2 + k \\ &= \log \frac{m}{2k} + k \ . \end{aligned}$$

Um die Aussage nun für ein k zu zeigen, welches keine Zweierpotenz ist, werden wir eine vollständige Induktion nach der Tiefe d von T^*_{min} machen. Wir beginnen mit $d = 2$ — für kleinere d ist die Blattanzahl stets eine Potenz von 2 — und betrachten dort den einzigen Fall $k = 3$. Die Behauptung folgt sofort mit $\log \frac{2^d}{k} + k \approx 3.415 > 3 = \# U_{min}$.



Gilt unsere Abschätzung für alle komprimierten Verteilungsbäume mit Tiefe kleiner d, so können wir nun die Anzahl der inneren Knoten von T^*_{min} rekursiv angeben. Sei dazu k' die größte Zweierpotenz kleiner k, also $k' < k < 2k'$. Dann gilt nach Konstruktion von T^*_{min}, dass alle Blätter eines minimalen komprimierten Verteilungsbaumes mit k' auch in T^*_{min} enthalten sind. Deren Anzahl ist nach den vorausgegangenen Überlegungen $\log \frac{m}{2k'} + k'$. Dazu kommen allerdings nun noch die Knoten, welche für die weiteren $k - k'$ Blätter gebraucht werden. Ihre Menge sei mit U' bezeichnet. Dann gilt also

$$\# \, U_{min} = \log \frac{m}{2k'} + k' \; + \; \# \, U' \; .$$

Der von U' induzierte Teilbaum stellt aber selbst wieder einen minimalen komprimierten Verteilungsbaum mit Tiefe $\log(k') < d$ und $k - k'$ Blättern dar. Also folgt nach Induktionsvoraussetzung

$$\begin{aligned}
\# \, U_{min} &\geq \log \frac{m}{2k'} + k' \; + \; \log \frac{k'}{2(k-k')} + k - k' \\
&= \log \frac{m}{2k'} + \log \frac{k'}{2(k-k')} + k \\
&= \log \frac{m}{4(k-k')} + k \\
&= \log \frac{m}{2k + 2(k-2k')} + k \\
&> \log \frac{m}{2k} + k \; ,
\end{aligned}$$

da mit $k < 2k'$ auch $2(k - 2k') < 0$ gilt. Demnach ist unser Satz bewiesen. □

4.5 Effiziente Datenstruktur komprimierter Verteilungsbäume

Nachdem wir nun abschätzen können, wie viele Knoten wir in etwa bei einem komprimierten Verteilungsbaum zu erwarten haben, müssen wir uns nun noch überlegen, wie wir diesen platzeffizient speichern können. Dies soll jedoch derart geschehen, dass das Traversieren des Baumes — also das Auslesen der jeweiligen binären Wahrscheinlichkeiten entsprechend der Spur der Nachricht — ohne Zeitverlust geschehen kann.

Wir werden dazu ab sofort, entsprechend den Überlegungen aus der Einleitung des vorangegangenen Abschnittes, den komprimierten Verteilungsbaum T^* ohne die Blätter des ursprünglichen Baumes T betrachten. Wir wollen aber noch eine weitere praxisrelevante Änderung vornehmen. Streng nach der Definition 4.7 würde die Werteabbildung w von T^* reellwerte Ergebnisse liefern. Ähnlich zu den implementierungspezifischen Überlegungen aus Kapitel 3 werden wir nun nicht weiter mit der Wahrscheinlichkeitsverteilung p, sondern einer ganzzahligen Häufigkeitsverteilung $f = (f_1, f_2, \dots, f_m)$, welche p induziert, argumentieren. Die Änderungen bei der Konstruktion der Bäume sind minimal, da wir mit den bekannten Hilfssummen

$$s_i := \begin{cases} s_{2i} + s_{2i+1} & \text{falls } 1 \leq i \leq m-1 \\ f_{i-m+1} & \text{falls } m \leq i \leq 2m-1 \end{cases}$$

die Werteabbildung nun durch

$$w(v) := \begin{cases} s_v & \text{falls } 1 \leq v \leq m-1 \\ f_{v-m+1} & \text{falls } m \leq v \leq 2m-1 \end{cases}$$

mit ganzen Zahlen definieren können. Eine besondere Bedeutung kommt dabei der Summe $F_m = \sum_{i=1}^{m} f_i$ zu, da deren Logarithmus die maximale Bitgröße eines f_i angibt.

Definition 4.25 *Sei $f = (f_1, f_2, \dots, f_m)$ eine Häufigkeitsverteilung. Dann bezeichnet*

$$\Delta_f := \left\lceil \log \left(\sum_{i=1}^{m} f_i \right) \right\rceil$$

die maximale Anzahl an Bits, die zur Speicherung einer Häufigkeit von f benötigt werden.

Um die Speicherung eines komprimierten Verteilungsbaumes zu beschreiben, werden wir uns der folgenden Sprechweise bedienen.

Definition 4.26 *Sei ein Baum $T = (V, E)$ mit der Abbildung $w : V \to \mathbb{N}_0$ gegeben. Unter einer* (binären) Linearisierung *von T, kurz $\langle T \rangle$, verstehen wir die eindeutige Zuordnung von T und w zu einer binären Zeichenkette aus $\mathbb{B}^*$.*

Aus der Definition geht hervor, dass zur Speicherung eines Baumes sowohl dessen Inhalt (die Abbildung w), als auch Struktur (V und E) zugeordnet werden muss. Es gibt bestimmt mannigfaltige Möglichkeiten, den Inhalt eines Baumes per se zu komprimieren. Wir werden jedoch aus Gründen der Zeiteffizienz auf diesen Ansatz völlig verzichten und konzentrieren uns statt dessen auf die Speicherung der Struktur. Für die Speicherung eines Wertes werden wir deswegen annehmen, dass $w(v)$ durch seine Kodierung als Binärzahl $\text{bin}\big(w(v)\big)$ abgespeichert wird.

Als erstes, einfaches Beispiel dazu wenden wir uns noch einmal einem unkomprimierten Verteilungsbaum $T = (V, E)$ aus Abschnitt 4.2 zu. Sei dabei m die Anzahl seiner Blätter. Bereits bei deren Konstruktion haben haben wir bei den Mengen V und E auf die Möglichkeit einer effizienten Speicherung Rücksicht genommen. Betrachten wir nun als Linearisierung den Bitvektor

$$\langle T \rangle := \Big(\text{bin}\big(w(1)\big), \text{bin}\big(w(2)\big), \dots, \text{bin}\big(w(m-1)\big) \Big) \ ,$$

so ist die Struktur von T implizit klar. Schließlich gilt nach Konstruktion für die Kinder eines Knotens w, dass diese $2v$ und $2v+1$ sind. Achtet man nun noch darauf, dass jeder Abschnitt der Bitfolge mit führenden Nullen bis zu einer Größe von Δ_f aufgefüllt wird, so ist eine Auswertung von $w(v)$ durch das Auslesen der Bits $(v-1)\cdot\Delta_f$ bis $v\cdot\Delta_f-1$ direkt möglich. Die Länge von $\langle T\rangle$ ist demnach

$$|\langle T\rangle| = (m-1)\cdot\Delta_f\ .$$

Jede Linearisierung eines komprimierten Verteilungsbaumes wird sich an dieser Größe messen lassen müssen. Ein direktes Übertragen des Ansatzes scheitert nun zumal daran, dass die Eltern/Kind-Zuordnung des i-ten Vektor-Eintrages auf die $2i$-ten, bzw. $2i+1$-ten, Einträge nicht mehr stimmen muss. Würde man versuchen, dies z.B. durch das Einfügen von Positionszeigern abzufangen, so würde doch ein erheblicher zusätzlicher Platzbedarf anfallen. Bezeichnet n die Anzahl der inneren Knoten, so wären dafür etwa $2n\log n$ Bits notwendig. Man kann hier jedoch mit einer etwas ausgefeilteren Speicherung der Struktur von T^* ansetzen.

Dazu veranschauliche man sich zunächst das folgende wohl bekannte Ergebnis, welches noch auf Cayley zurückgeht (vgl. [Cay81]). Zwei alternative Beweise können [Odl84] oder [GS95] entnommen werden.

Satz 4.27 *Es gibt $\frac{1}{n+1}\binom{2n}{n}$ verschiedene Bäume mit n Knoten.*

Aus der Stirling'schen Formel folgt nun, dass $\frac{1}{n+1}\binom{2n}{n} = \frac{1}{\sqrt{2}e^2}2^{2n} + \mathcal{O}(n)$ ist, weswegen zur Speicherung der Struktur eines allgemeinen Baumes mit n Knoten wenigstens $\log\left(\frac{1}{n+1}\binom{2n}{n}\right) = 2n + o(n)$ Bits benötigt werden. Wir werden nun eine Möglichkeit der Linearisierung der Struktur betrachten, die dieses theoretische Optimum erreicht.

Dazu seien die Werte der Knoten von $V^* = \{v_1, v_2, \ldots, v_n\}$ mit $v_i < v_{i+1}$ einfach wie oben als binärer Vektor

$$t := \Big(\text{bin}\big(w(v_1)\big), \text{bin}\big(w(v_2)\big), \ldots, \text{bin}\big(w(v_n)\big)\Big)$$

gegeben. Die Binärdarstellung wird auch dabei mit führenden Nullen aufgefüllt. Das Geheimnis zur direkten Ausgabe von $w(v)$ liegt dann in der Bestimmung des Ranges i von v.

Definition 4.28 *Sei $T^* = (V^*, E^*)$ ein komprimierter Verteilungsbaum. Zu einem Knoten $v \in V^*$ bezeichnet*

$$\text{rang}(v) := \#\left\{u \in V^* \,|\, u \le v\right\}$$

den Rang *von v.*

Mit dieser Definition entsprechen dann stets $\text{bin}\big(w(v)\big)$ den Bits von t ab Position $\big(\text{rang}(v)-1\big)\cdot\Delta_f$ bis $\text{rang}(v)\cdot\Delta_f - 1$. Ist $s \in \mathbb{B}^*$ eine geeignete Bitsequenz, um den Rang eines Knotens, also die Struktur innerhalb von t, effizient zu bestimmen, so bildet $\langle T\rangle = (s,t)$ eine geeignete Linearisierung von T.

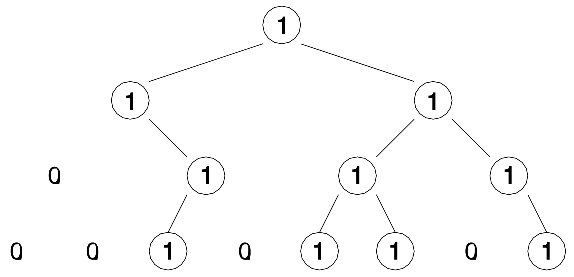


Abbildung 4.4: Bitmuster-Sequenz eines Binärbaumes. Der dargestellte Baum wird ebenenweise zu $s = 1\,11\,0111\,00101101$ codiert.

Howard und Vitter deuten in [HV92] ein Verfahren an, welches die Struktur s eines komprimierten Verteilungsbaumes als einfaches *Bitmuster* (*bitmap*) Ebene für Ebene speichert. Dabei werden im Vergleich zu einem vollständigen Binärbaum vorhandene Knoten als 1 und fehlende Knoten als 0 interpretiert. Die Abbildung 4.4 stellt diese Kodierung dar.

Wird nun der Baum traversiert, so wird die Sequenz einmal von Anfang bis Ende durchlaufen. Beginnend mit $v_0 = 1$ werden ab dem aktuellen Knoten v entweder die nächsten v oder $v + 1$ Bits der Sequenz s gelesen, je nachdem, ob wir zum linken oder rechten Kind wollen. Der Rang des Nachfahren von v entspricht dann dem Rang von v plus der Anzahl der gelesenen Einsen der Teilsequenz.

Dieses Verfahren hat aber gleich zwei Mängel. Zum einen ist die Länge von s stets gleich der Anzahl der Knoten eines vollständigen Baumes mit m Blättern, also $m - 1$, und zwar ungeachtes dessen, wie dünn der Baum in Wirklichkeit gewesen sein mag. Des weiteren werden somit auch stets m Schritte benötigt, um den Baum der Tiefe $\log m$ zu traversieren. Zwar wird in [HV92] noch eine Verbesserung angesprochen, mit welcher bereits nach dem Durchlaufen der Bits der oberen $\log m - 1$ Ebenen ein direktes Anspringen der untersten Ebene möglich ist, dies reduziert jedoch Bit- und Schrittgröße auf lediglich $\frac{1}{2}m$. Wir werden uns deswegen einer anderen Grundidee zuwenden.

Jacobson schlägt in seiner Dissertation [Jac89b] eine Kodierung der Struktur vor, welche bereits früher von Zaks eingeführt wurde (vgl. [Zak80]). Diese bildet einen Baum auf die sogenannte Zaks-Sequenz ab, bei welcher jedem Knoten imaginäre Blätter angehängt werden, so dass der Baum anschliessend regulär ist. Danach werden die originalen Knoten als 1 und die imaginären als 0 interpretiert und Ebene für Ebene hintereinander gehängt. Als Beispiel dient Abbildung 4.5. Obwohl bei den hier gewählten kleinen Bäumen die eingangs betrachtete Bitmuster-Sequenz kürzer zu sein scheint ist leicht ersichtlich, dass die Nullen am Ende der Zaks-Sequenz und damit die unterste Ebene der imaginären Blätter nicht wirklich benötigt werden. Die verkürzte Sequenz wäre damit auf jeden Fall nicht größer als das naive Bitmuster.

Da der imaginär erweiterte Baum regulär ist, ergibt sich für einen Baum mit insgesamt n Knoten nach Folgerung 4.4 eine Sequenz der Länge $2n + 1$. Die Kodierung der Struktur ist somit asymptotisch optimal. Jacobson formuliert nun in [Jac89a] eine Strategie zur Traversierung, welche in $\mathcal{O}(\log^2 m)$ Schritten den Baum von der Wurzel bis zu einem Blatt durchläuft und dabei $o(n)$ zusätzlicher Bits an Information bedarf. Diese Zusatzinforma-

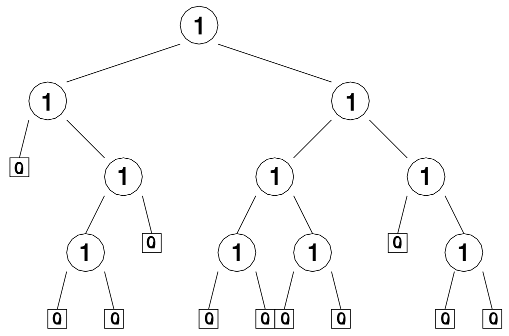


Abbildung 4.5: Zaks-Sequenz eines Binärbaumes. Der dargestellte Baum wird ebenenweise zu $s = 1\,11\,0111\,101101\,00000000$ codiert.

tion stellt ein sogenanntes *Verzeichnis* (*directory*) dar, welches bei gegebenen Knoten v in $\mathcal{O}(\log m)$ Schritten $\mathrm{rang}(v)$ bestimmt. Die Funktionsweise des Verzeichnisses kann in [Jac89b] nachgelesen werden.

Diese zusätzlichen $\mathcal{O}(\log m)$ Schritte zur Rangbestimmung stammen im wesentlichen von einer binären Suche her. Die Konstruktion des Verzeichnisses macht es leider nicht möglich, von diesem sequentiellen Suchansatz abzurücken. Munro und Raman lösen dieses Problem in [MR97], indem sie ihre Ergebnisse zu balancierten Klammerstrukturen (balaced parantheses) durch einen geeigneten Isomorphismus auf binäre Bäume übertragen. Demnach benötigt ein Traversierungsschritt zwar noch immer $\mathcal{O}(\log m)$ einzelne Bitoperationen; erlaubt die Implementierung aber etwa wie in Hardware das simultane Auslesen von $\mathcal{O}(\log m)$ konsekutiven Bits, so kann der gesuchte Nachfahre in konstanter Zeit bestimmt werden.

Somit ist es also insgesamt möglich, trotz der kompakten Speicherung von T^* als $\langle T^* \rangle = (s, t)$, den komprimierten Verteilungsbaum wie gehabt in Zeit $\mathcal{O}(\log m)$ von der Wurzel bis zu einem Blatt zu durchlaufen. Bezeichnet dabei — entsprechend Satz 4.15 — k die Anzahl der von 0 verschiedenen Symbolhäufigkeiten, so folgt für die Größe der Linearisierung

$$|\langle T^* \rangle| = \big(\Delta_f + 2\big) \cdot n + o(n) ,$$

wobei

$$\log \frac{m}{2k} + k \leq n \leq k \log \frac{2m}{k} - 1 .$$

4.6 Fazit

Wir haben die Idee von Howard und Vitter aufgegriffen und die arithmetische Kodierung für beliebige Alphabete mittels binärer arithmetischer Kodierung realisiert. Wir haben dazu den Begriff der Verteilungsbäume spezifiziert und gezeigt, wie man diese in bezüglich der Alphabetgröße linearer Zeit konstruieren kann. Darüber hinaus konnten wir beweisen, dass die beiden Kodierungen die gleiche Codewort-Länge erwarten lassen.

Wir haben ferner mit den komprimierten Verteilungsbäumen die Möglichkeiten der platzeffizienten Realisierung dieses Ansatzes untersucht. Wir konnten dabei die obere

Schranke zur Knotenanzahl von Howard und Vitter beweisen und analog eine untere Schranke einführen. Wir haben gezeigt, dass diese Schranken „eng“ sind; d.h., dass es insbesondere im Falle von Zweierpotenzen komprimierte Verteilungsbäume gibt, bei denen die Gleichheit mit der Schranke gilt.

Wie aus den Abbildungen 4.6 und 4.7 ersichtlich ist, lassen sich jedoch bei wenig schlanken, komprimierten Verteilungsbäumen (etwa $k \geq \frac{1}{2}m$) — wenn überhaupt — nur geringe Einsparungen gegenüber den einfachen Verteilungsbäumen erwarten. Ein echtes Einsparpotential wird erst bei dünn besetzten Wahrscheinlichkeitsverteilungen deutlich (z.B. $k \leq \frac{1}{4}m$). Natürlich muss man sich hier a priori die Frage stellen, ob eine Einkürzung des Alphabetes Σ nicht sinnvoller sei. Jedoch gerade bei stark kontextabhängigen Modellierungen wie Markov-Ketten oder PPM, die zwar das volle Alphabet benötigen, aber auch über dünn besetzte Kontext-Verteilungen verfügen, kann diese Technik eine interessante Option darstellen.

Mit der Methode von Jacobson, beziehungsweise Munro und Raman, lassen sich diese Bäume effizient speichern, so dass sich die Schrittanzahl bei der Kodierung mit Verteilungsbäumen von der binären arithmetischen Kodierung nur um einen konstanten Faktor unterscheidet. Dies kann bei zeitkritischen Anwendungen jedoch bereits ein ausschlaggebendes Kriterium sein. Wenn etwa bei einer Hardware-Implementierung auf Smartcards statt eines Dekodierungsschrittes auf Σ nun iterativ $\log(\# \Sigma)$ Schritte auf dem binären Alphabet verlangt werden, so kann dies mitunter aus Gründen der Laufzeit und des Energiebedarfs nicht vertretbar sein.

	$k=1$	$\ldots$	$k=\frac{m}{16}$	$k=\frac{m}{8}$	$k=\frac{m}{4}$	$k=\frac{m}{2}$	$k=m$
obere Schranke: $k \cdot \log \frac{2m}{k} - 1$	$\log m$	$\ldots$	$\frac{5}{16}m-1$	$\frac{1}{2}m-1$	$\frac{3}{4}m-1$	$m-1$	$m-1$
untere Schranke: $\log \frac{m}{2k} + k$	$\log m$	$\ldots$	$\frac{1}{16}m+3$	$\frac{1}{8}m+2$	$\frac{1}{4}m+1$	$\frac{1}{2}m$	$m-1$

Abbildung 4.6: Einige Platzschranken für komprimierte Verteilungsbäume

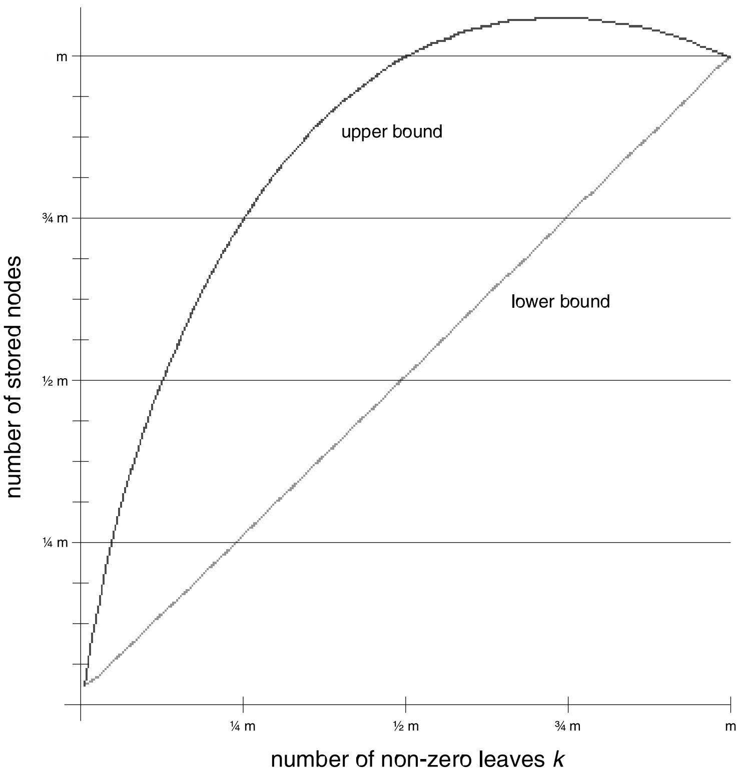


Abbildung 4.7: Obere und untere Schranke der Anzahl gespeicherter Knoten

5 Konzept eines Dekodierschaltkreises mit Tabellen-Arithmetik

„Aw, people can come up with statistics to prove anything, Kent. Fourteen percent of all people know that.“

— Homer J. Simpson

Eines der Kernprobleme bei der Implementierung von arithmetischer Kodierung in Hardware ist der stark iterative Charakter des Verfahrens. So kann mit der Dekodierung des nächsten Symbols erst dann begonnen werden, wenn das Endintervall des aktuellen Symbols bestimmt ist. Verschärft wird dieses Verhalten in dem Ansatz von Witten, Neal und Cleary noch dadurch, dass zu diesem Zweck strikt sequentiell eine Suche nach dem jeweiligen Unterintervall und ein wiederholtes Expandieren desselben von Nöten sind. Das Verfahren scheint daher zumindest problematisch, wenn nicht gar für die Realisierung in Hardware ungeeignet zu sein.

Die Tabellen-Arithmetik nach Howard und Vitter stellt diesbezüglich eine Verbesserung dar, da das jeweilige Folgeintervall in nur einer Tabellenauswertung bestimmt werden kann. Jedoch gibt es zunächst zwei Schwachstellen. Zum einen ist auch hier eine Suche nach dem richtigen Zustand und der passenden Aufteilung des Intervalls innerhalb der Tabelle notwendig. Des Weiteren beziehen sich die Tabellen nur auf binäre Alphabete. Würde man das Verfahren mit Hilfe der Verteilungsbäume aus Kapitel 4 einsetzen, so würden durch diese weitere, streng sequentielle Schritte bei der Traversierung des Baumes entstehen.

Wir wollen daher einen Lösungsvorschlag präsentieren, welcher diesen Stolpersteinen aus dem Weg geht, indem zum Einen statt Suchvorgängen nur direkte Zugriffe verwendet werden und zudem die Technik der Arithmetik-Tabellen auf beliebige Alphabete verallgemeinert wird.

Dazu werden wir im ersten Abschnitt die Idee des Schaltkreises kurz skizzieren. In den beiden darauf folgenden Abschnitten beweisen wir, wie sich durch die Verwendung so genannter geordneter, dyadischer Verteilungen eine Reihe von Problemen komplett vermeiden lassen. Danach können wir im vierten Abschnitt eine konkrete Implementierung des Schaltkreis vorschlagen und ihn bezüglich seiner Größe analysieren. Anschließend betrachten wir die Möglichkeiten zur verlustbehafteten Approximation beliebiger Verteilungen durch dyadische Verteilungen. Deren Praxistauglichkeit soll danach in Abschnitt sechs empirisch untersucht werden. Abschließend wollen wir unsere Ergebnisse noch in einem Fazit zusammenfassen.

5.1 Idee des Schaltkreises

Bevor wir die Arbeitsweise des Schaltkreises skizzieren, ist es noch notwendig, die Rahmenbedingungen abzustecken, unter welchen später die Komplexität des Dekodierers analysiert und bewertet werden soll. Wie aus den Betrachtungen der folgenden Abschnitte deutlich wird, ist der Einsatz des vorgestellten Ansatzes leider nur für eine sehr beschränkte Anzahl an Szenarien sinnvoll.

Wichtigste Parameter eines Dekodierers mit Tabellen-Arithmetik sind die Feinheit N des Basisintervalls sowie die Kardinalität m des zu Grunde liegenden Symbolalphabetes Σ. Sowohl die Größenkomplexität des Schaltkreises und die erforderliche Breite der Kommunikationswege (Leiterbahnen), als auch die Größe der zu speichernden Statistik-Daten hängen elementar von diesen Kenngrößen ab.

Dabei wirkt sich eine große Feinheit N sehr stark auf die Komplexität aus. In Abschnitt 5.4 wird noch genauer auf diese Abhängigkeit eingegangen. Aber auch insbesondere ein großes Symbolalphabet und demzufolge ein großes m hat verheerende Folgen; dies nicht zuletzt, weil ein großer Parameter m eine große Feinheit N bedingt. Man vergegenwärtige sich dazu nur, dass die maximal darstellbare Wahrscheinlichkeit einer Verteilung bei der Tabellen-Arithmetik nur $1 - \frac{m-1}{N}$ beträgt, da wir eine Null-Wahrscheinlichkeit eines Symbols nicht gestatten wollen.

Aus diesem Grund konzentrieren sich die Betrachtungen auf einen konkreten Anwendungsfall und stellen lediglich die Tendenzen bei abweichenden Kenngrößen dar. Wir wollen dazu annehmen, dass dem Kodierer ein Symbolalphabet mit $2 \leq m \leq 16$ Symbolen zu Grunde liegt. Interessant ist hier vor allem das Alphabet $\{0, 1, 2, \ldots, 9, a, \ldots, f\}$ der Hexadezimal-Symbole mit $m = 16$. Als Feinheit wählen wir zunächst $N = 128$, werden aber auch davon abweichende Werte wie $N = 32$, $N = 64$ oder gar $N = 256$ in Betracht ziehen.

Die Konzentration auf eben diese Kenndaten wird aus den beobachteten Werten und Abhängigkeiten gerechtfertigt, welche im Rahmen des Projektes „Online Codekompression für Smartcards“ in Kooperation mit der Infineon Technologies AG ermittelt wurden. Eine Allgemeingültigkeit kann hier natürlich nicht unterstellt werden.

Der hier vorgestellte Ansatz verwendet im Grunde die Tabellen-Arithmetik von Ho-

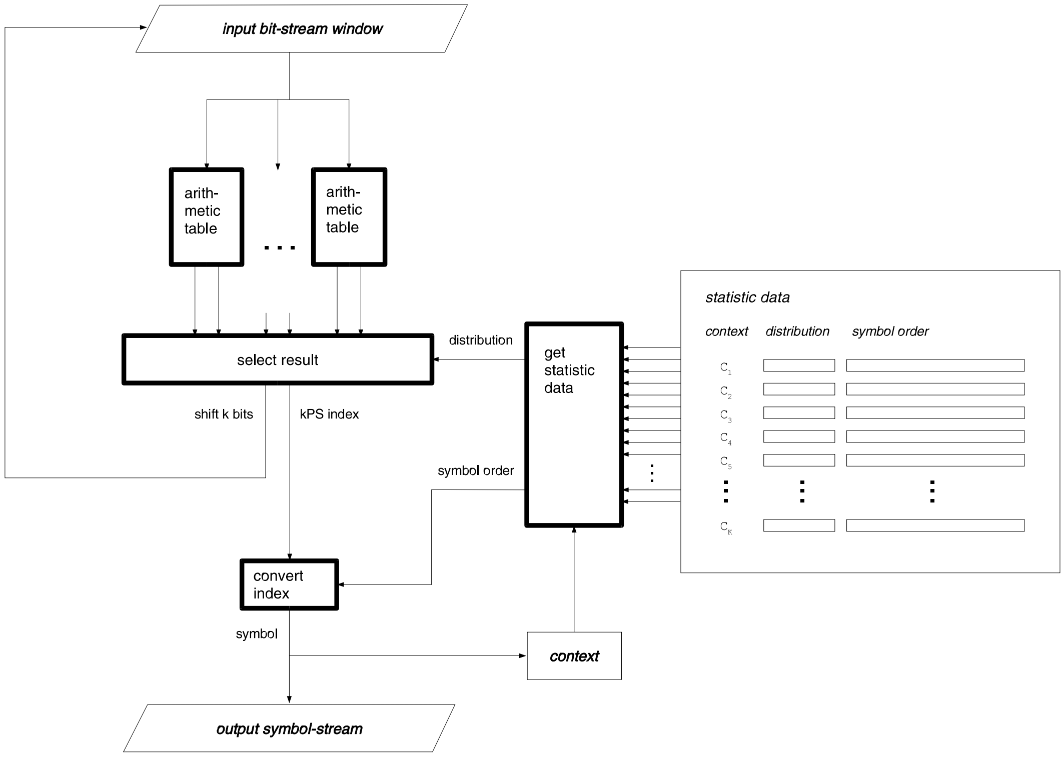


Abbildung 5.1: Skizze eines Dekodierschaltkreises mit Tabellen-Arithmetik

ward und Vitter. Die Verringerung der benötigten Taktzyklen soll dabei durch eine starke Parallelisierung der Berechnung in einem tiefeneffizienten Schaltkreis erfolgen. Des Weitern soll eine Vermeidung von Suchvorgängen innerhalb der Tabelle und auf den benötigten Statistik-Daten erreicht werden.

Wir wollen dabei annehmen, dass dem Kodierer ein kontextgesteuertes, semi-adaptives Modell zu Grunde liegt. Zur Dekodierung sind dessen Statistik-Daten Teil der Eingabe. Wir wollen dies in unserem Schaltkreis derart verstehen, dass jedem Kontext C eine bestimmte Verteilung f und eine Permutation π zugeordnet sind. Ob diese Daten aus dem Speicher bezogen werden oder fest in Hardware vorliegen, bleibt dabei von uns unbetrachtet.

Wir wollen ferner davon ausgehen, dass der Schaltkreis selbst in der Lage ist zu erkennen, wann ein entsprechender Datenblock vollständig dekodiert wurde. Am einfachsten ist dies durch eine feste Symbol-Blockgröße vorstellbar; der Schaltkreis würde die Anzahl der dekodierten Symbole einfach mitzählen und nach dem letzten entsprechend die Arbeit einstellen. Ein Dekodieren eines beliebig langen Blockes mit Hilfe eines speziellen Blockende-Symbols, welches bei der arithmetischen Kodierung ebenfalls denkbar ist, wird in dem vorliegenden Ansatz nicht weiter betrachtet, könnte aber ebenso realisiert werden.

Abbildung 5.1 zeigt eine Skizze unseres Schaltkreises. Dieser besteht im Wesentlichen aus den vier aktiven Baugruppen *Arithmetik-Tabellen* (arithmetic tables), *Statistik-Modul* (get statistic data), *Ergebnis-Filter* (select result) und *Index-Konverter* (convert index), auf deren genaue Aufgaben und Eigenschaften im Abschnitt 5.4 näher eingegangen wird.

Dazu kommt noch ein spezielles Register, welches den aktuellen Kontext speichert sowie die Gesamtheit der Statistik-Daten des Dekodierers. Grob gesprochen liegt je Symbol der folgende, logische Ablauf zu Grunde.

1. In den Arithmetik-Tabellen wird, parallel zu jeder möglichen Verteilung, das zum Eingabestrom gehörende Ergebnis bestimmt.

2. Gleichzeitig wählt das Statistik-Modul, in Abhängigkeit des aktuellen Kontextes, eine Verteilung und eine Permutation aus.

3. Gesteuert von der aktuellen Verteilung wird das richtige Ergebnis aus allen Berechnungen der Arithmetik-Tabellen herausgefiltert.

4. Der dekodierte „Index“ wird im Index-Konverter mit Hilfe der Permutation in das dazugehörige Symbol umgewandelt.

5. Damit wird auch der Kontext auf den neuesten Stand gebracht. Die führenden Bits des Eingabestroms werden nicht mehr benötigt und somit entfernt.

Es ist leicht zu erkennen, dass der Dekodierschaltkreis in einigen Punkten von der herkömmlichen Arbeitsweise eines Tabellendekodierers abweicht. Dabei fällt insbesondere auf, dass wir an keiner Stelle mehr von dem aktuellen Intervall oder Zustand sprechen. Ferner wird nicht mehr das Symbol, sondern stellvertretend ein Index kodiert und dekodiert. Wir wollen diese Modifikationen in den folgenden beiden Abschnitten durch die Betrachtung von so genannten geordneten, dyadischen Verteilungen sowohl motivieren als auch rechtfertigen.

5.2 Geordnete Verteilungen

Bevor wir den Schaltkreis detailliert beschreiben, wollen wir zuerst noch einige eher theoretische Betrachtungen anstellen. Wir werden feststellen, dass die Größe unseres Schaltkreises unter anderem sehr stark von der Anzahl der möglichen Intervalle sowie der Anzahl der zu betrachtenden Verteilungen eines Tabellendekodierers abhängt. Unser Ziel wird es nun sein, diese beiden Größen im Vorhinein einzuschränken. Dazu werden wir zunächst die Anzahl an geordneten Verteilungen betrachten, bevor wir im folgenden Abschnitt einige interessante Ergebnisse zu geordneten, dyadischen Verteilungen aufzeigen.

Beginnen wir mit der Anzahl der potentiell zu betrachtenden Wahrscheinlichkeitsverteilungen eines Tabellendekodierers. Wir wollen — aus Gründen der kombinatorischen Handhabbarkeit — die Verteilungen von m Symbolen auf einem Intervall der Länge k als Zeichenketten der Länge k über dem binären Alphabet $\mathbb{B}$ identifizieren.

Dabei entspricht das i-te Segment des k großen Intervalls genau dem i-ten Bit des k langen Wortes. Es soll in dem Wort stets genau dann eine 1 stehen, wenn das jeweilige Segment den Beginn eines neuen Teilintervalls unter der betrachteten Verteilung markiert. Abbildung 5.2 illustriert diese Kodierung. Betrachten wir dazu die folgende Definition.

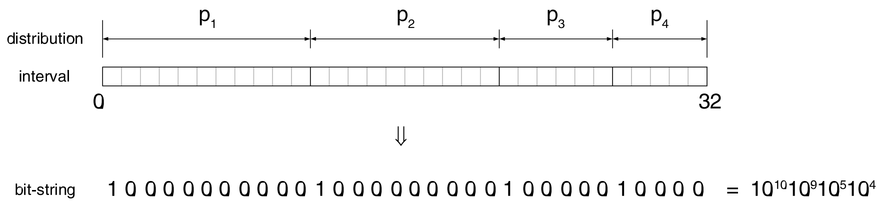


Abbildung 5.2: Kodierung einer Verteilung als binäre Zeichenkette für $k = 32$ und $m = 4$

Definition 5.1 *Die Menge $P_m(k)$ sei definiert durch*

$$P_m(k) \ := \ \{w \in \mathbb{B}^* \,|\, w \textit{ beginnt mit einer } 1,\ |w| = k \textit{ und } h(w) = m\}$$

wobei $h(w)$ das Hamming-Gewicht des Wortes w bezeichnet.

Beobachtung 5.2 *Die Kardinalität der Menge $P_m(k)$ entspricht genau der Anzahl aller möglichen Verteilungen von m Symbolen auf einem Intervall der Länge k.*

Diese Beobachtung sollte einleuchten, denn wenn eine feste Anordnung der Alphabetsymbole gegeben ist, so entspricht jede Bit-Kette aus $P_m(k)$ genau einer möglichen Wahrscheinlichkeitsverteilung, und umgekehrt kann jeder Verteilung eindeutig ein String zugeordnet werden, der in $P_m(k)$ liegt. Betrachten wir zunächst einmal zu gegebenem k die Kardinalität von $P_m(k)$.

Lemma 5.3 *Es gilt für alle $k, m \in \mathbb{N}$ mit $m < k$:*

$$\#\, P_m(k) = \binom{k-1}{m-1}$$

Beweis: Da zu jedem Element aus $P_m(k)$ mit der führenden 1 die erste Position bereits bestimmt ist gilt es nun noch, $m - 1$ weitere Einsen auf die verbleibenden $k - 1$ Plätze zu verteilen. Dies entspricht der Auswahl einer $m - 1$-elementigen Teilmenge (die zu wählenden Positionen für die Einsen) aus einer $k-1$-elementigen Menge. Die Kombinatorik besagt, dass es genau $\binom{k-1}{m-1}$ solche Teilmengen gibt. □

Ein Tabellenkodierer für m Symbole nach dem Prinzip von Howard und Vitter würde zu einem aktuellen Intervall der Länge k jede beliebige Verteilung durch eine Verteilung aus $P_m(k)$ approximieren. Anders ausgedrückt sollte also auch jedes Element aus $P_m(k)$ in der Dekodiertabelle bei Zuständen der Länge k berücksichtigt werden. Hier ist bereits der Zusammenhang zwischen der Komplexität eines Dekodierschaltkreises und der Anzahl der Verteilungen zu erkennen.

Konzentrieren wir unseren Blick nun lediglich auf das Startintervall $[0, N)$ der Länge N, so ist das Ergebnis bereits ernüchternd. Die Kardinalität der Menge $P_m(N)$ ist schon für relativ kleine Parameter N und m unverhältnismäßig groß. Die Tabelle 5.3 gibt ein paar beispielhafte Werte an.

Es zeigt sich also, dass ohne weitere Einschränkungen eine ziemlich große Anzahl an denkbaren Verteilungen von m Symbolen bei einem Tabellendekodierer mit Feinheit N

	$N = 32$	$N = 64$	$N = 128$	$N = 256$
$m = 2$	31	65	127	255
$m = 4$	4495	39711	333375	2731135
$m = 8$	2629575	553270671	$\approx 8.93 \cdot 10^{10}$	$\approx 1.28 \cdot 10^{13}$
$m = 16$	300540195	$\approx 1.22 \cdot 10^{14}$	$\approx 1.17 \cdot 10^{19}$	$\approx 6.30 \cdot 10^{23}$

Abbildung 5.3: Kardinalität der Menge $P_m(N)$

vorkommen. Dies war bei der binären Version aus Kapitel 3 noch kein Problem, da mit $m = 2$ stets $\binom{N-1}{m-1} = N - 1$ gilt. Bei Parametern $N = 128$ und $m = 16$ ist ein Wert von über 10 Trillionen Tabellen-Zeilen jedoch eindeutig indiskutabel.

Wir wollen daher in unserem Dekodierer nicht die ganze Menge $P_m(N)$ betrachten und dennoch alle darin enthaltenen Verteilungen zulassen. Der Trick, den wir dafür heranziehen wollen, liegt darin, dass sich die Reihenfolge der Symbolwahrscheinlichkeiten bei der arithmetischen Kodierung nicht auf die Kompressionsrate auswirkt, so lange nur bei Kodierung und Dekodierung dieselbe Reihenfolge verwendet wird.

Des Weiteren kommt uns entgegen, dass die Arithmetik-Tabellen von Verteilungen, welche sich nicht in den einzelnen Wahrscheinlichkeitsgrößen, sondern nur in deren Reihenfolge unterscheiden, sich völlig analog verhalten. Daher wollen wir fortan nur Arithmetik-Tabellen für einen Repräsentanten dieser äquivalenten Verteilungen in unserem Dekodierer zulassen.

Definition 5.4 *Eine Wahrscheinlichkeitsverteilung $p = (p_1, p_2, \ldots, p_m)$ heißt* absteigend (bzw. aufsteigend) geordnet*, falls stets $p_i \geq p_{i+1}$ (bzw. $p_i \leq p_{i+1}$) gilt. Analog heißt eine Häufigkeitsverteilung f* geordnet*, falls die durch f induzierte Wahrscheinlichkeitsverteilung geordnet ist.*

Wir werden also ab sofort nur noch absteigend geordnete Verteilungen betrachten.[1] Dies entspricht der Interpretation der binären Symbole als MPS und LPS, welche wir bereits in Abschnitt 3.4 eingeführt haben. Der Vorteil dieser Sichtweise liegt auf der Hand, wenn wir deren Anzahl untersuchen. Die Menge dieser Verteilungen lässt sich analog zu Definition 5.1 beschreiben.

Definition 5.5 *Die Menge $Q_m(N)$ sei definiert durch*

$$Q_m(k) := \left\{ w \in \mathbb{B}^* \,\middle|\, w = \underbrace{10\ldots0}_{n_1}\underbrace{10\ldots0}_{n_2}\ldots\underbrace{10\ldots0}_{n_m},\ |w| = k,\ \textit{und}\ n_1 \geq n_2 \geq \ldots \geq n_m \right\}.$$

Beobachtung 5.6 *Die Kardinalität der Menge $Q_m(k)$ entspricht genau der Anzahl aller möglichen geordneten Verteilungen von m Symbolen auf einem Intervall der Länge k. Offensichtlich gilt $Q_m(k) \subseteq P_m(k)$.*

[1] Natürlich kann man für aufsteigend geordnete Verteilung völlig analoge Beobachtungen anstellen und auch dieselben Ergebnisse erzielen.

Eine beliebige Verteilung $x \in P_m(N)$ kann dabei als ein Tupel (y, π) ausgedrückt werden, wobei $y \in Q_m(N)$ und π eine Permutation auf den m Symbolen von x ist, so dass y die absteigend geordnete Verteilung x beschreibt.

Diese Darstellung ist offensichtlich nicht eindeutig. Wenn zwei Symbole unter der Verteilung x gleich wahrscheinlich sind, so gibt es auch wenigstens zwei Permutationen, welche x sortieren würden. Allerdings wird durch ein Paar (y, π) stets genau eine Verteilung beschrieben.

Es stellt sich leider als äußerst schwierig heraus, die Kardinalität von $Q_m(k)$ in geschlossener Form anzugeben. Das folgende Lemma erlaubt uns aber, die Anzahl der Elemente rekursiv zu bestimmen.

Lemma 5.7 *Wir definieren die Zählabbildung $\chi : \mathbb{N}^3 \to \mathbb{N}$ rekursiv wie folgt:*

$$\chi(N, 1, l) := \begin{cases} 0 & \textit{falls } l < N \\ 1 & \textit{falls } l \geq N \end{cases}$$

und für alle $m \geq 2$:

$$\chi(N, m, l) := \sum_{j=\lceil \frac{N}{m} \rceil}^{\min\{l, N-m+1\}} \chi(N - j, m - 1, j).$$

Dann gilt für alle $N, m \in \mathbb{N}$:

$$\# Q_m(N) \;=\; \chi(N, m, N)$$

Beweis: Wir werden zeigen, dass $\chi(N, m, l)$ stets die Anzahl der geordneten Aufteilungen eines Intervalls mit Feinheit N in m Teile, wobei kein Abschnitt größer als l ist, darstellt. Dazu genügt eine vollständige Induktion nach der Anzahl der Abschnitte m.

Sei also zunächst $m = 1$, d.h. wir wollen ein Intervall der Länge N in genau einen Abschnitt kleiner l zerlegen. Dann gibt es genau dann eine Unterteilung, nämlich einfach das ganze Intervall, wenn $N \leq l$ gilt, anderenfalls ist dies unmöglich. Es folgt sofort die Gleichheit zu $\chi(N, 1, l)$ und somit der Induktionsanfang.

Gelte nun zu $m \geq 2$ die Behauptung bereits für $m - 1$. Wir zeigen, dass sie dann auch für m gilt. Betrachten wir zunächst, welche Möglichkeiten wir haben, von einem N großen Intervall das Vorderste zu wählen, so dass es nicht größer als l ist. Bezeichne j dessen Länge. Da es der größte Teilabschnitt ist, darf er auch nicht kleiner als der Durchschnitt sein, also $j \geq \lceil \frac{N}{m} \rceil$. Andererseits muss jedes Teilintervall kleiner l sein, somit auch das größte. Des Weiteren darf es natürlich auch nicht $N - (m - 1)$ übersteigen, da sonst nicht genügend für die anderen $m - 1$ Abschnitte übrig ist, so dass jeder wenigstens Größe 1 hat. Es folgt also $\lceil \frac{N}{m} \rceil \leq j \leq \min\{l, N - (m - 1)\}$.

Ist die Größe j des ersten Teilintervalls gewählt, so beschreibt nach Induktionsvoraussetzung $\chi(N - j, m - 1, j)$ genau die Anzahl der geordneten Unterteilungen auf dem Rest, von denen kein Teilabschnitt länger als j ist. Das bedeutet, dass dies genau die Unterteilungen sind, welche zusammen mit der ersten immer noch der Größe nach geordnet sind. Nach den Betrachtungen zur Wahl von j beschreibt die Summe

$$\sum_{j=\lceil \frac{N}{m} \rceil}^{\min\{l,N-m+1\}} \chi(N-j, m-1, j)$$

also die gesuchte Anzahl und es folgt die Induktionsbehauptung.

Es gilt also, dass $\chi(N, m, l)$ die Anzahl der geordneten Aufteilungen von Intervallen der Größe N in m Teile mit Größenschranke l zählt. Da kein Teilintervall jemals größer als das Ganze sein kann, beschreibt demnach $\chi(N, m, N)$ genau die Kardinalität von $Q_m(N)$. □

Mittels dynamischer Programmierung kann die Rekursion $\chi(N, m, N)$ einfach berechnet werden. Die Tabelle 5.4 zeigt hierzu einige Auswertungen.

	$N = 32$	$N = 64$	$N = 128$	$N = 256$
$m =$ 2	16	32	64	128
$m =$ 4	249	1906	14905	117874
$m =$ 8	919	55974	4602893	461346215
$m =$ 16	231	112540	161410965	$\approx 6.76 \cdot 10^{11}$

Abbildung 5.4: Kardinalität der Menge $Q_m(N)$

Betrachten wir nun die Kardinalität der Menge $Q_m(N)$, so zeigt sich, dass sich die Beschränkung in der Tat lohnt. Für den Fall mit $N = 32$ und $m = 16$ würde mit lediglich 231 geordneten Verteilungen statt etwa 300 Millionen beliebiger Verteilungen nur noch ein geringer Anteil in den Dekodiertabellen für das Intervall $[0, N)$ berücksichtigt werden. Für $N = 128$ und $m = 16$ sehen die Werte zwar ebenfalls deutlich besser aus, die absolute Anzahl an Verteilungen ist aber immer noch zu groß. Deswegen werden wir uns im folgenden Abschnitt um noch eine weitere Verringerung bemühen.

5.3 Dyadische Verteilungen

Wir haben im vorausgegangenen Abschnitt gesehen, wie sich die Anzahl der möglichen Verteilungen ohne Kompressionsverlust drastisch reduzieren lässt. Wir wollen in diesem Abschnitt diesbezüglich noch einen Schritt weiter gehen. Allerdings wird uns dies leider nicht ohne einen echten Verlust in der Kompressionsrate gelingen. Die empirischen Ergebnisse aus Abschnitt 5.6 legen jedoch nahe, dass dieser sich in Grenzen hält.

Wir werden uns dazu auf eine weitere, echte Teilmenge der geordneten Verteilungen einschränken. Die Rede ist von den so genannten geordneten, dyadischen Verteilungen.

Definition 5.8 *Eine Wahrscheinlichkeitsverteilung $p = (p_1, p_2, \ldots, p_m)$ heißt* dyadisch*, falls stets ein $l_i \in \mathbb{N}_0$ existiert, so dass $p_i = 2^{-l_i}$ gilt. Analog heißt eine Häufigkeitsverteilung f* dyadisch*, falls die durch f induzierte Wahrscheinlichkeitsverteilung dyadisch ist.*

*Zu $L \in \mathbb{N}$ sprechen wir von einer L-*beschränkt dyadischen Verteilung*, falls für alle i stets $l_i \leq L$ gilt.*

Wir werden ausschließlich Verteilungen für einen Tabellendekodierer mit einer Feinheit $N = 2^k$ betrachten, welche ein Zweierpotenz ist. In diesem Fall läßt sich auch eine leicht abgewandelte Charakterisierung für beschränkt dyadische Häufigkeitsverteilungen geben.

Lemma 5.9 *Sei $N = 2^k$ eine Zweierpotenz. Eine Häufigkeitsverteilung $f = (f_1, f_2, \ldots, f_m)$ mit $\sum_{i=1}^{m} f_i = N$ ist genau dann k-beschränkt dyadisch, wenn alle Häufigkeiten Zweierpotenzen sind, also stets ein $k_i \in \mathbb{N}_0$ existiert mit $f_i = 2^{k_i}$.*

Beweis: Der Zusammenhang folgt sofort aus der Definition mit $l_i := k - k_i \leq k$. □

Wir wollen, wie im Abschnitt zuvor, die Anzahl an geordneten, k-beschränkten dyadischen Verteilungen eines Intervalls der Größe 2^k bestimmen. Dazu kodieren wir auch diese Verteilungen als Bitfolge.

Definition 5.10 *Die Menge $R_m(k)$ sei definiert durch*

$$R_m(k) \ := \left\{ w \in \mathbb{B}^* \ \middle| \ \begin{array}{c} w = \underbrace{10\ldots0}_{n_1}\underbrace{10\ldots0}_{n_2}\ldots\underbrace{10\ldots0}_{n_m} \ \textit{mit } |w| = k,\ n_1 \geq n_2 \geq \ldots \geq n_m \\ \textit{und } \forall i \in \{1\ldots m\} \ \exists k_i \in \mathbb{N}_0 \ \textit{mit } n_i = 2^{k_i} \end{array} \right\}.$$

Beobachtung 5.11 *Die Kardinalität der Menge $R_m(2^k)$ entspricht genau der Anzahl aller möglichen geordneten, k-beschränkt dyadischen Verteilungen von m Symbolen auf einem Intervall der Länge 2^k. Offensichtlich gilt $R_m(2^k) \subseteq Q_m(2^k) \subseteq P_m(2^k)$.*

Die obige Beobachtung gilt im Allgemeinen nicht für Intervalle, deren Länge keine Zweierpotenz ist. In diesem Fall könnten alle n_i, also alle Häufigkeiten, durchaus Zweierpotenzen sein, ohne dass die induzierten Wahrscheinlichkeiten dyadisch sind.

Auch für $R_m(2^k)$ scheint die Bestimmung der Kardinalität in geschlossener Form schwierig zu sein. Eine alternative Sichtweise, die wir in Abschnitt 5.5 noch ausführlicher bemühen wollen, betrachtet eine dyadische Verteilung $q = \left(2^{-l_1}, 2^{-l_2}, \ldots, 2^{-l_m}\right)$ als regulären Binärbaum mit m Blättern, welche jeweils auf den Tiefen $l_1, l_2, \ldots, l_m$ hängen. Selbst in der Literatur, welche sich kombinatorisch mit diesem Thema beschäftigt, existiert für deren Anzahl offenbar keine exakte Formel (vgl. [LLW86]). Lediglich eine approximative Schranke kann aus [FO84] abgeleitet werden.

Für die kleinen, von uns betrachteten Kenndaten genügt es aber auch hier, die Kardinalität über eine Rekursion auszurechnen.

Lemma 5.12 *Wir definieren die Zählabbildung $\rho : \mathbb{N}^3 \to \mathbb{N}$ rekursiv wie folgt:*

$$\rho(N, 1, l) := \begin{cases} 0 & \textit{falls } N \textit{ keine Zweierpotenz} \\ 0 & \textit{falls } l < N \\ 1 & \textit{sonst} \end{cases}$$

und für alle $m \geq 2$:

$$\rho(N,m,l) := \sum_{j=\lceil \log \lceil \frac{N}{m} \rceil \rceil}^{\min\{\lfloor \log l \rfloor, \lfloor \log N-m+1 \rfloor\}} \rho(N-2^j, m-1, 2^j)$$

Dann gilt für alle Zweierpotenzen $N = 2^k$ und $m \in \mathbb{N}$:

$$\# R_m(2^k) \;=\; \rho(2^k, m, 2^k)$$

Beweis: Zum Beweis werden wir zeigen, dass $\rho(N,m,l)$ stets genau der Anzahl der geordneten m-Tupel $f = (f_1, f_2, \ldots, f_m) \in \mathbb{N}^m$ entspricht, bei welchen $\sum_{i=1}^{m} f_i = N$ gilt und alle f_i Zweierpotenzen kleiner oder gleich l sind. Dazu führen wir eine vollständige Induktion nach m.

Betrachte den Fall $m = 1$. Dann gibt es mit $f = (N)$ genau dann nur dieses eindeutige 1-Tupel, wenn $N < l$ gilt und N eine Zweierpotenz ist. Anderenfalls kann es ein solches f nicht geben. Es gilt also die Gleichheit mit $\rho(N,1,l)$ und somit der Fall $m = 1$.

Sei nun vorausgesetzt, dass bei $m \geq 2$ die Behauptung für $m-1$ gilt. Wir wollen dann zeigen, dass sie auch für m gilt. Betrachten wir zunächst, welche Möglichkeiten sich für den ersten und, auf Grund der Anordnung von f, größten Tupeleintrag ergeben. Da dieser natürlich nicht kleiner als der Durchschnitt aller Einträge sein kann, ist er also größer oder gleich $\lceil \frac{N}{m} \rceil$. Dann ist 2^j, wobei $j \in \mathbb{N}$ minimal bezüglich $2^j \geq \lceil \frac{N}{m} \rceil$ ist, die kleinste Möglichkeit für den ersten Eintrag. Andererseits soll auch $2^j \leq l$ gelten. Ferner muss $2^j \leq N - (m-1)$ sein, da sonst nicht genug für den Rest übrig bleibt, so dass jeder Eintrag mindestens 1 ist. Insgesamt sind also alle Zweierpotenzen 2^j mit $\lceil \frac{N}{m} \rceil \leq 2^j \leq \min\{l, N-m+1\}$ möglich. Anders ausgedrückt können wir genau die Exponenten j wählen, für die $\log \lceil \frac{N}{m} \rceil \leq j \leq \log\bigl(\min\{l, N-m+1\}\bigr)$ gilt. Da $j \in \mathbb{N}$, sind somit Werte zwischen $\lceil \log \lceil \frac{N}{m} \rceil \rceil$ und $\min\bigl\{\lfloor \log l \rfloor, \lfloor \log N-m+1 \rfloor\bigr\}$ möglich.

Steht der erste Eintrag mit 2^j fest, so gilt es noch die restlichen $N-2^j$ geordnet zu auf m-1 Einträge zu verteilen, so dass kein Wert größer als $2^j \leq l$ ist. Nach Induktionsvorrausetzung beschreibt $\rho(N-2^j, m-1, 2^j)$ genau die Anzahl der dazu möglichen $(m-1)$-Tupel. Addiert man diese nun über alle legalen 2^j auf, so folgt die Summe aus der Behauptung.

Somit beschreibt also $\rho(N,m,l)$ die Anzahl der geordneten Aufteilungen von Intervallen der Größe N in m Teile mit Größenschranke l, so dass jeder Abschnitt eine Zweierpotenz ist. Da kein Teilabschnitt größer als das ganze Intervall sein kann, gibt demnach $\rho(2^k, m, 2^k)$ genau die Kardinalität von $R_m(2^k)$ an. □

Auch diese Berechnung kann mit dynamischer Programmierung durchgeführt werden. Abbildung 5.5 zeigt einige dieser Werte.

Beeindruckenderweise ist die Kardinalität von $R_m(N)$ nicht sehr groß. Eine Approximation beliebiger Verteilungen durch geordnete, dyadische Verteilungen würde somit der Größeneffizienz eines Tabellendekodierers sehr zugute kommen. Abschnitt 5.5 betrachtet einige Möglichkeiten dazu, eine geeignete dyadische Verteilung zu bestimmen.

Doch fast noch wichtiger als deren Anzahl sind die Eigenschaften, welche ein arithmetischer Kodierer bei dyadischen Verteilungen aufweist. Wir werden sehen, dass wir es

	$N = 32$	$N = 64$	$N = 128$	$N = 256$
$m = 2$	1	1	1	1
$m = 4$	2	2	2	2
$m = 8$	10	15	16	16
$m = 16$	12	59	181	406

Abbildung 5.5: Kardinalität der Menge $R_m(N)$

nur noch mit dem Startintervall zu tun haben und darüber hinaus follow-on Sequenzen vollständig vermeiden. Diese Eigenschaften werden die effiziente Implementierung des Dekodierers in Hardware erst ermöglichen.

Satz 5.13 *Sei ein Tabellendekodierer für m Symbole mit Feinheit $N = 2^k$ gegeben, welcher nur geordnete, k-beschränkt dyadische Verteilungen verwendet. Dann gilt:*

a) Am Ende eines Dekodierschrittes wird stets zum Startintervall $[0, N)$ expandiert.

b) Es treten keine follow-on Sequenzen auf.

Bevor wir uns dem Beweis dieses wichtigen Satzes nähern können, müssen wir aber zunächst die Unterteilungs- und Expansionsschritte des arithmetischen Kodierers etwas genauer kennen lernen. Das folgende Lemma formuliert eine interessante Erkenntnis über die Unterteilung mittels geordneter, dyadischer Verteilungen.

Lemma 5.14 *Sei $N = 2^k$ und $f = \left(2^{k_1}, 2^{k_2}, \ldots, 2^{k_m}\right)$ eine absteigend geordnete, dyadische Häufigkeitsverteilung mit $\sum_{i=1}^{m} 2^{k_i} = 2^k$ für ein $k \in \mathbb{N}$. Gilt nun $k_1 \leq k - \lambda$ für ein $\lambda \in \mathbb{N}$, so existieren Indizes $r_1, r_2, \ldots, r_{2^\lambda} \in \{1, \ldots, m\}$ mit $r_1 < r_2 < \ldots < r_{2^\lambda}$, so dass für alle $j \in \left\{1, \ldots, 2^\lambda\right\}$ gilt:*

$$\sum_{i=1}^{r_j} 2^{k_i} \;=\; j \cdot 2^{k-\lambda}.$$

Mit anderen Worten: Im Abstand von $2^{k-\lambda}$ definiert f auf $[0, N)$ eine Intervallgrenze.

Beweis: Es gilt zunächst zu bemerken, dass wegen $k_1 \leq k - \lambda$ auch $k_1 + \lambda \leq k$ und somit $2^\lambda \cdot 2^{k_1} = 2^{k_1+\lambda} \leq 2^k$ gilt. Da 2^{k_1} die größte Häufigkeit der Verteilung f ist, sich alle 2^{k_i} jedoch zu 2^k aufsummieren müssen, können wir daraus schließen, dass die Anzahl der Häufigkeiten $m \geq 2^\lambda$ sein muss. Deshalb macht es durchaus Sinn, von 2^λ paarweise verschiedenen Indizes $r_j \in \{1, \ldots, m\}$ zu sprechen.

Wir werden die Aussage induktiv nach $j \in \left\{1, \ldots, 2^\lambda\right\}$ beweisen. Zeigen wir also zuerst die Behauptung für $j = 1$. Falls $k_1 = k - \lambda$, folgt diese trivialerweise mit $r_1 = 1$. Gelte daher im Folgenden $k_1 < k - \lambda$ und somit $2^{k_1} < 2^{k-\lambda}$.

Sei $r \in \mathbb{N}$ maximal gewählt, so dass $\sum_{i=1}^{r} 2^{k_i} < 2^{k-\lambda}$. Dies existiert nach obiger Annahme. Da $\sum_{i=1}^{m} 2^{k_i} = 2^k$, gilt damit insbesondere $r < m$. Betrachten wir nun die Differenz beider Seiten der Ungleichung, so gilt für

$$2^{k-\lambda} - \sum_{i=1}^{r} 2^{k_i} > 0 \ ,$$

dass nach Multiplikation mit $2^{-(k-\lambda)-1}$ jeder Subtrahend kleiner 1, aber größer 0 ist und den Hauptnenner $2^{k_r-(k-\lambda)-1}$ hat, denn schließlich gilt $k-\lambda > k_i \geq k_r$ für alle $i = 1 \ldots r$. Damit folgt

$$2^{-1} - \sum_{i=1}^{r} 2^{k_i-(k-\lambda)-1} \;\geq\; 2^{k_r-(k-\lambda)-1}\,,$$

und nach weiterer Multiplikation mit $2^{k-\lambda+1}$

$$2^{k-\lambda} - \sum_{i=1}^{r} 2^{k_i} \;\geq\; 2^{k_r}.$$

Wenden wir unseren Blick nun auf die Häufigkeit des $(r+1)$-ten Symbols. Da $r < m$, muss ein solches in f existieren. Angenommen, es gelte $2^{k_{r+1}} < 2^{k-\lambda} - \sum_{i=1}^{r} 2^{k_i}$. Dann wäre auch $\sum_{i=1}^{r+1} 2^{k_i} < 2^{k-\lambda}$, und es folgt der Widerspruch zur maximalen Wahl von r. Würde hingegen $2^{k_{r+1}} > 2^{k-\lambda} - \sum_{i=1}^{r} 2^{k_i}$ gelten, so folgt $2^{k_{r+1}} > 2^{k_r}$, also $k_{r+1} > k_r$, was im Widerspruch zu der Ordnung der Verteilung f steht.

Demnach muss also $2^{k_{r+1}} = 2^{k-\lambda} - \sum_{i=1}^{r} 2^{k_i}$ sein, und es folgt die Behauptung für $j = 1$ mit $r_1 = r + 1$.

Sei im Folgenden die Behauptung für ein $j < 2^{\lambda}$ als wahr vorausgesetzt. Wir wollen dann die Aussage für $j+1$ beweisen. Falls $\sum_{i=1}^{r_j+1} 2^{k_i} = (j+1) \cdot 2^{k-\lambda}$, so folgt diese sofort mit $r_{j+1} = r_j + 1$, und wir sind fertig. Sei also $\sum_{i=1}^{r_j+1} 2^{k_i} < (j+1) \cdot 2^{k-\lambda}$.

Sei nun $r \in \mathbb{N}$ mit $r > r_j$ maximal bezüglich $\sum_{i=1}^{r} 2^{k_i} < (j+1) \cdot 2^{k-\lambda}$. Ein solches r existiert nach der vorausgegangenen Überlegung. Da somit auch $\sum_{i=1}^{r} 2^{k_i} < 2^{k}$ ist, gilt insbesondere $r < m$. Betrachten wir auch hier die Differenz beider Seiten der Ungleichung, so gilt nach Induktionsvoraussetzung

$$\begin{aligned} 0 &< (j+1) \cdot 2^{k-\lambda} - \sum_{i=1}^{r} 2^{k_i} \\ &= 2^{k-\lambda} + \underbrace{j \cdot 2^{k-\lambda} - \sum_{i=1}^{r_j} 2^{k_i}}_{=0 \text{ nach IV}} - \sum_{i=r_j+1}^{r} 2^{k_i} \\ &= 2^{k-\lambda} - \sum_{i=r_j+1}^{r} 2^{k_i}. \end{aligned}$$

Mit dem gleichen Hauptnenner-Argument wie im Fall $j = 1$ folgt somit

$$2^{k-\lambda} - \sum_{i=r_j+1}^{r} 2^{k_i} \;\geq\; 2^{k_r}.$$

Analog zum Fall $j = 1$ genügt es nun auch hier das $(r+1)$-te Symbol zu betrachten. Dieses muss existieren, da $r < m$. Nehmen wir an, dass $2^{k_{r+1}} < 2^{k-\lambda} - \sum_{i=r_j+1}^{r} 2^{k_i}$ gelten würde, so folgt $\sum_{i=r_j+1}^{r+1} 2^{k_i} < 2^{k-\lambda}$ und damit

$$\begin{aligned}\sum_{i=1}^{r+1} 2^{k_i} &= \sum_{i=1}^{r_j} 2^{k_i} + \sum_{i=r_j+1}^{r+1} 2^{k_i} \\ &\overset{\text{IV}}{=} j \cdot 2^{k-\lambda} + \sum_{i=r_j+1}^{r+1} 2^{k_i} \\ &< j \cdot 2^{k-\lambda} + 2^{k-\lambda} \\ &= (j+1) \cdot 2^{k-\lambda}.\end{aligned}$$

Dies steht aber im Widerspruch zur Maximalität von r und kann daher nicht sein. Andererseits impliziert $2^{k_{r+1}} > 2^{k-\lambda} - \sum_{i=r_j+1}^{r} 2^{k_i}$, dass $2^{k_{r+1}} > 2^{k_r}$ und somit $k_{r+1} > k_r$. Dies widerspricht der Anordnung der k_i.

Also folgt $2^{k_{r+1}} = 2^{k-\lambda} - \sum_{i=r_j+1}^{r} 2^{k_i}$ und damit $\sum_{i=1}^{r+1} 2^{k_i} = (j+1) \cdot 2^{k-\lambda}$, was die Behauptung mit $r_{j+1} = r+1$ für $j+1$ beweist. Auf Grund der Konstruktion mit $r_{j+1} = r_j + 1$ (trivialer Fall) oder $r_{j+1} = r+1 > r_j + 1$ ist damit auch gezeigt, dass stets $r_j < r_{j+1}$ gilt. □

Folgerung 5.15 *Sei $N = 2^k$. Mit den Voraussetzungen aus Lemma 5.14 gilt, solange $f \neq (N)$, dass es eine Intervallgrenze bei $\frac{1}{2}N$ geben muss.*

Beweis: Da f nicht trivial ist, gilt $k_1 < k$. Wende Lemma 5.14 mit $\lambda = 1$ und $j = 1$ an. Dann existiert ein Index r_1, so dass $\sum_{i=1}^{r_1} 2^{k_i} = 2^{k-\lambda} = 2^{k-1}$ gilt. □

Nach dieser Einsicht über die Unterteilungsschritte bei dyadischen Verteilungen wenden wir uns nun der Expansion des aktuellen Intervalls zu. Es stellt sich die Frage, ob die obige Eigenschaft auch nach einem oder mehreren Expansionsschritten erhalten bleibt. Dazu betrachte man zunächst das folgende Lemma.

Lemma 5.16 *Gegeben sei ein Tabellendekodierer für m Symbole mit Feinheit $N = 2^k$, welcher nur absteigend geordnete, dyadische Verteilungen verwendet. Sei darüber hinaus $[a, b) \subset [0, N)$ das aktuelle Intervall, welches aus einem Unterteilungsschritt des Startzustandes $[0, N)$ hervorgegangen ist. Dann existiert nach dem ersten Expansionsschritt von $[a, b)$ in $[\alpha, \beta)$ eine absteigend geordnete, dyadische Häufigkeitsverteilung $g = \left(2^{k_1}, 2^{k_2}, \ldots, 2^{k_\mu}\right)$ mit $\sum_{i=1}^{\mu} 2^{k_i} = 2^k$ für μ Symbole und ein $r \in \{1, \ldots, \mu\}$, so dass*

$$\alpha = \sum_{i=1}^{r} 2^{k_i} \quad \textit{und} \quad \beta = \sum_{i=1}^{r+1} 2^{k_i}.$$

Mit anderen Worten: Auch $[\alpha, \beta)$ könnte eine Unterteilung durch eine geordnete, dyadische Verteilung sein.

Beweis: Es gilt $[a, b) \neq [0, N)$, denn sonst erfüllt bereits die Verteilung $g = (N)$ mit $\mu = 1$ die Behauptung trivialerweise. Bezeichne $f = \left(2^{k_1}, \ldots, 2^{k_m}\right)$ die geordnete Häufigkeitsverteilung, welche das Basisintervall $[0, N)$ in $[a, b)$ unterteilt hat. Es existiert somit ein $s \in \mathbb{N}$ mit $\sum_{i=1}^{s} 2^{k_i} = a$ und $\sum_{i=1}^{s+1} 2^{k_i} = b$.

Da laut Folgerung 5.15 nach dem Unterteilungsschritt eine Intervallgrenze auf $\frac{1}{2}N$ liegt, existiert auch ein $t \in \mathbb{N}$ mit $\sum_{i=1}^{t} 2^{k_i} = \frac{1}{2}N$. Weil somit kein Teilintervall, also auch nicht $[\alpha, \beta)$, über $\frac{1}{2}N$ sattelt, wird im ersten Expansionsschritt entweder eine untere oder obere Expansion auf $[\alpha, \beta)$ angewendet. Wir werden nun in einer Fallunterscheidung für jede dieser beiden Möglichkeiten eine dyadische Häufigkeitsverteilung g sowie die Symbolanzahl μ angeben.

Betrachten wir zunächst den Fall, dass eine untere Expansion $[a, b)$ in $[\alpha, \beta)$ überführt hat. Dann gilt $\alpha = 2a$ und $\beta = 2b$. Sei $\mu := t$ definiert. Wir definieren für alle $i = 1 \ldots \mu$ die Verteilung g durch $g_i = 2 \cdot 2^{k_i} = 2^{k_i+1}$. Da f geordnet war, ist auch g geordnet. Damit gilt:

$$\alpha = 2a = 2 \cdot \sum_{i=1}^{s} 2^{k_i} = \sum_{i=1}^{s} 2^{k_i+1}$$

und

$$\beta = 2b = 2 \cdot \sum_{i=1}^{s+1} 2^{k_i} = \sum_{i=1}^{s+1} 2^{k_i+1} \; .$$

Da

$$\sum_{i=1}^{\mu} g_i = \sum_{i=1}^{t} 2^{k_i+1} = 2 \cdot \sum_{i=1}^{t} 2^{k_i} = 2 \cdot \frac{1}{2}N = N$$

folgt mit $r = s$ die Behauptung.

Wenden wir uns nun dem Fall zu, dass eine obere Expansion vorgelegen hat. Somit gilt $\alpha = 2a - N$ und $\beta = 2b - N$. Definiere $\mu := m - t$ und für alle $i = 1 \ldots \mu$ die Verteilung g durch $g_i = 2 \cdot 2^{k_{i+t}} = 2^{k_{i+t}+1}$. g erbt damit die Anordnung von f. Es folgt:

$$\begin{aligned}
\alpha = 2a - N = 2 \cdot \sum_{i=1}^{s} 2^{k_i} - N &= 2 \sum_{i=1}^{t} 2^{k_i} + 2 \sum_{i=t+1}^{s} 2^{k_i} - N \\
&= 2 \cdot \frac{1}{2}N + \sum_{i=t+1}^{s} 2^{k_i+1} - N = \sum_{i=t+1}^{s} 2^{k_i+1} = \sum_{i=1}^{s-t} 2^{k_{i+t}+1}
\end{aligned}$$

und

$$\begin{aligned}
\beta = 2b - N = 2 \cdot \sum_{i=1}^{s+1} 2^{k_i} - N &= 2 \sum_{i=1}^{t} 2^{k_i} + 2 \sum_{i=t+1}^{s+1} 2^{k_i} - N \\
&= 2 \cdot \frac{1}{2}N + \sum_{i=t+1}^{s+1} 2^{k_i+1} - N = \sum_{i=t+1}^{s+1} 2^{k_i+1} = \sum_{i=1}^{s-t+1} 2^{k_{i+t}+1} \; .
\end{aligned}$$

Da ferner

$$\begin{aligned}\sum_{i=1}^{\mu} g_i &= \sum_{i=1}^{m-t} 2^{k_{i+t}+1} = 2 \cdot \sum_{i=1}^{m-t} 2^{k_{i+t}} = 2 \cdot \sum_{i=t+1}^{m} 2^{k_i} \\ &= 2 \cdot \left(\sum_{i=1}^{m} 2^{k_i} - \sum_{i=1}^{t} 2^{k_i} \right) = 2 \cdot \left(N - \frac{1}{2} N \right) = 2 \cdot \frac{1}{2} N = N \;,\end{aligned}$$

folgt auch in diesem Fall mit $r = s - t$ die Behauptung. □

Natürlich interessieren wir uns nicht wirklich für die Verteilung g aus dem vorausgegangenen Lemma. Zur Kompression soll selbstverständlich f verwendet werden. Allerdings erlaubt deren reine Existenz eine Verallgemeinerung von Lemma 5.14 und Folgerung 5.15 auf alle Zwischenschritte von der Unterteilung bis zur letzten Expansion eines Intervalls.

Lemma 5.17 *Seien die Voraussetzungen aus Lemma 5.16 gegeben. Dann existiert nach jedem Expansionsschritt für das neue Intervall $[\alpha, \beta)$ eine absteigend geordnete, dyadische Häufigkeitsverteilung $g = \left(2^{k_1}, 2^{k_2}, \ldots, 2^{k_\mu}\right)$ mit $\sum_{i=1}^{\mu} 2^{k_i} = 2^k$ für μ Symbole und ein $r \in \{1, \ldots, \mu\}$, so dass:*

$$\alpha = \sum_{i=1}^{r} 2^{k_i} \quad \textit{und} \quad \beta = \sum_{i=1}^{r+1} 2^{k_i}.$$

Nach jedem Expansionsschritt gilt dann entweder $[\alpha, \beta) = [0, N)$ oder $\alpha \geq \frac{1}{2}N$ oder $\beta \leq \frac{1}{2}N$.

Beweis: Wir beweisen die Behauptung durch vollständige Induktion nach der Anzahl n der Expansionsschritte. Für den Induktionsanfang $n = 1$ genügt es, Lemma 5.16 zu betrachten. Es existiert somit für $[\alpha, \beta)$ eine geordnete, dyadische Verteilung g, welche das Intervall aus $[0, N)$ unterteilt. Falls nun $[\alpha, \beta) \neq [0, N)$, so garantiert Folgerung 5.15, dass eine Intervallgrenze bei $\frac{1}{2}N$ existiert. Demnach kann lediglich $[\alpha, \beta) \subseteq \left[0, \frac{1}{2}N\right)$ oder $[\alpha, \beta) \subseteq \left[\frac{1}{2}N, N\right)$ gelten und es gilt somit entweder $\alpha \geq \frac{1}{2}N$ oder $\beta \leq \frac{1}{2}N$.

Gelte also zunächst die Aussage für n Expansionsschritte. Wir wollen zeigen, dass sie dann ebenfalls nach dem $(n+1)$-ten Schritt gilt. Sei $[a, b)$ das Intervall vor diesem Schritt. Dann muss $[a, b) \neq [0, N)$ gelten, da sonst kein $(n + 1)$-ter Schritt nötig wäre.

Nach Voraussetzung existiert eine geordnete, dyadische Verteilung g, welche das Intervall $[a, b)$ aus $[0, N)$ unterteilt. Mit der Technik aus dem Beweis von Lemma 5.16 lässt sich dann völlig analog eine geordnete, dyadische Verteilung g' für das Intervall $[\alpha, \beta)$ mit der geforderten Eigenschaft konstruieren.

Mit der Verteilung g gilt nach Folgerung 5.15, dass es eine Intervallgrenze unter g bei $\frac{1}{2}N$ geben muss. Die gleiche Argumentation wie im Fall $n = 1$ liefert dann, dass entweder $[\alpha, \beta) = [0, N)$ oder $\alpha \geq \frac{1}{2}N$ oder $\beta \leq \frac{1}{2}N$ gilt. □

Somit haben wir das Verhalten der Unterteilungs- und Expansionsschritte eines Dekodierers mit geordneten, dyadischen Verteilungen ausreichend beschrieben. Der Beweis von Satz 5.13 fällt nun besonders leicht.

Beweis: (von Satz 5.13) Nach dem letzten Expansionsschritt gilt nach 5.17 entweder $[\alpha, \beta) = [0, N)$ oder $\alpha \geq \frac{1}{2}N$ oder $\beta \leq \frac{1}{2}N$. Die letzten beiden Fälle können jedoch nicht eintreten, da dann das Intervall noch expandierbar wäre. Also folgt der erste Teil der Aussage.

Da zudem, solange $[\alpha, \beta) \neq [0, N)$, stets eine Intervallgrenze bei $\frac{1}{2}N$ existiert, wird niemals die mittlere Expansion angewandt. Da lediglich bei mittleren Expansionen follow-on Sequenzen entstehen, gilt somit auch Teil b). □

Im Prinzip haben wir bei dem Übergang zu dyadischen Verteilungen das alleinige Gebiet der arithmetischen Kodierung verlassen. Da stets zum Basisintervall $[0, N)$, also dem Startzustand, expandiert wird, sind die einzelnen Codewörter je Symbol von einander unabhängig. Ohne follow-on Sequenzen kann zudem $C(a)$ für alle $a \in \Sigma$ ohne Berücksichtigung der Nachfolgenden Symbole angegeben werden. Die Kodierung eines Wortes $w = w_1 w_2 \dots w_n \in \Sigma^*$ ist somit einfach die Konkatenation der Codewörter $C(w_i)$. Die entstehenden Code-Fragmente je Symbol könnten also ebenso aus einem beliebigen, anderen kompakten Code C', z.B. der Huffman-Kodierung, auf den Elementen von Σ entstanden sein. Wir haben uns dennoch die Flexibilität des Verfahrens gegenüber wechselnden Verteilungen erhalten.

Die Eigenschaften aus Satz 5.13 sind von elementarer Bedeutung, um die Größenkomplexität unseres Schaltkreises einzuschränken. Die Anzahl der Zustände fällt als Faktor für die Arithmetik-Tabellen nun nicht mehr ins Gewicht. Das folgende Korollar garantiert uns zudem noch eine Begrenzung der benötigten Leiterbahnen.

Korollar 5.18 *Sei ein Tabellendekodierer für m Symbole mit Feinheit $N = 2^k$ gegeben, welcher stets geordnete, k-beschränkt dyadische Verteilungen verwendet. Dann genügen je die führenden $\log N = k$ Bits einer Eingabe, um das nächste Symbol eindeutig zu dekodieren.*

Beweis: Nach dem Unterteilungsschritt aus $[0, N)$ hat das resultierende Intervall wenigstens die Größe 1. Da in jedem Expansionsschritt dessen Größe verdoppelt wird und es letztendlich in dem Basisintervall der Größe N endet, sind maximal $\log N$ Verdopplungen möglich. Jeder Expansionsschritt korrespondiert dabei zu einem Bit des Codewortes. Laut Lemma 3.11 liefert die arithmetische Kodierung einen präfixfreien Code, weswegen die Dekodierung eindeutig ist. □

Mit diesem Hintergrund ausgestattet können wir nun (endlich) die konkrete Konstruktion unseres Schaltkreises angeben.

5.4 Baugruppen und deren Komplexität

Nachdem wir nun zur Genüge die Vorteile der ausschließlichen Verwendung geordneter, dyadischer Verteilungen untersucht haben, wollen wir uns nun den Dekodierschaltkreis im Detail vornehmen. Wir haben dessen prinzipielle Arbeitsweise bereits in Abschnitt 5.1 vorgestellt. Hier noch einmal die einzelnen Arbeitsschritte des Dekodierers.

1. In den Arithmetik-Tabellen wird, parallel zu jeder möglichen Verteilung, das zum Eingabestrom gehörende Ergebnis bestimmt.

2. Gleichzeitig wählt das Statistik-Modul, in Abhängigkeit des aktuellen Kontextes, eine Verteilung und eine Permutation aus.

3. Gesteuert von der aktuellen Verteilung wird das richtige Ergebnis aus allen Berechnungen der Arithmetik-Tabellen herausgefiltert.

4. Der dekodierte Index wird im Index-Konverter mit Hilfe der Permutation in das dazugehörige Symbol umgewandelt.

5. Damit wird auch der Kontext auf den neuesten Stand gebracht. Die führenden Bits des Eingabestroms werden nicht mehr benötigt und somit entfernt.

Nach unseren Ergebnissen aus Satz 5.13 sollte nun klar sein, wieso wir auf die Betrachtung der Intervalle ganz verzichtet haben: Es wird lediglich das Basisintervall $[0, N)$ vorkommen. Ein weiterer wichtiger Aspekt liegt in der Verwendung geordneter Verteilungen. In Anlehnung an die Bezeichnungen des binären Dekodierers wollen wir dabei folgende Sprechweise verwenden.

Notation 5.19 *Sei eine Verteilung f für die Symbole des Alphabetes $\Sigma = \{a_1, a_2, \dots, a_m\}$ gegeben. Dann bezeichne für $k \leq m$ der Ausdruck k*PS *das k-*wahrscheinlichste Symbol *(*k^{th} probable symbol*) von Σ. Wir nennen k auch den* Index *des k-wahrscheinlichsten Symbols. Insbesondere bezeichnet* MPS *das wahrscheinlichste und* LPS *das unwahrscheinlichste Symbol unter der gegebenen Verteilung.*

Da wir statt der originalen Verteilung eine geordnete Verteilung f verwenden, macht es keinen Sinn, von der Häufigkeit eines Symbols $a \in \Sigma$ in f zu sprechen. Deswegen wird in unserem Verfahren statt a der Index von a kodiert, bzw. dekodiert. Diese Zuordnung macht letzten Endes die Permutation, also die Umrechnung vom Index in das ursprüngliche Symbol, notwendig.

Betrachten wir aber nun die vier zentralen Baugruppen im Einzelnen. Bei der Implementierung des Dekodierers in Hardware soll es uns vor allem auf eine tiefeneffiziente Realisierung ankommen. Diese garantiert uns, dass ein Dekodierschritt in nur wenigen Taktzyklen, am besten sogar nur einem Zyklus, durchgeführt wird. Dazu werden wir im Wesentlichen die Ausgabe einer Baugruppe als eine Boole'sche Abbildung der Eingabewerte in disjunktiver Normalform (DNF) angeben.

DNF-Schaltkreise haben mit nur einer Ebene an *AND*-Gattern und einer abschließenden Disjunktion eine für uns optimale Tiefe.[2] Es ist allerdings bekannt, dass diese dann mitunter eine immense Anzahl an Gattern benötigen. Der hier vorgestellte Ansatz macht diesbezüglich keine Ausnahme. Wir werden im Folgenden jedoch feststellen, dass für Kenndaten mit $N = 128$ und $m = 16$, sowie der eingeschränkten Anzahl an Verteilungen, deren Größe stets im Rahmen bleibt.

[2]Wir wollen dabei zur Bestimmung von Tiefe und Größe die eventuelle Negationen der Eingabebits und die Disjunktion der Ausgabe nicht mitzählen.

prefix			next state		shift		output	
0	$\mapsto$	(	$[0,128)$	,	1	,	MPS	)
1000	$\mapsto$	(	$[0,128)$	,	4	,	2PS	)
10010	$\mapsto$	(	$[0,128)$	,	5	,	3PS	)
10011	$\mapsto$	(	$[0,128)$	,	5	,	4PS	)
$\vdots$			$\vdots$		$\vdots$		$\vdots$	
11110	$\mapsto$	(	$[0,128)$	,	5	,	15PS	)
11111	$\mapsto$	(	$[0,128)$	,	5	,	LPS	)

Abbildung 5.6: Beispiel einer einzelnen Dekodiertabelle für $N = 128$ und $m = 16$ zum Zustand $[0, 128)$ und zur Verteilung $\left(\frac{1}{2}, \frac{1}{16}, \frac{1}{32}, \frac{1}{32}, \ldots, \frac{1}{32}, \frac{1}{32}\right)$

Die Bausteine zur Tabellen-Arithmetik sind das Kernstück des Schaltkreises. Die Konstruktion dieser Tabellen folgt dabei denselben Regeln, wie wir sie bereits für zwei Symbole in Abschnitt 3.5 vorgeführt haben. D.h. es werden die Unterteilungs- und Expansionsschritte quasi „von Hand“ vorberechnet und das Ergebnis notiert. Sind dabei der aktuelle Zustand und die Verteilung gegeben, so stellt sich eine Tabellenzeile als Abbildung aus dem jeweiligen Eingabestrom in einem Tripel (I, k, s) dar, wobei I das resultierende nicht-expandierbare Intervall nach allen Expansionsschritten (also den neuen Zustand), k die Anzahl der benötigten Expansionen (und somit auch die Anzahl der Shift-Operationen auf das Eingabe-Fenster) und s das dekodierte Symbol bezeichnen.

Ein Beispiel einer solchen Zuordnung kann Abbildung 5.6 entnommen werden. Zu jeder möglichen Verteilung wird die entstehende Tabelle durch einen eigenen Schaltkreis realisiert. Dieser wertet die kodierte Eingabe aus und liefert das erwähnte Tripel als Ausgabe. Hierbei wird jedoch nicht die komplette Eingabe eins zu eins verglichen, sondern vielmehr die zu einem Präfix der Eingabe passende Ausgabe bestimmt. Korollar 5.18 garantiert dabei, dass lediglich die führenden $\log N$ Bits dabei betrachtet werden müssen.

Beobachtung 5.20 *Bei einem Tabellendekodierer für m Symbole mit einer Feinheit von $N = 2^k$, welcher nur geordnete, k-beschränkt dyadische Verteilungen verwendet, kann eine Arithmetik-Tabelle durch* $m \cdot (\lceil \log\log N \rceil + \lceil \log m \rceil)$ AND-*Gatter mit Eingangsgrad maximal* $\log N + 1$ *und einer Schaltkreistiefe von* 1 *realisiert werden.*

Beweis: Da sich die Anzahl der Shift-Operationen zwischen 1 und $\log N$ bewegt, kann diese mit $\lceil \log\log N \rceil$ Bits kodiert werden. Jedes der m Symbole kann mit $\lceil \log m \rceil$ Bits eindeutig dargestellt werden.

Somit kann also eine mögliche Ausgabe mit $\lceil \log \log N \rceil + \lceil \log m \rceil$ kodiert werden. Im Prinzip genügt es nun, jedes Ausgabe-Bit mit den dazugehörigen Präfix-Bits der Eingabe $x_1, \ldots, x_{\log N}$ durch ein *AND*-Gatter zu verknüpfen, um gegebenenfalls die Ausgabe durchzuschalten. Die *AND*-Gatter haben demnach maximalen Eingangsgrad $\lceil \log N \rceil + 1$ und die Tiefe beträgt dabei jeweils genau 1.

Da für jedes der m abzubildenden Symbole eine solche Schaltung parallel angegeben werden kann, folgt die Beobachtung. Der Schaltkreis, welcher eine Arithmetik-Tabelle realisiert, ist in Abbildung 5.7 skizziert. □

Wir sehen also, dass die Schaltkreisgröße einer Arithmetik-Tabelle nur von N und m abhängt. Darüber hinaus benötigen wir je Verteilung eine solche Tabelle. Es werden also insgesamt $\# R_m(N)$ Schaltkreise gebraucht.

Betrachten wir nun das Statistik-Modul. Um die gewünschte Verteilung und Permutation zu ermitteln, werden wir keine Suche verwenden. Vielmehr werden wir, gesteuert durch den Kontext, einfach die gesuchte Ausgabe durchschalten. Der zu Grunde liegende Schaltkreis arbeitet dabei nach dem gleichen Prinzip wie die Arithmetik-Tabellen.

Da es $m!$ Permutationen auf m Symbolen gibt, müssen mindestens $\lceil \log m! \rceil$ Bits bereitstehen, um diese eindeutig zu beschreiben. Wir werden uns aus Gründen, die gleich noch etwas klarer werden, sogar eine etwas ineffizientere Speicherung durch $m \lceil \log m \rceil$ Bits erlauben. Dazu werden — quasi als Liste angeordnet — die $\lceil \log m \rceil$ Bits, welche ein Alphabetsymbol identifizieren, in der aufsteigenden Reihenfolge der Indizes hintereinander geschrieben. Für $m = 16$ ist diese Darstellung nur etwa $1,45$ mal länger als notwendig.

Beobachtung 5.21 *Sei c die Anzahl der Kontexte eines Tabellendekodierers für m Symbole mit einer Feinheit $N = 2^k$, welcher nur geordnete, k-beschränkt dyadische Verteilungen verwendet. Dann kann das Statistik-Modul durch $c \cdot \left(\lceil \log \# R_m(N) \rceil + m \lceil \log m \rceil \right)$* AND-*Gatter mit maximalem Eingangsgrad $\lceil \log c \rceil + 1$ und einer Schaltkreistiefe von 1 realisiert werden.*

Beweis: Zur Darstellung der c Kontexte und der $\# R_m(N)$ Verteilungen genügen $\lceil \log c \rceil$, bzw. $\lceil \log \# R_m(N) \rceil$ Bits. Die Permutationen können wir, wie oben angegeben, durch $m \lceil \log m \rceil$ Bits ausdrücken.

Nun kann jedes der Ausgabe-Bits einfach mit den $\lceil \log c \rceil$ Bits durch ein *AND*-Gatter verknüpft werden, welches die entsprechende Verteilung bzw. Permutation durchschaltet. Insgesamt resultieren also $c \cdot \left(\lceil \log \# R_m(N) \rceil + m \lceil \log m \rceil \right)$ solche Gatter. Ein Schaltkreis für das Statistik-Modul ist in Abbildung 5.8 skizziert. □

Wir haben bereits gesehen, dass $\# R_m(N)$ für unsere Kenndaten keine großen Werte annimmt. Die Schaltkreisgröße ist also im Wesentlichen von der Anzahl der Kontexte abhängig. Das Finden eines geeigneten Modells, welches eine adäquate Kompression ermöglicht und dennoch nur eine geringe Zahl an Kontexten betrachtet, ist jedoch eine Kunst für sich, welche wir in diesem Text nicht weiter betrachten wollen. Ein interessanter Ansatz zur Reduzierung der Kontexte kann z.B. der Arbeit von Hoppe in [Hop03] entnommen werden.

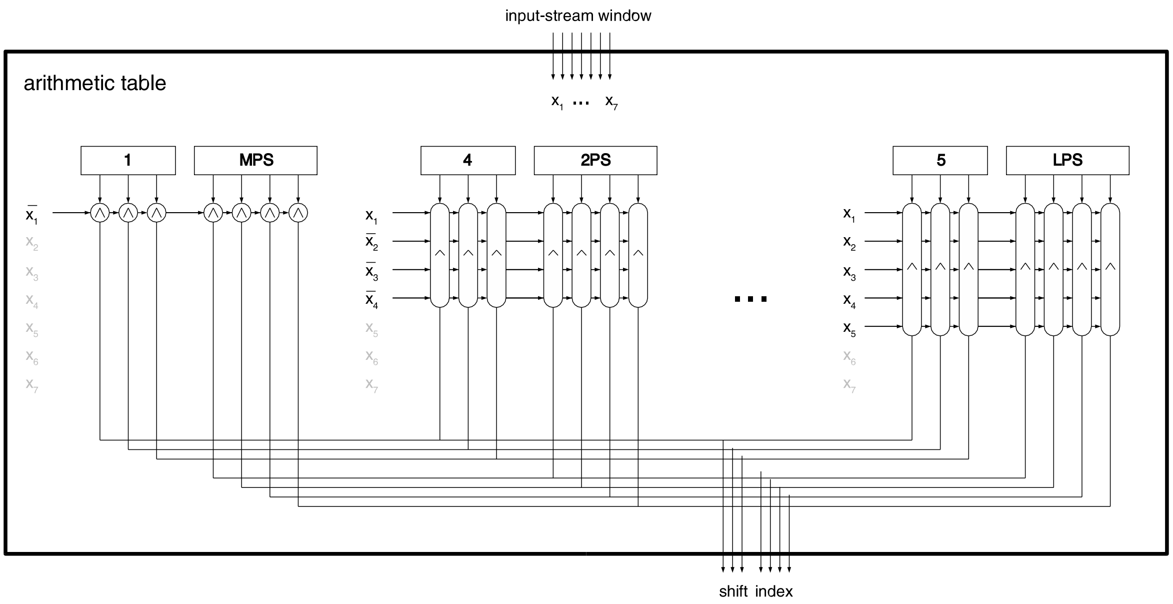


Abbildung 5.7: Schaltkreis-Skizze einer Dekodiertabelle für $N = 128$ und $m = 16$ zum Zustand $[0, 128)$ und zur Verteilung $\left(\frac{1}{2}, \frac{1}{16}, \frac{1}{32}, \frac{1}{32}, \ldots, \frac{1}{32}, \frac{1}{32}\right)$

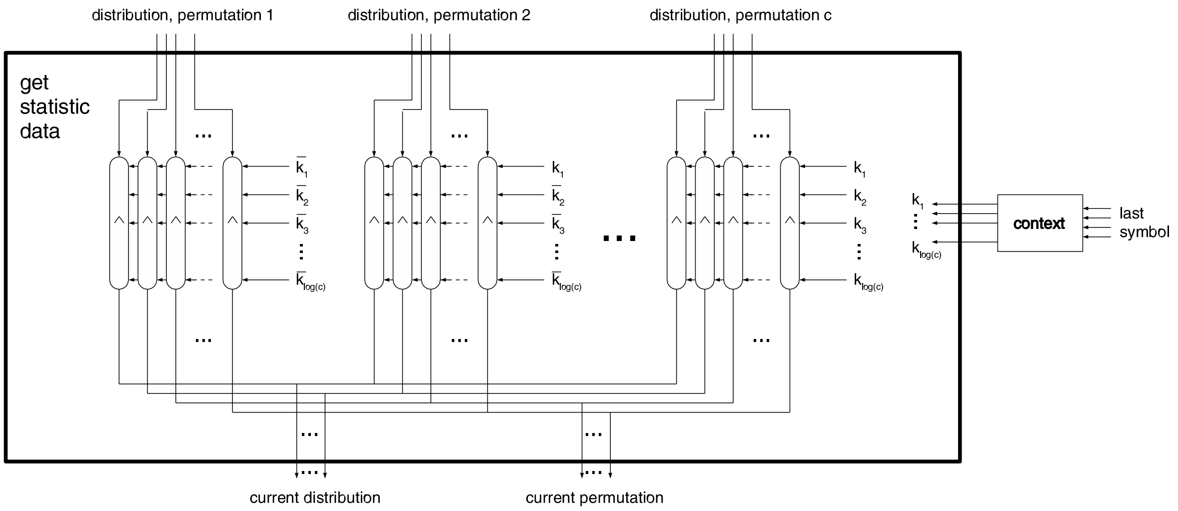


Abbildung 5.8: Schaltkreis-Skizze des Statistik-Moduls für c Kontexte und $m = 16$

Werfen wir nun einen Blick auf den Ergebnis-Filter. Auch diese Baugruppe schaltet aus den Eingaben lediglich das Ergebnis in Abhängigkeit von Zustand und Verteilung durch.

Beobachtung 5.22 *Bei einem Tabellendekodierer für m Symbole mit Feinheit N, kann das gesuchte Ergebnis durch $\# R_m(N) \cdot \left(\lceil \log\log N \rceil + \lceil \log m \rceil\right)$ AND-Gatter mit maximalem Eingangsgrad $\lceil \log \# R_m(N) \rceil + 1$ in einer Schaltkreistiefe von 1 aus allen Ausgaben der Arithmetik-Tabellen herausgefiltert werden.*

Beweis: Zur Darstellung der Verteilungen sind $\lceil \log \# R_m(N) \rceil$ Bits ausreichend. Aus Beobachtung 5.20 wissen wir, dass ein Ergebnis einer Arithmetik-Tabelle mit $\lceil \log\log N \rceil + \lceil \log m \rceil$ Bits ausgedrückt werden kann.

Wie in den vorausgegangenen Beobachtungen können nun einfach alle Eingabe-Bits jeweils mit den $\lceil \log \# R_m(N) \rceil$ steuernden Bits in einem *AND*-Gatter zusammengeschaltet werden. Demnach folgt die Behauptung. Abbildung 5.9 skizziert diesen Schaltkreis. □

Es fehlt nun nur noch die Betrachtung des Index-Konverters. Wie bereits oben erwähnt wurde, wollen wir die Permutation, welche Index und Symbol zuordnet, als eine nach dem Index sortierte „Liste" von Symbolen verstehen. Aus dieser kann dann, gesteuert vom dekodierten Index, die Bitdarstellung des Ergebnisses mit der gleichen Methode wie oben selektiert werden.

Beobachtung 5.23 *Sei ein Tabellendekodierer für m Symbole gegeben. Der Index-Konverter kann dann mit $m \cdot \lceil \log m \rceil$ AND-Gatter mit maximalem Eingangsgrad $\lceil \log m \rceil + 1$ in einer Schaltkreistiefe von 1 implementiert werden.*

Beweis: Wie bereits betrachtet, benötigt die Darstellung der Permutation $m \cdot \lceil \log m \rceil$ Bits. Jede dieser Eingaben kann nun mit den entsprechenden $\lceil \log m \rceil$ Bits, welche einen Index kodieren, durch ein *AND*-Gatter verknüpft werden. Abbildung 5.10 zeigt diesen Schaltkreis. □

Wie aus den Beobachtungen zu erkennen ist, weist die Gesamtgröße unseres Schaltkreises — wenn man die logarithmischen Faktoren einmal außer Acht lässt — vor allem eine direkte Abhängigkeit von der Größe des Symbolalphabetes m und der Anzahl der geordneten, dyadischen Verteilungen bezüglich m und N auf. Beim Statistik-Modul geht darüber hinaus auch noch die Anzahl der zu betrachtenden Kontexte ein. Als Beispiel für deren Größe können die Ergebnisse aus dem Projekt „Online Codekompression für Smartcards" herangezogen werden. Die dort entwickelten Modelle betrachteten zwischen $c = \mathcal{O}(m)$ Kontexte für etwa JavaCard Quellcode und $c = \mathcal{O}(m^2)$ Kontexte bei Assembler Code. Abbildung 5.11 hat die resultierenden Schaltkreisgrößen einmal tabellarisch zusammengefasst. Es zeigt sich, dass selbst diese groben oberen Schranken die Kapazitäten moderner Smartcards mit einigen Hunderttausend Gattern nicht überschreiten. Dies spricht für die prinzipielle Machbarkeit des Konzeptes.

Die reine Betrachtung als Boole'scher Schaltkreis blendet natürlich einige, in der Praxis sehr wichtige Details aus. Zum Einen sind in Wirklichkeit *AND*-Gatter nicht dass

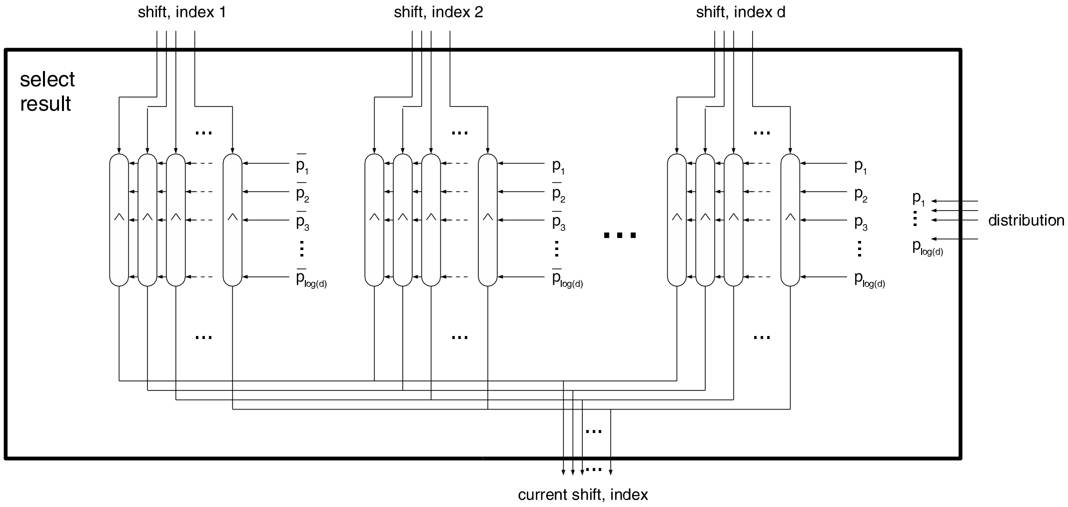


Abbildung 5.9: Schaltkreis-Skizze des Ergebnis-Filters für d Verteilungen

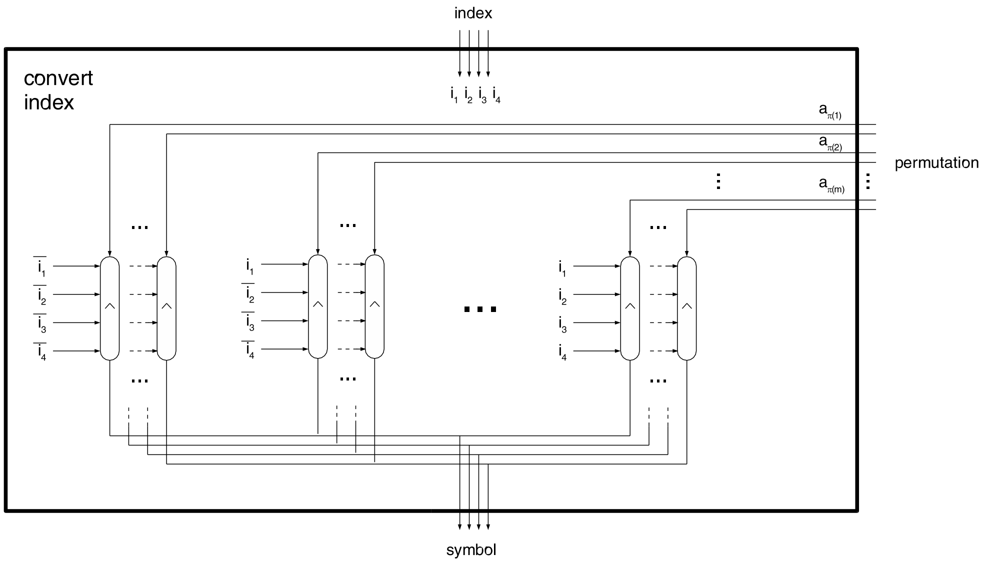


Abbildung 5.10: Schaltkreis-Skizze des Index-Konverters für $m = 16$

	$N = 32$, $m = 16$, $c \approx m$	$N = 128$, $m = 16$, $c \approx m$	$N = 128$, $m = 16$, $c \approx m^2$	SK-Größe ($\approx$)
Arithmetik-Tabellen	1344	20272	20272	$\# R_m(N) \cdot m \cdot \log(\ldots)$
Statistik-Modul	1088	1152	18432	$c \cdot m \cdot \log(\ldots)$
Ergebnis-Filter	84	1267	1267	$\# R_m(N) \cdot \log(\ldots)$
Index-Konverter	64	64	64	$m \cdot \log(\ldots)$
Summe	2580	22755	40035	

Abbildung 5.11: Größenschranken für den Dekodierschaltkreis

Maß aller Dinge. Aus elektrotechnischer Sicht sind unter anderem noch Abschätzungen betreffend des Energiebedarfs, des Flächenverbrauchs oder gar der Kreuzungswege von Leiterbahnen von unverzichtbarer Bedeutung.

Ferner bietet unser Konzept noch mannigfaltige Möglichkeiten zur praktischen Verbesserung, auf welche wir nicht eingegangen sind. Die strikt getrennten Gruppen von *AND*-Gattern könnten effizienter zusammengefasst werden, um bei ähnlichen Schaltungen dieselben Teilschaltkreise nicht immer wieder und wieder zu implementieren. Aus Energiegründen macht es auch durchaus Sinn, den Schaltkreis zur aktuellen Arithmetik-Tabelle im Voraus zu bestimmen und nur diesen mit Energie zu versorgen, statt $\# R_m(N) - 1$ überflüssige Berechnungen durchzuführen. Außerdem könnte der doch sehr fordernden Leiterbahnbreite von $m \log m$ für die Permutation aus dem Weg gegangen werden, indem man die Möglichkeiten des jeweiligen Prozessors zur indirekten Adressierung ausnutzt und lediglich ROM-Speicheradressen übermitteln würde.

Diese Argumente machen die Notwendigkeit einer stärker realitätsbezogenen Analyse deutlich. Erste Untersuchungen durch die Infineon Technologies AG München mit Hilfe schaltkreisoptimierender Simulationssoftware (Infineon C11FL) ergaben bereits eine geringe Flächenkomplexität bei sehr schneller Bearbeitungszeit des Schaltkreises. Die ermittelten Ergebnisse bezüglich Latenz und Flächenbedarf sind in Abbildung 5.12 zusammengefasst.

Latenz (in ns)	2	3	4	5
Fläche (in μm^2)	44422	49755	27800	26700

Abbildung 5.12: Latenz und Flächenbedarf des optimierten Schaltkreises

Dessen ungeachtet ging es in unseren Betrachtungen auch lediglich um das Aufzeigen der konzeptionellen Machbarkeit.

5.5 Approximation durch dyadische Verteilungen

Der hier vorgestellte Dekodierschaltkreis soll, wie in den vorausgegangenen Abschnitten hergeleitet, nur geordnete, dyadische Verteilungen verwenden. Der Übergang auf geordnete Verteilungen ist dabei unter Verwendung des Index-Konverters recht einfach und dazu in der Kompressionsrate verlustfrei realisierbar.

Bei der Approximation durch dyadische Verteilungen haben wir leider nicht so viel Glück. Da bei der Kodierung und Dekodierung nicht die exakten Wahrscheinlichkeiten verwendet werden, wird dies die Kompressionsrate verschlechtern. Definition 2.11 hat uns mit der relativen Entropie bereits ein Maß vorgestellt, mit dem sich dieser Verlust beschreiben lässt. Wir können dies analog zu Lemma 3.19 zeigen.

Lemma 5.24 *Betrachte einen Tabellenkodierer mit Feinheit $N = 2^L$. Sei p die Wahrscheinlichkeitsverteilung der Quelle. Wird stattdessen eine dyadische Verteilung q zur Kompression verwendet, so gilt für den erwarteten Kompressionsverlust ε in Bits:*

$$\varepsilon \leq \mathrm{D}\left(p \| q\right) \ .$$

Beweis: Seien $p = (p_1, p_2, \ldots, p_m)$ und $q = (q_1, q_2, \ldots, q_m)$ gegeben, wobei stets $q_i = 2^{-l_i}$ für ein $l_i \in \mathbb{N}_0$ gilt. Dann ist $l_i = \log \frac{1}{q_i}$ die genaue Codewort-Länge des Symbols a_i bei dem Tabellenkodierer, im Gegensatz zu $\left\lceil \log \frac{1}{p_i} \right\rceil + 1$ bei der ursprünglichen *WNC*-Kodierung. Es folgt

$$\varepsilon = \sum_{i=1}^{m} p_i \cdot \left(\log \frac{1}{q_i} - \left\lceil \log \frac{1}{p_i} \right\rceil - 1 \right) \leq \sum_{i=1}^{m} p_i \cdot \left(\log \frac{1}{q_i} - \log \frac{1}{p_i} \right) = \sum_{i=1}^{m} p_i \log \frac{p_i}{q_i} = D\left(p \| q\right)$$

und somit die Behauptung. □

Das Ziel ist es also, zu $N = 2^L$ aus p eine geordnete, L-beschränkt dyadische Verteilung $q = \left(2^{-l_1}, 2^{-l_2}, \ldots, 2^{-l_m}\right)$ zu entwickeln, welche $\sum p_i l_i$ minimiert. Diese Problematik ist bereits von KNUTH untersucht worden (vgl. [Knu71]), allerdings in einer Formulierung über die minimale, erwartete Blatttiefe in binären Suchbäumen, und vor allem ohne die L-Beschränktheit.

LARMORE und HIRSCHBERG lösten in [LH90] das L-beschränkte Problem — ebenfalls in leicht abgewandelter Form über das so genannte Coin-Collectors-Problem — in einer effizienten Laufzeit von $\mathcal{O}(mL)$. Der Package-Merge-Algorithmus, welchen sie dafür entwickelten, arbeitet im Wesentlichen mit Mengenoperationen, um das Coin-Collectors-Problem zu lösen, und einer $\mathcal{O}(mL)$-Zeit-Reduktion, um die Lösung zu übertragen.

Der BRCI-Algorithmus, welcher in einer Laufzeit von $\mathcal{O}(m)$ einen L-tiefenbeschränkten Suchbaum approximiert, wurde von MILIDIU und LABER präsentiert (vgl. [ML97]). Auch dieser Algorithmus verwendet leider einige etwas komplexere Operationen auf Bäumen. Darüber hinaus garantiert er bei unseren doch recht kleinen Kenndaten von $N = 128$ und $m = 16$ theoretisch nur einen Kompressionsverlust von unter 39%.[3] Obwohl der tatsächliche Approximationsfaktor sicherlich besser wäre, als es die beweisbare Schranke vermuten

[3] Bei geeignet großen Parametern N und m liefert der Algorithmus jedoch auch wesentlich aussagekräftigere Schranken.

lässt, stellt sich daher die Frage, ob ein komplexerer Algorithmus überhaupt notwendig ist.

Im Prinzip entspricht die Minimierung des Terms $\sum p_i l_i$ exakt der Suche nach einem kompakten Code für p_i. Diese Beobachtung führt daher zu einer alternativen, algorithmisch wesentlich einfacheren Approximations-Heuristik als die beiden oben angesprochenen Methoden. Man bestimme zu einer beliebigen Häufigkeitsverteilung f einfach einen geeigneten präfixfreien, kompakten Code, z.B. einen Huffman-Code zu f. Dessen Kodierung definiert dann nach Lemma 5.25 durch seine Codewort-Längen eine dyadische Verteilung.

Lemma 5.25 *Sei ein präfixfreier, kompakter Code C auf dem Alphabet $\Sigma = (a_1, a_2, \ldots, a_m)$ zu der Verteilung $p = (p_1, p_2, \ldots, p_m)$ gegeben. Bezeichne $l_i := |C(a_i)|$ die Länge des Codewortes von a_i. Dann beschreibt $q = \left(2^{-l_1}, 2^{-l_2}, \ldots, 2^{-l_m}\right)$ eine dyadische Verteilung.*

Beweis: q ist nach Konstruktion sicherlich dyadisch. Wir müssen aber noch zeigen, dass q überhaupt einen Wahrscheinlichkeitsvektor beschreibt, also dass $\sum_{k=1}^{m} 2^{-l_k} = 1$ gilt. Dazu führen wir eine Induktion nach m. Für $m = 2$ ist die Behauptung dann sofort klar, da für $\Sigma = \mathbb{B}$ mit $|C(0)| = 1$ und $|C(1)| = 1$ die einzige Möglichkeit für einen kompakten Code gegeben ist.

Es gelte die Aussage für m. Wir wollen nun zeigen, dass sie dann auch für $m+1$ gilt. Sei ein präfixfreier, kompakter Code C zu Σ und p mit $m+1$ Symbolen gegeben. Dann müssen zunächst mindestens zwei Symbole a_i und a_j existieren, welche beide die maximale Codewort-Länge besitzen. Gäbe es nur ein Symbol maximaler Länge, so könnte man dieses ohne Verletzung der Präfixfreiheit um wenigstens eine Stelle einkürzen und so der Kompaktheit widersprechen. Darüber hinaus muss es mindestens zwei Symbole a_i und a_j geben, welche maximale Länge haben und sich nur in der letzten Stelle unterscheiden. Auch hier würde anderenfalls durch ein Verkürzen der Codewörter der Widerspruch folgen.

Wir ersetzen nun a_i und a_j durch ein neues Symbol $a' \notin \Sigma$, welchem wir die Wahrscheinlichkeit $p' := p_i + p_j$ zuordnen. Als Kodierung soll a' den gemeinsamen Teil von $C(a_i)$ und $C(a_j)$ ohne die letzte Stelle erhalten, also $l' := |C(a')| = l_i - 1 = l_j - 1$. Dann beschreibt C auf dem Alphabet $\Sigma' = \left(\Sigma \setminus \{a_i, a_j\}\right) \cup \{a'\}$ mit m Symbolen noch immer eine präfixfreie Kodierung. Diese ist kompakt, da im Erwartungswert lediglich der Term $p_i l_i + p_j l_j$ durch $p' l'$ ersetzt wurde. Würde eine weitere Codierung mit Codewort-Längen λ_k und

$$\sum_{k \neq i, k \neq j} p_k \lambda_k + p' l' < \sum_{k \neq i, k \neq j} p_k l_k + p' l'$$

existieren, so würde auch

$$\sum_{k=1}^{m+1} p_k \lambda_k < \sum_{k=1}^{m+1} p_k l_k$$

gelten, was der vorausgesetzten Kompaktheit von C widerspräche. Demnach folgt aus der Induktionsvoraussetzung, dass

$$\sum_{k \neq i, k \neq j} 2^{-l_k} + 2^{-l'} = 1$$

gilt. Da nach Konstruktion $l' = l_i - 1 = l_j - 1$, folgt somit auch

$$2^{-l'} = 2^{-l_i+1} = 2 \cdot 2^{-l_i} = 2^{-l_i} + 2^{-l_i} = 2^{-l_i} + 2^{-l_j} \ .$$

Einfaches Einsetzen in der obigen Summe beweist dann die Behauptung. □

Allerdings haben wir bisher völlig außer Acht gelassen, dass nur Verteilungen auf einem Intervall der Länge $N = 2^L$, also L-beschränkt dyadische Verteilungen, zulässig sind. Wir werden deshalb als erste Heuristik folgenden Ansatz verfolgen. Zunächst wird mittels Huffman-Kodierung eine unbeschränkt dyadische Verteilung berechnet. Anschließend werden eventuell zu lange Codewörter durch lokale Veränderungen im Huffman-Baum eingekürzt. Dabei muss auch ein kurzes Codewort verlängert werden, um nicht die Summeneigenschaft $\sum_{k=1}^{m} 2^{-l_k} = 1$ zu verletzen.

Kernidee des Huffman-LI Algorithmus ist es, den optimalen Huffman-Baum nur an den Blättern mit geringer Wahrscheinlichkeit zu manipulieren. Um die relative Entropie so klein wie möglich zu halten, wird also als Auswahl-Heuristik stets ein Codewort mit geringstmöglicher Häufigkeit verlängert.

Algorithmus 5.26 *(Huffman-LI Heuristik)*

Eingabe: *Verteilung* $f = (f_1, f_2, \ldots, f_m) \in \mathbb{N}^m$ *und* $L \in \mathbb{N}$ *mit* $\log m \leq L \leq m$.

1. *Bilde Huffman-Baum zu* f. *Es bezeichne* k_i *die Tiefe des Knotens zu* f_i.
2. *Sortiere die* k_i *absteigend zur verketteten Liste* $K[1..m]$.
3. *Solange* $K[1] > L$, *wiederhole 4 – 6:*
4. *Finde minimales* j *mit* $K[j] < L$.
5. *Hänge die Knoten zu* $K[1]$ *und* $K[j]$ *auf eine gemeinsame Tiefe, dazu setze* $K[1] \leftarrow K[j] + 1$, $K[j] \leftarrow K[j] + 1$ *und* $K[2] \leftarrow K[2] - 1$.
6. *Erhalte die Sortierung von* $K[1..m]$.
7. *Gib* $\left(2^{L-K[m]}, \ldots, 2^{L-K[2]}, 2^{L-K[1]}\right)$ *aus.*

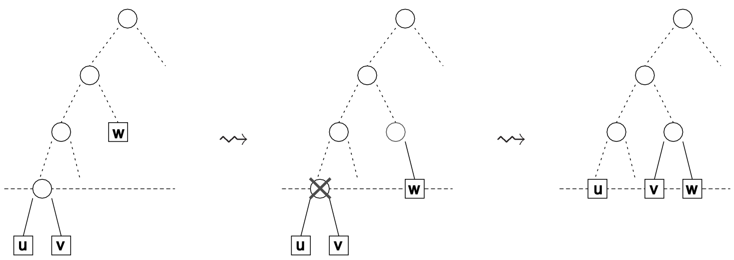
Es fällt offensichtlich nicht schwer, den Algorithmus derart abzuändern, so dass auch gleichzeitig die Permutation π, welche den Sortierschritten entspricht, bestimmt wird. Wir wollen aber noch die Korrektheit des Algorithmus nachweisen.

Satz 5.27 *Der Algorithmus* Huffman-LI(f, L) *berechnet zur Verteilung* f *und zu* $L \in \mathbb{N}$ *in Laufzeit* $\mathcal{O}(m^2)$ *eine geordnete,* L*-beschränkt dyadische Verteilung.*

Beweis: Aus Lemma 5.25 folgt, dass nach Schritt 2 bereits $\sum_{k=1}^{m} 2^{-K[i]} = 1$, also auch $\sum_{k=1}^{m} 2^{L-K[i]} = 2^L$ gilt. Wir wollen zeigen, dass während eines Schleifendurchlaufs diese Eigenschaft nicht verletzt wird. Die einzigen Werte, welche verändert werden, sind $K[1]$, $K[2]$ und $K[j]$. Da diese aus einer kompakten Kodierung hervorgingen, gibt es wenigstens zwei Codewörter maximaler Länge, also $K[1] = K[2]$. Bezeichne zum Beweis der Invariante $\kappa[1]$, $\kappa[2]$ und $\kappa[j]$ die neuen Werte von $K[1]$, $K[2]$ und $K[j]$ nach Schritt 5. Dann gilt:

$$\begin{aligned}
2^{L-\kappa[1]} + 2^{L-\kappa[2]} + 2^{L-\kappa[j]} &= 2^{L-K[j]+1} + 2^{L-K[2]+1} + 2^{L-K[j]-1} \\
&= 2 \cdot 2^{L-K[2]} + 2 \cdot 2^{L-K[j]-1} \\
&= 2^{L-K[2]} + 2^{L-K[2]} + 2 \cdot 2^{-1} \cdot 2^{L-K[j]} \\
&= 2^{L-K[1]} + 2^{L-K[2]} + 2^{L-K[j]} \ .
\end{aligned}$$

Somit wird der Wert der Summe nicht verändert. Intuitiv entspricht ein solcher Schritt dem Verschieben eines Knotens v mit Tiefe größer L im Huffman-Baum. Dazu muss ein Knoten w mit Tiefe kleiner L tiefer gehangen werden, um Platz zu schaffen. Der Geschwisterknoten v kann dabei auch eine Ebene höher gehangen werden.



Ein j mit $K[j] < L$ muss existieren, sobald wenigstens ein $K[i]$, also insbesondere $K[1]$, größer L ist. Anderenfalls würde, falls alle $K[i] \geq L$ und $K[1] > L$, die Summe $\sum_{k=1}^{m} 2^{L-K[i]} < \sum_{k=1}^{m} 2^0 = m \leq 2^L$ den obigen Beobachtungen widersprechen.

Die Schleife muss terminieren, da in jedem Schritt mit $K[1]$ mindestens ein weiterer Wert nach dem Schleifenrumpf kleiner oder gleich L ist. Gleichzeitig wird niemals ein $K[j]$ größer als L. Auf Grund ihrer Konstruktion ist $\left(2^{L-K[m]}, \ldots, 2^{L-K[2]}, 2^{L-K[1]}\right)$ sicherlich dyadisch. Da nach dem letzten Schleifendurchlauf stets $K[i] \leq L$ gilt, ist die Ausgabe zudem noch L-beschränkt.

Sowohl der Huffman-Baum als auch die Sortierung können mit den bekannten Algorithmen in einer Laufzeit von $\mathcal{O}(m \log m)$ bestimmt werden. Die Schleife aus Algorithmus 5.26 wird maximal $(m-L)$-mal durchlaufen. Bei einer Implementierung von Schritt 4 als lineare Suche werden dort auch maximal $m-L$ Schritte benötigt, da je Iteration wenigstens ein Blatt höher als Tiefe L gehangen wird, was zu einer Schranke von $\mathcal{O}\left((m-L)^2\right) \subseteq \mathcal{O}(m^2)$ führt. Die Aktualisierung der Sortierung aus Schritt 6 kann dabei en passant mit einer konstanten Anzahl an Korrekturen geschehen. Die Berechnung der Ausgabe ist durch wiederholte Multiplikation mit 2 und Wiederverwendung alter Rechenergebnisse in $\mathcal{O}(m)$ Operationen möglich. Es folgt die Laufzeitschranke. □

Wir werden die Güte der Approximation aus 5.26 nicht beweisen. Vielmehr werden wir noch eine zweite Heuristik betrachten und deren Kompressionsverluste im nächsten Abschnitt empirisch betrachten. Wo wir beim Huffman-LI Algorithmus einen theoretisch optimalen Fall behutsam an unsere Nebenbedingungen angepasst haben, wollen wir nun auf direktem Wege eine recht gute Lösung angeben und versuchen, diese anschließend zu optimieren. Der Vergleich dieser gegensätzlichen Grundkonzepte soll uns dann ein Gefühl für deren Qualität geben.

Vorbild unserer zweiten Heuristik ist die Kodierung nach SHANNON (vgl. [Sha48]), bei welcher einem Symbol mit Wahrscheinlichkeit p_i direkt ein Codewort der Länge $\left\lceil \log \frac{1}{p_i} \right\rceil$ zugeordnet wird. Aus dem Satz von Kraft folgt dann sofort, dass es eine derartige präfixfreie Kodierung geben muss. Leider ist der Shannon-Code nicht kompakt,[4] weswegen wir Lemma 5.25 nicht anwenden können. Wir können aber folgende Beobachtung treffen.

[4] Als einfaches Gegenbeispiel betrachte man den Shannon-Code zur Verteilung $\left(\frac{2}{3}, \frac{1}{3}\right)$.

Lemma 5.28 *Sei ein m-Tupel $f = \left(2^{k_1}, 2^{k_2}, \ldots, 2^{k_m}\right) \in \mathbb{N}^m$ gegeben. Sei ferner $k \in \mathbb{N}$ mit $\sum_{i=1}^{m} 2^{k_i} \leq 2^k$ und bezeichne $D := 2^k - \sum_{i=1}^{m} 2^{k_i}$. Dann gilt entweder*

(i) $D = 0$, oder

(ii) es existiert ein k_j, so dass $D \geq 2^{k_j}$.

Beweis: Es sei angenommen, das $D < 2^{k_i}$ für alle k_i gilt. Anderenfalls würde Fall *(ii)* gelten und es wäre nichts zu zeigen. Dann gilt auch $0 \leq D < 2^{k_j}$ für $k_j = \min\{k_1, \ldots, k_m\}$. Betrachten wir nun

$$0 \leq 2^k - \sum_{i=1}^{m} 2^{k_i} < 2^{k_j} \ ,$$

so folgt nach Multiplikation mit 2^{-k-1}

$$0 \leq 2^{-1} - \sum_{i=1}^{m} 2^{k_i-k-1} < 2^{k_j-k-1} \ .$$

Da $k_j \leq k_i \leq k$ für alle k_i gilt, sind somit alle Subtrahenden im Summenterm größer 0, aber kleiner 1 und besitzen den Hauptnenner 2^{k_j-k-1}. Demnach gilt für ein $a \in \mathbb{N}_0$

$$0 \leq a \cdot 2^{k_j-k-1} = 2^{-1} - \sum_{i=1}^{m} 2^{k_i-k-1} < 2^{k_j-k-1}$$

und nach weiterer Multiplikation mit 2^{k+1}

$$0 \leq a \cdot 2^{k_j} = 2^k - \sum_{i=1}^{m} 2^{k_i} < 2^{k_j} \ .$$

Somit kann lediglich $a = 0$ gelten, und es folgt $D = 0$. □

Eine Interpretation von Lemma 5.28 besagt nun, dass man, wenn eine Liste f von Zweierpotenzen gegeben ist, sukzessive geeignete Elemente der Liste verdoppeln kann und f so zu einer beschränkt dyadischen Verteilung heranwächst. Für unsere zweite Heuristik gehen wir daher wie folgt vor. Zunächst bilden wir mit Hilfe des Shannon-Codes Codewort-Längen. Sollten diese größer als L geworden sein, so kürzen wir deren Länge auf L ein. Da wir dadurch eventuell die Kraft-Ungleichung verletzen, verlängern wir einfach alle Codewörter solange um Eins, bis diese wieder gilt. Auch dabei werden wir als maximale Länge nur L erlauben.

Ist dies getan, so verringern wir die Codelängen der wahrscheinlichsten Symbole wieder so lange, bis in der Kraft-Ungleichung die Gleichheit gilt. Diese Auswahl-Heuristik sorgt dafür, dass häufige Symbole größere dyadische Häufigkeiten erhalten und somit die erwartete Codewort-Länge, und damit der Fehler, verringert wird.

Algorithmus 5.29 *(Shannon-LI Heuristik)*
Eingabe: *Verteilung $f = (f_1, f_2, \ldots, f_m) \in \mathbb{N}^m$ und $L \in \mathbb{N}$ mit $\log m \leq L \leq m$.*

1. *Bezeichne* $F_m \leftarrow \sum_{i=1}^{m} f_i$.
2. *Bilde mit* $k_i \leftarrow \left\lceil \log \frac{F_m}{f_i} \right\rceil$, *bzw.* $k_i \leftarrow \infty$ *falls* $f_i = 0$, *die Codewort-Längen des Shannon-Codes zu* f. *Beschränke* k_i *durch* $k_i \leftarrow \min\{k_i, L\}$.
3. *Sortiere die* k_i *aufsteigend zur verketteten Liste* $K[1..m]$.
4. *Solange* $\sum_{i=1}^{m} 2^{L-K[i]} > 2^L$, *setze alle* $K[i]$ *auf* $K[i] \leftarrow \min\{K[i]+1, L\}$.
5. *Solange* $\sum_{i=1}^{m} 2^{L-K[i]} < 2^L$, *wiederhole 6 – 8:*
6. *Finde minimales* j *mit* $2^{L-K[j]} \leq 2^L - \sum_{i=1}^{m} 2^{L-K[i]}$.
7. *Setze* $K[j]$ *auf* $K[j] \leftarrow K[j] - 1$.
8. *Erhalte die Sortierung von* $K[1..m]$.
9. *Gib* $\left(2^{L-K[1]}, 2^{L-K[2]}, \ldots, 2^{L-K[m]}\right)$ *aus.*

Satz 5.30 *Der Algorithmus* Shannon-LI(f, L) *berechnet zur Verteilung* f *und* $L \in \mathbb{N}$ *in Laufzeit* $\mathcal{O}(mL)$ *eine geordnete,* L*-beschränkt dyadische Verteilung.*

Beweis: Falls die Schleifen aus Schritt 4 und 5 terminieren, steht die Korrektheit des Algorithmus außer Frage. Nach der ersten ist die Summe $\sum_{i=1}^{m} 2^{L-K[i]} \leq 2^L$ und wird auf Grund der Wahl von j auch nicht wieder größer als 2^L. Wenn die zweite Schleife terminiert, muss also $\sum_{i=1}^{m} 2^{L-K[i]} = 2^L$ gelten, und die Ausgabe ist auf Grund ihrer Konstruktion dyadisch. Da die $K[i]$ niemals größer L oder kleiner 0 werden, folgt auch die L-Beschränktheit. Die Ordnung auf den $K[i]$ bleibt währenddessen erhalten.

Die Schleife in Schritt 4 muss terminieren, da sonst in ihrem Rumpf die $K[i]$ schrittweise erhöht werden, solange wenigstens ein $K[i] < L$ ist. Die $(K[1], \ldots, K[m])$ streben also strikt gegen $(L, \ldots, L)$. Spätestens dann gilt aber $\sum_{k=1}^{m} 2^{L-K[i]} = \sum_{k=1}^{m} 2^0 = m \leq 2^L$. Aber auch die zweite Schleife hält nach einer beschränkten Zahl an Iterationen. Lemma 5.28 besagt, dass wir stets einzelne $2^{L-K[j]}$ verdoppeln, also $K[j]$ dekrementieren können, ohne dass deren Summe 2^L übersteigt. Da sofort $\sum_{k=1}^{m} 2^{L-K[i]} > 2^L$ gelten würde, sobald ein $K[j] < 1$ sinkt, kann ein $K[j]$ nicht beliebig oft gewählt werden. Laut 5.28 muss also, sobald der letzte Dekrement-Schritt durchgeführt wurde, die Gleichheit folgen, und die Schleife terminiert.

Die L-beschränkten k_i aus der Shannon-Kodierung können in $\mathcal{O}(m)$ bestimmt werden. Deren Sortierung erfolgt in $\mathcal{O}(m \log m) \subseteq \mathcal{O}(mL)$. Der Summenterm der Schleife in Schritt 4 ist garantiert kleiner als $m \cdot 2^L \leq 2^{2L}$. Da im Prinzip der Wert der Summe in jedem Durchlauf halbiert wird, wird die Schleifenbedingung nach $\mathcal{O}(L)$ Schritten unwahr. Wegen $\mathcal{O}(m)$ Operationen im Rumpf benötigt die erste Schleife eine Laufzeit von $\mathcal{O}(mL)$.

Die zweite Schleife kann derart implementiert werden, dass die Differenz D, wie oben betrachtet, nur einmal berechnet werden muss. Da je Durchlauf stets nur $K[j]$ geändert wird, kann D somit in konstanter Zeit aktualisiert werden. Darüber hinaus kann man die Schleife auch als Greedy-Algorithmus formulieren, welcher — bei $K[1]$ angefangen — versucht, schrittweise die $K[j]$ maximal zu erhöhen, und dann zu $K[j+1]$ weitergeht. Da kein $K[j] > L$ sein kann, folgt als Laufzeit $\mathcal{O}(mL)$; ein stetes Sortieren ist bei dieser Implementierung nicht notwendig. Insgesamt folgt die Laufzeitschranke. □

Da die Approximation der Verteilungen im Voraus berechnet werden kann und dann beim eigentlichen Kodieren und Dekodieren zur Verfügung steht, ist deren Laufzeit nur von untergeordnetem Interesse. Implementierungstechnisch wirkt die simple Shannon-LI Heuristik attraktiver, da keine komplexeren Datenstrukturen wie Huffman-Bäume benötigt werden. Wirklich interessant ist jedoch der Vergleich des zu erwartenden Kompressionsverlustes. Diesem Thema werden wir uns im nächsten Abschnitt widmen.

5.6 Empirische Ergebnisse

Nachdem wir nun alle Komponenten unseres Dekodierers kennen gelernt haben, wollen wir nun auch das tatsächliche Kompressionsverhalten auf realistischen Testdaten betrachten. Dabei ist insbesondere ein Blick auf die Folgen interessant, welche die Kompromisse und Zugeständnisse gegenüber einem herkömmlichen arithmetischen Kodierer nach Witten, Neal und Cleary haben.

Als Testdaten wurde Assembler-Code für kryptographische Anwendungen auf Smartcards der Infineon Technologies AG München verwendet. Als Symbolalphabet der Größe $m = 16$ diente die Darstellung dieser Opcodes als Hexadezimalsymbole. Zur Erstellung der Statistik-Daten wurde eine semi-adaptive, kontextgesteuerte Modellierung verwendet, welche im Rahmen des Projektes „Online Codekompression für Smartcards“ in der Arbeitsgruppe von Prof. Dr. Blömer an der Universität Paderborn entwickelt wurde.

Beginnen wir mit unseren Untersuchungen bei den beiden Approximations-Heuristiken. Da sich der Huffman-LI und der Shannon-LI Algorithmus aus verschiedenen Richtungen auf die Suche nach einer optimalen, dyadischen Verteilung begeben haben, interessiert uns, wie stark sich jeweils die approximierte Verteilung von der ursprünglichen unterscheidet. Als Distanzbegriff dazu haben wir bereits in Folgerung 5.24 die relative Entropie motiviert.

Die Abbildungen 5.13 und 5.14 zeigen beide Heuristiken an Hand der Abhängigkeit zwischen dem Kompressionsverhalten der ursprünglichen Verteilung, angegeben durch $H(p)$, und dem maximal erwarteten Kompressionsverlust $D(p\|q)$ bzw. der erwarteten Codewort-Länge $ECL(p\|q)$ unter Verwendung der approximierten Verteilung q an Stelle von p. Es ist leicht zu erkennen, dass sich beide Algorithmen in ihrem Ergebnis nicht sehr stark unterscheiden.

Man kann außerdem die im Grunde sehr einleuchtende Tendenz ausmachen, dass stark unbalancierte Verteilungen einen wesentlich höheren Verlust erleiden als diejenigen, welche bereits relativ gleichverteilt waren. Dies erklärt sich recht einfach dadurch, dass die Gleichverteilung bei $N = 2^k$ bereits eine dyadische Verteilung ist. Andererseits wird bei unbalancierten Verteilungen, also $H(p) \approx 0$, ein Symbol mit Wahrscheinlichkeit ≈ 1 stets auf die dyadisch maximale Wahrscheinlichkeit $\frac{1}{2}$ abgebildet. Dies führt zu einer Verlängerung des Codewortes um fast ein ganzes Bit durch das wahrscheinlichste Symbol, was bei $m = 16$ einen Verlust von bis zu 25% bedeutet.

Andererseits lässt sich aber bereits ablesen, dass in dem praxisrelevanten und dadurch interessanten Fall einer Entropie zwischen $0,4$ und $0,7$ ein Kompressionsverlust von lediglich fünf bis sieben Prozentpunkten zu erwarten ist. Allerdings unterliegen diese Werte noch einer gewissen Streuung. Um zu untersuchen, ob vielleicht das eine Verfahren gegenüber dem anderen eine systematische Schwäche bei gewissen Verteilungen aufweist,

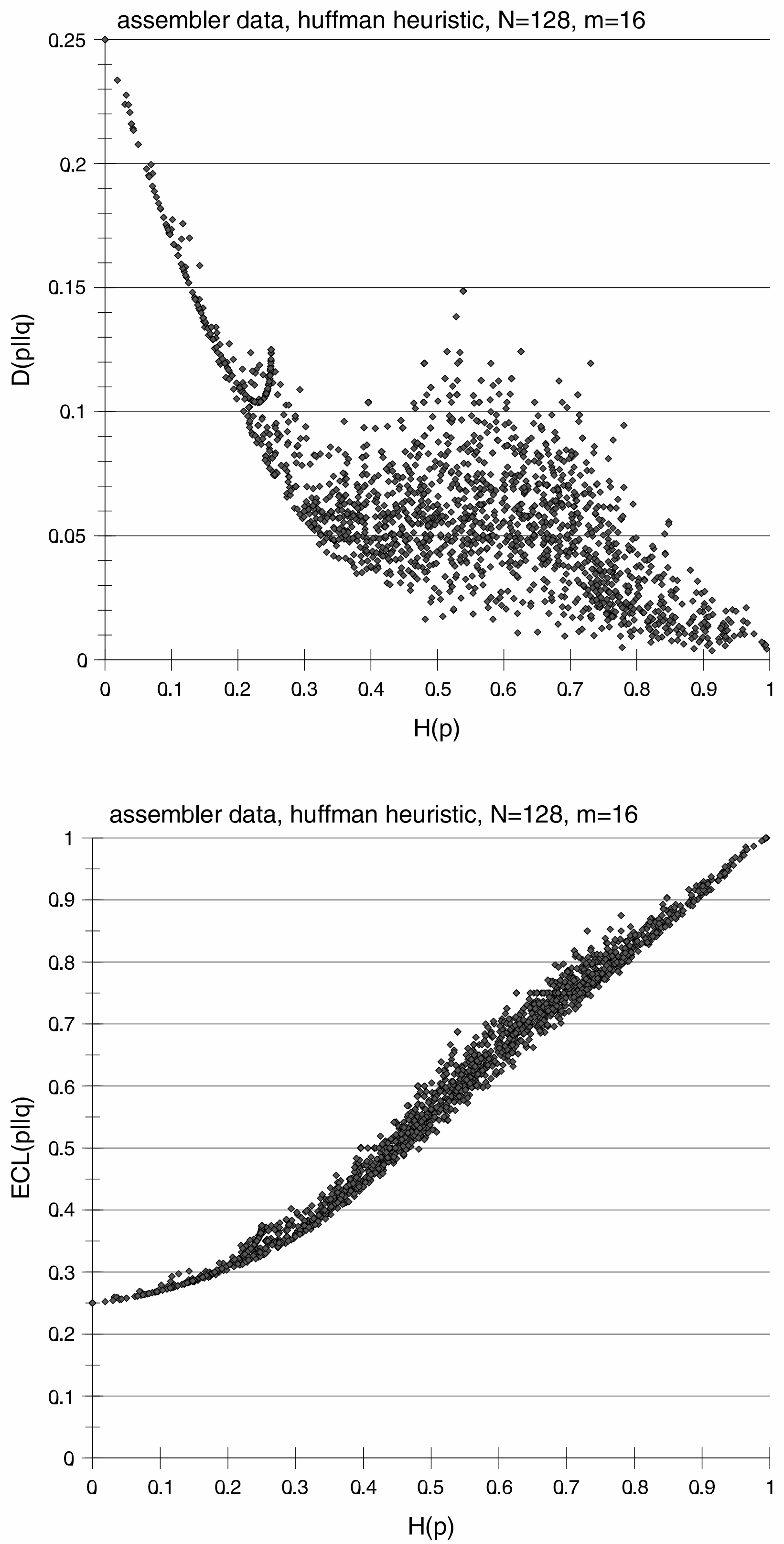


Abbildung 5.13: Kompressionsverlust eines Tabellendekodierers mit $N = 128$ und $m = 16$ bei der Huffman-LI-Approximation von p mit dyadischer Verteilung q.

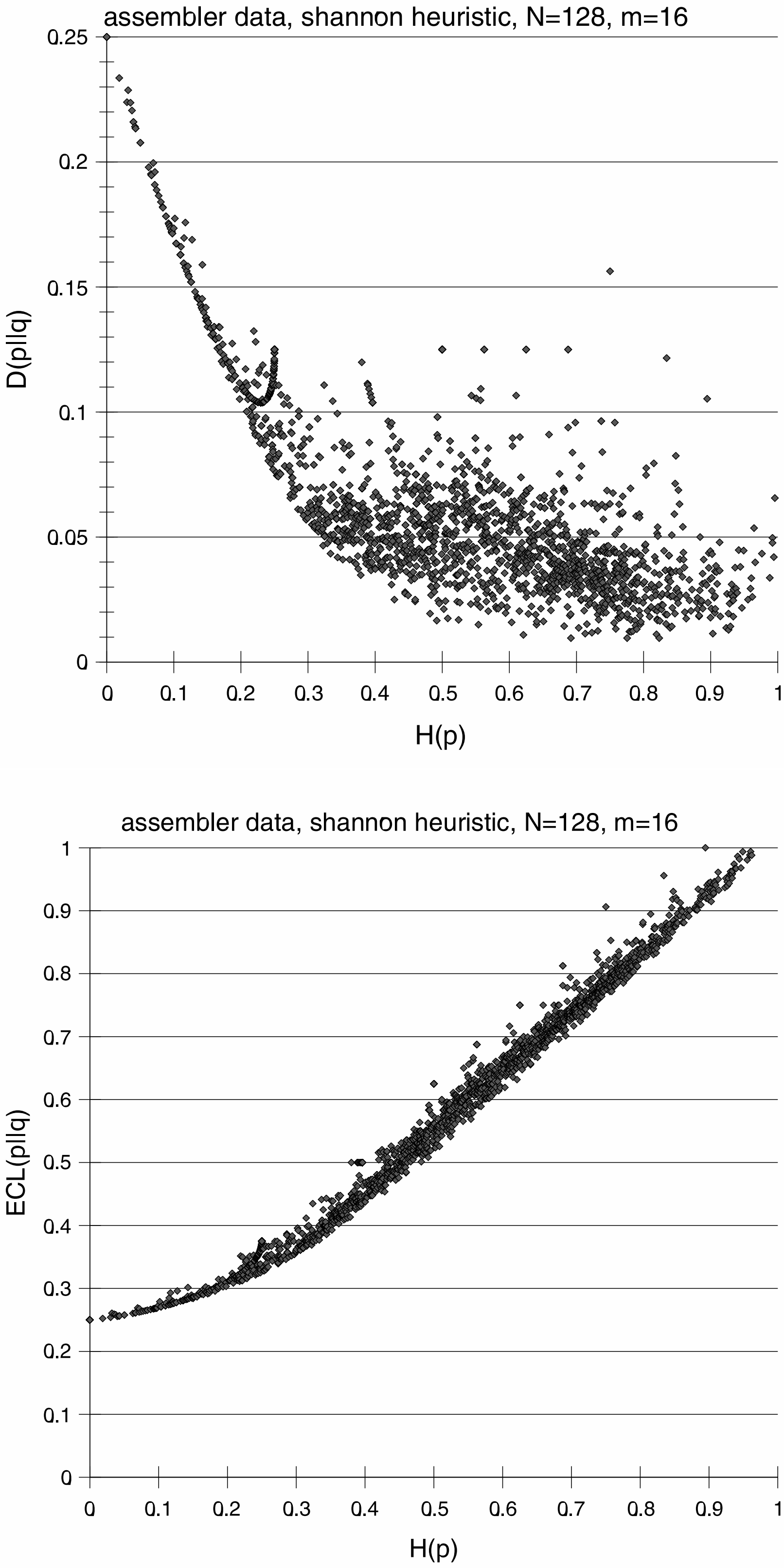
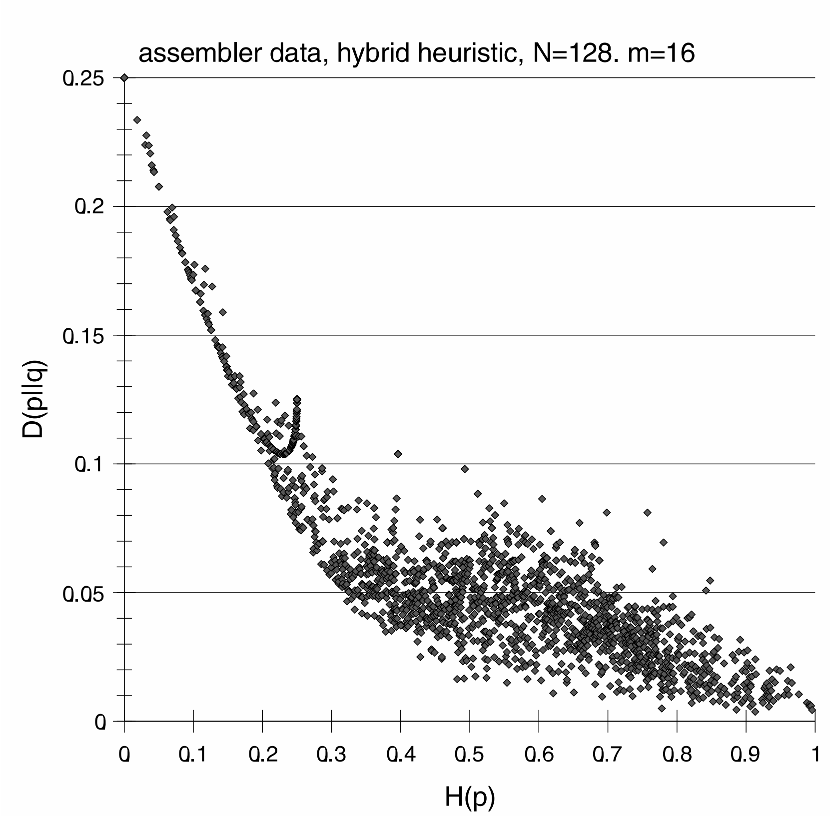


Abbildung 5.14: Kompressionsverlust eines Tabellendekodierers mit $N = 128$ und $m = 16$ bei der Shannon-LI-Approximation mit dyadischen Verteilungen.



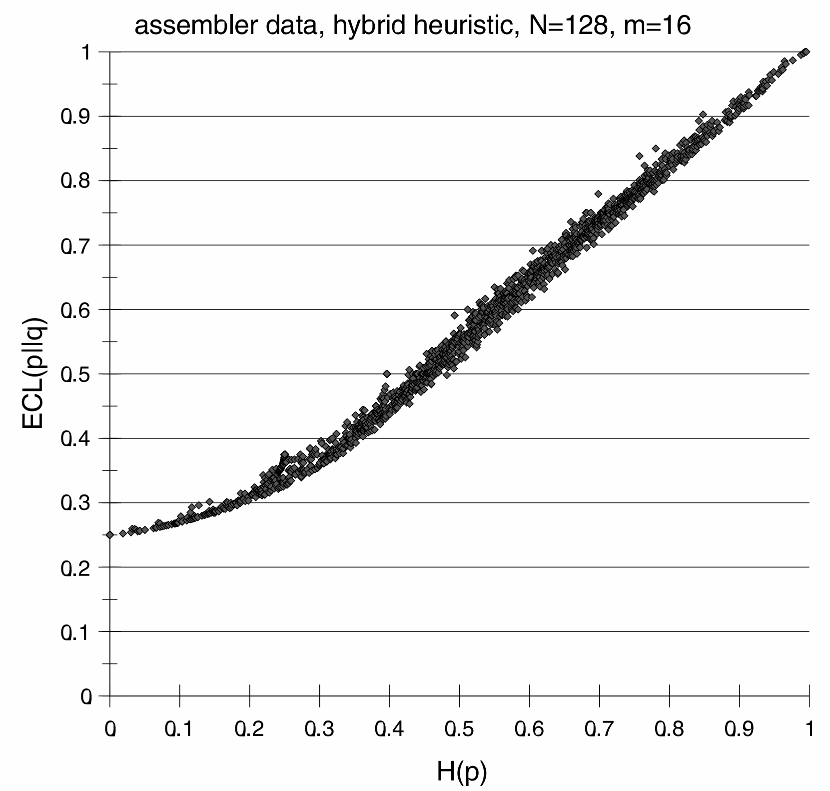
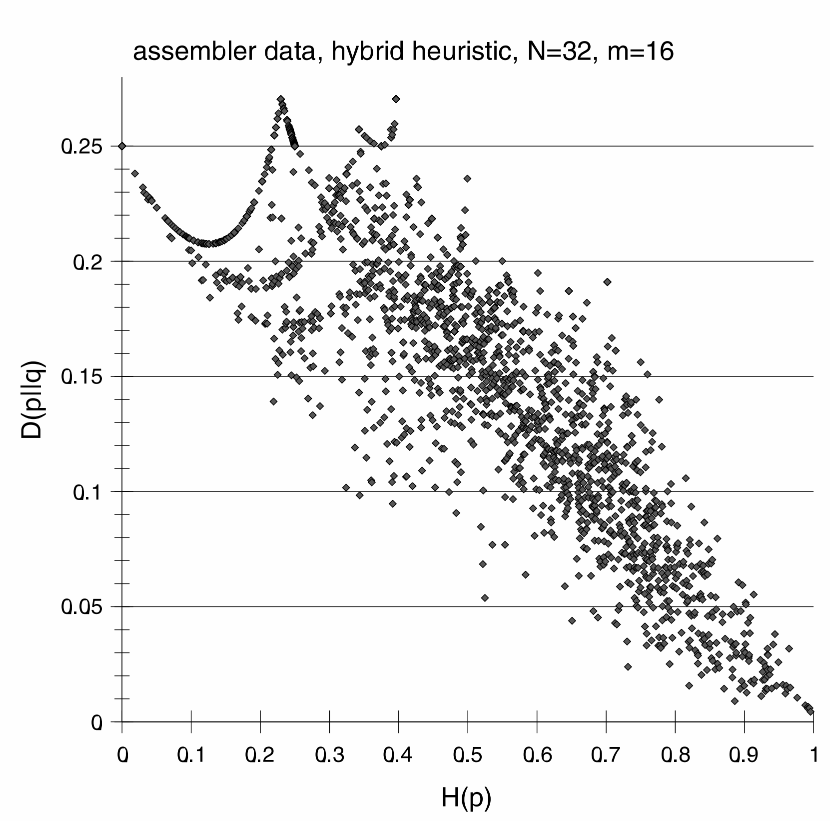


Abbildung 5.15: Kompressionsverlust eines Tabellendekodierers mit $N = 128$ und $m = 16$ bei hybrider Approximation von p mit dyadischer Verteilung q.



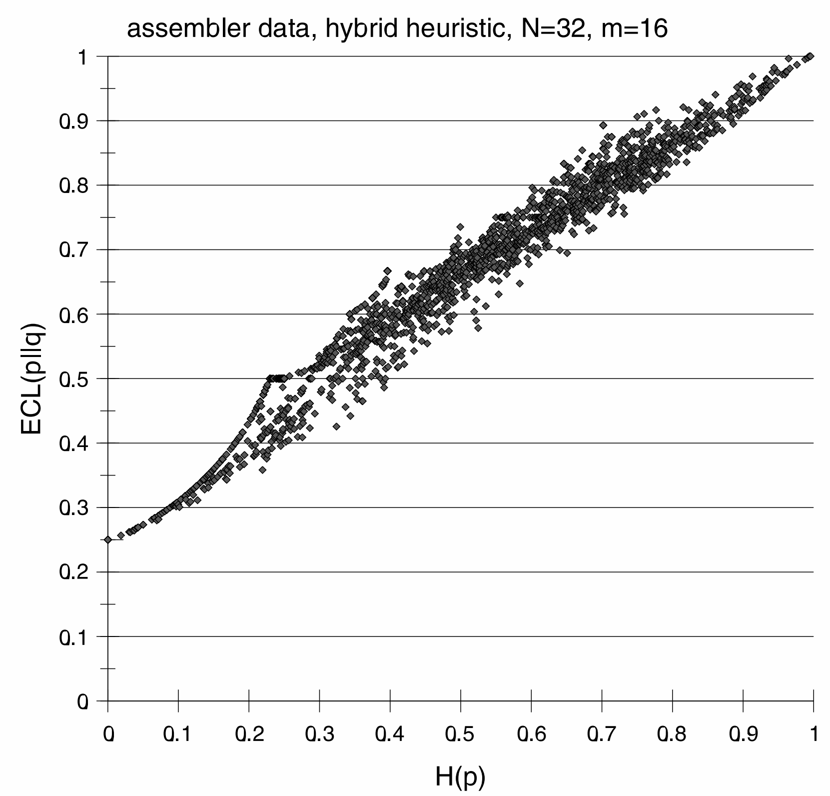


Abbildung 5.16: Kompressionsverlust eines Tabellendekodierers mit $N = 32$ und $m = 16$ bei hybrider Approximation mit dyadischen Verteilungen.

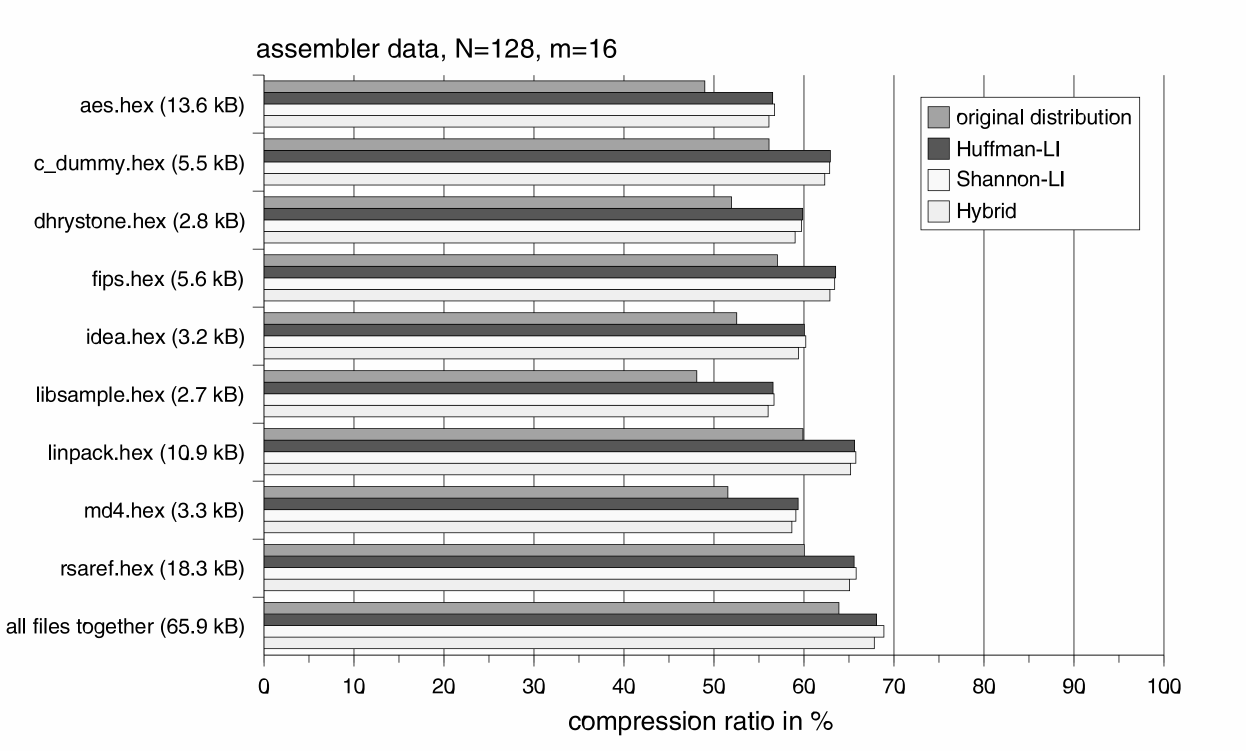


Abbildung 5.17: Vergleich der Kompressionsraten verschiedener Approximations-Heuristiken bei $N = 128$ und $m = 16$

wollen wir daher noch eine weitere, hybride Approximations-Heuristik aus unseren beiden gegebenen Algorithmen konstruieren. Dazu approximieren wir einfach die originale Verteilung durch jeweils Huffman-LI bzw. Shannon-LI und suchen uns die gegenüber $D(p\|q)$ vielversprechenste, dyadische Verteilung aus. Sollte nun ein Verfahren bei gewissen Verteilungen stets schwache Ergebnisse liefern, so haben wir nun die Hoffnung, dass das andere Verfahren diese abfängt.

Algorithmus 5.31 *(Hybride Heuristik)*
Eingabe: *Verteilung* $f = (f_1, f_2, \ldots, f_m) \in \mathbb{N}^m$ *und* $L \in \mathbb{N}$ *mit* $\log m \leq L \leq m$.
1. *Bestimme* $g_H \leftarrow$ Huffman-LI(f,L).
2. *Bestimme* $g_S \leftarrow$ Shannon-LI(f,L).
3. *Falls* $\mathrm{D}(f\|g_H) \leq \mathrm{D}(f\|g_S)$, *so gib* g_H *aus.*
4. *Falls* $\mathrm{D}(f\|g_S) < \mathrm{D}(f\|g_H)$, *so gib* g_S *aus.*

Dabei bezeichnet $D(f\|g)$ die relative Entropie der durch f beziehungsweise g induzierten Wahrscheinlichkeitsverteilungen.

Betrachten wir nun in Abbildung 5.15 das Verhalten der hybriden gegenüber unseren beiden anderen Heuristiken so stellt sich dieses als nur leicht besser dar. Der erwartete Kompressionsverlust bei Verteilungen mit $H(p) \approx 0$ liegt noch immer bei um die 25%, bei $0,4 \leq H(p) \leq 0,7$ dürfte er noch immer etwa zwischen vier und sechs Prozentpunkten liegen. Allerdings scheint die hybride Heuristik eine geringere Streuung aufzuweisen.

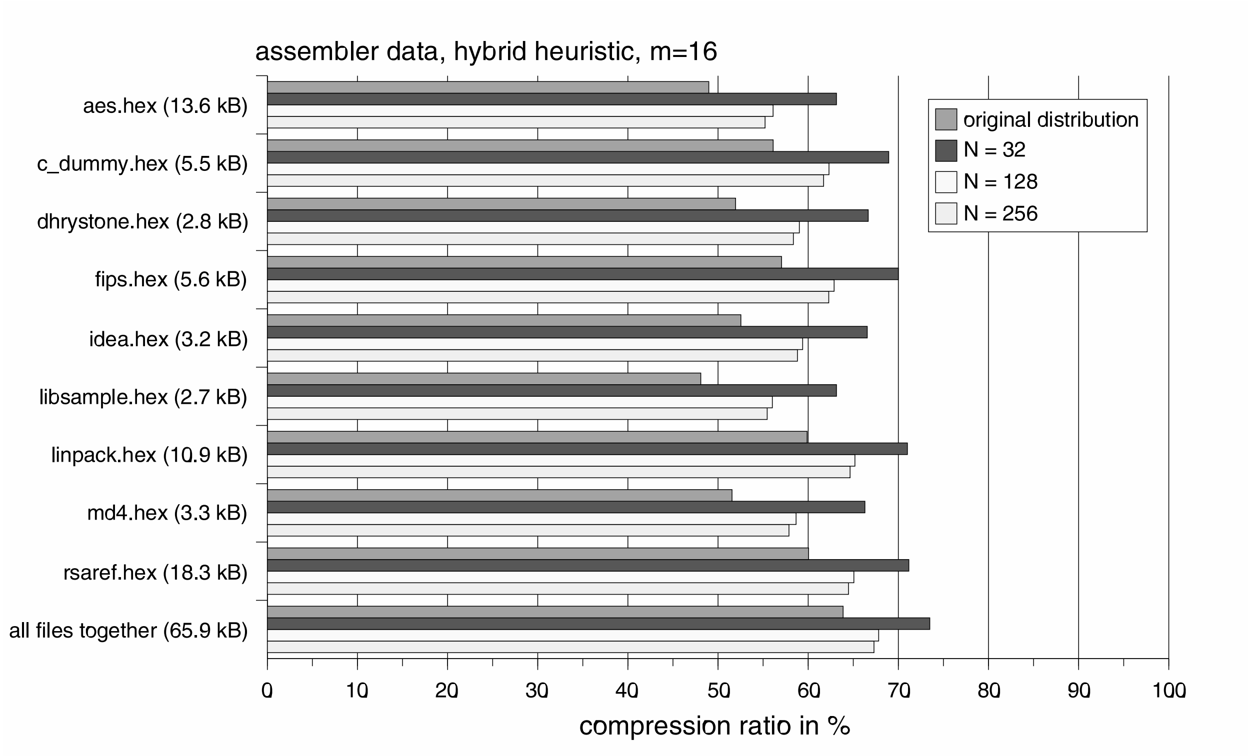


Abbildung 5.18: Vergleich der Kompressionsraten verschiedener Feinheiten bei $m = 16$ und hydrider Approximation

Werfen wir nun auch einen Blick auf die tatsächlichen Kompressionsraten. Dabei spielen nun natürlich nicht nur die relative Distanz der einzelnen Verteilungen, sondern auch deren Häufigkeiten ein Rolle. Da z.B. sehr unbalancierte Verteilungen mit $H(p) \approx 0$ in Wirklichkeit nicht so häufig vorkommen, wird deren großer Approximationsverlust auch nur geringe Auswirkungen haben. Abbildung 5.17 zeigt die Kompressionsraten der verschiedenen Ansätze zu einigen Beispieldateien. Insgesamt zeigt sich ein Kompressionsverlust gegenüber arithmetischer Kodierung mit optimaler Verteilung von vier bis acht Prozentpunkten. Weder Huffman-LI noch Shannon-LI erweisen sich dabei im Vergleich zu der anderen Heuristik als überlegen. Erwartungsgemäß liefert der hybride Ansatz jedoch noch eine Verbesserung um $0,5$ bis zu 1 Prozentpunkten.

Zu guter Letzt wollen wir aber auch noch die Auswirkungen einer alternativen Wahl der Feinheit N auf die Kompressionsrate betrachten. Die Ergebnisse können Abbildung 5.18 entnommen werden. Offensichtlich wirkt sich eine geringe Feinheit wie $N = 32$ sehr deutlich auf die Kompressionsrate aus. Abbildung 5.16 zeigt, dass dies in der verlustreichen Approximation durch 5-beschränkt dyadische Verteilungen begründet ist. Der Verlust steigt auf Werte zwischen 10% und 15%. Wählen wir eine größere Feinheit wie $N = 256$, so würde dies nach unseren Betrachtungen aus Abschnitt 5.4 mit der Kardinalität von $R_m(N)$ auch die Schaltkreisgröße ungefähr verdoppeln. Der zu erwartende Kompressionsgewinn von unter einem Prozentpunkt dürfte dies jedoch kaum rechtfertigen. Die Wahl einer Feinheit von $N = 128$ erweist sich also als guter Kompromiss.

5.7 Fazit

Wir haben in diesem Kapitel einen tiefeneffizienten Dekodierschaltkreis für tabellenbasierte, arithmetische Kodierung konkret vorgeschlagen. Der Ansatz stellt dabei eine Verallgemeinerung des binären Tabellenkodierers nach Howard und Vitter auf beliebige Alphabete dar. Wir haben uns dabei auf praxistaugliche Kenngrößen mit Feinheit $N = 128$ und einer Alphabetgröße $m = 16$ konzentriert.

Es konnte gezeigt werden, dass mittels Approximation durch geordnete, dyadische Verteilungen dessen Realisierung in Hardware, etwa auf modernen Smartcards, möglich ist. Der beobachtete Verlust gegenüber einer optimalen Komprimierung eines echten Smartcard-Anwendungsprogrammcodes beträgt dabei in der Regel unter acht Prozentpunkte. Eine Realisierung des Konzeptes wird im Rahmen des Projektes „Online Codekompression für Smartcards“ in Kooperation mit der Infineon Technologies AG München erwogen.

Wir konnten dabei durch eine Größenanalyse unseres Schaltkreises sowie kombinatorischen Betrachtungen über die Anzahl der auftretenden Verteilungen Tendenzen zu abweichenden Kenngrößen aufzeigen. Es hat sich gezeigt, dass bei geringerer Feinheit zwar die Schaltkreisgröße sinkt, jedoch starke Kompressionsverluste durch den Approximationsschritt zu erwarten sind. Andererseits würden größere Werte für N keine spürbare Steigerung der Kompressionsrate, jedoch eine Vervielfachung des Hardwarebedarfs bringen. Die Alphabetgröße m, welche sehr stark mit der zu Grunde liegenden Modellierung verbunden ist, erlaubt — wie im Rahmen des Projektes gezeigt — ab $m = 16$ gute Kompressionsraten. Wir konnten zeigen, dass eine Verdopplung von m trotz der Aussicht auf eine bessere Modellbildung allerdings auch eine Verdopplung der Schaltkreisgröße mit sich zieht.

Ferner konnten noch einige interessante Eigenschaften bei der Verwendung von dyadischen Verteilungen bei arithmetischer Kodierung bewiesen werden. Wir haben darüber hinaus zwei sehr einfache, heuristische Approximations-Algorithmen zur Bestimmung einer L-beschränkt dyadischen Verteilung mit minimaler zu erwartender Distanz zur Ausgangsverteilung vorgestellt und empirisch deren Gleichwertigkeit aufgezeigt. Obwohl wir keinen Approximationsfaktor bewiesen haben, legt deren Verhalten auf den betrachteten Kenngrößen eine annähernde Optimalität nahe. Nähere Untersuchungen dazu sowie Überlegungen bezüglich beweisbarer Schranken zum Approximationsverlust stellen bestimmt eine interessante Fortführung dieser Betrachtungen dar.

„I love deadlines. I specially love the swooshing sounds they make as they fly by.“

— Douglas Adams

Abbildungsverzeichnis

1.1 Varianten der Online Codekompression 2

3.1 Prinzip der arithmetischen Kodierung bei $\#\,\Gamma = 8$ 12
3.2 Prinzip der *SFE*-Kodierung bei $\#\,\Gamma = 8$ 13
3.3 Prinzip der rekursiven *SFE*-Kodierung bei $\Gamma = \mathbb{B}^3$, also $\#\,\Gamma = 8$ 17
3.4 Intuition zur kontextgesteuerten Modellbildung (Quelle: [BCW90]) 19
3.5 Prinzip der Codewort-Bestimmung bei *WNC*-Kodierung 21
3.6 Renormalisierung mittels *(a)* unterem, *(b)* oberem und *(c)* mittlerem Expansionsschritt 23
3.7 Arithmetische Kodierung als endlicher Automat 27
3.8 Auszug aus einer Kodiertabelle zur Feinheit $N = 8$ 28
3.9 Auszug aus einer verkleinerten Kodiertabelle zur Feinheit $N = 8$ 31
3.10 Binärer Unterteilungsschritt von $[0, 8)$ bei $\mathrm{prob}(X = MPS) \approx \frac{5}{8}$ 33
3.11 Auszug aus einer Dekodiertabelle zur Feinheit $N = 8$ 34

4.1 Struktur eines Verteilungsbaumes mit $m = 6$ 39
4.2 Beispiel eines komprimierten Verteilungsbaumes zu der Wahrscheinlichkeitsverteilung $p = \left(0, 0, 0, 0, \frac{1}{2}, 0, 0, 0, \frac{1}{8}, 0, \frac{1}{4}, 0, 0, 0, 0, \frac{1}{8}\right)$ 46
4.3 Skizze von komprimierten Verteilungsbäumen mit $k = 4$. *(a)* beschreibt den ungünstigsten Fall für die obere Schranke aus Satz 4.15, *(b)* den günstigsten Fall für die untere Schranke. 47
4.4 Bitmuster-Sequenz eines Binärbaumes. Der dargestellte Baum wird ebenenweise zu $s = 1\,11\,0111\,00101101$ codiert. 58
4.5 Zaks-Sequenz eines Binärbaumes. Der dargestellte Baum wird ebenenweise zu $s = 1\,11\,0111\,101101\,00000000$ codiert. 59
4.6 Einige Platzschranken für komprimierte Verteilungsbäume 61
4.7 Obere und untere Schranke der Anzahl gespeicherter Knoten 61

5.1 Skizze eines Dekodierschaltkreises mit Tabellen-Arithmetik 65
5.2 Kodierung einer Verteilung als binäre Zeichenkette für $k = 32$ und $m = 4$. 67
5.3 Kardinalität der Menge $P_m(N)$ 68
5.4 Kardinalität der Menge $Q_m(N)$ 70
5.5 Kardinalität der Menge $R_m(N)$ 73
5.6 Beispiel einer einzelnen Dekodiertabelle für $N = 128$ und $m = 16$ zum Zustand $[0, 128)$ und zur Verteilung $\left(\frac{1}{2}, \frac{1}{16}, \frac{1}{32}, \frac{1}{32}, \ldots, \frac{1}{32}, \frac{1}{32}\right)$ 80

5.7 Schaltkreis-Skizze einer Dekodiertabelle für $N = 128$ und $m = 16$ zum Zustand $[0, 128)$ und zur Verteilung $\left(\frac{1}{2}, \frac{1}{16}, \frac{1}{32}, \frac{1}{32}, \ldots, \frac{1}{32}, \frac{1}{32}\right)$ 82
5.8 Schaltkreis-Skizze des Statistik-Moduls für c Kontexte und $m = 16$ 82
5.9 Schaltkreis-Skizze des Ergebnis-Filters für d Verteilungen 84
5.10 Schaltkreis-Skizze des Index-Konverters für $m = 16$ 84
5.11 Größenschranken für den Dekodierschaltkreis 85
5.12 Latenz und Flächenbedarf des optimierten Schaltkreises 85
5.13 Kompressionsverlust eines Tabellendekodierers mit $N = 128$ und $m = 16$ bei der Huffman-LI-Approximation von p mit dyadischer Verteilung q. . . . 93
5.14 Kompressionsverlust eines Tabellendekodierers mit $N = 128$ und $m = 16$ bei der Shannon-LI-Approximation mit dyadischen Verteilungen. 94
5.15 Kompressionsverlust eines Tabellendekodierers mit $N = 128$ und $m = 16$ bei hybrider Approximation von p mit dyadischer Verteilung q. 95
5.16 Kompressionsverlust eines Tabellendekodierers mit $N = 32$ und $m = 16$ bei hybrider Approximation mit dyadischen Verteilungen. 96
5.17 Vergleich der Kompressionsraten verschiedener Approximations-Heuristiken bei $N = 128$ und $m = 16$ 97
5.18 Vergleich der Kompressionsraten verschiedener Feinheiten bei $m = 16$ und hydrider Approximation 98

Literaturverzeichnis

[Abr63] Norman Abramson. *Information Theory and Coding*. McGraw-Hill, 1963. (chapter notes).

[BCW90] Thimothy C. Bell, John G. Cleary, and Ian H. Witten. *Text Compression*. Prentice Hall Advanced Reference Series: Computer Science. Prentice-Hall, Inc., 1990.

[BK93] Abraham Bookstein and Shmuel T. Klein. Is huffman coding dead? *Computing*, 50:279–296, 1993.

[BSMM00] Ilja N. Bronstein, Konstantin A. Semendjajew, Gerhard Musiol, and Heiner Mühlig. *Taschenbuch der Mathematik*. Verlag Harry Deutsch AG, 5th edition, 2000.

[Cay81] Arthur Cayley. On the analytical forms called trees. *American Journal of Mathematics*, 4(1/4):266–268, 1881.

[CLR90] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. The MIT Press, 1990.

[CT91] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. John Wiley and Sons, Inc., 1991.

[Fan49] Robert M. Fano. Transmission of information. Technical Report 65, The Research Laboratory of Electronics, MIT, Cambridge, 1949.

[FO84] Philippe Flajolet and Andrew M. Odlyzko. Limit distributions for coefficients of iterates of polynomials with applications to combinatorial enumerations. *Mathematical Proceedings of the Cambridge Philosophical Society*, 96(237):237–253, 1984.

[GS95] Ira M. Gessel and Richard P. Stanley. Algebraic enumeration. In R. Graham, M. Grötschel, and L. Lovász, editors, *Handbook of Combinatorics*, volume 2, chapter 21, pages 1021–1069. Elsevier Science, 1995.

[Hop03] Ulrich Hoppe. Markovprozesse zur Modellierung bei Codekompression. Master's thesis, Universität Paderborn, Institut für Informatik, 2003.

[Huf52] David A. Huffman. A method for the construction of minimum redundancy codes. In *Proceedings of the Institute of Radio Engineers (IRE)*, volume 40(9), pages 1098–1101, 1952.

[HV92] Paul G. Howard and Jeffrey Scott Vitter. Practical implementations of arithmetic coding. Technical Report 92-18, Brown University, Department of Computer Science, 1992.

[Jac89a] Guy Jacobson. Space-efficient static trees and graphs. In *Proceedings of the 30th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 549–554, 1989.

[Jac89b] Guy Jacobson. Succinct static data structures. Technical Report CMU-CS-89-112, Carnegie Mellon University, School of Computer Science, 1989.

[Jel68] Frederick Jelinek. *Probabilistic Information Theory*. McGraw-Hill, 1968.

[Knu71] Donald E. Knuth. Optimal binary search trees. *Acta Informatica*, 1:14–25, 1971.

[Kra49] Leon G. Kraft. A device for quantizing, grouping, and coding amplitude modulated pulses. Master's thesis, Department of Electrical Engineering, MIT, Cambridge, 1949.

[Kun03] Daniel Kuntze. Tabellenbasierte arithmetische Kodierung. Bachelor's Thesis, Universität Paderborn, Institut für Informatik, 2003.

[LH90] Lawrence L. Larmore and Daniel S. Hirschberg. A fast algorithm for optimal length-limited huffman codes. *Journal of the ACM (JACM)*, 37(3):464–473, 1990.

[LLW86] C.C. Lee, D.T. Lee, and C.K. Wong. Generating binary trees of bounded height. *Acta Informatica*, 23(5):529–544, 1986.

[LR86] Glen G. Langdon and Jorma J. Rissanen. Compression of black-white images with arithmetic coding. *IEEE Transactions on Communications*, COM-29(6):858–867, 1986.

[LW99] Haris Lekatsas and Wayne Wolf. SAMC: A code compression algorithm for embedded processors. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 18(12):1689–1701, 1999.

[Mac03] David MacKay. *Information Theory, Inference and Learning Algorithms*. Cambridge University Press, 2003.

[McM56] Brockway McMillan. Two inequalities implied by unique decipherability. *IEEE Transactions on Information Theory*, IT-2:115–116, 1956.

[ML97] Ruy L. Milidiu and Eduardo S. Laber. Improved bounds on the inefficiency of lengthrestricted prefix codes. Technical Report Monografias em Ciencia da Computac ao 33/97, Departamento de Inform'atica, PUC-RJ, Brasil, 1997.

[MNW98] Alistair Moffat, Radford M. Neal, and Ian H. Witten. Arithmetic coding revisited. *ACM Transactions on Information Systems*, 16(3):256–294, 1998.

[MR97] J. Ian Munro and Venkatesh Raman. Succinct representation of balanced parentheses, static trees and planar graphs. In *Proceedings of the 38th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 118–126, 1997. (corrected version of 1999).

[Odl84] Andrew M. Odlyzko. Some new methods and results in tree enumeration. Congressus Numerantium 42, 27-52, 1984.

[Pas76] Richard C. Pasco. *Source Coding Algorithms for Fast Data Compression*. PhD thesis, Department of Electrical Engeneering, Stanford University, 1976.

[Ris76] Jorma J. Rissanen. Generalized kraft inequality and arithmetic coding. *IBM Journal of Research and Development*, 20:198–203, 1976.

[Say00] Khalid Sayood. *Introduction to Data Compression*. Academic Press, 2nd edition, 2000.

[Sed88] Robert Sedgewick. *Algorithms*. Addison-Wesley, 2nd edition, 1988.

[Sha48] Claude E. Shannon. A mathematical theory of communication. *Bell Systems Technical Journal*, 27, 1948. (reprinted version with corrections).

[WNC87] Ian H. Witten, Radford M. Neal, and John G. Cleary. Arithmetic coding for data compression. *Communications of the ACM*, 30(6):520–540, 1987.

[Zak80] Shmuel Zaks. Lexicographic generation of ordered trees. *Theoretical Computer Science*, 10:63–82, 1980.