DIE VERWENDUNG HIERARCHISCH STRUKTURIERTER SPRACHNETZWERKE ZUR REDUNDANZARMEN CODIERUNG VON TEXTEN

Vom Fachbereich 19
der Technischen Hochschule Darmstadt
zur Erlangung der Würde eines
Doktor-Ingenieurs (Dr.-Ing.)
genehmigte Dissertation

Dipl.-Ing.
Jochen Meyer
geboren am 19. Februar 1960

Referent: Prof. Dr.-Ing. W. Hilberg
Korreferent: Prof. Dr. rer. nat. W. Endres
Tag der Einreichung: 10. 2. 1989
Tag der mündlichen Prüfung: 16. 6. 1989

D 17
Darmstädter Dissertation

Diese Arbeit entstand während meiner Tätigkeit als wissenschaftlicher Mitarbeiter am Institut für Datentechnik der Technischen Hochschule Darmstadt.

Mein besonderer Dank gilt Herrn Prof. Dr.-Ing. W. Hilberg für die interessante Themenstellung, für seine fruchtbaren Anregungen und Diskussionen sowie die stetige Förderung dieser Arbeit.

Ebenso danke ich Herrn Prof. Dr. rer. nat. W. Endres für sein großes Interesse an der Arbeit, für seine wertvollen fachlichen und persönlichen Hinweise und für die Übernahme des Korreferats.

Ein herzliches Dankeschön auch an die Herren Dipl.-Ing. Kurt Siedenburg und Dipl.-Ing. Michael Lipp für die Durchsicht des Rohmanuskripts sowie an alle Kollegen, die mich mit ihrer Kritik und zahlreichen Anregungen unterstützten.

Eine zeitweilige finanzielle Förderung dieser Arbeit wurde dankenswerterweise von der Deutschen Forschungsgemeinschaft gewährt.

Diese Arbeit ist meinen Eltern gewidmet, die mir die wesentlichen Voraussetzungen für meine Berufsausbildung boten.

Inhaltsverzeichnis

1 Einleitung

Sprache spielt bei der zwischenmenschlichen Kommunikation eine wesentliche Rolle. Damit eine gute Verständigung auch bei Umweltgeräuschen oder undeutlicher Aussprache gewährleistet ist, enthalten sprachliche Äußerungen weglaßbare Elemente, die keine zusätzlichen Informationen liefern, sondern lediglich die Grundinformation stützen. Auch in Büchern und Handschriften bewirken überflüssige Informationen eine meist zufriedenstellende Lesbarkeit, selbst wenn einige Buchstaben oder Wörter schlecht gedruckt oder geschrieben wurden. Diesen Sachverhalt kann man demonstrieren, indem man alle Vokale eines Satzes wegläßt: N∗rm∗l∗rw∗∗s∗ b∗r∗∗t∗t ∗s k∗∗n∗ Schw∗∗r∗gk∗∗t∗n, ∗∗n∗n S∗tz, b∗∗ d∗m ∗ll∗ V∗k∗l∗ f∗hl∗n, z∗ ∗ntz∗ff∗rn.

Nachrichtentechniker bezeichnen die überflüssigen Anteile einer Nachricht bekanntlich als „Redundanz“. Bei einem „gestörten Übertragungskanal“ erhöhen redundante Informationen die Wahrscheinlichkeit, daß gestörte Nachrichten komplett rekonstruiert werden können. Dazu muß der Empfänger (beispielsweise ein Leser des obigen Satzes) wissen, wie „normale“ Nachrichten aufgebaut sind, um eventuelle Fehler erkennen und korrigieren zu können.

In manchen Fällen ist die in Nachrichten enthaltene Redundanz allerdings unerwünscht: Zum Beispiel wird der Speicherplatzbedarf beim elektronischen Speichern von Texten durch die Textredundanz unnötig vergrößert. Prinzipiell ist es möglich, diese überflüssigen Informationen durch eine geeignete Codierung zu eliminieren, so daß man mit einen Bruchteil des ursprünglichen Speicherplatzes auskommt.

Für eine redundanzarme Codierung benötigt man ein Codierungssystem, das einen Speicher enthält, in dem „Wissen“ darüber gespeichert ist, in welcher Weise eine Nachricht aufgebaut wird: Zum Beispiel könnte in einem Codierungssystem für Texte ein Wörterbuch benutzt werden, in dem alle Wörter einer Sprache enthalten sind. Zusätzlich könnte man noch abspeichern, welche Wörter aufeinanderfolgen dürfen, d. h. man speichert die „Assoziationen“ der Wörter. Beim Codieren werden damit von vornherein unmögliche Buchstaben- und Wortkombinationen außer Betracht gelassen, wodurch eine Redundanzreduktion erreicht wird.

In der vorliegenden Arbeit wird ein solches „assoziatives“ Codierungssystem für Texte beschrieben, das nach Vorschlägen von W. Hilberg aufgebaut wurde. Wesentlicher Bestandteil dieses Systems sind hierarchisch angeordnete Sprachnetzwerke, die in einer „Lernphase“ anhand eines Lerntextes aufgebaut werden und in denen „Sprachwissen“ gespeichert ist. Mit dieser hierarchischen Struktur, die auch „semantischer Speicher“ genannt wird, können beim Codieren von Texten mittlere Codelängen in der Größenordnung von einigen Millibit pro Buchstabe erreicht werden.

Da die Hilberg'sche Speicherstruktur nicht nur speziell für Texte konzipiert wurde, sondern für beliebige sequentielle Signale, die wegen starker innerer Bindungen einen hohen Anteil an Redundanz aufweisen, sind neben der Textcodierung auch andere

Anwendungen für einen semantischen Speicher denkbar: Beispielsweise könnte man in der neuen Speicherhierarchie typische Signalmuster abspeichern, um anschließend eine Mustererkennung durchzuführen. Derartige Erkennungsprobleme treten z. B. bei der Spracherkennung oder bei der Bildverarbeitung auf, so daß der semantische Speicher möglicherweise auch hier vorteilhaft eingesetzt werden kann. Zunächst soll die neue Speicherorganisation allerdings nur in Hinblick auf eine redundanzarme Textcodierung untersucht werden.

Vor der Beschreibung des realisierten semantischen Speichers, mit dem man Texte redundanzarm codieren kann, wird zuerst das für die Untersuchungen verwendete Textmaterial vorgestellt. Danach werden die Ergebnisse einiger Textanalysen präsentiert, anhand derer sichergestellt werden soll, daß das Codierungssystem mit den Gegebenheiten der zu speichernden Texte harmoniert. Außerdem werden einige konventionelle Textcodierungsverfahren (z. B. Optimalcodierung) untersucht, damit Vergleiche mit dem neuen Verfahren ermöglicht werden.

Die vorliegende Arbeit entstand während meiner Tätigkeit als wissenschaftlicher Mitarbeiter am Institut für Datentechnik der Technischen Hochschule Darmstadt. Da es sich bei der gewählten Thematik weitgehend um wissenschaftliches „Neuland“ handelt, konnte hierfür lediglich auf die theoretischen Überlegungen von Prof. W. Hilberg [41]–[70] zurückgegriffen werden. Die Anwendung des vorgeschlagenen semantischen Speichers für einen Textcodierer zeigte jedoch, daß die Realisierung große Schwierigkeiten bereitete. Im Zuge der Messungen und Simulationen, deren Ergebnisse ich jeweils in Institutsberichten [73]–[84] zusammenfaßte, konnten diese Schwierigkeiten schließlich nacheinander beseitigt werden, wobei auch zahlreiche Studien- und Diplomarbeiter [86]–[114], die mit der Lösung von Teilproblemen beauftragt wurden, beteiligt waren.

2 Die für die Untersuchungen verwendeten Texte

Für die im nächsten Kapitel beschriebenen Textanalysen wird umfangreiches Textmaterial benötigt. Im Rahmen der noch relativ jungen wissenschaftlichen Disziplin der „linguistischen Datenverarbeitung“ entstand eine Reihe von Textsammlungen in maschinenlesbarer Form, die als „Standardtexte“ oder „Textkorpora“ bezeichnet werden [1]. Ein derartiges Textkorpus besteht in der Regel aus einem repräsentativen Querschnitt von Texten einer bestimmten Sprache, eines speziellen Fachgebietes oder eines Autors.

Aus dem Bonner Institut für Kommunikationsforschung und Phonetik stammt das sogenannte „LIMAS-Korpus“, das aus verschiedenen deutschen Texten der Gegenwart (z. B. Trivialliteratur, Zeitungstexte, Romane, Fachaufsätze etc.) aufgebaut ist [2]. Die Konzeption des LIMAS-Korpus folgt dem Vorbild des englischen „BROWN-Corpus“, weil damals (ca. 1970) neben linguistischen Untersuchungen auch Probleme der automatischen Übersetzung angegangen werden sollten. Genau wie der englische Standardtext enthält das LIMAS-Korpus 500 Textstellen, wobei jeder dieser Textauszüge aus etwa 2 000 Wortformen besteht. Daraus resultiert ein Gesamtumfang von ca. 10^6 Wortformen, unter denen etwa 116 000 *verschiedene* enthalten sind. Für das gesamte LIMAS-Korpus werden etwa 8 Megabyte Speicherplatz benötigt.

Ein Textausschnitt des LIMAS-Korpus ist in folgendem Bild wiedergegeben.

```
*BA *BT *NR +GRUNDBEGRIFFE DER +DATENVERARBEITUNG *ET *BK +ZUR          159*040875
+TERMINILOGIE DER +BEGRIFFE '' +COMPUTER '' UND ''                      159*040876
+DATENVERARBEITUNGSANLAGE '' *EK +WER EINEN +BLICK IN DIE               159*040877
UMFANGREICHE BESCHREIBENDE +LITERATUR DES +FACHGEBIETS *BH              159*040878
+DATENVERARBEITUNG *EH WIRFT , DER WIRD FESTSTELLEN , WIE GROS$ DIE    159*040879
+ZAHL VON UNTERSCHIEDLICHEN +AUSDRU$CKEN IST , DIE FAST SYNONYN ZUR    159*040880
+BEZEICHNUNG DES +COMPUTERS VERWENDET WERDEN . +SELBST IN DEN          159*040881
+MELDUNGEN DER +PRESSE , DIE SICH NICHT IN ERSTER +LINIE AN            159*040882
+FACHLEUTE , SONDERN AN BREITE +LESERSCHICHTEN WENDEN , IST DIE        159*040883
UNEINHEITLICHE +VERWENDUNG ZAHLREICHER , MITEINANDER KONKURRIERENDER   159*040884
+BEGRIFFE FESTZUSTELLEN . +SO FINDEN WIR NEBENEINANDER ETWA *BH        159*040885
+RECHENANLAGE , +RECHNER , +RECHENAUTOMAT , +COMPUTER ,                159*040886
+DATENVERARBEITUNGSANLAGE , +DATENVERARBEITUNGSSYSTEM , +E+D+V *TK     159*040887
+ANLAGE , +D+V *TK +ANALGE , +D+V *TK +SYSTEM *EH UND VIELE            159*040888
+AUSDRU$CKE MEHR , DIE EIGENTLICH ALLE DASSELBE BEZEICHNEN WOLLEN UND 159*040889
NUR IN DEN SELTENSTEN +FA$LLEN BEWUS$T DEM EINEN ODER ANDEREN +ASPEKT 159*040890
ZUR +GELTUNG VERHELFEN , DER URSPRU$NGLICH IN DEN VERSCHIEDENEN       159*040891
+NAMEN ZUM +AUSDRUCK GEBRACHT WERDEN SOLLTE . +AM +ANFANG STAND       159*040892
ZWEIFELLOS DER +BEGRIFF '' +COMPUTER '' , DER IN +AMERIKA GEPRA$GT     159*040893
WURDE UND VOM ENGLISCHEN +VERB *BH TO COMPUTE *EH ( = RECHNEN )       159*040894
HERGELEITET IST . +DAS +BESTREBEN , DIESEN IN UNSERER +SPRACHE ALS    159*040895
```

Bild 2.1: Ausschnitt aus dem LIMAS-Korpus [2]. (Druckfehler wurden nicht verbessert.)

Am rechten Zeilenrand befinden sich Quellen- und Zeilen-Numerierungen: Anhand der Quellen-Nummer können dem ausführlichen Titelverzeichnis des LIMAS-Korpus

Informationen über einzelne Texte (Titel, Herkunft etc.) entnommen werden: Zum Beispiel stammt die Quelle 159 (siehe Bild 2.1) aus einem populärwissenschaftlichen Buch über Computer.

Bei der Durchsicht des obigen Textausschnittes fallen mehrere Dinge auf, wobei es sich hier um folgende Konventionen handelt, die eine linguistische Auswertung erleichtern:

- Es werden nur große Buchstaben verwendet. Zum Beispiel wird die Wortform „der“ als Zeichenfolge „`DER`“ dargestellt.
- Im Klartext groß geschriebene Buchstaben werden mit Plus-Zeichen kenntlich gemacht: Zum Beispiel lautet die Zeichenfolge „`+COMPUTER`“ im Klartext „Computer“.
- Umlaute werden mit Dollar-Zeichen dargestellt. Beispielsweise codiert man „ä“ und „ß“ mit Hilfe der Zeichenfolgen „`A$`“ und „`S$`“.
- Zwischen den Satzzeichen und den ihnen vorangehenden Wortformen befindet sich jeweils ein Leerzeichen. So wird z. B. „ist,“ umgesetzt zu „`IST␣,`“. Hierbei handelt es sich um eine in der linguistischen Datenverarbeitung übliche Konvention [3], durch die eine einfache Erkennung der Wortgrenzen möglich ist: Alle Zeichen zwischen zwei aufeinanderfolgenden Leerzeichen werden als isolierte Wortform behandelt.
- Es treten recht häufig kryptische Abkürzungen auf, die alle mit einem Sternchen beginnen. Hiermit werden einerseits Satzzeichen mit mehreren Bedeutungen (z. B. Gedankenstriche, Bindestriche, Trennstriche etc.) unterschieden; der Bindestrich für Komposita wird beispielsweise mit „`*TK`“ codiert. Auf der anderen Seite werden bestimmte, für linguistische Auswertungen wichtige Textteile abgegrenzt, wie z. B. mit „`*BT`“ und „`*ET`“ eine Titelüberschrift gekennzeichnet wird.

Die Entwicklung des neuen Textcodierers basiert auf der Vorstellung eines lernenden Systems, d. h. das System eignet sich automatisch das für die Codierung notwendige „Sprachwissen“ an, indem es einen „Lerntext durchliest“. Diese Lernphase kann prinzipiell mit den Vorgängen verglichen werden, die in einem Kind ablaufen, wenn es lesen lernt: In der Regel legt man Kindern einfache Kinderbücher vor und überfordert sie nicht mit zu schwieriger Literatur. Aufgrund dieser Überlegung wurde als Lerntext für das Codierungssystem zuerst ein besonders einfacher Text, nämlich ein Lesebuch des ersten Schuljahres [4] bereitgestellt, wobei dieser Text aus etwa 5 000 Wortformen besteht. Ein Auszug dieses Lesebuchs, das unter Beachtung der Konventionen des LIMAS-Korpus (siehe oben) in maschinenlesbare Form überführt wurde, ist in Bild 2.2 wiedergegeben. Während der Entwicklung des Codierungssystems zeigte es sich jedoch, daß man zum Laden des Sprachspeichers wesentlich größere Textmengen als dieses Lesebuch benötigt: Um signifikante Ergebnisse zu erhalten, wurde dann schließlich wieder auf das LIMAS-Korpus zurückgegriffen.

```
+DER SU$S$E +BREI .

+ES WAR EINMAL EIN ARMES FROMMES +MA$DCHEN , DAS LEBTE MIT SEINER +MUTTER
ALLEIN , UND SIE HATTEN NICHTS MEHR ZU ESSEN . +DA BEGEGNETE IHM EINE ALTE
+FRAU , DIE WUS$TE SEINEN +JAMMER SCHON UND SCHENKTE IHM EIN +TO$PFCHEN . +ZU
DEM SOLLT' ES SAGEN : " +TO$PFCHEN KOCH ! " , SO KOCHTE ES GUTEN SU$S$EN
+HIRSEBREI , UND WENN ES SAGTE : " +TO$PFCHEN STEH ! " , SO HO$RTE ES WIEDER
AUF ZU KOCHEN . +DAS +MA$DCHEN BRACHTE DEN +TOPF SEINER +MUTTER HEIM , UND NUN
BRAUCHTEN SIE KEINEN +HUNGER MEHR ZU ERLEIDEN , DENN SIE AS$EN SU$S$EN +BREI ,
SOOFT SIE WOLLTEN . +EINMAL WAR DAS +MA$DCHEN AUSGEGANGEN , DA SPRACH DIE
+MUTTER : " +TO$PFCHEN KOCH ! " +DA KOCHTE ES , UND SIE AS$ SICH SATT . +NUN
WOLLTE SIE , DAS$ DAS +TO$PFCHEN WIEDER AUFHO$REN SOLL , ABER SIE WUS$TE DAS
+WORT NICHT . +ALSO KOCHTE ES FORT , UND DER +BREI STIEG U$BER DEN +RAND HINAUS
UND KOCHTE IMMERZU , DIE +KU$CHE UND DAS GANZE +HAUS VOLL UND DAS ZWEITE +HAUS
UND DANN DIE +STRAS$E , ALS WOLLT ES DIE GANZE +WELT SATT MACHEN , UND DIE
GRO$S$TE +NOT ENTSTAND , UND KEIN +MENSCH WUS$TE SICH DA ZU HELFEN . +ENDLICH ,
ALS NUR NOCH EIN EINZIGES +HAUS U$BRIG WAR , DA KAM DAS +KIND HEIM UND SPRACH
NUR : " +TO$PFCHEN STEH ! " , DA STAND ES UND HO$RTE AUF ZU KOCHEN . +UND WER
WIEDER IN DIE +STADT WOLLTE , DER MUS$TE SICH DURCHESSEN .
```

Bild 2.2: Ausschnitt aus dem Lesebuch [4].

In den folgenden Ausführungen wird nicht der in der Linguistik übliche Begriff „Wortform“, mit dem man die Beugungsformen der Wörter bezeichnet [3], verwendet, sondern der einfachere und etwas unschärfere Ausdruck „Wort“, um schwerfällige Konstruktionen wie „Wortformenhäufigkeiten“ oder „Wortformenbuch“ zu vermeiden.

3 Statistische Untersuchungen von Texten

Es werden zunächst die klassischen Textuntersuchungen von Zipf, Shannon und Küpfmüller kurz in Erinnerung gerufen, wobei es sich hierbei um Messungen der Häufigkeitsverteilung von Wörtern sowie des mittleren Informationsgehaltes von Texten handelt. Da diese Analysen für die Entwicklung des neuen Codierungssystems allerdings nicht ausreichten, wurden eigene, umfangreiche Textuntersuchungen durchgeführt, deren Ergebnisse im Anschluß an die klassischen Untersuchungen präsentiert werden. Im Rahmen dieser neuen Experimente wurde z. B. die Häufigkeitsverteilung von Wortpaaren gemessen.

3.1 Die Häufigkeitsverteilung von Wörtern

G. K. Zipf untersuchte Ende der vierziger Jahre dieses Jahrhunderts u. a. die Häufigkeitsverteilungen der Wörter in Texten [6]. Da diese Zipf'schen Experimente anhand des LIMAS-Korpus [2] nachvollzogen werden sollen, wird angegeben, wie man die Häufigkeitsverteilung von Wörtern bestimmen kann:

1. Zuerst muß ein Text ausgewählt werden, der untersucht werden soll.
2. Dann wird eine Liste angelegt, die alle *verschiedenen* Wörter dieses Textes enthält.
3. Zu jedem Wort dieser Liste wird bestimmt, wie oft es in dem Text vorkommt.
4. Danach sortiert man die Liste so, daß die Häufigkeiten der Wörter von vorne nach hinten abnehmen.
5. Die Wörter werden entsprechend ihrem Listenplatz durchnumeriert.
6. Jedem Wort in der Liste sind demnach zwei Parameter zugeordnet:
   a) die Listen-Nummer oder Ordnungszahl n und
   b) die absolute Häufigkeit $a(n)$.

   Die Wertepaare aller Wörter werden schließlich in ein Diagramm eingetragen, dessen Achsen logarithmisch geteilt sind.

Für das LIMAS-Korpus wurde die Tabelle nach obiger Anleitung ermittelt. Der Anfang dieser Liste ist in Tabelle 3.1 zu sehen. Bei dieser Liste fällt auf, daß auch Satzzeichen als „Wörter“ behandelt werden. Das hängt mit der automatischen Auswertung zusammen, bei der eine Zeichenfolge zwischen zwei aufeinanderfolgenden Leerzeichen als ein Wort interpretiert wird.

Werden die Wertepaare der vollständigen Tabelle nun wie oben beschrieben in ein Diagramm eingetragen, dann erhält man den in Bild 3.1 gezeigten Zusammenhang.

Ordnungszahl n	Wort	Häufigkeit $a(n)$	Ordnungszahl n	Wort	Häufigkeit $a(n)$
1	,	70 610	11	DES	9 608
2	.	50 925	12	IST	9 303
3	DER	37 229	13	DAS	9 055
4	DIE	33 013	14	MIT	8 592
5	UND	28 341	15	SICH	8 436
6	''	17 746	16	NICHT	7 906
7	IN	17 206	17	AUF	7 678
8	DEN	12 335	18	*TK	6 923
9	VON	11 034	19	DEM	6 610
10	ZU	10 856	20	EINE	6 597

Tabelle 3.1: Die 20 häufigsten Wörter des LIMAS-Korpus [2].

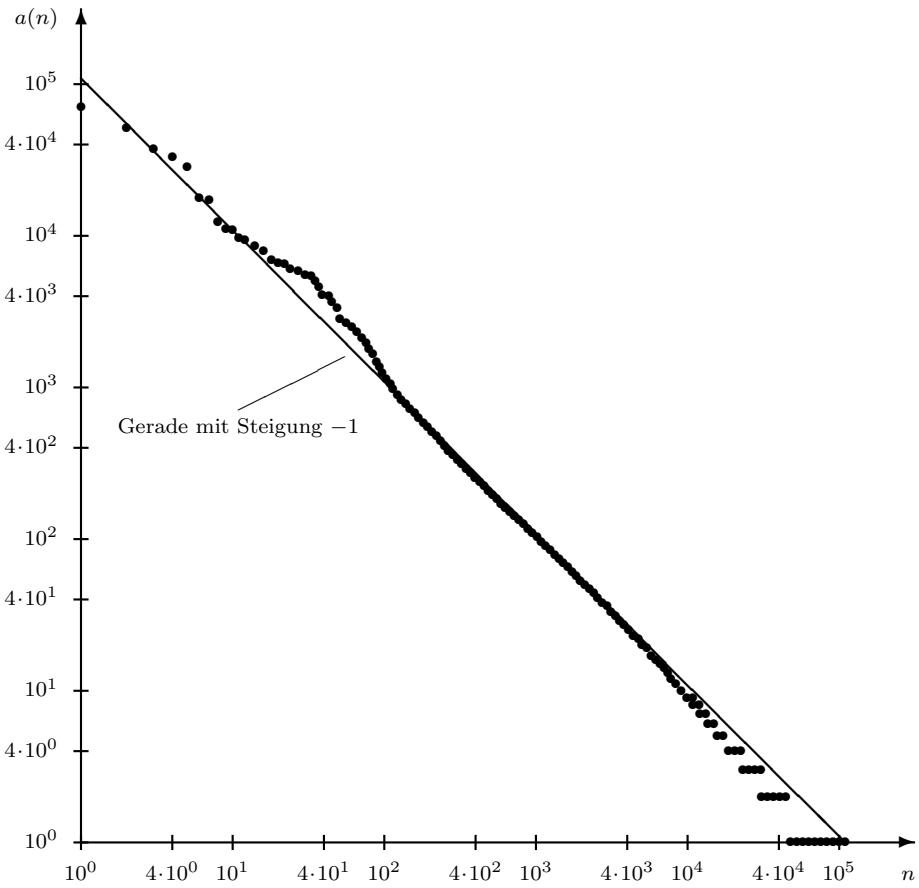

Bild 3.1: Die absoluten Häufigkeiten $a(n)$ der sortierten Wörter des LIMAS-Korpus in Abhängigkeit ihrer Ordnungszahlen n (Zipf'sches Gesetz [6]).

Zunächst einmal ist erstaunlich, daß die Wertepaare näherungsweise auf einer Geraden liegen. Außerdem fällt auf, daß diese Gerade mit der Steigung -1 abfällt. Zipf untersuchte zahlreiche Texte verschiedener Sprachen und Autoren und mußte überrascht feststellen, daß dieser Zusammenhang für *beliebige* Texte gilt. Bis heute hat man keine Erklärung für dieses eigentümliche Phänomen und ermittelt selbst für Sprachen, die dem Englischen und Deutschen völlig fremd sind, (z. B. für die chinesische Sprache) immer wieder den gleichen Zusammenhang [71], der im folgenden als „Zipf'sches Gesetz" bezeichnet wird.

Der in Bild 3.1 gezeigte lineare Zusammenhang soll nun mathematisch erfaßt werden:

Gegeben sei ein Text, der aus s Wörtern besteht (beispielsweise das LIMAS-Korpus mit $s \approx 10^6$). Darunter sind insgesamt n_0 verschiedene Wörter enthalten (z. B. $n_0 \approx 116\,000$). Die Summe der absoluten Häufigkeiten $a(n)$ aller Wörter muß identisch sein mit der Textlänge

$$s = \sum_{n=1}^{n_0} a(n). \tag{3.1}$$

Anhand der Geraden in Bild 3.1 bestimmt man

$$\log\Big(a(n)\Big) + \log(n) = \text{const} = \log(n_0). \tag{3.2}$$

Dabei kann die Konstante in guter Näherung durch $\log(n_0)$ ersetzt werden. Durch Auflösen dieser Gleichung nach $a(n)$ berechnet man dann

$$a(n) = \frac{n_0}{n} \qquad \text{mit } 1 \le n \le n_0. \tag{3.3}$$

Setzt man dieses Ergebnis in Gleichung (3.1) ein, so ergibt sich

$$s = \sum_{n=1}^{n_0} \frac{n_0}{n} = n_0 \sum_{n=1}^{n_0} \frac{1}{n}. \tag{3.4}$$

Die Summe in dieser Gleichung kann für große n_0 näherungsweise durch ein Integral ersetzt werden (siehe auch [72]). Man bestimmt

$$s \approx n_0 \int_1^{n_0} \frac{\mathrm{d}n}{n} = n_0 \cdot \Big[\ln n\Big]_1^{n_0} \qquad \text{für } n_0 \gg 1. \tag{3.5}$$

Schließlich erhält man den angenäherten Zusammenhang zwischen der Textlänge s und der Anzahl n_0 der verschiedenen Wörter zu

$$s \approx n_0 \ln n_0 \qquad \text{für } n_0 \gg 1. \tag{3.6}$$

Neben der Zipf'schen Näherung (siehe Bild 3.1) existieren noch weitere, genauere Approximationen der Worthäufigkeiten [7, 8]:

M. Joos geht bei seiner Näherung zwar auch von einem linearen Zusammenhang im Diagramm mit logarithmisch geteilten Achsen (siehe Bild 3.1) aus, setzt jedoch die

Steigung der Geraden mit $-(1+\varepsilon)$ an, wobei die Konstante ε eine kleine positive Zahl ist [7]. Aus Gleichung (3.3) wird dann

$$a_{\mathrm{j}}(n) = \left(\frac{n_0}{n}\right)^{(1+\varepsilon)} \qquad \text{mit } 0 < \varepsilon \ll 1. \tag{3.7}$$

B. Mandelbrot gibt eine noch genauere Approximation an, indem er die für kleine n festgestellten Abweichungen der Joos'schen Näherung von der tatsächlichen Häufigkeitsverteilung berücksichtigt [7, 8]. Das kann erreicht werden durch geeignete Wahl der Konstanten δ in der Gleichung

$$a_{\mathrm{m}}(n) = \left(\frac{n_0}{n+\delta}\right)^{(1+\varepsilon)} \qquad \text{mit } \delta > 0 \text{ und } 0 < \varepsilon \ll 1. \tag{3.8}$$

Im folgenden reicht jedoch die Zipf'sche Näherung (Gleichung (3.3)) aus, da es nicht so sehr auf eine genaue quantitative Beschreibung der Häufigkeitsverteilung, sondern lediglich auf ihren prinzipiellen Verlauf ankommt.

3.2 Die klassischen Untersuchungen der Text-Entropie

Nach C. E. Shannon kann der mittlere Informationsgehalt H (Entropie) einer Nachrichtenquelle mit Hilfe folgender Gleichung berechnet werden, wobei $p(n)$ die relative Häufigkeit des n-ten Nachrichtensymbols ist und insgesamt N verschiedene Symbole existieren [9, 10]:

$$H = -\sum_{n=1}^{N} p(n)\ \mathrm{ld}\, p(n) \qquad \text{(in bit/Symbol)}. \tag{3.9}$$

Geht man z. B. von einem Alphabet mit 27 Zeichen (26 Buchstaben und einem Leerzeichen) aus, die alle gleichhäufig vorkommen, so bestimmt man mit obiger Gleichung (die Bedeutung des dabei verwendeten Index 0 wird weiter unten noch erklärt)

$$H_0 = -27 \cdot \frac{1}{27} \cdot \mathrm{ld}\, \frac{1}{27} \approx 4{,}75 \text{ bit/Buchstabe}. \tag{3.10}$$

Unter Berücksichtigung der wirklichen Buchstabenhäufigkeiten englischer Texte berechnete Shannon mit Gleichung (3.9) einen Wert von $H_1 = 4{,}03$ bit/Buchstabe [11].

In Gleichung (3.9) werden die Abhängigkeiten zwischen aufeinanderfolgenden Buchstaben außer Betracht gelassen. Man errechnet damit stets einen zu großen Wert für die Entropie, da durch die Bindungen der Buchstaben untereinander ein wesentlicher Anteil an Redundanz enthalten ist. Mit Hilfe von Buchstabenketten (sogenannten „k-Grammen“) und deren Häufigkeitsverteilungen ermittelte Shannon genauere Ergebnisse, wobei er folgende Gleichung verwendete [11]:

$$H_k = -\sum_{m=1}^{M}\sum_{n=1}^{N} p(m_{k-1}, n)\ \mathrm{ld}\, p(n|m_{k-1}) \qquad \text{(in bit/Symbol)}. \tag{3.11}$$

Dabei bedeuten m_{k-1} die m-te Symbolkette, die aus $(k-1)$ Nachrichtensymbolen besteht, M die Anzahl der verschiedenen Ketten, $p(m_{k-1}, n)$ die relative Häufigkeit, mit der die Kombination aus der m-ten Kette und dem n-ten Symbol auftritt (siehe z. B. [12]: dort sind die Häufigkeiten der deutschen „Digramme“ wiedergegeben), und $p(n|m_{k-1})$ die bedingte oder *Übergangs*wahrscheinlichkeit, mit der das n-te Symbol auf die m-te Kette folgt. Shannon bestimmte mit Hilfe von Gleichung (3.11) die Werte $H_2 = 3{,}32$ bit/Buchstabe und $H_3 = 3{,}1$ bit/Buchstabe.

Die mittleren Informationsgehalte H_4, H_5 etc. konnte Shannon nicht auf diese Art berechnen, da die Tabellen mit den Häufigkeiten zu umfangreich und damit nicht verfügbar waren. Allerdings war zu dieser Zeit die Zipf'sche Näherung für die Häufigkeitsverteilung der Wörter [6] schon bekannt, die Shannon in Gleichung (3.9) einsetzte und damit die Entropie der englischen Wörter zu $H_\mathrm{w} = 11{,}82$ bit/Wort oder (auf Buchstaben bezogen) $H_\mathrm{w} = 2{,}14$ bit/Buchstabe ermittelte.

Schließlich führte Shannon noch Experimente durch, bei denen er Versuchspersonen den Anfang eines Textes vorlegte und die nächsten Buchstaben oder Wörter voraussagen ließ. Aufgrund dieser Untersuchungen schätzte er ab, daß unter Berücksichtigung *sämtlicher* Bindungen, die zwischen den Buchstaben und Wörtern englischer Texte vorhanden sind, der untere Grenzwert der Entropie etwa $H_\mathrm{g} = 1$ bit/Buchstabe betragen müßte. Anhand des nächsten Bildes erkennt man, wie die Entropie H_k mit wachsendem k, d. h. mit zunehmender Berücksichtigung der inneren Bindungen, dem Grenzwert H_g zustrebt.

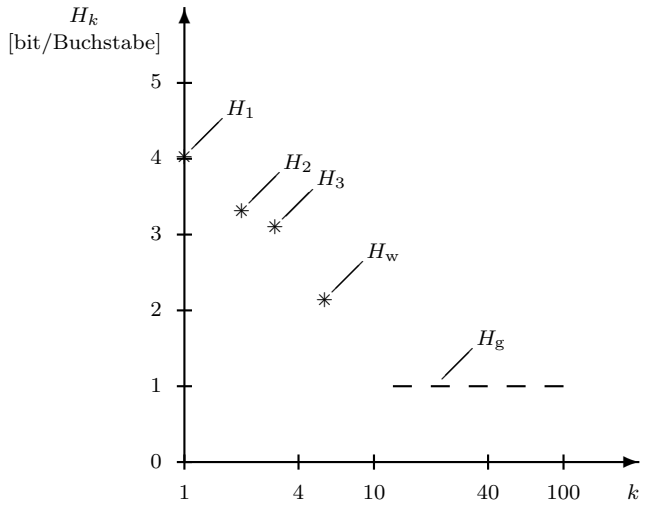

Bild 3.2: Näherungswerte für die Entropie H_k der englischen Schriftsprache, wobei alle Bindungen innerhalb von Ketten aus k Buchstaben berücksichtigt wurden. Mit H_w wird die Entropie von Wörtern und mit H_g der Shannon'sche Grenzwert bezeichnet (nach Werten aus [11]; vergleiche auch mit [13]).

K. Küpfmüller, der Shannons Untersuchungen für die deutsche Sprache nachvollzog, ermittelte ganz ähnliche Werte und erstellte mit diesen ein Diagramm, das nur geringfügig von Bild 3.2 abweicht [13].

In jüngster Zeit werden allerdings die Grenzwerte, die sowohl Shannon als auch Küpfmüller für den redundanzfreien mittleren Informationsgehalt von englischen und deutschen Texten angegeben haben, angezweifelt: W. Hilberg äußerte den berechtigten Einwand, bei den klassischen Experimenten habe man nur verhältnismäßig kurze Texte verwendet, so daß *semantische* Zusammenhänge gar nicht in die Messungen eingehen konnten. Wenn man diese inhaltlichen Bindungen allerdings von vornherein außer Betracht lassen will und lediglich *formale*, d. h. grammatikalische und syntaktische Abhängigkeiten berücksichtigt, dann liegen die klassischen Grenzwerte, bei denen es sich wie erwähnt um Schätzwerte handelt, sicher in der richtigen Größenordnung: Hilberg konnte nämlich mit Hilfe der „deterministischen Informationstheorie“ einen unteren Grenzwert für die Entropie von Texten *berechnen*, der mit den klassischen Ergebnissen gut übereinstimmt [42, 43, 48, 49]:

$$H_{\mathrm{h}} = \frac{\mathrm{ld}\, N}{4} \approx 1{,}19 \text{ bit/Buchstabe} \qquad \text{mit } N = 27. \tag{3.12}$$

In diese Gleichung muß man lediglich die Größe des verwendeten Alphabets (z. B. $N = 27$ Zeichen: 26 Buchstaben und ein Leerzeichen) einsetzen.

Stellt man sich dagegen hypothetische Entropie-Messungen mit sehr langen Texten (z. B. mit ganzen Romanen oder sogar mit Romansammlungen eines Autors) vor, so wird man mit Hilfe dieser außerordentlich aufwendigen Experimente auch *semantische* Abhängigkeiten erfassen können. Dann wäre aber der Shannon'sche Grenzwert lediglich ein Zwischenergebnis und die „wirkliche“ Entropie (bezogen auf Buchstaben) würde schließlich bei beliebig langen Texten asymptotisch gegen einen sehr kleinen Wert gehen (siehe dazu auch Bild 3.2). Die durchgeführten Codierungsexperimente, die in den folgenden Kapiteln noch beschrieben werden, zeigen jedenfalls, daß es prinzipiell möglich ist, den „Grenzwert“ von 1 bit/Buchstabe zu unterschreiten.

3.3 Die Häufigkeitsverteilungen von Vorgänger- und Nachfolgerwörtern

Zipf hat bei seinen Experimenten die absoluten Häufigkeiten $a(n)$ bestimmt [6]. Bei diesen Häufigkeiten gehen Abhängigkeiten zwischen den Wörtern nicht ein. Man kann jedoch über die Zipf'schen Untersuchungen hinausgehen und anstatt der Worthäufigkeiten die Häufigkeiten der „Assoziationen“ zu Wörtern berücksichtigen. Eine erste Möglichkeit besteht darin, die Bindungen zwischen aufeinanderfolgenden Wörtern mit einer bedingten Häufigkeitsverteilung $a_{\mathrm{n}}(m|n)$ zu erfassen, die angibt, wie oft das m-te Wort in einem Text unter der Bedingung, daß es *nach* dem n-ten Wort steht, vorkommt. Dabei wird das m-te Wort innerhalb aller Nachfolgerwörter des n-ten Wortes gezählt. Für alle n_0 Wörter des Wortvorrates existiert jeweils eine Häufigkeitsverteilung $a_{\mathrm{n}}(m|n)$ mit $n = \text{const}$. Da es sehr aufwendig ist, alle n_0 Verteilungen zu bestimmen, wurden stichprobenartig einige Wörter ausgewählt, um auf diese Art einen Überblick über die Häufigkeitsverteilungen der

Nachfolgerwörter zu erhalten. Die Ergebnisse dieser Untersuchungen wurden in dem Institutsbericht [76] zusammengefaßt. Nach dem gleichen Verfahren konnten auch die Häufigkeiten $a_{\mathrm{v}}(m|n)$ der Vorgängerwörter ermittelt werden, wobei dafür diejenigen Wörter erfaßt wurden, die im untersuchten Text jeweils *vor* einem bestimmten Wort stehen.

3.3.1 Die Häufigkeitsverteilungen der Nachfolgerwörter

In Bild 3.3 sind die absoluten Häufigkeiten $a_{\mathrm{n}}(m|8)$ aller Wörter wiedergegeben, die im LIMAS-Korpus auf das Wort „DEN“ folgen. Nach Tabelle 3.1 gilt für dieses Wort $n = 8$. Es fällt auf, daß die Häufigkeiten in Bild 3.3 entlang einer Geraden angesammelt sind, wobei diese Gerade, die nach der Methode der kleinsten Fehlerquadrate [14] eingezeichnet wurde, im Vergleich zu derjenigen in Bild 3.1 etwas flacher verläuft. Bild 3.4 zeigt die Häufigkeiten $a_{\mathrm{n}}(m|16)$ der Nachfolger von „NICHT“ ($n = 16$), die um eine Gerade mit der Steigung $-0{,}95$ verstreut liegen.

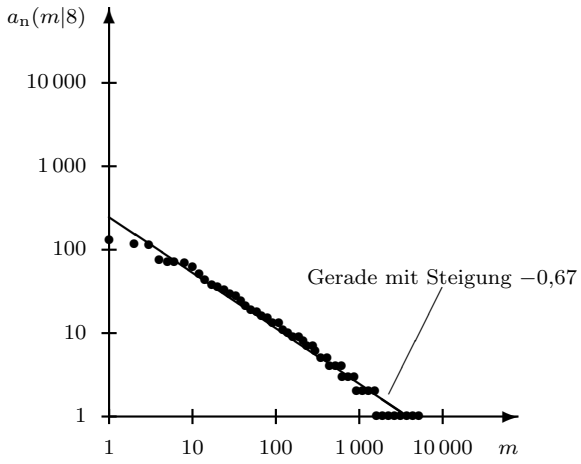

Bild 3.3: Die Häufigkeiten $a_{\mathrm{n}}(m|8)$ der Nachfolger von „DEN“ ($n = 8$).

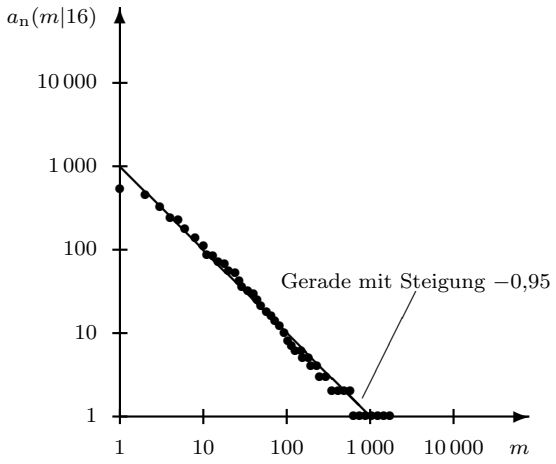

Bild 3.4: Die Häufigkeiten $a_{\mathrm{n}}(m|16)$ der Nachfolger von „NICHT“ ($n = 16$).

Die Häufigkeitsverteilungen der Nachfolger der anderen untersuchten Wörter verlaufen im Diagramm mit logarithmisch geteilten Achsen ebenfalls annähernd linear, wobei die Geradensteigungen alle im Bereich um -1 liegen [76]. Die negativen Werte dieser Steigungen, die in den folgenden Berechnungen mit $\alpha(n)$ bezeichnet werden, sind in Bild 3.5 wiedergegeben.

Die Häufigkeit $a_{\mathrm{n}}(m|n)$ kann ähnlich wie bei Gleichung (3.3) direkt aus den Bildern 3.3 und 3.4 angesetzt werden zu

$$a_{\mathrm{n}}(m|n) = \left(\frac{v_{\mathrm{n}}(n)}{m}\right)^{\alpha(n)} \qquad \text{mit } 1 \le m \le v_{\mathrm{n}}(n). \tag{3.13}$$

Dabei wird mit $v_{\mathrm{n}}(n)$ die später noch zu bestimmende Anzahl der verschiedenen Wörter, die im untersuchten Text auf das n-te Wort folgen, bezeichnet. Diese Anzahl wird auch „Verzweigungsgrad“ genannt.

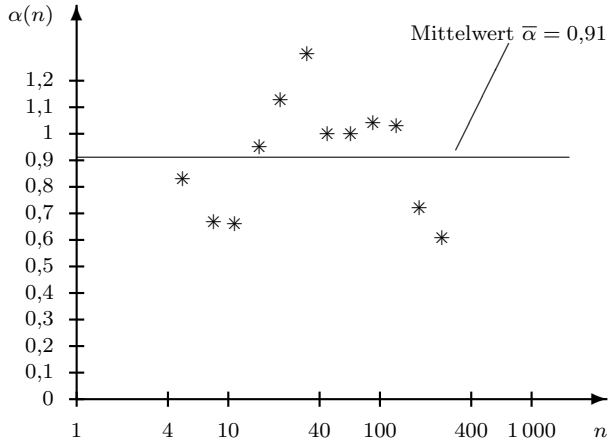

Bild 3.5: Die negativen Werte $\alpha(n)$ der Geradensteigungen.

Der Mittelwert von $\alpha(n)$ beträgt $\overline{\alpha} = 0{,}91$ (siehe Bild 3.5). Alle Werte $\alpha(n)$ sind relativ dicht um diesen Wert gestreut, so daß man in erster Näherung $\alpha(n) = 1$ in Gleichung (3.13) einsetzen kann. Damit ergibt sich eine der Gleichung (3.3) ganz ähnliche Beziehung für die Häufigkeiten der Nachfolgerwörter

$$a_{\mathrm{n}}(m|n) = \frac{v_{\mathrm{n}}(n)}{m} \qquad \text{mit } 1 \le m \le v_{\mathrm{n}}(n). \tag{3.14}$$

3.3.2 Die Häufigkeitsverteilungen der Vorgängerwörter

Entsprechend den Häufigkeiten $a_{\mathrm{n}}(m|n)$ wurden die Häufigkeiten $a_{\mathrm{v}}(m|n)$ bestimmt, indem nun die *Vorgänger*wörter untersucht wurden. Die Bilder 3.6 und 3.7 geben die Häufigkeiten der Vorgängerwörter von „DEN“ und „NICHT“ wieder (vergleiche auch mit den Bildern 3.3 und 3.4).

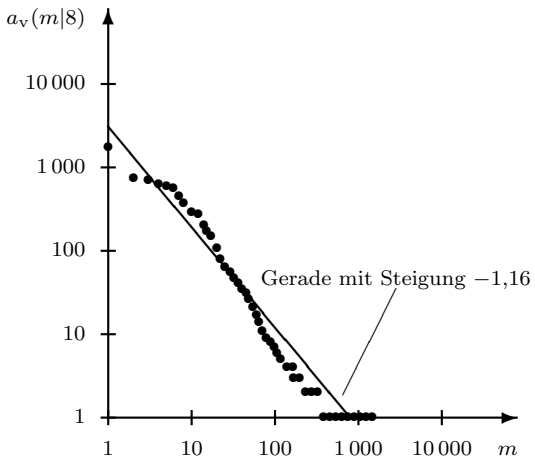

Bild 3.6: Die Häufigkeiten $a_{\mathrm{v}}(m|8)$ der Vorgänger von „DEN“ ($n = 8$).

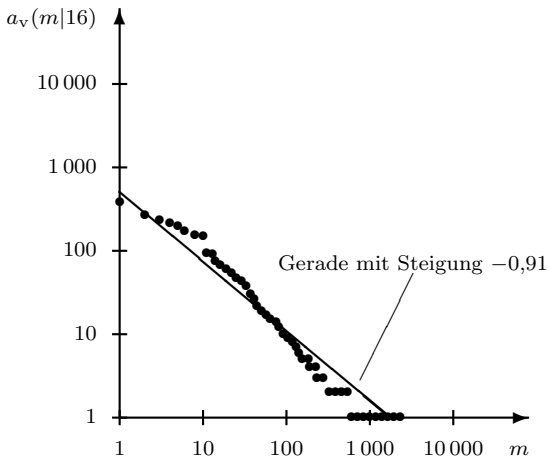

Bild 3.7: Die Häufigkeiten $a_{\mathrm{v}}(m|16)$ der Vorgänger von „NICHT“ ($n = 16$).

Auch hier sind die Häufigkeitsverteilungen annähernd Geraden, wobei die Steigungen wieder um den Wert -1 gestreut liegen. Die negativen Geradensteigungen, die mit $\beta(n)$ bezeichnet werden, sind im folgenden Bild zu sehen [76].

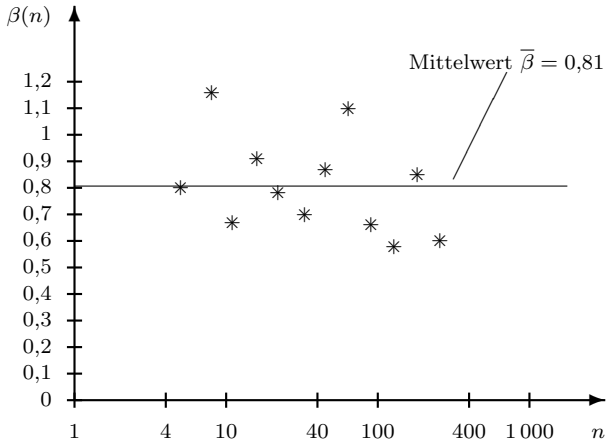

Bild 3.8: Die negativen Werte $\beta(n)$ der Geradensteigungen.

Der Mittelwert von $\beta(n)$ ist mit $\overline{\beta} = 0{,}81$ etwas kleiner als derjenige von $\alpha(n)$. Hier soll ebenfalls die Näherung $\beta(n) = 1$ verwendet werden. Mit der im nächsten Abschnitt erläuterten Anzahl $v_{\mathrm{v}}(n)$ der verschiedenen Vorgängerwörter des n-ten Wortes berechnet man entsprechend den Gleichungen (3.13) und (3.14)

$$a_{\mathrm{v}}(m|n) = \left(\frac{v_{\mathrm{v}}(n)}{m}\right)^{\beta(n)} = \frac{v_{\mathrm{v}}(n)}{m} \qquad \text{mit } \beta(n) = 1 \text{ und } 1 \le m \le v_{\mathrm{v}}(n). \tag{3.15}$$

3.4 Der Verzweigungsgrad von Wörtern

Im Rahmen der Untersuchungen von $a_{\mathrm{n}}(m|n)$ und $a_{\mathrm{v}}(m|n)$ wurde die Anzahl $v_{\mathrm{n}}(n)$ der verschiedenen Nachfolger für einige ausgewählte Wörter bestimmt (siehe den Institutsbericht [76]), außerdem die Anzahl $v_{\mathrm{v}}(n)$ der verschiedenen Vorgänger. Diese Werte werden im folgenden auch „Verzweigungsgrade“ genannt.

3.4.1 Die gemessene Verzweigungsgradverteilung der Wörter

In Bild 3.9 sind die Meßwerte $v_{\mathrm{n}}(n)$ und $v_{\mathrm{v}}(n)$ sowie die Häufigkeiten $a(n)$ eingetragen, wobei $v_{\mathrm{n}}(n)$ mit einem „$\times$“, $v_{\mathrm{v}}(n)$ mit einem „$+$“ und $a(n)$ mit einem „$\bullet$“ gekennzeichnet ist. Zur Orientierung ist außerdem eine Gerade mit der Steigung -1 eingezeichnet. Die Anzahl $v_{\mathrm{n}}(n)$ der Nachfolger und die Anzahl $v_{\mathrm{v}}(n)$ der Vorgänger streuen um einen gemeinsamen Mittelwert, der ungefähr parallel zu $a(n)$ verläuft. Aus diesem Grund wird im folgenden nur noch der Verzweigungsgrad $v(n)$ verwendet, der sowohl die Anzahl der Nachfolger als auch die Anzahl der Vorgänger bezeichnet:

$$v(n) = v_{\mathrm{n}}(n) = v_{\mathrm{v}}(n). \tag{3.16}$$

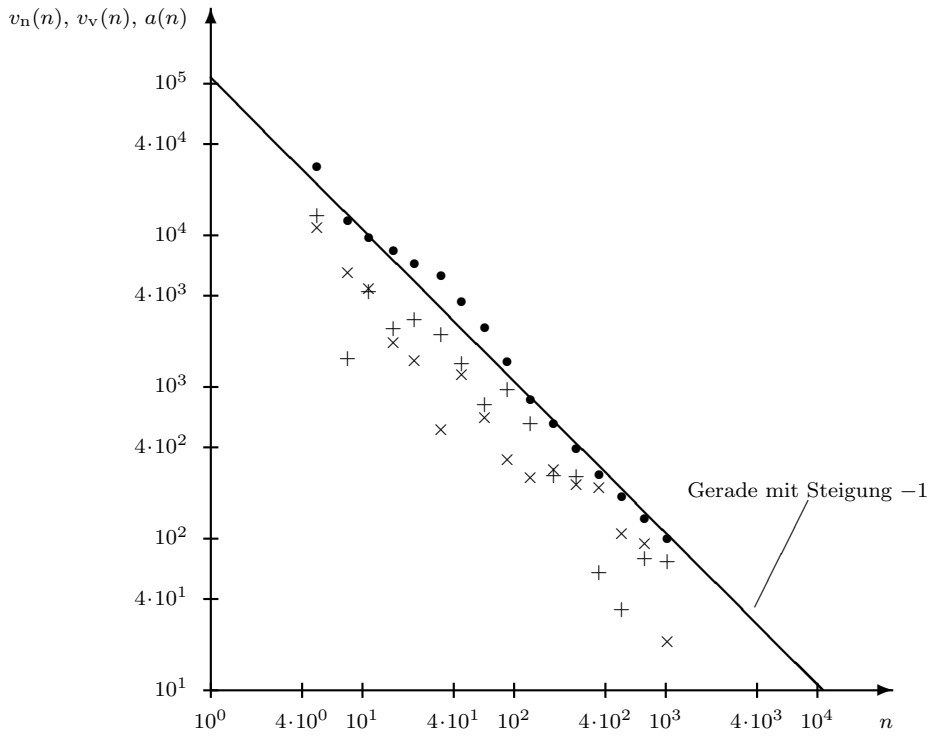

Bild 3.9: Gegenüberstellung der Meßwerte der Anzahl $v_n(n)$ der verschiedenen Nachfolgerwörter ($\hat{=}$ „$\times$“), der Anzahl $v_v(n)$ der verschiedenen Vorgängerwörter ($\hat{=}$ „$+$“) und der absoluten Häufigkeiten $a(n)$ ($\hat{=}$ „$\bullet$“).

Für jedes Wort gilt demnach in erster Näherung, daß die Anzahl seiner Nachfolger mit der Anzahl seiner Vorgänger übereinstimmt, d. h. die *Verzweigungen* der Wörter symmetrisch sind. Wegen $\alpha(n) = \beta(n) = 1$ (siehe oben) stimmen dann auch die Häufigkeitsverteilungen $a_n(m|n)$ und $a_v(m|n)$ überein. Mit den Gleichungen (3.14) und (3.15) berechnet man die bedingte Häufigkeit $a(m|n)$, mit der das m-te Wort im untersuchten Text neben dem n-ten Wort steht:

$$a(m|n) = \frac{v(n)}{m} \qquad \text{mit } 1 \le m \le v(n). \tag{3.17}$$

Im folgenden soll mit Hilfe der idealisierten Häufigkeitsverteilungen eine Näherung des Verzweigungsgrades $v(n)$ rechnerisch ermittelt werden.

3.4.2 Die berechnete Verzweigungsgradverteilung der Wörter

Voraussetzung bei der Berechnung von $v(n)$ sei, daß sich die Häufigkeitsverteilungen eines Textes ideal nach den Gleichungen (3.3) und (3.17) verhalten. Die einzige Unbekannte in diesen Gleichungen ist die Verzweigungsgradverteilung $v(n)$. Nach folgendem Ansatz kann $v(n)$ bestimmt werden: Die Summe der Häufigkeiten $a(m|n)$

aller Nachfolger eines Wortes muß identisch sein mit der absoluten Häufigkeit $a(n)$ dieses Wortes, d. h. es gilt mit Gleichung (3.17)

$$a(n) = \sum_{m=1}^{v(n)} a(m|n) = \sum_{m=1}^{v(n)} \frac{v(n)}{m} = v(n) \sum_{m=1}^{v(n)} \frac{1}{m}. \tag{3.18}$$

Für große $v(n)$ kann die Summe in dieser Gleichung näherungsweise in ein Integral überführt werden (vergleiche auch mit Gleichung (3.5)). Man berechnet

$$a(n) \approx v(n) \int_{1}^{v(n)} \frac{\mathrm{d}m}{m} = v(n) \cdot \Big[\ln m\Big]_1^{v(n)} \qquad \text{für } v(n) \gg 1. \tag{3.19}$$

Daraus erhält man (vergleiche auch mit Gleichung (3.6))

$$a(n) \approx v(n) \ln v(n) \qquad \text{für } v(n) \gg 1. \tag{3.20}$$

Es ist nicht möglich, diese Gleichung explizit nach $v(n)$ aufzulösen. Mit Hilfe eines einfachen Iterationsverfahrens können allerdings einzelne Funktionswerte berechnet werden. Als ersten Näherungswert für $v(n)$ setzt man beispielsweise $a(n)$ ein: Mit Gleichung (3.3) gilt dann

$$v_1(n) = a(n) = \frac{n_0}{n}. \tag{3.21}$$

Die i-te Näherung bestimmt man nach Gleichung (3.20) zu

$$v_i(n) = \frac{a(n)}{\ln v_{i-1}(n)} = \frac{n_0}{n \cdot \ln v_{i-1}(n)}. \tag{3.22}$$

Die Werte $v_i(n)$ kann man numerisch für wachsende i sehr einfach berechnen. Dabei zeigt sich, daß $v_i(n)$ schnell gegen einen konstanten Wert konvergiert, sofern die Bedingung $a(n) \gg 1$ erfüllt ist. In Bild 3.10 sind, mit „$*$“ bezeichnet, einige auf diese Art berechnete Werte von $v(n)$ für $n_0 = 10^5$ (entspricht etwa dem Wortvorrat des LIMAS-Korpus) eingetragen. Man erkennt anhand dieses Bildes, daß der berechnete Verzweigungsgrad $v(n)$ annähernd linear im Diagramm mit logarithmisch geteilten Achsen verläuft, wobei die Steigung der Näherungsgeraden $-0{,}8$ beträgt. Dieser Wert gilt allerdings nur für Wortvorräte in der Größenordnung von $n_0 = 10^5$, da die berechneten Werte von $v(n)$ entlang einer leicht gekrümmten Linie liegen, so daß für größere Wortvorräte die Näherungsgerade etwas steiler abfällt.

Mit der negativen Geradensteigung ψ kann man nun $v(n)$ wie folgt ansetzen (siehe dazu auch Bild 3.10 und Gleichung (3.3)):

$$v(n) = \Big(a(n)\Big)^{\psi} = \left(\frac{n_0}{n}\right)^{\psi} \qquad \text{mit } 1 \le n \le n_0. \tag{3.23}$$

Für Wortvorräte in der Größenordnung des LIMAS-Korpus bestimmt man mit der Geradensteigung aus Bild 3.10, d. h. mit $\psi = 0{,}8$ dann

$$v(n) = \left(\frac{n_0}{n}\right)^{0{,}8} \qquad \text{mit } 1 \le n \le n_0 \text{ und } n_0 \approx 10^5. \tag{3.24}$$

Wegen der zugrundegelegten Idealisierungen und Approximationen gilt dieser Zusammenhang selbstverständlich nur näherungsweise. Für die weiteren Untersuchungen und die Entwicklung des Codierungssystems ist mit dieser Gleichung jedoch der prinzipielle Verlauf des Verzweigungsgrades $v(n)$ zufriedenstellend erfaßt.

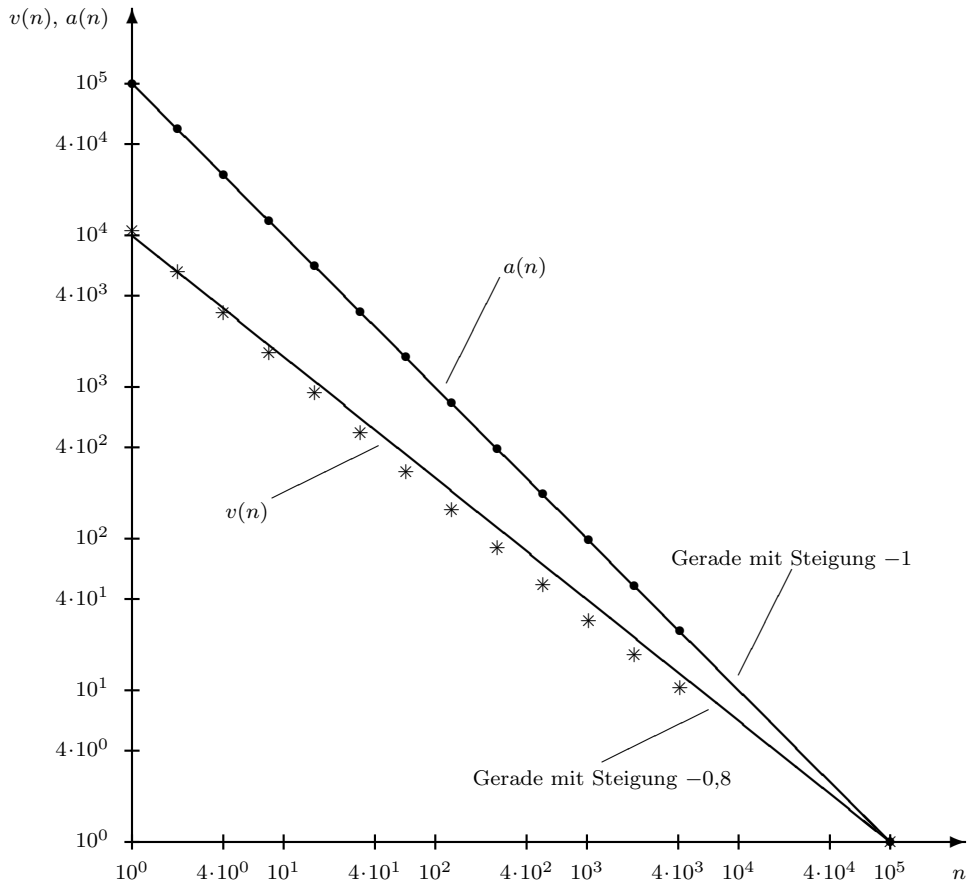

Bild 3.10: Berechnete Werte für die Verzweigungsgradverteilung $v(n)$ ($\widehat{=}$ „$*$“) und für die Häufigkeiten $a(n)$ ($\widehat{=}$ „$\bullet$“) mit $n_0 = 10^5$ (siehe auch die Gleichungen (3.21) und (3.22) sowie Gleichung (3.3)).

3.5 Die Häufigkeitsverteilungen von Wortfolgen

Zipf mußte für seine Untersuchungen der Worthäufigkeiten einen erheblichen Aufwand treiben, da ihm die Mittel der heutigen Computertechnik nicht zur Verfügung standen. Zur Bestimmung der nach Häufigkeiten sortierten Wortliste, deren Anfang Tabelle 3.1 zeigt, anhand eines Textes mit etwa 10^6 Wörtern benötigt man heutzutage dagegen nur knapp 10 CPU-Minuten auf einer VAX 8530, so daß es keine allzugroßen Schwierigkeiten bereitet, „erweiterte Zipf-Untersuchungen“ durchzuführen, die vor 40 Jahren noch jeden Rahmen gesprengt hätten: Zum Beispiel konnten in jüngeren Untersuchungen, deren Ergebnisse in den Institutsberichten [77, 78] zusammengefaßt sind, jetzt auch verschiedene statistische Verteilungen häufiger Wortfolgen sowie die Häufigkeitsverteilung von Wortpaaren ermittelt werden.

3.5.1 Die gemessene Häufigkeitsverteilung von Wortpaaren

Die Häufigkeitsverteilung $a_{\text{paar}}(n)$ der Wortpaare wurde entsprechend den Worthäufigkeiten $a(n)$ anhand einer Liste, in der die nach Häufigkeiten sortierten Wort-

paare eines Textes eingetragen wurden, bestimmt (vergleiche auch mit Tabelle 3.1). In dem folgenden Bild sind die Häufigkeiten $a_{\text{paar}}(n)$ der Wortpaare des gesamten LIMAS-Korpus aufgetragen, wobei auch hier wieder ein linearer Zusammenhang festgestellt werden kann. Allerdings verläuft die Gerade in diesem Bild, die wieder nach der Methode der kleinsten Fehlerquadrate [14] (lineare Regression) ermittelt wurde, mit einer Steigung von $-0{,}71$ flacher als diejenige in Bild 3.1.

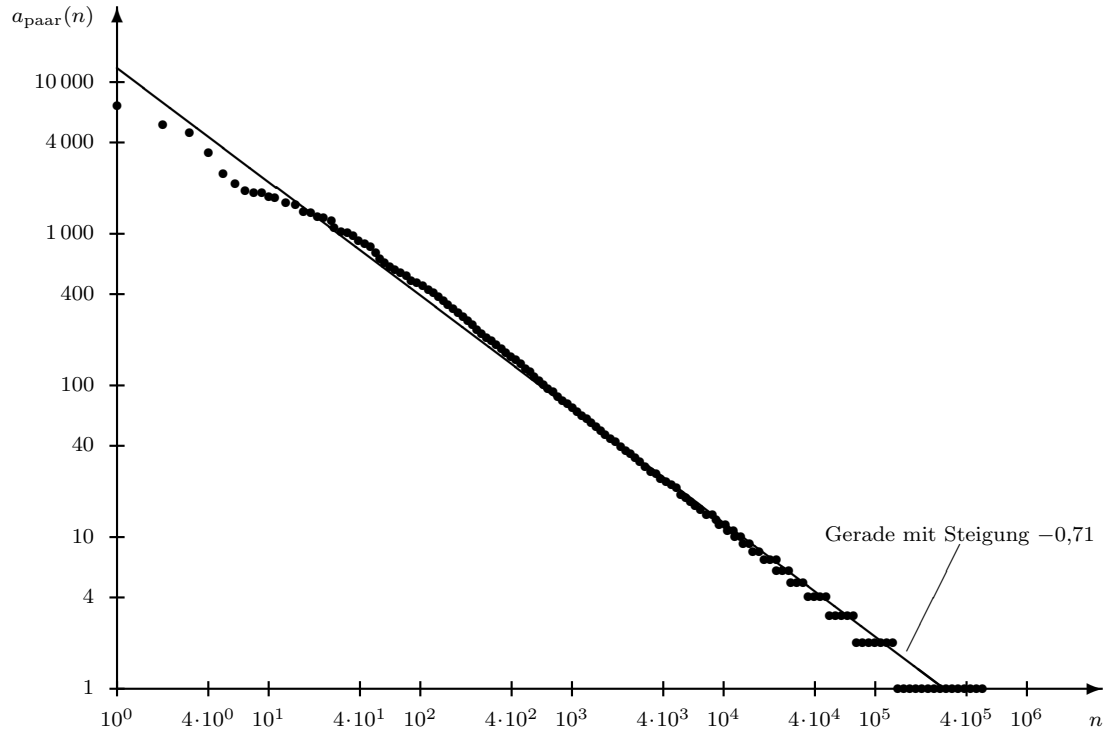

Bild 3.11: Die absoluten Häufigkeiten $a_{\text{paar}}(n)$ der sortierten Wortpaare des LIMAS-Korpus in Abhängigkeit ihrer Ordnungszahlen n.

Zur Untermauerung dieser Ergebnisse wurde die beschriebene Untersuchung für mehrere verschieden lange Textausschnitte wiederholt [78]. Dabei stellte sich heraus, daß die Häufigkeiten $a_{\text{paar}}(n)$ im logarithmischen Diagramm immer annähernd linear verlaufen (siehe Bild 3.11) und die Geradensteigungen im Bereich zwischen $-0{,}63$ und $-0{,}71$ liegen. Mit zunehmender Textlänge werden die Geraden geringfügig steiler. Anhand der Ergebnisse der vorangegangenen Untersuchungen kann die Verteilung $a_{\text{paar}}(n)$ auch rechnerisch ermittelt werden.

3.5.2 Die berechnete Häufigkeitsverteilung von Wortpaaren

Wegen des linearen Zusammenhangs in Bild 3.11 kann $a_{\text{paar}}(n)$ wie folgt angesetzt werden (vergleiche auch mit Gleichung (3.23)):

$$a_{\text{paar}}(n) = \left(\frac{n_{\text{paar}}}{n}\right)^{\vartheta} \qquad \text{mit } 1 \le n \le n_{\text{paar}}. \tag{3.25}$$

Dabei entspricht ϑ dem negativen Wert der Geradensteigung in Bild 3.11 und n_{paar} der Anzahl der verschiedenen Wortpaare, die im untersuchten Text vorkommen.

Summiert man die Verzweigungsgrade $v(n)$ aller n_0 Wörter auf, so erhält man die Anzahl n_{paar} der Wortpaare. Unter Berücksichtigung von Gleichung (3.23) und der Näherung der Summe durch ein Integral berechnet man

$$\begin{aligned} n_{\text{paar}} &= \sum_{n=1}^{n_0} v(n) = \sum_{n=1}^{n_0} \left(\frac{n_0}{n}\right)^{\psi} = n_0^{\psi} \sum_{n=1}^{n_0} \frac{1}{n^{\psi}} \approx n_0^{\psi} \int_1^{n_0} n^{-\psi}\,\mathrm{d}n \\ &= n_0^{\psi} \cdot \left[\frac{1}{1-\psi} \cdot n^{(1-\psi)}\right]_1^{n_0} \qquad \text{für } \psi \neq 1 \text{ und } n_0 \gg 1. \end{aligned} \tag{3.26}$$

Das Ergebnis lautet

$$n_{\text{paar}} \approx \frac{n_0^{\psi}}{1-\psi} \cdot \left(n_0^{(1-\psi)} - 1\right) \qquad \text{für } \psi \neq 1 \text{ und } n_0 \gg 1. \tag{3.27}$$

Setzt man in diese Gleichung die Werte des LIMAS-Korpus ein ($n_0 = 10^5$ und $\psi = 0{,}8$), so berechnet man

$$n_{\text{paar}} \approx \frac{10^{5 \cdot 0{,}8}}{1-0{,}8} \cdot \left(10^{5 \cdot 0{,}2} - 1\right) = \frac{10^4}{0{,}2} \cdot 9 = 450\,000. \tag{3.28}$$

Dieser Wert stimmt mit dem gemessenen Wert von $n_{\text{paar}} = 534\,714$ in der Größenordnung überein.

Für die Berechnung des Exponenten ϑ benötigt man die Häufigkeit $a_{\text{paar}}(1)$ des häufigsten Wortpaares, die identisch ist mit der Häufigkeit $a(1|1)$ des häufigsten Nachfolgers des häufigsten Wortes (siehe dazu auch die Gleichungen (3.25), (3.17) und (3.23)):

$$a_{\text{paar}}(1) = n_{\text{paar}}^{\vartheta} = a(1|1) = v(1) = n_0^{\psi}. \tag{3.29}$$

Durch Logarithmieren erhält man

$$\vartheta \cdot \lg n_{\text{paar}} = \psi \cdot \lg n_0. \tag{3.30}$$

Nun wird diese Gleichung nach ϑ aufgelöst:

$$\vartheta = \psi \cdot \frac{\lg n_0}{\lg n_{\text{paar}}}. \tag{3.31}$$

Mit $n_0 = 10^5$, $\psi = 0{,}8$ und dem Näherungswert aus Gleichung (3.28) bestimmt man den Exponenten ϑ wie folgt zu

$$\vartheta \approx 0{,}8 \cdot \frac{\lg 10^5}{\lg 450\,000} \approx 0{,}708. \tag{3.32}$$

Dieser Wert weicht nur geringfügig von dem gemessenen Wert $\vartheta = 0{,}71$ ab (siehe Bild 3.11). Zur Überprüfung der Gleichungen (3.27) und (3.31) wurden die Meßwerte, die aus Untersuchungen mit anderen Textlängen stammen, mit den jeweiligen berechneten Werten verglichen [78]. Für Texte in der Größenordnung des

LIMAS-Korpus wurden bei dem Exponenten ϑ Abweichungen bis zu 0,5 % und bei n_{paar} bis zu 16 % festgestellt. Bei kleineren Texten sind wegen der dann nicht mehr gültigen Näherung $\psi = 0{,}8$ (der Verlauf von $v(n)$ in Bild 3.10 entspricht in Wirklichkeit keinem exakt linearen Zusammenhang!) die berechneten Werte des Exponenten ϑ um etwa 10 % und diejenigen von n_{paar} sogar um 30 % zu groß.

3.5.3 Die Häufigkeitsverteilung von Folgen aus vier Wörtern

Entsprechend der Auswertung von Wortpaaren wurden auch Folgen aus vier Wörtern untersucht. Die anhand eines Ausschnittes von 60 % des LIMAS-Korpus ermittelte Häufigkeitsverteilung $a_{\text{vier}}(n)$ ist im nächsten Bild wiedergegeben. Der Arbeitsspeicher des für die Untersuchungen verwendeten Rechners (VAX 8530) ist nicht groß genug, um alle Viererfolgen des gesamten LIMAS-Korpus aufzunehmen.

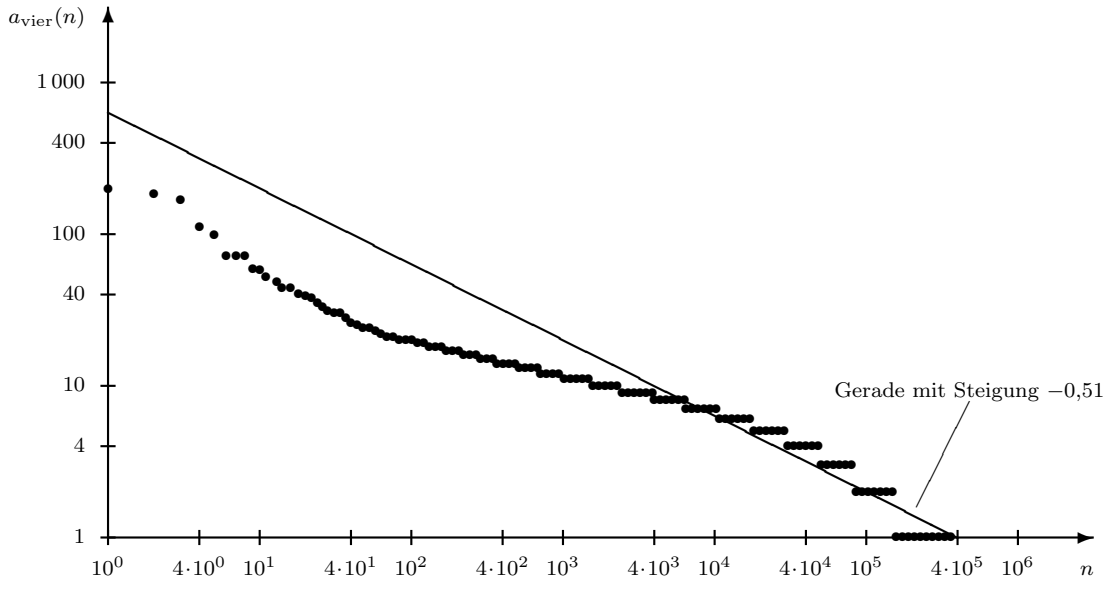

Bild 3.12: Die absoluten Häufigkeiten $a_{\text{vier}}(n)$ der sortierten Folgen aus vier Wörtern in Abhängigkeit ihrer Ordnungszahlen n (Textausschnitt: 60 % des LIMAS-Korpus).

Entgegen den Verteilungen, die in den Bildern 3.1 und 3.11 gezeigt sind und praktisch ideal linear verlaufen, weist die Häufigkeitsverteilung $a_{\text{vier}}(n)$ in Bild 3.12 eine deutliche Krümmung auf. Dennoch kann für $a_{\text{vier}}(n)$ in guter Näherung eine Regressionsgerade angesetzt werden, die mit einer Steigung von $-0{,}51$ noch flacher verläuft als diejenige in Bild 3.11. Auch die Untersuchungen von $a_{\text{vier}}(n)$ für andere Textlängen ergaben stets annähernd lineare Zusammenhänge im logarithmischen Diagramm, wobei die Steigungen im Bereich zwischen $-0{,}29$ und $-0{,}51$ liegen und die Geraden für längere Texte etwas steiler abfallen [78]. Die Tendenz, die aus den Bildern 3.1, 3.11 und 3.12 abzulesen ist, soll nun zur qualitativen Bestimmung der Häufigkeitsverteilungen längerer Wortfolgen benutzt werden.

3.5.4 Die Häufigkeitsverteilungen sehr langer Wortfolgen

Ausgehend von der Häufigkeitsverteilung $a_f(n)$ der Folgen aus f Wörtern kann man Aussagen über den prinzipiellen Verlauf der Häufigkeitsverteilung $a_{2f}(n)$ der Folgen aus $2f$ Wörtern treffen. Voraussetzung dabei soll sein, daß sämtliche Häufigkeitsverteilungen im logarithmischen Diagramm linearen Zusammenhängen gehorchen. Für Einzelwörter ($f = 1$) und Wortpaare ($f = 2$) trifft dies nahezu exakt zu, für Viererfolgen ($f = 4$) nur in guter Näherung (siehe die Bilder 3.1, 3.11 und 3.12). Entsprechend Gleichung (3.25) gilt dann

$$a_f(n) = \left(\frac{n_f}{n}\right)^{\varphi(f)} \qquad \text{mit } 1 \le n \le n_f. \tag{3.33}$$

Dabei bezeichnet n_f die Anzahl der verschiedenen Folgen aus f Wörtern und $\varphi(f)$ den negativen Wert der Geradensteigung. Analog dazu erhält man

$$a_{2f}(n) = \left(\frac{n_{2f}}{n}\right)^{\varphi(2f)} \qquad \text{mit } 1 \le n \le n_{2f}. \tag{3.34}$$

Da die Wortfolgen, die aus $2f$ Wörtern bestehen, aus zwei direkt benachbarten Folgen aus f Wörtern zusammengesetzt sind, ist die Anzahl der unterschiedlichen längeren Wortfolgen ersichtlich größer, d. h.

$$n_{2f} \ge n_f. \tag{3.35}$$

Die häufigste Folge aus $2f$ Wörtern kann nicht häufiger auftreten als die häufigste Folge aus f Wörtern. Es gilt

$$a_{2f}(1) \le a_f(1). \tag{3.36}$$

In Bild 3.13 sind $a_f(n)$ und $a_{2f}(n)$ nach den Gleichungen (3.33) und (3.34) aufgetragen, wobei die Bedingungen aus den Gleichungen (3.35) und (3.36) erfüllt sind.

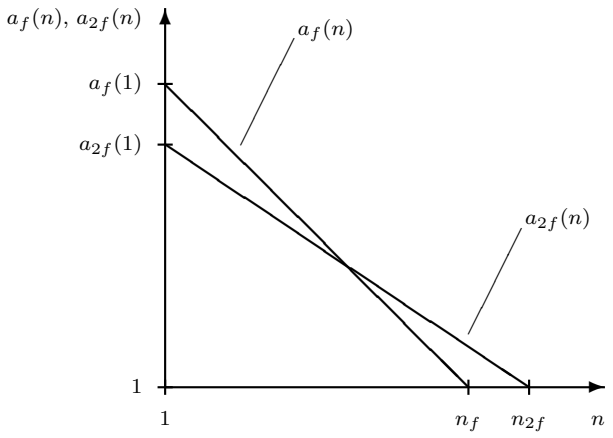

Bild 3.13: Der prinzipielle Verlauf der Häufigkeitsverteilungen $a_f(n)$ und $a_{2f}(n)$ sehr langer Wortfolgen.

Die Häufigkeitsverteilung $a_{2f}(n)$ verläuft flacher als diejenige von $a_f(n)$. Die Exponenten $\varphi(f)$ und $\varphi(2f)$ müssen demnach folgender Ungleichung genügen (siehe dazu Bild 3.13 und die Gleichungen (3.33) und (3.34)):

$$0 \leq \varphi(2f) \leq \varphi(f). \tag{3.37}$$

Daraus folgt, daß die Häufigkeitsverteilungen längerer Wortfolgen sich immer mehr einer Parallelen zur Abszisse annähern. Eine Veranschaulichung dieses Sachverhalts ist mit Bild 3.14 gegeben: Dort sind die negativen Steigungen $\varphi(f)$ der Geraden aus den Bildern 3.1, 3.11 und 3.12 als Funktion der Wortfolgenlänge f eingezeichnet.

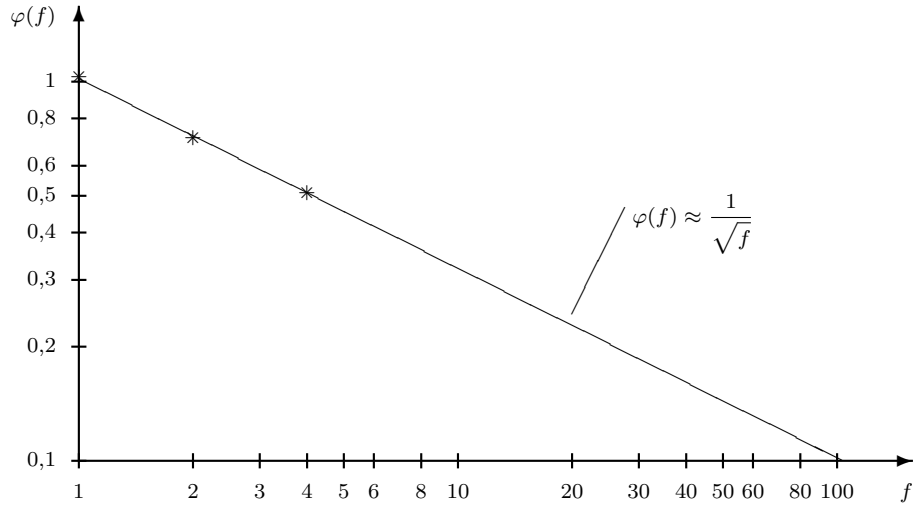

Bild 3.14: Der Exponent $\varphi(f)$ als Funktion der Wortfolgenlänge f.

Durch Extrapolieren findet man, daß die Häufigkeitsverteilung $a_{100}(n)$ der Folgen aus 100 Wörtern den Exponent $\varphi(100) \approx 0{,}1$ besitzt und die zugehörige Regressionsgerade mit einer Steigung von etwa $-0{,}1$ im Diagramm mit logarithmisch geteilten Achsen sehr flach verläuft, d. h. daß alle diese Folgen praktisch nur einmal vorkommen. Daß diese grobe Abschätzung zumindest tendenziell richtig ist, zeigt folgende Überlegung: Je länger eine Wortfolge ist, desto höher ist die Wahrscheinlichkeit, daß sie ein seltenes Wort enthält. Die absolute Häufigkeit der gesamten Wortfolge ist dann maximal so groß wie die Häufigkeit des seltensten Wortes, das in ihr vorkommt.

Für die Verzweigungsgrade $v_f(n)$ und die Häufigkeiten $a_f(n)$ der Folgen aus f Wörtern gilt folgende Ungleichung (siehe dazu auch Bild 3.10):

$$1 \leq v_f(n) \leq a_f(n). \tag{3.38}$$

Mit den Häufigkeiten $a_f(n)$ ist damit eine obere Schranke für die Verzweigungsgradverteilung $v_f(n)$ gegeben.

Die in diesem Kapitel diskutierten Ergebnisse von Textanalysen bilden eine ausreichende Basis für die Entwicklung eines Codierungssystems, das auf der Verwendung von „Assoziationen“ beruht. Vor der Beschreibung dieses neuen Systems werden jedoch erst einige konventionelle Codierungsverfahren untersucht.

4 Die klassischen Methoden der Textcodierung

Hier sollen zuerst die klassischen Codierverfahren daraufhin untersucht werden, bis zu welchem Grad der Redundanzreduktion man damit kommen kann.

Durch die Erfindung der Telegraphie ist es möglich geworden, Texte (z. B. Telegramme) über weite Entfernungen praktisch ohne Zeitverzögerung zu übermitteln. Dazu müssen die Texte allerdings erst umgeformt werden, da bei der Telegraphie nur zwei verschiedene Zeichen, d. h. Zustände, übertragen werden können:

1. Es fließt ein Strom: Der Telegraph schreibt eine Linie.
2. Es fließt kein Strom: Der Telegraph schreibt nichts und spult nur das Papier weiter.

S. Morse löste dieses Problem, indem er jedem Buchstaben eine bestimmte Folge von Punkten und Strichen eindeutig zuordnete [15]. Mit dieser Methode konnte er Texte *codieren* und übertragen. Auf der Empfängerseite mußte dann die Punkt-Strich-Folge in die ursprünglichen Texte zurückgewandelt, d. h. *decodiert* werden.

4.1 Universalcodes

Im Laufe der Zeit wurde die Codierung von Texten weiterentwickelt: Zum Beispiel verwendet man beim Fernschreiber zur Übertragung einen festen Takt und faßte zuerst jeweils fünf zeitlich aufeinanderfolgende binäre Zustände zu einem „5-Schritt-Code“ zusammen [15]. Dadurch konnte man insgesamt $2^5 = 32$ Zustände unterscheiden, so daß außer den 26 Buchstaben auch noch Satz- und Steuerzeichen in die *Codetabelle* aufgenommen wurden. Da dieser Code nicht gegen Übertragungsfehler geschützt war, führte man einen „7-Schritt-Code“ ein, der neben der gesicherten Übertragung von Texten auch dazu benutzt wurde, Daten in elektronische Rechenanlagen einzugeben.

Heute haben sich auf dem Gebiet der Computertechnik im wesentlichen zwei Codes durchgesetzt, der ASCII (**A**merican **S**tandard **C**ode for **I**nformation **I**nterchange) und der EBCDIC (**E**xtended **B**inary **C**oded **D**ecimal **I**nterchange **C**ode) [16]. Neben großen und kleinen Buchstaben, Satzzeichen, Ziffern und mathematischen Zeichen enthalten beide Codetabellen spezielle Steuerzeichen, die zum Speichern und Übertragen benötigt werden. Beim EBCDIC werden die Zeichen mit 8 bit codiert (der Buchstabe „`a`“ entspricht z. B. der Bitfolge „10000001“). Der ASCII ist nach DIN 66 003 ein ISO-7-bit-Code [16], d. h. die Zeichen werden jeweils mit 7 bit dargestellt (der Buchstabe „`a`“ z. B. durch „1100001“) und es können 128 verschiedene Zeichen codiert werden (z. B. ist das LIMAS-Korpus [2] aus ASCII-Zeichen aufgebaut). Da moderne Speicher meistens byte-organisiert sind, würde bei der Verwendung dieses 7-bit-Codes pro Zeichen jeweils 1 bit ungenutzt bleiben. Aus

diesem Grunde existieren zahlreiche 8-bit-Versionen des ASCII, bei denen insgesamt 256 verschiedene Zeichen zur Verfügung stehen. Ein derartiger Code ist z. B. der „Multinational Character Set“ von DEC, der außer den ASCII-Zeichen noch 128 weitere Zeichen enthält, unter denen neben einigen Sonderzeichen z. B. auch akzentuierte Buchstaben der französischen Sprache zu finden sind [17].

Gemeinsamer Vorteil der erwähnten Codes ist, daß man beliebige Texte, z. B. auch Quelltexte für Programme, mit der gleichen auf einen Buchstaben bezogenen Codelänge darstellen kann. Deshalb sollen diese Codes mit „Universalcodes“ bezeichnet werden. Wenn man allerdings nur Texte z. B. der deutschen Sprache codieren will, so ist die Verwendung dieser Universalcodes wegen der mittleren Codelänge von 8 bit/Buchstabe ungünstig, da nach K. Küpfmüller der Informationsgehalt eines deutschen Buchstabens im Mittel etwa 1,3 bit beträgt [13] und der Code daher mehr als 80 % Redundanz enthält. Deshalb sollte man in diesem Fall einen Code verwenden, der der deutschen Sprache besonders angepaßt ist. Mit diesem Code könnten alle *deutschen* Texte redundanzarm codiert werden. Allerdings kann dieser Code dann für andere Sprachen wieder sehr ungünstig sein.

4.2 Der Optimalcode

Bei einem Universalcode wie z. B. dem ASCII wird jedes Zeichen mit der gleichen Codelänge (z. B. 7 bit) codiert. Die Codelänge eines Zeichens sollte aber mit seinem *Informationsgehalt* übereinstimmen. Bei einem gegebenen Alphabet $\mathcal{A}$, das aus den Zeichen $\mathcal{A} = \{Z_1, Z_2, \ldots, Z_N\}$ besteht, kann man nach Shannon den Informationsgehalt $I(n)$ des Symbols Z_n bestimmen, wenn man die Auftrittshäufigkeiten $\mathcal{P} = \{p(1), p(2), \ldots, p(N)\}$ der Zeichen kennt [9, 10]. Es gilt

$$I(n) = -\operatorname{ld} p(n) \qquad \text{(in bit/Symbol)}. \tag{4.1}$$

D. A. Huffman beschrieb eine Methode, mit der man jedem Zeichen des Alphabets $\mathcal{A}$ einen Code zuordnen kann, dessen Länge $C(n)$ dem Informationsgehalt $I(n)$ des Zeichens ungefähr entspricht [18], d. h.

$$C(n) \approx I(n) = -\operatorname{ld} p(n) \qquad \text{(in bit/Symbol)}. \tag{4.2}$$

Der *mittlere* Informationsgehalt H der Nachrichtenquelle Q, welche Zeichen mit den Häufigkeiten $\mathcal{P}$ aussendet, berechnet man nach Gleichung (3.9). Mit einem ähnlichen Ansatz bestimmt man die mittlere Codelänge

$$\overline{C} = \sum_{n=1}^{N} p(n) \cdot C(n). \tag{4.3}$$

Man kann zeigen, daß die Entropie H (siehe Gleichung (3.9)) eine untere Schranke für die mittlere Codelänge $\overline{C}$ darstellt. Es gilt

$$\overline{C} \geq H. \tag{4.4}$$

Nur wenn für jedes Zeichen die Codelänge $C(n)$ mit dem Informationsgehalt $I(n)$ übereinstimmt, dann wird die *minimale* mittlere Codelänge $\overline{C} = H$ erreicht.

An zwei Beispielen soll der Huffman-Code, der auch „Optimalcode“ genannt wird, erläutert werden:

Ein Alphabet $\mathcal{A}_1$ enthalte $N = 4$ Zeichen, die alle gleichhäufig vorkommen. Für die Summe aller Häufigkeiten muß gelten

$$\sum_{n=1}^{N} p(n) = 1. \tag{4.5}$$

Daraus bestimmt man mit obiger Vorgabe

$$p(1) = p(2) = p(3) = p(4) = \frac{1}{N} = \frac{1}{4}. \tag{4.6}$$

Der Informationsgehalt eines Zeichens wird nach Gleichung (4.1) ermittelt zu

$$I(1) = I(2) = I(3) = I(4) = -\operatorname{ld}\frac{1}{4} = 2 \text{ bit/Symbol}. \tag{4.7}$$

Jedem Zeichen muß demnach ein 2-bit-Code zugeordnet werden, wenn man redundanzfrei codieren will. Zum Beispiel könnte man die Codes nach folgender Tabelle vergeben:

Zeichen	Code
Z_1	00
Z_2	01
Z_3	10
Z_4	11

Tabelle 4.1: Einfaches Beispiel für einen binären Code.

Ein Text, der nur aus Zeichen des Alphabets $\mathcal{A}_1$ besteht, kann anhand der Codes von Tabelle 4.1 codiert werden. Man erhält eine Folge der binären Symbole „0“ und „1“, die z. B. mit Hilfe eines binären Baumes, der aus einer Wurzel und mehreren Zweigen, Knoten und Blättern besteht (siehe Bild 4.1), in den ursprünglichen Text zurückverwandelt werden kann.

Von der Wurzel ausgehend gelangt man über Zweige und Knoten zu den Blättern, in denen die Zeichen stehen. Der Code beschreibt dabei den Weg durch den Baum, wobei eine „0“ eine Abzweigung nach links bedeutet und eine „1“ eine Abzweigung nach rechts (z. B. entspricht der Bitfolge „1001“ die Zeichenfolge „$Z_3\ Z_2$“ wie man leicht anhand von Bild 4.1 nachvollziehen und mit Tabelle 4.1 überprüfen kann). Wie im nächsten Beispiel noch demonstriert wird, muß die Länge der Codes nicht unbedingt konstant sein. Das Ende eines Codes wird durch das Erreichen eines

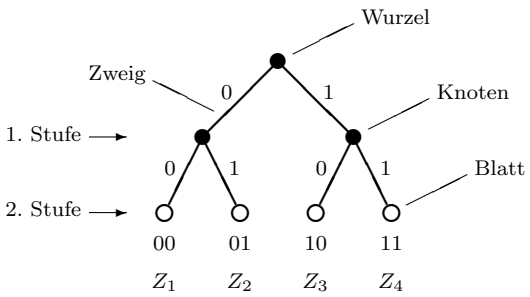

Bild 4.1: Beispiel für einen binären Baum.

Blattes signalisiert. Aus diesem Grund muß der binäre Baum so aufgebaut sein, daß die Zeichen nur in den Blättern und nicht in den Knoten enthalten sind, d. h. kein Code darf der Anfang eines anderen sein. Einen derartigen Code nennt man auch „präfixfrei“.

Die Bestimmung der Codes ist i. allg. nicht so einfach wie bei dem Beispiel in Tabelle 4.1. Unter Zuhilfenahme eines binären Baumes (siehe z. B. Bild 4.1) kann die Vergabe der Codes allerdings erleichtert werden: Aus der Häufigkeit $p(n)$ eines Zeichens berechnet man nach Gleichung (4.2) den Informationsgehalt $I(n)$ und die geforderte Codelänge $C(n)$; die Codelänge $C(n)$ entspricht der Position des Zeichens im binären Baum: Zum Beispiel hat das Zeichen Z_1 nach Gleichung (4.7) einen Informationsgehalt von $I(1) = 2$ bit/Symbol; in Bild 4.1 befindet sich dieses Zeichen auf der *zweiten* Stufe des Baumes, d. h. $I(n)$ gibt diejenige Stufe an, auf der man das Blatt mit dem n-ten Zeichen plazieren sollte. Bei einem zweiten Beispiel mit dem Alphabet $\mathcal{A}_2$, das in folgender Tabelle gezeigt ist, kann diese Forderung für jedes Zeichen erfüllt werden, da alle Häufigkeiten Zweierpotenzen sind.

Zeichen Z_n	Häufigkeit $p(n)$	Informationsgehalt $I(n)$
Z_1	1/4	2 bit/Symbol
Z_2	1/8	3 bit/Symbol
Z_3	1/8	3 bit/Symbol
Z_4	1/2	1 bit/Symbol

Tabelle 4.2: Beispiel für ein Alphabet mit Auftrittshäufigkeiten.

Ein möglicher binärer Baum, den man nach dieser Tabelle aufbauen kann, ist in Bild 4.2 gezeigt. Dem Baum aus diesem Bild können die Codes für alle Zeichen des Alphabets von Tabelle 4.2 entnommen werden (siehe Tabelle 4.3).

Für dieses Beispiel kann, wie man sieht, der Baum leicht konstruiert und die Verteilung der Codes abgelesen werden, da der Zeichenvorrat klein ist und die Häufigkeiten Zweierpotenzen sind. Bei realen Alphabeten sind die Auftrittshäufigkeiten $p(n)$ i. allg. keine Zweierpotenzen, sondern reelle Zahlen aus dem Intervall zwischen 0

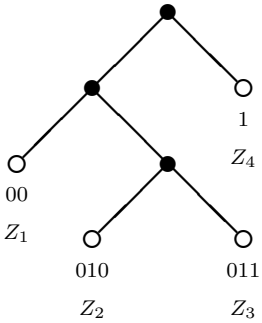

Bild 4.2: Binärer Baum für das Alphabet von Tabelle 4.2.

Zeichen	Code
Z_1	00
Z_2	010
Z_3	011
Z_4	1

Tabelle 4.3: Alphabet mit Codes nach Bild 4.2.

und 1. Daraus errechnen sich nach Gleichung (4.1) Informationsgehalte, die ebenfalls reelle Zahlen sind. Nach Gleichung (4.2) entspricht der Informationsgehalt eines Zeichens der geforderten Codelänge, d. h. der Position im binären Baum. Da die Stufen des Baumes ganzzahlig sind, muß man entsprechend auf- oder abrunden, um die Positionen der Zeichen zu erhalten. Wegen dieser Ungenauigkeiten ist der resultierende Code i. allg. nicht redundanzfrei. Nach dem Huffman-Verfahren kann man allerdings anhand der Häufigkeiten eines gegebenen Alphabets systematisch denjenigen binären Baum aufbauen, der zu dem bestmöglichen, d. h. redundanz*ärmsten* Code gehört [18, 19]. Die Redundanz, die in diesem Code noch enthalten ist, berechnet man mit den Gleichungen (4.3) und (3.9) zu

$$R = \overline{C} - H = \sum_{n=1}^{N} p(n) \cdot \Big[C(n) + \operatorname{ld} p(n)\Big] \qquad \text{(in bit/Symbol)}. \tag{4.8}$$

In der Regel ist diese Redundanz allerdings im Vergleich zu derjenigen, die man z. B. bei der Codierung mit ASCII-Zeichen feststellt, vernachlässigbar klein: Für das deutsche Alphabet läßt sich ein Optimalcode angeben, dessen mittlere Codelänge bei $\overline{C} = 4{,}127$ bit/Buchstabe liegt [20]. Berücksichtigt man Küpfmüllers Entropiewert für Buchstaben [13], dann beträgt die Redundanz dieses Optimalcodes weniger als 0,1 bit/Buchstabe.

Die von Huffman für Buchstaben vorgeschlagene Optimalcodierung läßt sich auch auf Wörter anwenden. (Huffman hätte dies wegen des damals ($\approx$ 1952) noch zu hohen Aufwands gar nicht realisieren können.) Weiterentwicklungen des Optimalcodes z. B. in dieser Richtung werden im nächsten Kapitel vorgestellt. Der Buchstaben-Optimalcode enthält noch relativ viel Redundanz, da die Beziehungen zwischen den Buchstaben nicht berücksichtigt werden. Eine stärkere Redundanzreduktion wäre zu

erreichen, wenn man bei der Bildung der Optimalcodes statt der Buchstaben-Häufigkeit $p(n)$ die bedingte Wahrscheinlichkeit $p(n|m_k)$ verwendet, mit der der n-te Buchstabe auf die m-te Kette aus k Buchstaben folgt (vergleiche auch mit Gleichung (3.11)).

4.3 Die Codierung mit Markoff-Ketten

4.3.1 Das klassische Verfahren

A. A. Markov befaßte sich als erster mit Untersuchungen der Häufigkeitsverteilungen von Buchstabenketten [20]. Nach ihm sind die „Markoff-Ketten“ benannt. Markoff-Ketten k-ter Ordnung sind Symbolfolgen, bei denen jeweils k benachbarte Symbole die bedingte Wahrscheinlichkeit $p(n|m_k)$ des nachfolgenden $(k+1)$-ten Symbols bestimmen (siehe auch Gleichung (3.11)). Mit diesen Übergangswahrscheinlichkeiten kann man einen Optimalcode erzeugen, der umso redundanzärmer ist, je größer man die Konstante k wählt:

Im Gegensatz zum Huffman-Code, bei dem man die Buchstaben isoliert voneinander codiert, werden nun die letzten k Buchstaben eines Textes bei der Codierung einbezogen. Unter Berücksichtigung der m-ten Buchstabenfolge muß man zur Bildung des Optimalcodes, d. h. zum Aufbau des binären Baumes (siehe oben), die Übergangswahrscheinlichkeiten $\mathcal{P}_m = \{p(1|m_k), p(2|m_k), \ldots, p(N|m_k)\}$ verwenden, wobei das Alphabet aus N Buchstaben besteht. Da für jede der $M = N^k$ Folgen jeweils eine Häufigkeitsverteilung $\mathcal{P}_m$ existiert, muß man zum Speichern der Häufigkeiten $N \cdot M = N^{(k+1)}$ Speicherzellen reservieren. Selbst wenn man die Häufigkeiten *binär* speichert (z. B. „0“ $\widehat{=}$ Übergang nicht möglich, „1“ $\widehat{=}$ Übergang ist möglich), benötigt man einen riesigen Speicher, wie folgende Abschätzung zeigt: Bei einem Alphabet mit 27 Zeichen (26 Buchstaben und einem Leerzeichen) muß der Speicher eine Kapazität von $27^{(k+1)}$ bit haben. Für $k = 3$ beträgt die Speichergröße schon ca. 0,5 Megabit. Nach Bild 3.2 sollte man $k \gg 3$ wählen, um eine ausreichende Redundanzreduktion zu erreichen. Dafür bräuchte man nach obiger Abschätzung allerdings einen Speicher von utopischen Ausmaßen, so daß die Verwendung von Markoff-Ketten für eine redundanzarme Codierung in der Praxis von vornherein ausscheidet.

Dennoch wurde das beschriebene Verfahren für $k = 3$ untersucht, wobei folgende neue Methode zur Redundanzmessung angewendet wurde, mit der die erreichte Redundanzreduktion allerdings nur *qualitativ* bestimmt werden konnte:

4.3.2 Die Redundanzmessung mit Zufallstexten

Wenn man einen Text mit einem binären Optimalcode (Huffman-Code) codiert, so ergibt sich eine „quasi-stochastische“ „0“-„1“-Folge, d. h. zwischen den binären

Symbolen können auf den ersten Blick keine statistischen Abhängigkeiten festgestellt werden. Bei der Decodierung dieser Folge erhält man wieder den ursprünglichen Text, sofern der Decodierer korrekt arbeitet. Prinzipiell ist es möglich, eine mit einem Zufallsgenerator erzeugte Bitfolge zu decodieren. Bei einer redundanz*freien* Codierung ist jeder auf diese Art decodierte Text sinnvoll, bei einer redundanz*armen* nur ein gewisser Bruchteil. Das Verhältnis von sinnvollen zu sinnlosen Texten gibt somit Aufschluß über die Güte des Codes. Da die Attribute „sinnvoll“ und „sinnlos“ subjektiv empfundene Bewertungen sind, kann dabei kein genauer Zahlenwert für die Redundanz des Codes angegeben werden. Die Bestimmung eines Meßwertes würde darüber hinaus die Auswertung von großen Mengen an „Zufallstexten“ erfordern. Dennoch ist eine *qualitative* Bewertung des Codes anhand eines relativ kleinen Textausschnittes möglich: Wenn eine Versuchsperson nicht entscheiden kann, ob es sich um einen „menschlichen“ oder um einen Zufallstext handelt, dann hat man einen redundanzarmen Code verwendet. Auf dieses „Zufallsexperiment“ wird bei späteren Untersuchungen zurückgegriffen.

Der Optimalcode, der auf Markoff-Ketten dritter Ordnung basiert, wurde mit zufällig erzeugten Bitfolgen getestet. Die dafür benötigten Häufigkeiten $p(n|m_3)$ wurden anhand des Lesebuchs [4] ermittelt. Unter den bei diesem Experiment entstandenen „Zufallswörtern“ ist ein erstaunlich hoher Anteil im deutschen Vokabular enthalten. Alle anderen Wörter kann man zumindest aussprechen, wie die Auswahl in Tabelle 4.4 zeigt [79]. Das deutet darauf hin, daß dieser Optimalcode nur noch verhältnismäßig wenig Redundanz enthält.

GEMAUSGELAGE	ROCKNAUT	PFLOCHEN	RÜHLTEISTERS
BAUMELAND	RISAUNE	NURZUNG	UHAMPELAT
BOCKNATTEN	SEGNEINE	HÖNSTRA	ISESELEINI
BARAUSEND	KNASSE	PINDELN	MÜSLINET
UNDHERADEIN	GUMACKTE	LADENEIT	INTABENTIN

Tabelle 4.4: Eine Auswahl an „Zufallswörtern“, die mit Hilfe von Markoff-Ketten dritter Ordnung entstanden sind und nicht im DUDEN stehen [79, 5].

4.3.3 Die Codierung mit Markoff-Ketten auf Wortebene

T. Höfling führte auch Untersuchungen von Markoff-Ketten durch, deren Elemente aus Wörtern bestehen [89]. Obwohl die oben angesprochenen Speicherplatzprobleme durch eine dynamische Speicherstruktur weitgehend entschärft werden konnten, reichte der verfügbare Speicher nur für Experimente bis zur Ordnung $k = 7$ aus. Das Codierungssystem, das Markoff-Ketten aus Wörtern verwendet, liefert etwa von der vierten Ordnung an brauchbare Ergebnisse, wie der Zufallstest zeigt: Die generierten Zufallstexte enthalten für $k \geq 4$ nur relativ wenig grammatikalische Fehler. Allerdings sind diese Texte inhaltlich völlig unsinnig, weil durch die zufällige

Ansteuerung dieses Codierers Texte erzeugt werden, die man als „semantisches Rauschen“ bezeichnen kann.

Auf die Codierung mit Markoff-Ketten soll im folgenden jedoch nicht weiter eingegangen werden, da dieses Verfahren wegen des immensen Speicherbedarfs schwierig zu realisieren ist.

4.4 Grundsätzliche Überlegungen zu neuen Methoden der Textcodierung

Bei der obigen Diskussion der praktisch verwendeten Codierungsverfahren fällt auf, daß man für eine Steigerung der Redundanzreduktion einen unverhältnismäßig hohen Aufwand treiben muß. Das liegt bei den erläuterten Verfahren sicherlich daran, daß ein recht künstlicher Ansatz gewählt wurde, der der Struktur der Sprache nicht genügend angepaßt ist. Als natürliche Elemente einer Sprache sind Silben, Wörter und Wortfolgen zu nennen, wohingegen Buchstaben erst durch einen relativ komplizierten Abstraktionsprozeß entstanden sind. Aus diesem Grunde sollte man bei der Codierung von Texten auf größere sprachliche Einheiten, z. B. Wörter, zurückgreifen.

Eine Codierung auf Wortebene wird zunächst durch die Vielzahl der Wörter einer Sprache erschwert. Neben der Grundform eines jeden Wortes existiert außerdem eine Reihe von Beugungsformen, die sich vom Schriftbild der Stammform unterscheiden. Die Zuordnung dieser „Wortformen“ zu den jeweiligen Stammformen, die einem Menschen in der Regel keine Schwierigkeiten bereitet, kann von einer Datenverarbeitungsanlage wegen der vielen Unregelmäßigkeiten einer Sprache nicht automatisch durchgeführt werden. In der linguistischen Datenverarbeitung ist es üblich, Wörter mit menschlicher Hilfe zu klassifizieren und in Lexika einzutragen, um auf diese Art eine *halb*automatische Verarbeitung zu erreichen [3]. T. Erler entwickelte nach diesem Vorbild mit Hilfe einer manuellen Klassifikation von Wortformen eine Codierung, die auf der Ausnutzung von typischen Satzstrukturen basiert [90, 79]. Dabei stellte sich heraus, daß dieses Verfahren wegen der manuellen Auswertung des umfangreichen Textmaterials nicht praktikabel ist. Man sollte deshalb auf menschliche Hilfe ganz verzichten und *voll*automatische Codierverfahren vorziehen.

D. Schmitt und M. Lauth versuchten, die Ähnlichkeiten der Wortformen, die zu einer Grundform gehören, für eine automatische Klassifikation auszunutzen [91, 92]. Obwohl recht gute Ergebnisse erzielt werden konnten, war wegen der vielen unregelmäßigen Beugungsformen eine fehler*freie* Zuordnung mit diesem Verfahren *nicht* möglich, so daß es sinnvoll erschien, jede in einem Text auftretende Wortform als eigenständiges Wort zu behandeln.

Im folgenden sollen Verfahren untersucht werden, mit denen man nach einem möglichst einfachen Prinzip und mit minimalem Aufwand eine maximale Redundanzreduktion erreichen kann. Allgemein läßt sich schon voraussagen, wie sich ein sehr

redundanzarmer Code verhalten muß: Ein Universalcode (siehe oben) hat für alle möglichen Texte die gleiche mittlere Codelänge. Dagegen kann man mit einem Code, der z. B. speziell auf Wörter und Redewendungen deutscher Zeitungen abgestimmt ist, diese Texte äußerst redundanzarm codieren. Für literarische Texte in deutscher Sprache ist dieser Code i. allg. nicht geeignet. Ein Optimalcode auf Buchstabenebene ist in dieser Hinsicht relativ unempfindlich, da es sich in beiden Fällen um *deutsche* Texte handelt und stilistische Merkmale den Aufbau des Optimalcodes nicht beeinflussen. Daher ist die Anzahl der verschiedenartigen Texte, die man mit einem speziellen Code redundanzarm codieren kann, umso kleiner, je weniger Redundanz dieser Code enthält.

Im folgenden Kapitel werden, wie oben schon erwähnt, neue Anwendungen des Optimalcodes auf ihre Tauglichkeit für eine redundanzarme Codierung geprüft.

5 Neue Anwendungen des Optimalcodes

Das Prinzip der Optimalcodierung, das bei den klassischen Anwendungen nur für Einzelbuchstaben realisiert wurde, soll nun für größere sprachliche Einheiten angewandt werden. Dabei soll untersucht werden, welche minimale mittlere Codelänge erreicht werden kann und welcher Aufwand dazu notwendig ist.

5.1 Die Optimalcodierung ganzer Wörter

Bei der Optimalcodierung von Wörtern treten wegen der großen Anzahl verschiedener Wörter zwei grundsätzliche Probleme auf: Für die Bestimmung der Codes nach dem Huffman-Verfahren [18] muß einerseits der gesamte Wortvorrat nach Häufigkeiten sortiert werden, was je nach verwendetem Sortieralgorithmus entweder speicher- oder rechenzeitintensiv ist, andererseits muß für die Codierung eine Codetabelle organisiert werden, bei der ein möglichst schneller Zugriff auf ein beliebiges Wort und dessen Code möglich ist. Diese Tabelle, die im folgenden „assoziatives Wörterbuch“ genannt wird, soll zuerst entwickelt werden.

5.1.1 Das assoziative Wörterbuch

Die Codetabelle, die für die Optimalcodierung von Buchstaben benötigt wird, kann wegen der geringen Buchstabenanzahl technisch sehr einfach realisiert werden, indem man einen adreßorganisierten Speicher (z. B. ein ROM $\hat{=}$ **R**ead **O**nly **M**emory) verwendet. Diese Methode scheidet bei der Optimalcodierung von Wörtern aus: Wenn man die Wörter in ihrer digitalen Form als Zahlen interpretiert und direkt zur Adressierung eines ROM benutzen würde, müßte dieser Speicher bei einer maximalen Wortlänge von beispielsweise 37 Buchstaben (LIMAS-Korpus) eine 174-bit-Adresse haben. Dieses nicht realisierbare ROM hätte demnach einen Adreßbereich von ca. $2 \cdot 10^{52}$ Adressen.

Da realistische Wortvorräte „nur“ in der Größenordnung 10^5–10^6 liegen, ist es gar nicht notwendig, für jede Adresse des „virtuellen“ Adreßraumes jeweils eine Speicherzelle zu reservieren. Mit Hilfe eines Assoziativspeichers (CAM $\hat{=}$ **C**ontent-**A**dressable **M**emory) kann nämlich eine „Adressenkonzentration“ von dem virtuellen auf den realen Adreßbereich erreicht werden, indem nur für die tatsächlich vorkommenden Adressen ($\hat{=}$ Wörter) Speicherzellen belegt werden (siehe [41]). In Bild 5.1 wird die Verwendung eines teilassoziativen Speichers für eine Codetabelle der Optimalcodes auf Wortebene an einem Beispiel demonstriert: Nach der Eingabe eines Wortes in dieses „assoziative Wörterbuch“ wird der betreffende Optimalcode, der im Datenteil gespeichert ist, direkt ausgegeben.

Obwohl dieses Konzept prinzipiell brauchbar erscheint, scheitert die Realisierung eines assoziativen Wörterbuches mit Hilfe eines CAM letztendlich an der Tatsache,

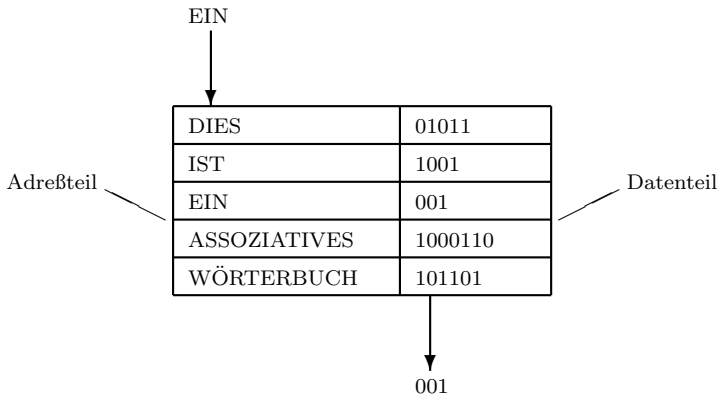

Bild 5.1: Die Verwendung eines teilassoziativen Speichers für ein „assoziatives Wörterbuch“, das bei einer Optimalcodierung auf Wortebene eingesetzt werden kann.

daß Assoziativspeicher mit den benötigten Speicherkapazitäten (> 1 Megabyte) heutzutage noch nicht verfügbar sind.

Es existiert jedoch eine bekannte Methode, mit der man das Verhalten eines Assoziativspeichers simulieren kann: die sogenannte „Hash-“ oder „Streuspeicherung“. Diesem Verfahren liegt eine „Schlüsseltransformation“ zugrunde, mit der man einen „Schlüssel“, d. h. ein bestimmtes Merkmal des zu speichernden Objektes, auf den realen Adreßbereich abbildet. Dieser Schlüssel kann z. B. eine virtuelle Adresse, die Länge eines Wortes oder die ASCII-Ordnungszahl des ersten Buchstabens sein. Die Transformationsvorschrift $A(S)$ wird auch „Hash-Funktion“ genannt, wobei mit S der Schlüssel und mit $A(S)$ die reale Adresse bezeichnet ist. Eine einfache Hash-Funktion, die gerne verwendet wird, ist die folgende [21]:

$$A(S) = S \bmod P. \tag{5.1}$$

Die Größe P des realen Adreßraumes sollte dabei eine Primzahl sein, damit die Schlüssel möglichst gleichmäßig auf die realen Adressen verteilt werden [21].

Da der Wertebereich der Schlüssel meist viel größer ist als der reale Adreßraum, kann es zu „Kollisionen“ kommen, wenn zwei Datenobjekte mit verschiedenen Schlüsseln auf eine gemeinsame reale Adresse abgebildet werden. Dieses Problem kann gelöst werden, indem man alle Datenobjekte, die einer einzigen realen Adresse zugeordnet sind, in einer verketteten Liste speichert, die über diese Adresse angesprochen wird. Dazu benötigt man ein „Zeigerfeld“, das aus P Zeigern (dem *realen* Adreßraum) besteht, und relativ komplexe Speicherzellen, die jeweils außer einem Wort samt zugehörigem Optimalcode einen sogenannten „Überlaufzeiger“ enthalten. Bild 5.2 zeigt diese Speicherorganisation an einem einfachen Beispiel, wobei als Hash-Funktion $A(\ell) = \ell \bmod 3$ benutzt wird und der Schlüssel ℓ der jeweiligen Wortlänge entspricht.

Unter Verwendung dieses Hash-Verfahrens implementierte W. Rüdiger mehrere PASCAL-Unterprogramme, die die Simulation eines assoziativen Wörterbuches ermöglichen [93, 22]. Obwohl der Schwerpunkt dieser Arbeit auf der prinzipiellen Lö-

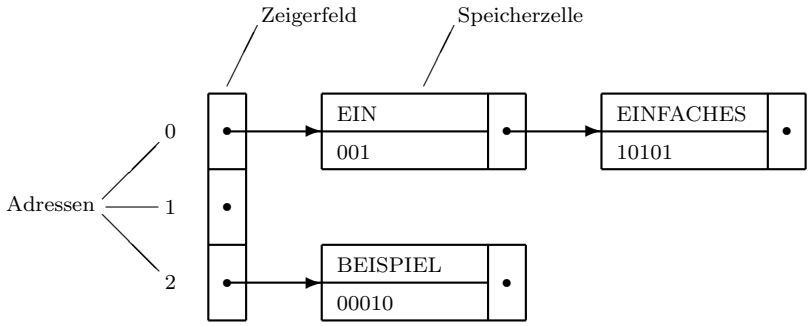

Bild 5.2: Die Realisierung eines assoziativen Wörterbuches mit Hilfe der Streuspeicherung (Hash-Funktion: *Adresse* = *Wortlänge* mod 3).

sung des Problems lag und nicht auf der Auswahl einer optimalen Hash-Funktion, konnten dennoch sehr kurze Suchzeiten ermittelt werden, wenn die Größe P des Adreßbereichs eine Primzahl ist, die die Anzahl n_0 der Wörter um etwa 20 % übersteigt: Bei dem Wörterbuch, das anhand des LIMAS-Korpus [2] aufgebaut wurde, benötigt man damit im Mittel nur 1,15 Vergleiche, um ein Wort im Speicher zu finden, d. h. es wird ein nahezu direkter Zugriff erreicht.

Eine Optimierung der Hash-Funktion für dieses assoziative Wörterbuch wurde von C. Herrmann durchgeführt [94]. Die von ihm vorgeschlagene Lösung ist allerdings programmtechnisch sehr aufwendig, da er für jede Wortlänge eine eigene, (für das LIMAS-Korpus) optimale Hash-Funktion bestimmte. Unter Berücksichtigung der Ergebnisse dieser Untersuchungen konnte schließlich eine gut streuende, aber dennoch einfache Hash-Funktion ermittelt werden, bei der jeweils die ersten und die letzten vier Buchstaben eines Wortes verwendet werden und man im Mittel 1,115 Vergleiche benötigt, um ein Wort im Speicher zu finden:

$$A(W) = \left[\sum_{i=1}^{4} \left(7^{(i-1)} \cdot W[i] \right) + \sum_{i=\ell-3}^{\ell} \left(7^{(i-\ell+7)} \cdot W[i] \right) \right] \bmod P \qquad \text{für } \ell > 4. \tag{5.2}$$

Dabei bezeichnet $W[i]$ die ASCII-Ordnungszahl des i-ten Buchstabens und ℓ die Länge des Wortes W. Für Wörter, die weniger als fünf Buchstaben lang sind, wird Gleichung (5.2) entsprechend vereinfacht:

$$A(W) = \begin{cases} \left[\sum_{i=1}^{\ell} \left(7^{(6 \cdot (i-1)/(\ell-1))} \cdot W[i] \right) \right] \bmod P & \text{für } 1 < \ell \leq 4; \\ W[1] \bmod P & \text{für } \ell = 1. \end{cases} \tag{5.3}$$

Neben den Software-Simulationen wurde auch ein assoziatives Wörterbuch in Hardware aufgebaut: H. Müller und B. Passinger realisierten unter Verwendung der Streuspeicherung einen Assoziativspeicher, der alle Wörter des LIMAS-Korpus aufnehmen kann [95]. Für die Berechnung der Hash-Funktion sowie für die anfallenden Verwaltungsaufgaben wurde dabei ein Mikroprozessor des Typs „MC 68 000“ verwendet und der Hash-Speicherbereich aus 31 EPROMs der Kapazität 64 kbyte zusammengesetzt. Mit diesem Hardware-Wörterbuch konnte die mittlere Zugriffszeit

auf ein Wort etwa um den Faktor 10 gegenüber der mittleren Zugriffszeit bei der Simulation gesenkt werden.

5.1.2 Die Erzeugung von Quasi-Optimalcodes

Mit dem Huffman-Verfahren kann ein optimaler Code ermittelt werden, indem zuerst der Symbolvorrat nach Auftrittshäufigkeiten sortiert und dann ein binärer Baum von den Blättern ausgehend, die man schrittweise zu Teilbäumen zusammenfügt, aufgebaut wird [18]. Die Wurzel des endgültigen Baumes ergibt sich erst nach dem letzten Aufbauschritt, weshalb diese Methode mit „Blatt-Wurzel-Verfahren“ bezeichnet werden soll. Da Sortierverfahren für große Symbolvorräte sehr aufwendig sind, wurde zur Erzeugung der Optimalcodes auf Wortebene eine andere Methode gesucht: S. Heidemann entwickelte eine vom bekannten Huffman-Verfahren abweichende Methode, mit der man die Symbole eines *unsortierten* Symbolvorrates sukzessive von der Wurzel her („Wurzel-Blatt“) in den binären Baum einbauen kann [96]. Diese neue Methode basiert auf dem nach C. E. Shannon und R. M. Fano bekannten Algorithmus, bei dem der Symbolvorrat in zwei Untergruppen mit gleichen Häufigkeitssummen zerlegt und jede Untergruppe auf diese Weise solange weiter aufgeteilt wird, bis alle Untergruppen nur noch aus einzelnen Symbolen bestehen [9, 10, 19]. Den „Wurzel-Blatt-Aufbau“ des Baumes mit diesem modifizierten Verfahren kann man sich wie folgt vorstellen: Die einzelnen Symbole werden nacheinander an der Wurzel des Baumes eingegeben und „wandern“ dann an denjenigen Platz, an dem sich die Aufteilung in Gruppen und Untergruppen mit gleichen Häufigkeitssummen am besten annähern läßt.

Im einzelnen lautet der Algorithmus:

1. Gegeben sei ein Symbolvorrat $\mathcal{A} = \{Z_1, Z_2, \ldots, Z_N\}$ mit den Auftrittshäufigkeiten $\mathcal{P} = \{p(1), p(2), \ldots, p(N)\}$, die i. allg. nicht nach Häufigkeiten sortiert vorliegen. Es gelte $N > 1$.

2. Man bilde aus den ersten beiden Symbolen Z_1 und Z_2 einen Teilbaum. Das Symbol, das als nächstes in diesen Baum eingebaut wird, soll Z_i genannt werden. Für $N > 2$ wird $Z_i = Z_3$ gesetzt. Für $N = 2$ ist der Baumaufbau beendet und es wird mit Schritt 5 fortgefahren.

3. Das Symbol Z_i wird von der Wurzel des bisher aufgebauten Baumes solange von Knoten zu Knoten weitergereicht, bis die passende Stelle im Baum gefunden ist. Die dafür benötigten Häufigkeiten der Knoten entsprechen den Summen der Häufigkeiten der jeweils nachfolgenden Knoten oder Blätter. Beim Einbau des Symbols Z_i können drei Fälle unterschieden werden:

   a) Der Knoten, bei dem sich das Symbol Z_i befindet, spaltet sich in zwei Knoten oder Blätter mit den Häufigkeiten $p(m)$ und $p(n)$ auf, wobei $p(m) \geq p(n)$ und $p(m) > p(i)$ ist. Wie in Bild 5.3 zu erkennen ist, wird in

diesem Fall das Symbol Z_i an den Knoten oder das Blatt mit der geringeren Häufigkeit (hier: $p(n)$) weitergereicht und mit Schritt 3 fortgefahren.

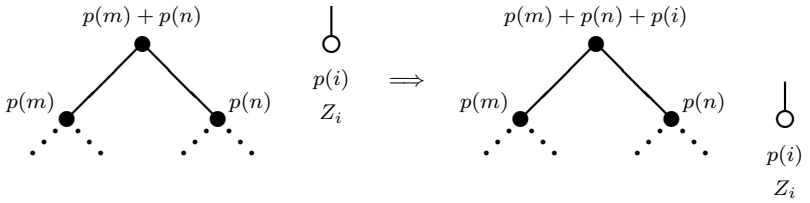

Bild 5.3: Fall 3a) beim Baumaufbau nach dem Wurzel-Blatt-Algorithmus.

b) Der Knoten, bei dem sich das Symbol Z_i befindet, spaltet sich in zwei Knoten oder Blätter mit den Häufigkeiten $p(m)$ und $p(n)$ auf, wobei $p(i) \geq p(m) \geq p(n)$ ist. In Bild 5.4 ist gezeigt, auf welche Weise hier ein neuer Knoten gebildet wird, an den der Knoten, bei dem sich das Symbol Z_i momentan befindet, und das Blatt mit dem Symbol Z_i angehängt wird.

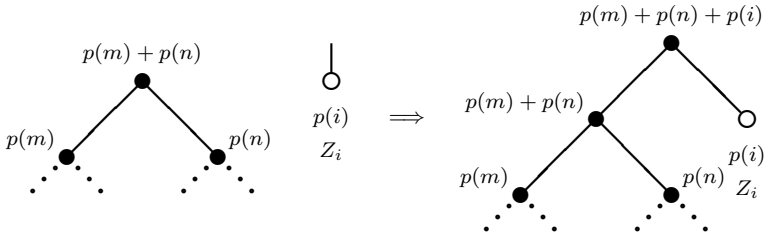

Bild 5.4: Fall 3b) beim Baumaufbau nach dem Wurzel-Blatt-Algorithmus.

c) Das Symbol Z_i befindet sich bei einem Blatt. In diesem Fall wird ein neuer Knoten gebildet, an den dieses Blatt und dasjenige mit dem Symbol Z_i angehängt wird (siehe Bild 5.5).

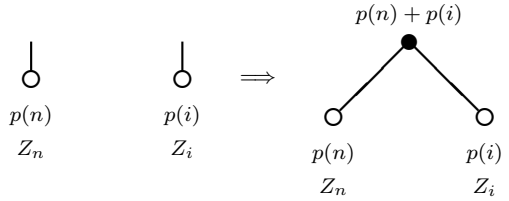

Bild 5.5: Fall 3c) beim Baumaufbau nach dem Wurzel-Blatt-Algorithmus.

4. Falls $i < N$ ist, wird mit Symbol Z_{i+1} bei Schritt 3 fortgefahren. Für $i = N$ ist der Baumaufbau beendet.

5. Dem binären Baum werden schließlich die Optimalcodes entnommen (siehe auch Bild 4.1 und Tabelle 4.1).

Anhand des Symbolvorrates aus Tabelle 4.2 (siehe Seite 26) soll ein binärer Baum mit Hilfe dieses Algorithmus aufgebaut werden: Zuerst wird mit den Symbolen Z_1 und Z_2 ein Teilbaum gebildet. Wegen $p(1) \geq p(2)$ und $p(1) > p(3)$ wird das Symbol Z_3 nach Schritt 3a) des oben erläuterten Algorithmus in den rechten Zweig dieses Baumes weitergereicht und befindet sich dann, wie in dem folgenden Bild gezeigt ist, bei dem Symbol Z_2.

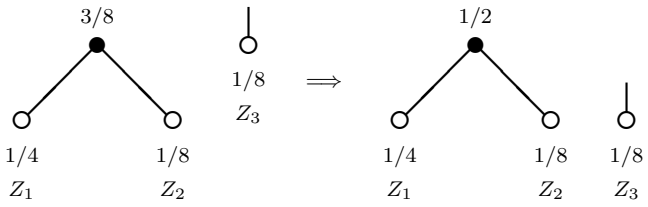

Bild 5.6: Die ersten beiden Phasen des Baumaufbaus mit dem Symbolvorrat aus Tabelle 4.2 auf Seite 26.

Da das Symbol Z_3 sich nun bei einem Blatt befindet, wird unter Berücksichtigung von Fall 3c) ein neuer Teilbaum aus diesen beiden Blättern gebildet. Nach dem Einbau des Symbols Z_4 (Schritt 3b)) erhält man dann den endgültigen Baum, der im nächsten Bild wiedergegeben ist (vergleiche auch mit dem Baum in Bild 4.2 auf Seite 27).

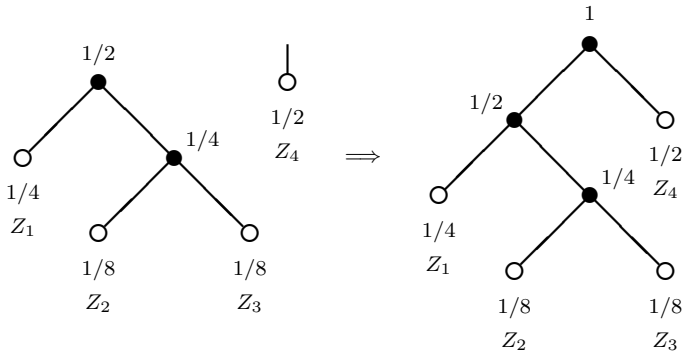

Bild 5.7: Die letzten beiden Phasen des Baumaufbaus mit dem Symbolvorrat aus Tabelle 4.2 auf Seite 26.

Wie schon erwähnt, wird mit dem Huffman-Algorithmus ein optimaler Code ermittelt [19]. Dagegen erzeugt man mit dem gerade erläuterten Wurzel-Blatt-Algorithmus, der einem modifizierten Fano-Shannon-Algorithmus entspricht, einen Code, der *nicht* optimal ist und noch etwas Redundanz enthält. In welcher Größenordnung sich diese Rest-Redundanz befindet, wird bei der Simulation des „Quasi-Optimalcodes“ auf Wortebene, die im folgenden dargelegt wird, ermittelt.

5.1.3 Die Ergebnisse der Simulation

Mit Hilfe des oben beschriebenen assoziativen Wörterbuches und des Wurzel-Blatt-Algorithmus für den Aufbau des binären Baumes konnte die Optimalcodierung von Wörtern auf einem Rechner simuliert werden [80]. Die einzelnen Schritte dieser Simulation sollen hier kurz erläutert werden:

Für den Baumaufbau benötigt man die Häufigkeiten aller Wörter. Deshalb wird zuerst das assoziative Wörterbuch (siehe Bild 5.2) erstellt, wobei in den Speicherzellen neben den Wörtern die *Häufigkeiten* anstelle der Optimalcodes abgelegt werden. Danach kann mit dem oben beschriebenen Algorithmus der binäre Baum aufgebaut werden. Anschließend bestimmt man anhand des Baumes die (Quasi-)Optimalcodes der Wörter und trägt sie in das assoziative Wörterbuch ein, wobei die nun nicht mehr benötigten Häufigkeiten überschrieben werden.

Beim Aufbau des binären Baumes tritt folgende Schwierigkeit auf: In den Blättern müssen sowohl die Häufigkeiten als auch die *Wörter* gespeichert werden (siehe z. B. Bild 5.7). Da ein Speichern in Klartext sehr aufwendig ist (die maximale Wortlänge im LIMAS-Korpus beträgt 37 Buchstaben!), werden in den Blättern nicht Wörter, sondern *Zeiger* auf die Wörter, die im assoziativen Wörterbuch in Klartext enthalten sind, gespeichert. Ein solcher Zeiger entspricht der Position eines Wortes im assoziativen Wörterbuch, die mit der Hash-Adresse $A(W)$ (siehe die Gleichungen (5.2) und (5.3)) und der Platznummer $\ddot{U}(W)$ des Wortes in der verketteten Überlaufliste (siehe Bild 5.2) gegeben ist. Diese beiden Werte können platzsparend gespeichert werden, indem man sie in folgender Weise zu einem Code $A^*(W)$ zusammenfaßt:

$$A^*(W) = A(W) + P \cdot \ddot{U}(W). \tag{5.4}$$

Dabei ist P wie bisher die Größe des Adreßbereichs. Bei bekanntem Code $A^*(W)$, der im folgenden als „Hash-Code“ bezeichnet wird, kann die Position eines Wortes im Wörterbuch bestimmt werden, indem man die Hash-Adresse $A(W)$ und die Listen-Nummer $\ddot{U}(W)$ nach folgenden Gleichungen berechnet:

$$A(W) = A^*(W) \bmod P. \tag{5.5}$$

$$\ddot{U}(W) = \frac{A^*(W) - A(W)}{P}. \tag{5.6}$$

Mit diesem Hash-Code ist es möglich, die Position eines Wortes mit nur 4 byte in den Blättern des Baumes zu speichern, d. h. bei der Simulation des Baumaufbaus wird Speicherplatz gespart.

Nach dem Aufbau des Codebaumes und dem Eintragen der Codes in das assoziative Wörterbuch wird ein Text codiert, indem jedes Wort des einlaufenden Textes im assoziativen Wörterbuch gesucht und sein dort gefundener Optimalcode in einem Massenspeicher abgelegt wird. Für das Decodieren wird dieser Massenspeicher bitweise ausgelesen und der dadurch festgelegte Weg durch den binären Baum durchschritten, bis ein Blatt erreicht wird. Mit Hilfe des darin enthaltenen Hash-Codes

kann man dann schließlich auf das gesuchte Wort im assoziativen Wörterbuch zugreifen. Dieser Vorgang wird solange wiederholt, bis alle Daten des Massenspeichers ausgelesen sind.

Bei der Simulation dieses Verfahrens konnten zur Erzeugung der Optimalcodes nur 40 % des LIMAS-Korpus ($\approx$ 400 000 Wörter) eingelesen werden, da der Arbeitsspeicher des verwendeten Rechners (VAX 11/780) für die Auswertung längerer Texte nicht ausreichte. Das assoziative Wörterbuch enthielt dabei insgesamt etwa 60 000 verschiedene Wörter. Die mittlere Codelänge, die bei der Optimalcodierung der Wörter erreicht wurde, beträgt $\overline{C}_{\text{opt}} = 1{,}66$ bit/Buchstabe. Die relativ große Abweichung von dem Küpfmüller'schen Meßwert für die Entropie der Wörter $H_{\text{w}} = 2{,}0$ bit/Buchstabe [13] läßt sich wie folgt erklären: Küpfmüller berechnete die auf Buchstaben bezogene Entropie der Wörter mit Hilfe der mittleren Wortlänge, wobei er die Leerzeichen, die auf die Wörter folgen, *nicht* berücksichtigte. Diese Leerzeichen gehen allerdings bei der Bestimmung der mittleren Codelänge ein, wodurch sich ein kleinerer Wert ergibt. Streng genommen müßte man bei der Berechnung der Entropie die Leerzeichen zwischen den Wörtern ebenfalls einbeziehen.

Wie bei einer späteren Untersuchung [97] für Buchstaben gezeigt werden konnte, wird mit dem Wurzel-Blatt-Algorithmus ein Quasi-Optimalcode erzeugt, der im Mittel 0,05 bit/Buchstabe „schlechter“ ist als der Huffman-Code. Wegen der notwendigen Ungenauigkeiten beim Baumaufbau (siehe Kapitel 4.2) enthält der Huffman-Code allerdings auch noch Redundanz, die man nach Gleichung (4.8) bestimmen kann: Diese Redundanz lag bei den Untersuchungen im Mittel bei 0,15 bit/Buchstabe [97]. Der mit dem Wurzel-Blatt-Algorithmus erzeugte Code enthält demnach eine Rest-Redundanz von etwa 0,2 bit/Buchstabe und ist somit ein nahezu optimaler Code.

5.1.4 Die Berechnung der mittleren Codelänge für große Wortvorräte

Da die Häufigkeitsverteilung der Wörter bekannt ist (Zipf'sches Gesetz [6]), kann man die Entropie der Wörter, d. h. die *minimale* mittlere Codelänge eines Optimalcodes auf Wortebene, nach Gleichung (3.9) berechnen. Dazu benötigt man die relativen Häufigkeiten $p(n)$ der Wörter, die man mit den absoluten Häufigkeiten $a(n)$, der Textlänge s und dem Wortvorrat n_0 unter Berücksichtigung von Gleichung (3.3) berechnet zu

$$p(n) = \frac{a(n)}{s} = \frac{n_0}{s \cdot n}. \tag{5.7}$$

Mit Gleichung (3.9) bestimmt man die Entropie der Wörter

$$H_{\text{w}} = \sum_{n=1}^{n_0} \frac{n_0}{s \cdot n} \cdot \operatorname{ld} \frac{s \cdot n}{n_0} = \left[\frac{n_0}{s} \cdot \sum_{n=1}^{n_0} \frac{\operatorname{ld} n}{n} + \frac{\operatorname{ld}(s/n_0)}{s} \cdot \sum_{n=1}^{n_0} \frac{n_0}{n} \right] \qquad \text{(in bit/Wort)}. \tag{5.8}$$

Die erste Summe in den eckigen Klammern kann für große n_0 näherungsweise mit Hilfe eines Integrals berechnet werden (vergleiche auch mit Gleichung (3.5)). Man approximiert

$$\sum_{n=1}^{n_0} \frac{\operatorname{ld} n}{n} \approx \int_1^{n_0} \frac{\operatorname{ld} n}{n}\,\mathrm{d}n = \frac{1}{\ln 2} \cdot \left[\frac{(\ln n)^2}{2}\right]_1^{n_0} = \frac{(\ln n_0)^2}{2 \cdot \ln 2} \qquad \text{für } n_0 \gg 1. \tag{5.9}$$

Das hier auftretende Integral kann in [14] nachgeschlagen werden. Man überprüft die angegebene Lösung aber ebenso leicht durch Ableiten der Stammfunktion.

Mit den Gleichungen (5.8), (5.9), (3.6) und (3.4) bestimmt man schließlich die Entropie H_w der Wörter, die die untere Schranke für die mittlere Codelänge $\overline{C}_\mathrm{opt}$ der Optimalcodes auf Wortebene darstellt (siehe Gleichung (4.4))

$$\overline{C}_\mathrm{opt} \geq H_\mathrm{w} \approx \left[\frac{\operatorname{ld} n_0}{2} + \operatorname{ld}(\ln n_0)\right] \qquad \text{(in bit/Wort)} \qquad \text{für } n_0 \gg 1. \tag{5.10}$$

Setzt man in diese Gleichung den Wortvorrat $n_0 = 60\,000$, der bei der oben beschriebenen Simulation verwendet wurde, ein, so erhält man $\overline{C}_\mathrm{opt} \geq 11{,}4$ bit/Wort. Mit der mittleren Wortlänge des LIMAS-Korpus von $\overline{\ell} = 5{,}7$ Buchstaben/Wort wird unter Berücksichtigung der auf die Wörter folgenden Leerzeichen dann $\overline{C}_\mathrm{opt} \geq 11{,}4/6{,}7$ bit/Buchstabe $= 1{,}70$ bit/Buchstabe berechnet. Dieser Wert liegt in der Nähe des gemessenen Wertes $\overline{C}_\mathrm{opt} = 1{,}66$ bit/Buchstabe.

Da die in Gleichung (5.10) angegebene Funktion für wachsende n_0 monoton steigend ist, kann mit dem Optimalcode auf Wortebene für realistische Wortvorräte der Shannon'sche Grenzwert von 1 bit/Buchstabe nie erreicht oder gar unterschritten werden. Für eine redundanzärmere Codierung muß man deshalb nach anderen Methoden suchen, wobei folgende Ansätze, zunächst nur in Hinblick auf Optimalcodierungen, zu sehen sind:

- Bei der Optimalcodierung ist es von großem Nachteil, daß man *vor* der Codierung die Codes bestimmen muß und dann nur noch diese *statische* Codetabelle verwendet. Günstiger wäre es, wenn sich der Code *dynamisch* dem zu codierenden Text optimal anpaßt. Auf die Problematik dieses dynamischen Optimalcodes wird im folgenden noch eingegangen.

- Eine Senkung der mittleren Codelänge wäre zu erreichen, wenn man größere sprachliche Einheiten als Wörter, z. B. häufig benutzte Wortfolgen, optimal codiert. Auch dieses Verfahren wird später diskutiert.

- Ein ganz wesentlicher Nachteil der Optimalcodierung auf Wortebene wurde bisher völlig außer Betracht gelassen: Bei der Codierung müssen *alle* Wörter eines Textes bekannt sein. Texte, die unbekannte Wörter, d. h. Wörter für die kein Optimalcode ermittelt wurde, enthalten, können nicht codiert werden. Ein Textcodierer, der unter realistischen Bedingungen arbeiten soll, muß allerdings in der Lage sein, unbekannte Wörter ebenfalls möglichst redundanzarm zu codieren. Eine derartige „universelle“ Optimalcodierung kann

z. B. realisiert werden, indem man bei einem unbekannten Wort auf einen Buchstaben-Optimalcode umschaltet.

5.2 Die dynamische Optimalcodierung

J. Sauerbrey untersuchte verschiedene Verfahren zur dynamischen Optimalcodierung [97]. Diejenige Methode, mit der die besten Codierungsergebnisse erzielt werden konnten, soll im folgenden an einer Codierung auf Buchstabenebene kurz beschrieben werden:

Sowohl dem Codierer als auch dem Decodierer stehen jeweils eine Codetabelle zur Verfügung, die die repräsentativen Optimalcodes z. B. der Buchstaben der deutschen Sprache enthält. Diese Codetabelle, die üblicherweise bei einer *statischen* Optimalcodierung verwendet wird, dient bei der *dynamischen* Codierung lediglich als „Starttabelle“, da während des Codiervorganges periodisch eine optimale Anpassung der Codes an den Text vorgenommen wird. Die Codierung erfolgt dabei in konstanten Intervallen, z. B. in Blöcken zu je 2 000 Buchstaben. Nach der Codierung des ersten Intervalls mit dem Startcode wird eine neue Codetabelle bestimmt, die diesem Intervall optimal angepaßt ist. Mit diesem neuen Code wird das zweite Intervall codiert, an dessen Ende erneut ein Code erzeugt wird, mit dem man den nächsten Abschnitt codiert. Dieses Verfahren hat den Vorteil, daß man dem Decodierer keine Codetabellen übermitteln muß, was sich in einer größeren mittleren Codelänge niederschlagen würde. Dem Decodierer stehen nämlich am Ende jedes Intervalls die notwendigen Informationen zur Verfügung, die er zur Bestimmung der jeweils neuen Codetabelle benötigt.

Obwohl dieses Verfahren zunächst vernünftig erscheint, können damit dennoch keine wesentlich besseren Ergebnisse erzielt werden als mit der statischen Optimalcodierung: Bei einer Intervallbreite von 2 000 Buchstaben war der dynamische Optimalcode auf Buchstabenebene im Mittel nur um ca. 0,01 bit/Buchstabe kürzer als der statische. Für einen guten dynamischen Optimalcode müssen nämlich zwei, sich gegenseitig widersprechende Bedingungen erfüllt sein: Einerseits soll der Codierer sich schnell an Änderungen der Häufigkeitsverteilungen anpassen, was nur mit kurzen Intervallen erreicht werden kann, andererseits muß die Intervallbreite möglichst groß gewählt werden, damit repräsentative Codes erzeugt werden. Sind die Intervalle zu klein, dann stellt sich das System zwar schnell auf die neuen Häufigkeitsverteilungen ein, die Textstichproben sind allerdings für die Ermittlung guter Codes nicht geeignet. Bei zu großer Intervallbreite reagiert der Codierer sehr träge auf Änderungen, wodurch man einen „quasi-statischen“ Code erhält und die Vorteile einer *dynamischen* Optimalcodierung nicht zum Tragen kommen.

Nach den Untersuchungen auf Buchstabenebene bestand die Hoffnung, dieses Verfahren sei vielleicht für die Codierung von Wörtern besser geeignet. Hierbei stellte sich jedoch heraus, daß die dynamische Optimalcodierung auf Wortebene ebenfalls unbefriedigende Ergebnisse liefert, da wegen der großen Anzahl verschiedener

Wörter die Intervallbreite sehr groß gewählt werden muß ($> 3\,000$ Wörter), um repräsentative Codes zu erhalten. Für kleine Intervallbreiten ($< 3\,000$ Wörter) wurden in den meisten Fällen sogar größere mittlere Codelängen ermittelt als bei der *statischen* Optimalcodierung auf Wortebene. Ein statischer Code, der anhand umfangreicher Texte erzeugt wird, ist für eine Codierung deshalb besser geeignet als ein dynamischer Code, der sich wegen zu kleiner Textstichproben nur unzureichend anpassen kann. Eine dynamische Optimalcodierung auf Wortebene ist demnach also *nicht* vorteilhaft.

5.3 Die Optimalcodierung der Nachfolgerwörter

A. Lüssem implementierte einen Textcodierer, bei dem die *Nachfolger* der Wörter optimal codiert werden [98]. Diese Codierung basiert auf der Verwendung von Markoff-Ketten erster Ordnung (siehe Kapitel 4.3) und ist wesentlich aufwendiger als die oben beschriebene Optimalcodierung auf Wortebene, da man für *jedes* der n_0 Wörter eine eigene Codetabelle benötigt: Wenn z. B. das m-te Wort codiert werden soll, das im Text nach dem n-ten Wort steht, so muß in der n-ten Codetabelle der Optimalcode des m-ten Wortes gesucht werden. Durch die Einbeziehung der Bindungen zwischen direkt aufeinanderfolgenden Wörtern wird verhindert, daß sinnlose Wortkombinationen codiert werden können, weshalb sich die mittlere Länge der Optimalcodes verringert. Bei den Untersuchungen konnte mit dieser Methode eine mittlere Codelänge von 0,69 bit/Buchstabe erreicht werden [99].

Allerdings tritt bei der Codierung *beliebiger* Texte folgende Schwierigkeit auf: Anhand eines *Lerntextes* ermittelt man die Häufigkeiten der Nachfolgerwörter und erzeugt alle n_0 Codetabellen; bei der Codierung eines *unbekannten* Textes kann es nun vorkommen, daß ein Wort in einem neuen Kontext auftaucht und nicht in der Codetabelle seines Vorgängers enthalten ist. Zur Lösung dieses Problems kann man beispielsweise mit einem speziellen „Umschaltwort“, das in alle Codetabellen eingetragen werden muß, auf die „gewöhnliche“ Optimalcodierung auf Wortebene umschalten. Diese Umschaltwörter, die beim Decodieren ausgeblendet werden, vergrößern die mittlere Codelänge allerdings erheblich, wie die Untersuchungen zeigten: Bei der Codierung unbekannter Texte benötigt man mit diesem Verfahren im Mittel 4,0 bit/Buchstabe, d. h. die Optimalcodierung auf Wortebene wäre in diesem Fall wesentlich günstiger.*

Prinzipiell ist es möglich, den oben geschilderten Codiervorgang nicht nur in Textrichtung, sondern auch in Text*gegen*richtung vorzunehmen: M. Böttger untersuchte die Optimalcodierung der *Vorgänger*wörter [99] und konnte dabei außerdem die in Gleichung (3.16) getroffene Annahme einer Symmetrie der Vorgänger- und Nachfolger-Verteilungen bestätigen. Die mittlere Codelänge, die mit der Vorgänger-Codierung gemessen wurde, weicht erst in der zweiten Nachkommastelle von derjenigen

* Auf das Problem, daß ein Wort nicht im assoziativen Wörterbuch enthalten ist und deshalb nicht optimal codiert werden kann, wird im folgenden noch eingegangen.

der Nachfolger-Codierung ab. Jedoch erfordert die Optimalcodierung der Vorgänger einen zusätzlichen Verarbeitungsschritt, da man den Text erst „invertieren“ muß, um ihn von hinten nach vorne codieren zu können, so daß die Nachfolger-Codierung günstiger erscheint.

Obwohl die Optimalcodierung der Nachfolgerwörter relativ aufwendig ist, erreicht man damit unter realistischen Bedingungen (Codierung unbekannter Texte) den Shannon'schen Grenzwert nicht. Deshalb soll im folgenden Abschnitt untersucht werden, ob man mit der Optimalcodierung größerer sprachlicher Einheiten bessere Ergebnisse erzielen kann.

5.4 Die Optimalcodierung von Wortfolgen

U. Langer entwickelte ein Textcodierungssystem, das die Codierung ganzer Wortfolgen ermöglicht [100]. Dazu wurden in einer Baumstruktur, die hier nicht näher erläutert werden soll, die häufigsten Redewendungen, die einem Lerntext entnommen waren, gespeichert. Beim Aufbau dieses Baumes wurden zunächst einmal alle Wortgruppen, die bis zu zehn Wörter lang waren, eingetragen, wobei seltene Wortfolgen, d. h. Folgen, die *nicht* nach einer bestimmten Zeit wiederholt im Lerntext vorkamen, gelöscht wurden [77]. Der größte Baum, der bei den durchgeführten Untersuchungen aufgebaut wurde, enthielt etwa 39 000 verschiedene Redewendungen. Mit Hilfe dieser Speicherstruktur konnte ein System simuliert werden, das beim Codieren eines Textes nach der jeweils längsten Wortfolge im Wortfolgenbaum sucht und diese Wortgruppe als Einheit optimal codiert.

Da mit dieser Methode die Bindungen zwischen aufeinanderfolgenden Wörtern erfaßt werden, erwartet man eine kleinere mittlere Codelänge als z. B. bei der Optimalcodierung von Wörtern: Die kürzeste mittlere Codelänge, die mit der Optimalcodierung von Wortfolgen erzielt werden konnte, liegt bei 1,38 bit/Buchstabe, wobei hier der Lerntext codiert wurde. Bei der Codierung fremder Texte konnte unter der Voraussetzung, daß alle Wörter im assoziativen Wörterbuch enthalten sind, eine mittlere Codelänge von 2,0 bit/Buchstabe erreicht werden, d. h. auch hier ist die Optimalcodierung von Wörtern auf jeden Fall vorzuziehen.

Für die Optimalcodierung größerer sprachlicher Einheiten als Wörter ist ein erheblicher Aufwand notwendig, der jedoch in keiner Weise gerechtfertigt ist, wenn man die Codierungsergebnisse betrachtet. Zwar kann unter günstigen Voraussetzungen eine unwesentlich kleinere mittlere Codelänge als bei der Optimalcodierung auf Wortebene erzielt werden, unter realistischeren Bedingungen ist der Wort-Optimalcode allerdings immer überlegen. Deshalb soll im folgenden Abschnitt ein „universeller“ Optimalcode entwickelt werden, mit dem auch Wörter, die nicht im assoziativen Wörterbuch enthalten sind, optimal codiert werden können. An dieser Stelle kann man schon abschätzen, daß die mittlere Codelänge bei diesem Verfahren in der Größenordnung von 2 bit/Buchstabe liegt und der Shannon'sche Grenzwert nicht unterschritten wird.

5.5 Die universelle Optimalcodierung auf Wortebene

Bei der Optimalcodierung von Wörtern kommt es vor, daß ein Wort nicht im assoziativen Wörterbuch gefunden wird und deshalb nicht codiert werden kann. In diesem Fall könnte man z. B. mit Hilfe eines „Umschaltwortes“, von dem oben schon die Rede war, auf einen Buchstaben-Optimalcode umschalten. Dieses Umschaltwort müßte im assoziativen Wörterbuch enthalten sein und einen Optimalcode besitzen. Nach der Codierung des unbekannten Wortes mit dem Buchstaben-Optimalcode könnte wieder auf die Wortebene zurückgeschaltet werden, indem ein Leerzeichen codiert wird.

N. Gümpel implementierte nach dieser Methode einen Textcodierer, mit dem man *beliebige* Texte codieren kann [101]. Anhand des Wörterbuches, das die 24 700 häufigsten Wörter des LIMAS-Korpus enthält, wurde damit bei der Codierung fremder Texte eine mittlere Codelänge von 2,23 bit/Buchstabe erreicht.

Dieser Wert kann gesenkt werden, wenn man unbekannte Wörter nicht in Buchstaben, sondern in *Silben* zerlegt, da es im Deutschen viele zusammengesetzte Wörter gibt, die aus relativ kurzen Wörtern oder Silben aufgebaut sind. Für die Codierung von Silben benötigt man zusätzlich ein assoziatives „Silbenbuch“, das ebenfalls mit Hilfe der Hash-Speicherung (siehe Bild 5.2) realisiert werden kann und die Optimalcodes der häufigsten Silben und aller ASCII-Zeichen enthält: Wenn in einem fremden Wort eine unbekannte Silbe auftritt, dann wird diese Silbe aus ASCII-Zeichen zusammengesetzt und codiert. Mit der Codierung des Leerzeichens am Wortende kehrt man schließlich wieder auf die Wortebene zurück. Problematisch ist bei diesem Verfahren die Erkennung der Silbengrenzen. Im Gegensatz zur Bestimmung der Wortgrenzen, die anhand der Leerzeichen sehr einfach erkannt werden können, benötigt man hier einen Algorithmus, der die deutschen Trennregeln möglichst gut beherrschen sollte.

5.5.1 Der Algorithmus zur Silbentrennung

Konventionelle Silbentrennprogramme benutzen in der Regel umfangreiche Wörterbücher, die sehr viele Wörter mit ihren Trennstellen enthalten [23]. Neue Wörter können manuell getrennt und nachträglich in diese Wörterbücher aufgenommen werden. Da diese Verfahren relativ aufwendig sind, wurde nach einer algorithmischen Methode gesucht, die *ohne* Wörterbuch auskommt: R. Sacher entwickelte einen Trennalgorithmus, der auf einfachen Trennregeln der deutschen Sprache basiert [102]. Dieser Silbentrenner, der noch relativ viele Wörter falsch trennte, wurde von H. Müller optimiert [103]. Die wichtigste Grundregel, die bei diesem Algorithmus angewandt wurde, ist, daß eine Folge von mehreren Konsonanten vor dem letzten getrennt wird (z. B. Silbe → Sil-be) [5].

Der entwickelte Trennalgorithmus soll nun erläutert werden: Das deutsche Alphabet $\mathcal{A}$, das alle deutschen Buchstaben und Umlaute enthält, besteht aus den Vokalen $\mathcal{V} = \{\mathtt{A}, \mathtt{E}, \mathtt{I}, \mathtt{O}, \mathtt{U}, \mathtt{Ä}, \mathtt{Ö}, \mathtt{Ü}, \mathtt{Y}, \mathtt{a}, \mathtt{e}, \mathtt{i}, \mathtt{o}, \mathtt{u}, \mathtt{ä}, \mathtt{ö}, \mathtt{ü}, \mathtt{y}\}$ und den Konsonanten $\mathcal{K} = \mathcal{A} \setminus \mathcal{V}$;

hierbei wird das „Ypsilon“ zu den Vokalen gezählt, da es im Deutschen meistens wie ein „I“ behandelt wird. Das Wort W wird nach folgender Methode getrennt:

1. Setze $i = 0$. Mit $W[i]$ wird der i-te Buchstabe des Wortes W bezeichnet.

2. Inkrementiere i solange um 1, bis $W[i] \in \mathcal{V}$. Wenn vorher das Wortende erreicht wird, dann fahre mit Schritt 8 fort.

3. Wenn $W[i+1] \in \mathcal{V} \wedge W[i+2] \in \mathcal{V}$, dann trenne W zwischen $W[i+1]$ und $W[i+2]$, inkrementiere i um 1 und fahre mit Schritt 2 fort.

4. Inkrementiere i solange um 1, bis $W[i] \in \mathcal{K}$. Wenn vorher das Wortende erreicht wird, dann fahre mit Schritt 8 fort.

5. Inkrementiere i solange um 1, bis $W[i] \in \mathcal{V}$. Wenn vorher das Wortende erreicht wird, dann fahre mit Schritt 8 fort.

6. Wenn i seit dem Wortanfang oder der letzten Trennstelle weniger als viermal inkrementiert wurde, dann fahre mit Schritt 4 fort.

7. Trenne das Wort W zwischen $W[i-2]$ und $W[i-1]$, dekrementiere i um 2 und fahre mit Schritt 2 fort, falls folgende Bedingungen *nicht* erfüllt sind:

   a) Sonderfall „SCH“: $W[i-3] \in \{\mathtt{S}, \mathtt{s}\} \wedge W[i-2] \in \{\mathtt{C}, \mathtt{c}\} \wedge W[i-1] \in \{\mathtt{H}, \mathtt{h}\}$. Wenn i seit dem Wortanfang oder der letzten Trennstelle weniger als sechsmal inkrementiert wurde, dann fahre mit Schritt 4 fort. Andernfalls trenne das Wort W zwischen $W[i-4]$ und $W[i-3]$, dekrementiere i um 4 und fahre mit Schritt 2 fort.

   b) Sonderfälle „CH“, „CK“, „SP“, „ST“, „TR“ und „TH“ mit Ausnahme von „HTH“ und „STH“: $(W[i-2] \in \{\mathtt{C}, \mathtt{c}\} \wedge W[i-1] \in \{\mathtt{H}, \mathtt{K}, \mathtt{h}, \mathtt{k}\}) \vee (W[i-2] \in \{\mathtt{S}, \mathtt{s}\} \wedge W[i-1] \in \{\mathtt{P}, \mathtt{T}, \mathtt{p}, \mathtt{t}\}) \vee (W[i-2] \in \{\mathtt{T}, \mathtt{t}\} \wedge W[i-1] \in \{\mathtt{R}, \mathtt{r}\}) \vee (W[i-3] \notin \{\mathtt{H}, \mathtt{S}, \mathtt{h}, \mathtt{s}\} \wedge W[i-2] \in \{\mathtt{T}, \mathtt{t}\} \wedge W[i-1] \in \{\mathtt{H}, \mathtt{h}\})$. Wenn i seit dem Wortanfang oder der letzten Trennstelle weniger als fünfmal inkrementiert wurde, dann fahre mit Schritt 4 fort. Andernfalls trenne das Wort W zwischen $W[i-3]$ und $W[i-2]$, dekrementiere i um 3 und fahre mit Schritt 2 fort.

   c) Sonderfälle „SCHL“, „SCHM“, „SCHN“, „SCHR“ und „SCHW“: $W[i-4] \in \{\mathtt{S}, \mathtt{s}\} \wedge W[i-3] \in \{\mathtt{C}, \mathtt{c}\} \wedge W[i-2] \in \{\mathtt{H}, \mathtt{h}\} \wedge W[i-1] \in \mathcal{S}$ mit den Sonderfallkonsonanten $\mathcal{S} = \{\mathtt{L}, \mathtt{M}, \mathtt{N}, \mathtt{R}, \mathtt{W}, \mathtt{l}, \mathtt{m}, \mathtt{n}, \mathtt{r}, \mathtt{w}\}$. Wenn i seit dem Wortanfang oder der letzten Trennstelle weniger als siebenmal inkrementiert wurde, dann fahre mit Schritt 4 fort. Andernfalls trenne das Wort W zwischen $W[i-5]$ und $W[i-4]$, dekrementiere i um 5 und fahre mit Schritt 2 fort.

8. Das Wort W ist vollständig in Silben aufgespalten.

In der folgenden Tabelle werden einige Wörter gezeigt, die mit diesem Algorithmus getrennt wurden. Für jedes Wort ist angegeben, bei welchem Schritt es getrennt wurde.

Wort	Silben	Schritt
Mauer	Mau-er	3.
springen	sprin-gen	7.
Eselei	Ese-lei	7.
Flasche	Fla-sche	7a)
Kuhstall	Kuh-stall	7b)
isotherm	iso-therm	7b)
Nachthemd	Nacht-hemd	7.
Wildschwein	Wild-schwein	7c)

Tabelle 5.1: Mit dem Silbentrennalgorithmus korrekt getrennte Wörter. In der rechten Spalte ist angegeben, bei welchem Schritt das Wort getrennt wurde.

Die Fehlerquote dieses Algorithmus wird mit etwa 5 % angegeben [103]. Der Silbentrenner versagt bei manchen Fremdwörtern (siehe dazu auch [24]) und bei zusammengesetzten Wörtern, wenn das zweite Wort mit zwei Konsonanten oder einem Vokal beginnt. Beispiele für fehlerhaft getrennte Wörter sind in der nächsten Tabelle gegeben.

Wort	Ergebnis des Algorithmus	Korrekte Trennung
Katastrophe	Ka-tas-trop-he	Ka-ta-stro-phe
Programm	Prog-ramm	Pro-gramm
Signal	Sig-nal	Si-gnal
Bandbreite	Bandb-rei-te	Band-brei-te
Waldeule	Wal-deu-le	Wald-eu-le
Häschen	Hä-schen	Häs-chen
Haustür	Hau-stür	Haus-tür
Waschmaschine	Wa-schma-schi-ne	Wasch-ma-schi-ne

Tabelle 5.2: Mit dem Silbentrennalgorithmus fehlerhaft getrennte Wörter.

Zur Verbesserung der obigen Trennmethode könnte man beispielsweise noch mehr Sonderfälle berücksichtigen. Jedoch zeigte sich bei den Untersuchungen, daß dies zu neuen Falschtrennungen führt, die vorher nicht aufgetreten waren (siehe dazu auch die letzten drei Beispiele in Tabelle 5.2). Aus diesem Grund wurde auf eine kompliziertere Sonderfallbehandlung verzichtet, die auch der Forderung eines möglichst

geringen Aufwands widersprochen hätte. Außerdem ist es gar nicht notwendig, einen *perfekten* Silbentrenner zu entwickeln, da der obige Algorithmus ausschließlich für die Verwendung in einem Textcodierer konzipiert wurde, bei dem es nicht darauf ankommt, grammatikalisch einwandfreie Silben zu codieren, sondern unbekannte Wörter in möglichst günstige Bruchstücke zu zerlegen, die redundanzarm codiert werden können. Obendrein liegt die Fehlerrate bei professionell eingesetzten Silbentrennern (z. B. bei dem für diese Arbeit verwendeten Textverarbeitungssystem [25, 26]) ebenfalls im Bereich bis zu 5 %, so daß diese relativ hohe Fehlerquote auch hier akzeptiert werden kann.

5.5.2 Die Ergebnisse der Simulation

J. Sallanz implementierte einen universellen Textcodierer, der nach dem oben skizzierten Verfahren arbeitet [104]. Das dafür verwendete assoziative Wörterbuch enthält die Optimalcodes der 24 700 häufigsten Wörter des LIMAS-Korpus, das assoziative Silbenbuch die Codes der 4 600 häufigsten Silben sowie die Optimalcodes aller ASCII-Zeichen. Bei der Codierung werden unbekannte Wörter aus Silben zusammengesetzt und unbekannte Silben aus ASCII-Zeichen. Der hierfür notwendige Wechsel der Codetabellen wird mit Hilfe eines Umschaltwortes erreicht (siehe oben). Mit dieser Methode konnte bei der Codierung *fremder* Texte eine mittlere Codelänge von $\overline{C}_{\text{uni}} = 2{,}12$ bit/Buchstabe erzielt werden.

Anhand der oben beschriebenen Verfahren wird ersichtlich, daß die Methode der Optimalcodierung offenbar nicht dazu geeignet ist, mit geringem Aufwand eine redundanzarme Codierung zu erreichen. Im folgenden soll deshalb nach neuen Methoden gesucht werden, die nicht allein auf der Auswertung von statistischen Häufigkeitsmessungen basieren. Wenn bei der Codierung die „Struktur“ der Sprache eingehen könnte, wären voraussichtlich bessere Codierungsergebnisse möglich.

6 Die Methode der assoziativen Textcodierung

Bei den im vorigen Kapitel erläuterten Optimalcode-Verfahren werden primär die Häufigkeiten der Textelemente (z. B. der Wörter) berücksichtigt, nicht dagegen ihre strukturelle Anordnung. Zwar schließt man mit der Verwendung des assoziativen Wörterbuches, das in Kapitel 5.1.1 beschrieben wurde, sinnlose Buchstabenkombinationen von vornherein aus, die Wörter dagegen können beliebig aneinandergereiht werden. W. Hilberg schlägt für die Erfassung der Bindungen der Wörter untereinander eine neuartige Speicherstruktur vor, die aus assoziativen Speichern besteht [50, 44, 41]. Assoziative Speicher, die schon recht lange bekannt sind und 1956 erstmals realisiert wurden [27, 28], setzt man vorteilhaft dann ein, wenn man bestimmte Informationen aus unstrukturierten Datenmengen sehr schnell finden muß, wie z. B. bei dem assoziativen Wörterbuch, das in Bild 5.1 auf Seite 33 gezeigt ist. Bevor die neue Verschaltung dieser Speicher im folgenden an einigen Beispielen erläutert wird, soll zunächst das bekannte Prinzip eines assoziativen Speichers kurz dargestellt werden.

Im Bild 6.2 ist ein assoziativer Speicher gezeigt, der alle Wörter des Textes aus Bild 6.1 mitsamt ihren Binärcodes enthält.

HUNDE BELLEN GERNE LAUT
KATZEN FRESSEN GERNE FISCH
HUNDE FRESSEN SELTEN FISCH

Bild 6.1: Beispieltext zur Erläuterung der assoziativen Textcodierung.

Bild 6.2: Beispiel für einen vollassoziativen Speicher, der alle Wörter des Textes aus Bild 6.1 sowie ihre Binärcodes enthält.

In diesem assoziativen Speicher sind zusammengehörige Daten zeilenweise abgelegt. Gibt man einen Teil einer bestimmten Zeile (z. B. ein Wort) in diesen Speicher

ein, so wird der Rest (z. B. der Code, der hier einem Binärcode konstanter Länge, d. h. einer Adresse entspricht) „assoziiert“ und ausgegeben. Man unterscheidet dabei *voll-* und *teil*assoziative Speicher: Der Speicher in Bild 5.1 ist teilassoziativ, da die Codespalte nicht zum Assoziieren benutzt werden kann. Bei einem vollassoziativen Speicher (siehe z. B. Bild 6.2) kann man dagegen alle Spalten zum Suchen verwenden: Der bekannte Teil einer Zeile wird an den Eingang angelegt, während man den Eingang des unbekannten Teils „maskiert“, d. h. in einem unbestimmten Zustand läßt. Der assoziative Speicher ergänzt dann den unbekannten Teil, den man am Ausgang abgreifen kann. Zum Beispiel gibt man beim Codieren des Textes aus Bild 6.1 ein Wort W_i in den assoziativen Speicher von Bild 6.2 ein und erhält den Code A_i dieses Wortes am Ausgang.

Bild 6.3: Das Codieren eines Wortes mit Hilfe des assoziativen Speichers aus Bild 6.2.

Bei den folgenden Beispielen soll von dem vereinfachten Fall ausgegangen werden, daß immer genau *eine* Zeile assoziiert wird: Die anderen Fälle, bei denen kein „Treffer“ oder Mehrfachassoziationen auftreten, werden zunächst außer Betracht gelassen.

Mit dem in Bild 6.3 gezeigten assoziativen Speicher, der im Prinzip dem assoziativen Wörterbuch in Bild 5.1 entspricht, kann man den Text aus Bild 6.1 codieren. Dabei ergibt sich folgende Bitfolge:

100	000	011	110
101	010	011	001
100	010	111	001

Tabelle 6.1: Mit dem assoziativen Speicher aus Bild 6.3 codierter Text.

Zum Decodieren gibt man die Codes in den assoziativen Speicher von Bild 6.2 ein und maskiert den Worteingang. Am Ausgang erscheinen dann die assoziierten Wörter, aus denen sich der ursprüngliche Text wiedergewinnen läßt (siehe Bild 6.4).

Bild 6.4: Das Decodieren eines Wortes mit Hilfe des assoziativen Speichers aus Bild 6.2.

Ein assoziativer Speicher muß nicht unbedingt in Hardware aufgebaut sein: Wie in Kapitel 5.1.1 erwähnt, können Assoziativspeicher mit sehr großen Kapazitäten heute noch nicht gefertigt werden, weshalb man sich damit begnügen muß, z. B. mit Hilfe der Streuspeicherung Massenspeicher mit quasi-assoziativem Verhalten zu konstruieren (siehe Bild 5.2 auf Seite 34 sowie [28, 29]).

Da bei dem obigen Codierbeispiel beliebige Wortkombinationen codierbar sind, ist der codierte Text (siehe Tabelle 6.1) nicht redundanzfrei. Wenn man nur die tatsächlich vorkommenden Wortkombinationen zuläßt und für eine Codierung benutzt, dann ist ein redundanzärmerer Code erreichbar. Zum Beispiel könnte man dazu die Vorgänger-Nachfolger-Beziehungen der Wörter verwenden, die in der sogenannten „Übergangsmatrix“ übersichtlich darstellbar sind.

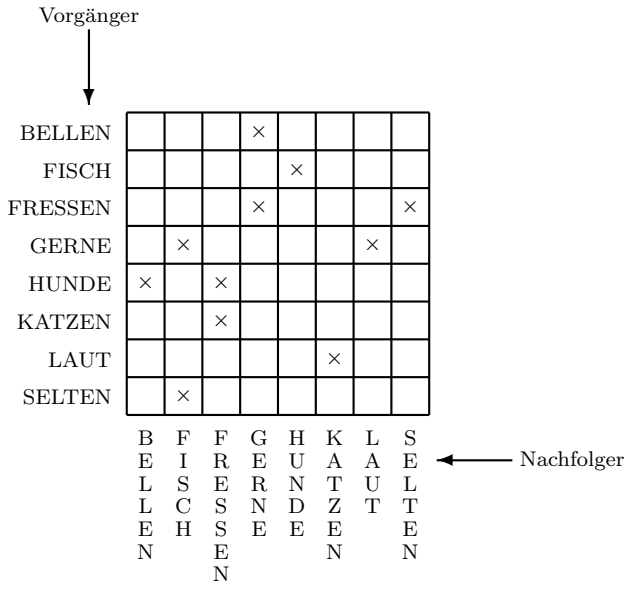

Bild 6.5: Die unsortierte Übergangsmatrix für den Text aus Bild 6.1.

Jedes „Sternchen“ in dieser Matrix entspricht einem erlaubten „Wort-Übergang“. Offensichtlich kommt nur ein Bruchteil aller theoretisch möglichen Wort-Übergänge, d. h. Wortpaare in dem Beispieltext vor. Diesen Bruchteil, der als „Füllungsgrad“ σ bezeichnet werden soll, berechnet man mit der Anzahl n_{paar} der Wortpaare (siehe Gleichung (3.27)) und der Anzahl n_0 der verschiedenen Wörter zu

$$\sigma = \frac{n_{\text{paar}}}{n_0^2}. \tag{6.1}$$

Für $n_0 = 10^5$ (LIMAS-Korpus [2]) ergibt sich mit dem Näherungswert aus Gleichung (3.28) dann

$$\sigma \approx 4{,}5 \cdot 10^{-5} \qquad \text{für } n_0 = 10^5. \tag{6.2}$$

Demnach kommt bei Texten in der Größenordnung des LIMAS-Korpus im Mittel unter etwa 20 000 möglichen Übergängen jeweils nur ein einziger tatsächlich vor, d. h. mit einer geeigneten Codierung wäre eine starke Redundanzreduktion möglich. W. Hilberg schlug hierfür ursprünglich vor, die Übergangsmatrix durch Spalten- und Zeilenvertauschungen so zu sortieren, daß die Übergänge nicht gleichmäßig verteilt, sondern gehäuft in sogenannten „Clustern“ angeordnet sind [51, 52, 53, 54]. In dem folgenden Bild ist die nach dieser Vorgabe umsortierte Matrix aus Bild 6.5 wiedergegeben, wobei man die Häufungen deutlich erkennen kann.

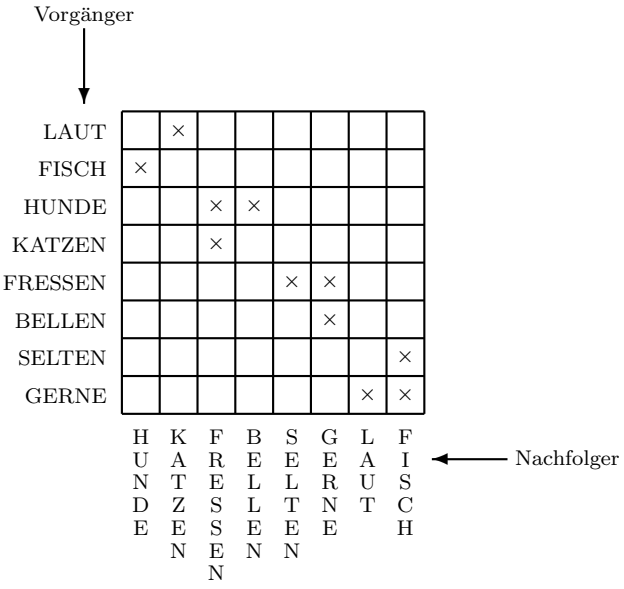

Bild 6.6: Die sortierte Übergangsmatrix für den Text aus Bild 6.1.

Der assoziative Speicher, der in Bild 6.2 gezeigt ist, kann folgendermaßen erweitert werden [44, 41]: Die Adresse A_i des Wortes W_i wird in zwei Teile aufgespalten, die als „Schlösser“ y und x bezeichnet werden (siehe Bild 6.7). Wenn zwei „Schlüssel“ Y_i und X_i eingegeben werden, die in diese Schlösser „passen“, dann werden die Nutzinformation I, die dem gespeicherten Wort W_i entspricht, und der Schlüssel Y_{i+1} ausgegeben. Dieser Schlüssel muß bei allen Nachfolgerwörtern des Wortes W_i passen. Mit dem *externen* Schlüssel X_{i+1} wird dann der Nachfolger W_{i+1} ausgewählt,

d. h. man benötigt zum Codieren dieses Wortes nicht die vollständige Adresse, die aus den Schlüsseln Y_{i+1} und X_{i+1} besteht, sondern nur einen Teil.

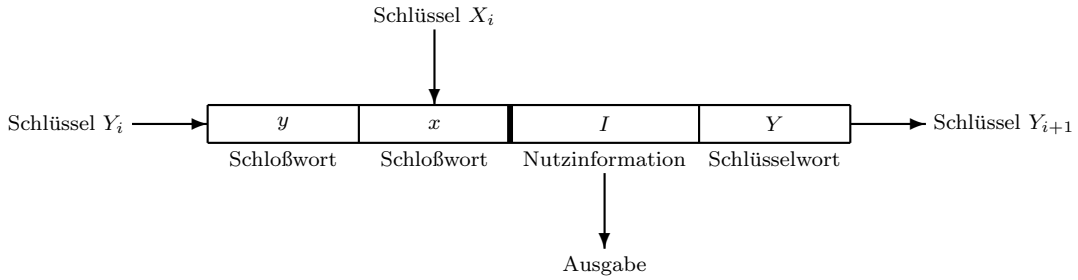

Bild 6.7: Die Hilberg'sche Einteilung einer Zeile eines assoziativen Speichers (nach [44, 41]).

Im folgenden Bild ist der auf diese Art erweiterte assoziative Speicher aus Bild 6.2 wiedergegeben, wobei die Übergangsmatrix von Bild 6.6 für die Verteilung der Schlösser und Schlüssel verwendet wurde: In diesem Beispiel haben die Wörter maximal zwei Nachfolger, d. h. der externe Schlüssel X_i besteht lediglich aus einer Bitstelle. In Bild 6.6 sind insgesamt vier Cluster zu erkennen: Der Schlüssel Y_i, der auch *Verkettungs*schlüssel genannt wird, muß demnach 2 bit breit sein.

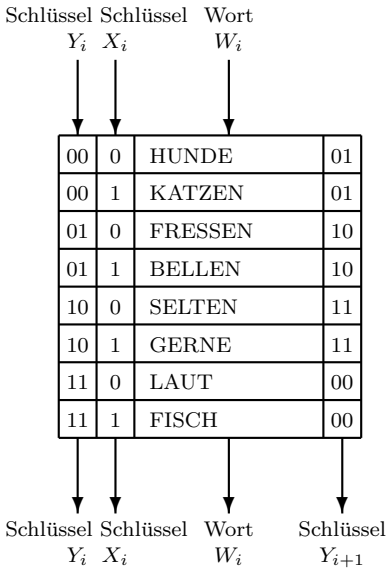

Bild 6.8: Organisation eines assoziativen Speichers nach W. Hilberg [44, 41].

Beim Codieren eines Textes gibt man das Wort W_i ein und maskiert die Spalte X_i. Die Schlüssel X_i und Y_{i+1} werden dann ausgegeben (siehe Bild 6.9). Anhand eines Beispielsatzes wird diese Speicherorganisation näher erläutert:

Der erste Satz des obigen Beispieltextes wird in Bild 6.10 codiert, wobei zur einfacheren Darstellung vier identische assoziative Speicher aus Bild 6.9 kaskadiert worden sind: Die Schlüssel Y_{i+1} werden dem jeweils nächsten Speicher zugeführt. Wenn man mit einer derartigen Speicheranordnung, die „assoziatives Feld" genannt

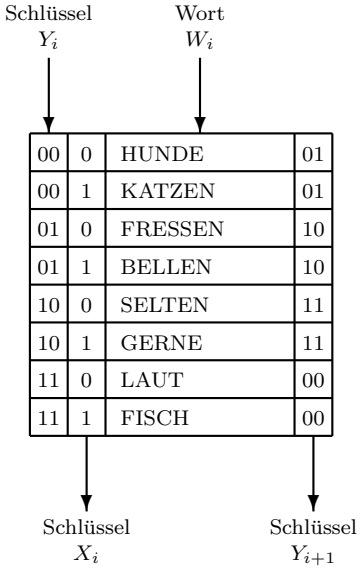

Bild 6.9: Das Codieren eines Wortes mit Hilfe des assoziativen Speichers aus Bild 6.8.

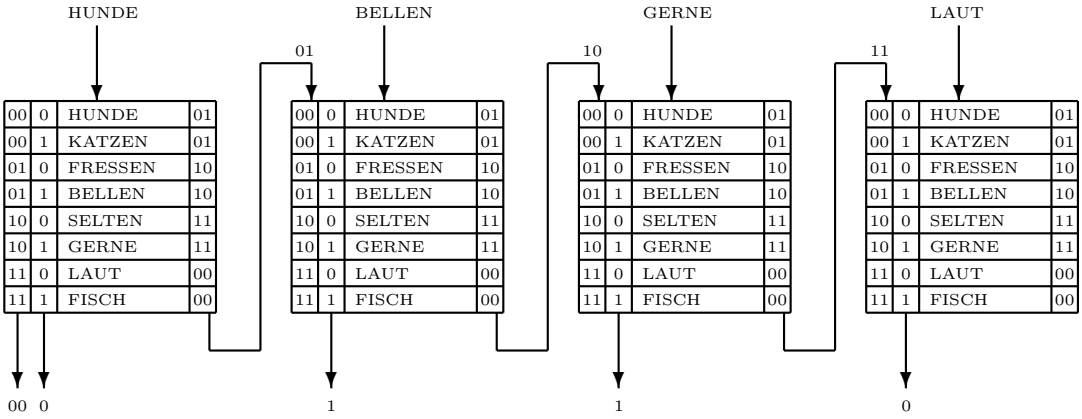

Bild 6.10: Das Codieren eines Textes mit Hilfe eines assoziativen Feldes.

wird [44], den gesamten Text aus Bild 6.1 codiert, erhält man einen Code, der wesentlich kürzer ist als derjenige von Tabelle 6.1: Mit der beschriebenen „assoziativen“ Codierung wird demnach eine erhebliche Redundanzreduktion erzielt.

00	0	1	1	0
	1	0	1	1
	0	0	0	1

Tabelle 6.2: Mit dem assoziativen Feld aus Bild 6.10 codierter Text.

Beim Decodieren maskiert man den Worteingang des assoziativen Speichers aus Bild 6.8 und gibt die Schlüssel Y_i und X_i ein. Das Wort W_i und der Schlüssel Y_{i+1} werden dann ausgegeben (siehe Bild 6.11). In Bild 6.12 ist der Decodiervorgang des in Bild 6.10 codierten Satzes gezeigt: Auch hier benutzt man vier kaskadierte assoziative Speicher.

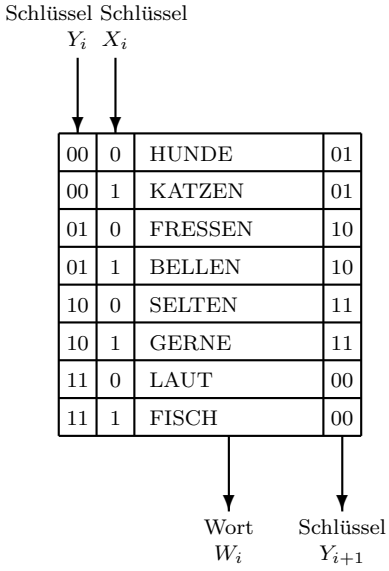

Bild 6.11: Das Decodieren eines Wortes mit Hilfe des assoziativen Speichers aus Bild 6.8.

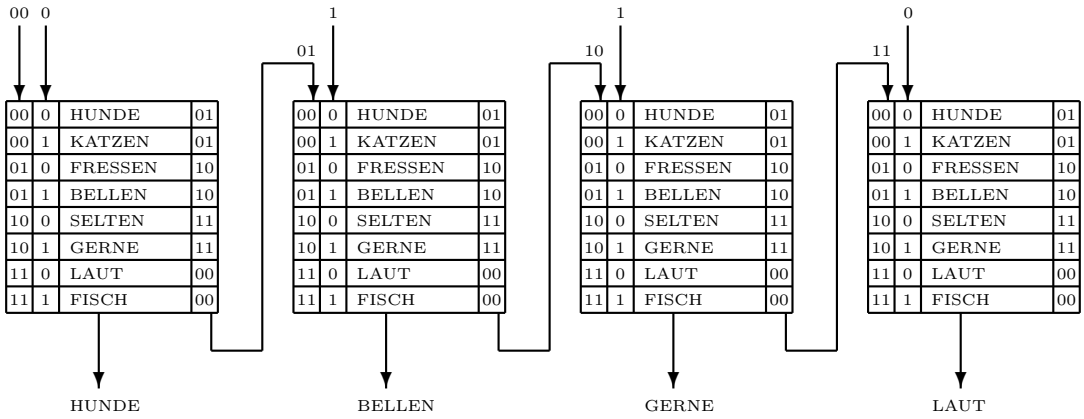

Bild 6.12: Das Decodieren eines Textes mit Hilfe eines assoziativen Feldes.

Mit dem in Bild 6.12 gezeigten assoziativen Feld können wesentlich mehr Sätze decodiert werden, als im Text von Bild 6.1 enthalten sind: An die Eingänge dieses Feldes muß man einen 6-bit-Code anlegen, d. h. es existieren insgesamt $2^6 = 64$ decodierbare Sätze. Variiert man nur die externen Schlüssel X_i für ein festes Y_1, mit dem man z. B. das Cluster derjenigen Wörter auswählt, die in dem Beispieltext von Bild 6.1 am Satzanfang stehen, dann schrumpft die Menge der decodierbaren Sätze auf $2^4 = 16$: Für $Y_1 = 00$ sind alle diese Sätze in Tabelle 6.3 aufgelistet.

Betrachtet man die Übergangsmatrix der Sätze aus Tabelle 6.3, die in Bild 6.13 wiedergegeben ist, so stellt man fest, daß die unvollständigen Cluster der Matrix von Bild 6.6 ergänzt worden sind: Bei der Codierung mit dem assoziativen Feld wird neben einem Wort W_i auch der Schlüssel Y_{i+1} ausgegeben, mit dem man das Cluster auswählt, das den Nachfolger W_{i+1} enthält; mit dem externen Schlüssel X_{i+1} kann dann *jedes* Wort dieses Clusters angesprochen werden. Wie man anhand von Tabelle 6.3 erkennt, sind die ergänzten Wort-Übergänge nicht immer sinnvoll.

X_1	X_2	X_3	X_4		Ausgegebener Satz
0	0	0	0	$\widehat{=}$	HUNDE FRESSEN SELTEN LAUT
0	0	0	1	$\widehat{=}$	HUNDE FRESSEN SELTEN FISCH
0	0	1	0	$\widehat{=}$	HUNDE FRESSEN GERNE LAUT
0	0	1	1	$\widehat{=}$	HUNDE FRESSEN GERNE FISCH
0	1	0	0	$\widehat{=}$	HUNDE BELLEN SELTEN LAUT
0	1	0	1	$\widehat{=}$	HUNDE BELLEN SELTEN FISCH
0	1	1	0	$\widehat{=}$	HUNDE BELLEN GERNE LAUT
0	1	1	1	$\widehat{=}$	HUNDE BELLEN GERNE FISCH
1	0	0	0	$\widehat{=}$	KATZEN FRESSEN SELTEN LAUT
1	0	0	1	$\widehat{=}$	KATZEN FRESSEN SELTEN FISCH
1	0	1	0	$\widehat{=}$	KATZEN FRESSEN GERNE LAUT
1	0	1	1	$\widehat{=}$	KATZEN FRESSEN GERNE FISCH
1	1	0	0	$\widehat{=}$	KATZEN BELLEN SELTEN LAUT
1	1	0	1	$\widehat{=}$	KATZEN BELLEN SELTEN FISCH
1	1	1	0	$\widehat{=}$	KATZEN BELLEN GERNE LAUT
1	1	1	1	$\widehat{=}$	KATZEN BELLEN GERNE FISCH

Tabelle 6.3: Auflistung aller mit dem assoziativen Feld von Bild 6.12 decodierbaren Sätze für den Initialisierungsschlüssel $Y_1 = 00$.

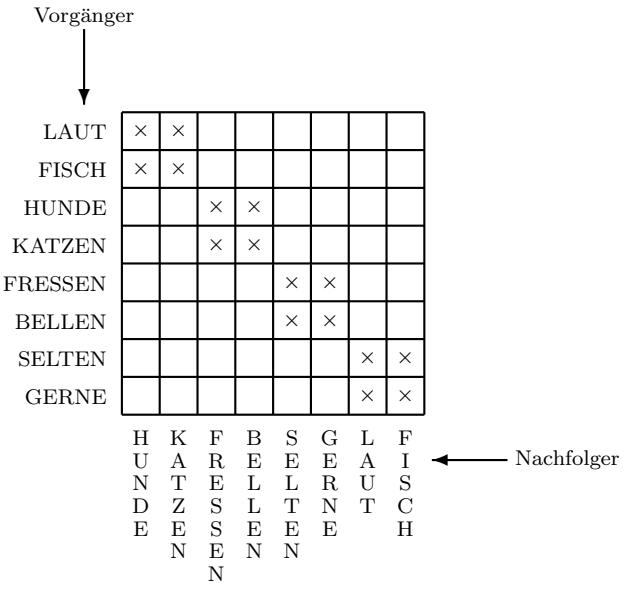

Bild 6.13: Die Übergangsmatrix für die Sätze aus Tabelle 6.3.

Für praktische Codierungsaufgaben ist das beschriebene Modell eines assoziativen Feldes mit separaten Clustern nicht brauchbar: M. Sturm und K. Engel scheiterten schon an dem Versuch, die Übergangsmatrix der Wörter eines relativ kurzen Textes ($\approx$ 50 verschiedene Wörter) so zu sortieren, daß die für die Schlösser- und Schlüsselvergabe notwendigen Cluster entstehen [105]. Die anfängliche Vermutung, die „Sternchen“ einer realen Übergangsmatrix seien gleichmäßig auf alle Zeilen ver-

teilt, konnte nämlich nicht bestätigt werden: Die Verzweigungsgradverteilung, die in Bild 3.10 auf Seite 17 wiedergegeben ist, zeigt, daß bei einer relativ geringen Anzahl der Matrixzeilen viele Übergänge möglich sind, während die meisten Zeilen nur wenige Sternchen enthalten. Wegen der Wörter, die viele Nachfolger haben, ist es nicht möglich, reale Matrizen zufriedenstellend zu „clustern".

M. El-Chami und D. Schmitt versuchten, die Schwierigkeiten, die beim Sortieren der Übergangsmatrix festgestellt wurden, zu umgehen, indem sie die Wörter in zwei Gruppen aufteilten [106]: Die Wörter mit vielen Nachfolgern wurden mit Optimalcodes, alle anderen Wörter unter Anwendung des assoziativen Feldes codiert. Obwohl beim Sortieren der hierfür verwendeten Übergangsmatrix, die nur die Übergänge zwischen den Wörtern mit wenigen Nachfolgern enthielt, bessere Ergebnisse als bei der vorangegangenen Untersuchung erzielt wurden, lag die mittlere Codelänge bei diesem Verfahren deutlich über derjenigen der „reinen" Optimalcodierung, da jedem Code ein „Entscheidungsbit" vorausgeschickt werden muß, das dem Decodierer ankündigt, ob ein Optimal- oder ein Assoziativcode folgt.

Nach diesen Voruntersuchungen analysierten A. Hantschel und M. Küpper das obige Clusterungsproblem systematisch, wobei sie die Vor- und Nachteile verschiedener Algorithmen ermittelten [107]. Das Ergebnis dieser Untersuchung ergab, daß die Übergangsmatrizen realer Texte nicht nach den obigen Vorgaben sortierbar sind, da sich die Wörter nicht in Gruppen, d. h. Cluster, einteilen lassen, die vollständig entkoppelt voneinander sind: Bei der Matrix, die in Bild 6.13 gezeigt ist, können auf die Wörter einer bestimmten Gruppe nur die Wörter *einer* anderen Gruppe folgen. Bei realen Matrizen existieren dagegen immer Wörter, deren Nachfolger auf mehrere Gruppen verteilt sind. Selbst wenn man versucht, diese Nachfolger in einer Gruppe zusammenzufassen, verstreut man die Nachfolger anderer Wörter, die schon in einem Cluster vereinigt waren, auf mehrere Gruppen.

C. Anastasiu untersuchte schließlich noch eine letzte Methode [108]: Wie aus der vorangegangenen Untersuchung hervorgeht, verursachen die Wörter, die sehr viele Nachfolger haben, starke Verkopplungen zwischen den Matrixzeilen; die maximale Anzahl der Nachfolger kann mit Hilfe eines Verfahrens, das in Kapitel 8 erläutert wird und auf der Verwendung eines Sprachnetzwerkes beruht, auf einen beliebigen Wert (z. B. 256 Nachfolger) begrenzt werden. Es bestand die Hoffnung, daß durch die Verringerung der Nachfolgeranzahl auch die Verkopplungen zwischen den Matrixzeilen abnehmen, wodurch sich die Matrix besser sortieren lassen müßte. Bei den Experimenten konnte diese Hoffnung jedoch nicht verifiziert werden: Die Übergangsmatrix realer Texte ist „quasi-stochastisch" mit Sternchen gefüllt und läßt sich nicht nach den vorgegebenen Bedingungen sortieren. Aus diesem Grund ist es nicht möglich, das assoziative Feld von Bild 6.10 für die Codierung großer Textmengen zu benutzen.

Es existiert jedoch eine Speicherorganisation, die praktisch das gleiche „Klemmenverhalten" hat wie diejenige aus Bild 6.8 und zuerst eine Variante für die Simulation eines assoziativen Feldes darstellte [81]: Mit Hilfe des assoziativen Speichers aus Bild 6.2 wandelt man einen Text zunächst in eine Codefolge ähnlich Tabelle 6.1 um;

nach der Codierung des ersten Wortes wird das zweite Wort unter Verwendung des in Bild 6.14 gezeigten assoziativen Speichers codiert. In diesem Speicher stehen die Nachfolger aller Wörter, wobei in der linken Spalte die Adressen A_i der Vorgänger und in der rechten Spalte die Adressen A_{i+1} der Nachfolger gespeichert sind. In der mittleren Spalte sind die Nachfolger jedes Wortes binär durchnumeriert. Das Nachfolgerwort wird mit dieser Nummer, die dem Schlüssel X_i in Bild 6.8 entspricht und deshalb analog dazu bezeichnet werden soll, codiert, wie man in Bild 6.15 sehen kann: Nach der Eingabe der Adressen zweier aufeinanderfolgender Wörter und der Maskierung der mittleren Spalte wird der Schlüssel X_{i+1} ausgegeben.

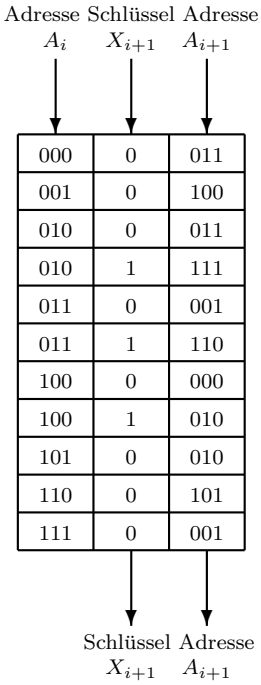

Bild 6.14: Die Grundstruktur für ein realisierbares Codierungssystem.

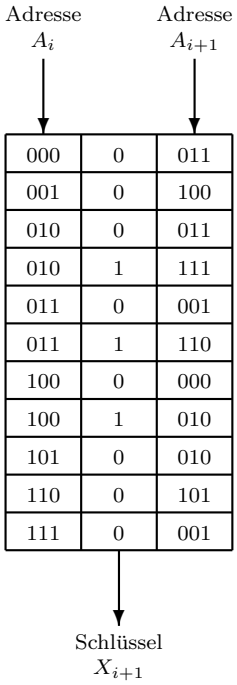

Bild 6.15: Das Codieren eines Wortes mit Hilfe des assoziativen Speichers aus Bild 6.14.

Diese neue Speicherorganisation soll anhand eines Beispiels näher erläutert werden: Der erste Satz des Textes aus Bild 6.1 wird in Bild 6.16 codiert. Analog zu Bild 6.10 kaskadiert man hier die assoziativen Speicher der Bilder 6.3 und 6.15. Nach der Umwandlung der Wörter W_i in ihre Adressen A_i und der Codierung des ersten Wortes mit der Adresse A_1 codiert man die Nachfolgerwörter, indem die Adressen der entsprechenden Wortpaare an die unteren assoziativen Speicher angelegt und die Codes X_{i+1} ausgegeben werden.

Der Code des vollständigen Textes aus Bild 6.1 ist in Tabelle 6.4 gezeigt: Das erste Wort dieses Textes wird mit der Adresse A_1 *absolut* codiert, für alle folgenden Wörter verwendet man eine Art *Differenz*-Code. Man erhält hier zwar eine andere Bitfolge als bei der Codierung mit dem assoziativen Feld aus Bild 6.10 (siehe Tabelle 6.2), die Codelängen sind für diesen Beispieltext allerdings identisch.

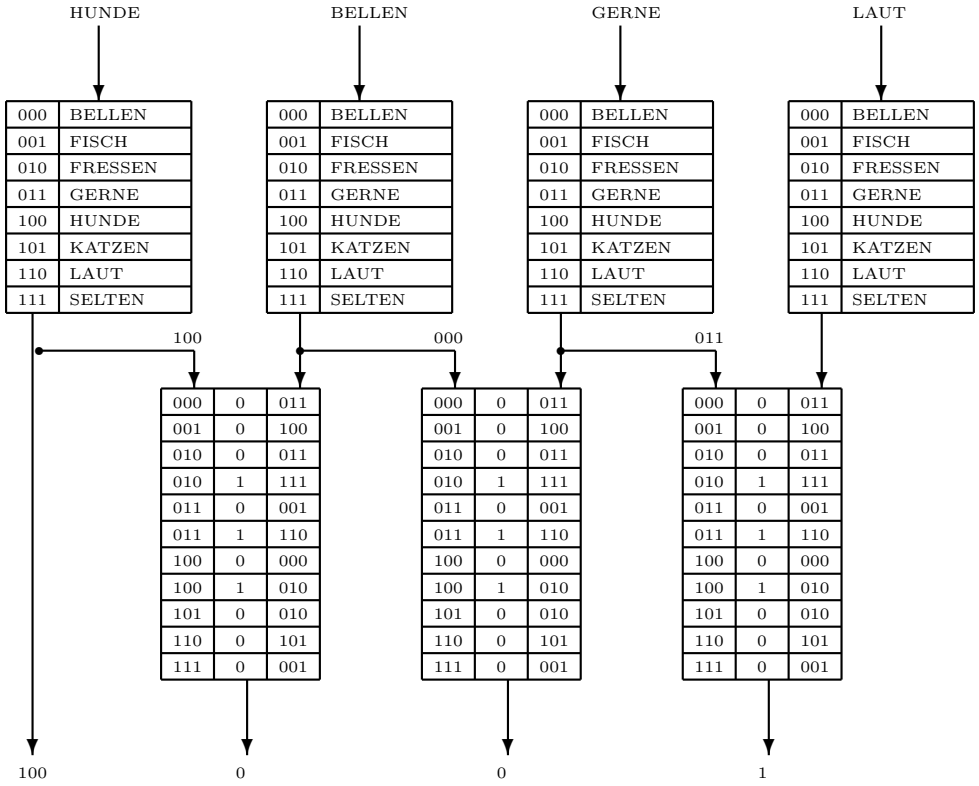

Bild 6.16: Das Codieren eines Textes mit Hilfe des erweiterten assoziativen Feldes.

100	0	0	1
0	0	0	0
0	1	1	0

Tabelle 6.4: Mit dem erweiterten assoziativen Feld aus Bild 6.16 codierter Text.

Beim Decodieren gibt man die Adresse A_i und den Schlüssel X_{i+1} in den assoziativen Speicher von Bild 6.14 ein und maskiert die rechte Spalte (siehe Bild 6.17). Die ausgegebene Adresse A_{i+1} des Nachfolgers wird danach mit Hilfe des assoziativen Speichers aus Bild 6.4 in das Wort W_{i+1} umgewandelt. Der Decodiervorgang des in Bild 6.16 codierten Satzes ist in Bild 6.18 wiedergegeben.

Mit diesem assoziativen Feld lassen sich ebenfalls mehr Sätze decodieren als im Text von Bild 6.1 enthalten sind: Variiert man die externen Schlüssel X_2–X_4 für die Initialisierungsadressen $A_1 \in \{100, 101\}$, die den Wörtern an den Satzanfängen des Beispieltextes entsprechen, dann sind insgesamt acht Sätze decodierbar, die in Tabelle 6.5 gezeigt sind (vergleiche auch mit Tabelle 6.3). Die Übergangsmatrix dieser Sätze entspricht derjenigen in Bild 6.6, d. h. anders als bei dem ursprünglichen assoziativen Feld (siehe die Bilder 6.12 und 6.13) werden hier keine neuen Wort-Übergänge ermöglicht. Da einige Wörter nur einen einzigen Nachfolger besitzen, kann man nicht jede Kombination der externen Schlüssel X_2–X_4 decodieren: Aus diesem Grund ist die Anzahl der Sätze in Tabelle 6.5 kleiner als in Tabelle 6.3.

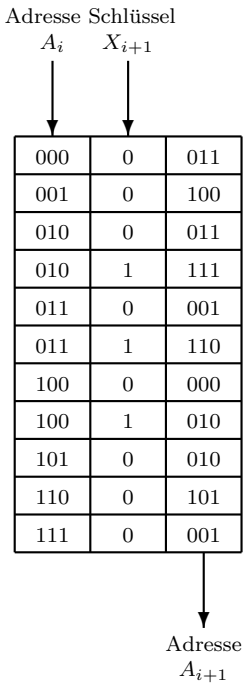

Bild 6.17: Das Decodieren eines Wortes mit Hilfe des assoziativen Speichers aus Bild 6.14.

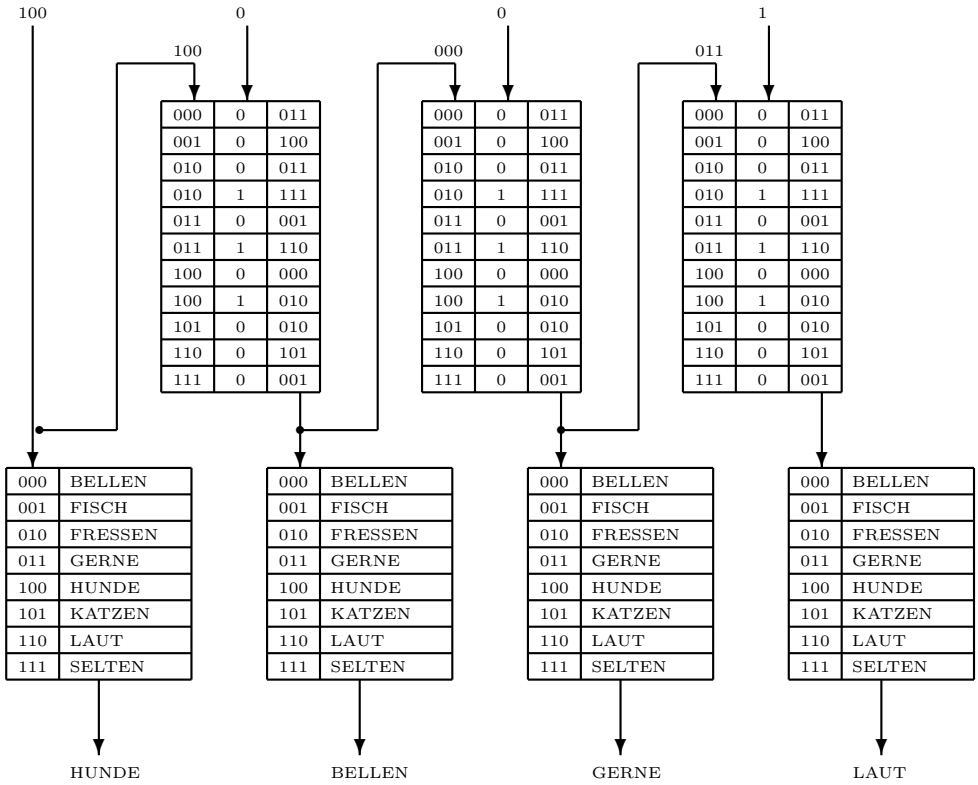

Bild 6.18: Das Decodieren eines Textes mit Hilfe des erweiterten assoziativen Feldes.

A_1	X_2	X_3	X_4		Ausgegebener Satz
100	0	0	0	$\widehat{=}$	HUNDE BELLEN GERNE FISCH
100	0	0	1	$\widehat{=}$	HUNDE BELLEN GERNE LAUT
100	1	0	0	$\widehat{=}$	HUNDE FRESSEN GERNE FISCH
100	1	0	1	$\widehat{=}$	HUNDE FRESSEN GERNE LAUT
100	1	1	0	$\widehat{=}$	HUNDE FRESSEN SELTEN FISCH
101	0	0	0	$\widehat{=}$	KATZEN FRESSEN GERNE FISCH
101	0	0	1	$\widehat{=}$	KATZEN FRESSEN GERNE LAUT
101	0	1	0	$\widehat{=}$	KATZEN FRESSEN SELTEN FISCH

Tabelle 6.5: Auflistung aller mit dem erweiterten assoziativen Feld von Bild 6.18 decodierbaren Sätze für die Initialisierungsadressen $A_1 = 100$ und $A_1 = 101$.

Der Aufwand, den man für die Codierung eines Textes treiben muß, ist bei der Verwendung des erweiterten assoziativen Feldes aus Bild 6.16 relativ groß: Man benötigt für jedes Wort (außer dem ersten) jeweils zwei assoziative Speicher, deren Zusammenschaltung im folgenden Bild gezeigt ist. Bei diesem Blockschaltbild wird das Wort W_i codiert, indem man die Adresse A_{i-1} des Vorgängers eingibt: Man erhält dann den Schlüssel X_i sowie die Adresse A_i.

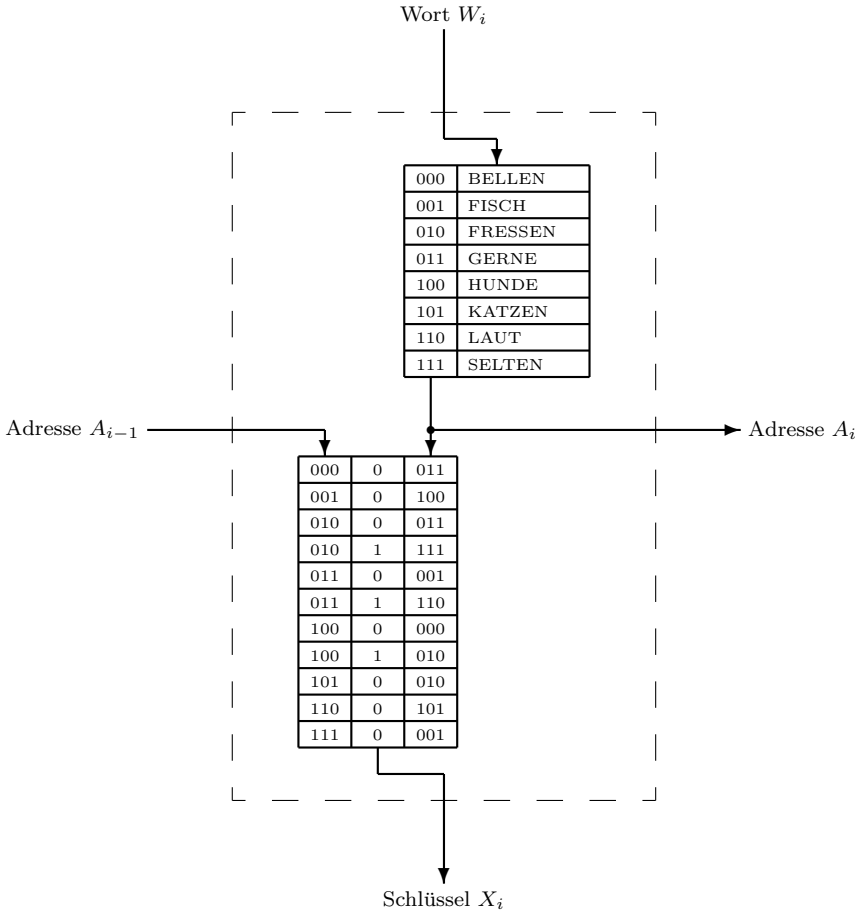

Bild 6.19: Speichermodul für das erweiterte assoziative Feld von Bild 6.16.

Unter Verwendung eines Zwischenspeichers (dieses Register wird häufig nicht explizit dargestellt!) kann mit der Speicheranordnung von Bild 6.19 ein assoziatives Feld realisiert werden, bei dem man die ausgegebenen Adressen A_i an den Eingang rückkoppelt. Diese Schaltung, die im folgenden „rückgekoppeltes assoziatives Wörterbuch“ oder „FAM“ ($\widehat{=}$ **F**eedback **A**ssociation **M**emory) genannt wird, ist im folgenden Bild gezeigt.

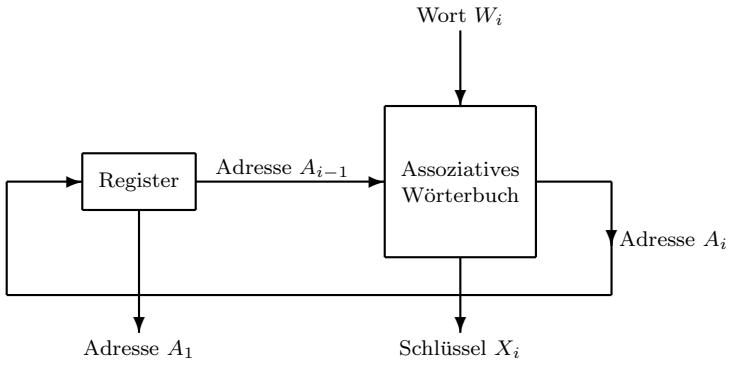

Bild 6.20: Blockschaltbild eines rückgekoppelten assoziativen Wörterbuches (FAM) für das Codieren von Texten.

Wie schon an den Beispielen zu sehen war, muß der Beginn der Codierung besonders behandelt werden, da hierbei das Vorgängerwort fehlt: Nach der Eingabe des ersten Wortes W_1 wird vom assoziativen Wörterbuch die Adresse A_1 abgegeben und in dem Register gespeichert. Die dort ausgegebene Adresse A_1 ist das erste Codewort. Danach kann das zweite Wort W_2 eingegeben werden. Dessen Codewort ist der Schlüssel X_2. Alle folgenden Wörter W_i codiert man auf diese Art, nachdem man die Adresse A_{i-1} des jeweiligen Vorgängers in das Register geschrieben hat. Die Adresse A_1 und die Folge der Schlüssel X_2, X_3, X_4, ... stellen den codierten Text dar und werden dem Decodierer übermittelt.

Für den Decodierer kann man eine entsprechende Schaltung entwerfen, wobei die beiden assoziativen Speicher aus Bild 6.19 lediglich anders verschaltet werden müssen: Bei dem Blockschaltbild, das in Bild 6.21 gezeigt ist, gibt man die Adresse A_i ein und erhält das decodierte Wort W_i. Nach dem Anlegen des Schlüssels X_{i+1} wird dann die Adresse A_{i+1} des nächsten Wortes ausgegeben (vergleiche auch mit Bild 6.18).

Das rückgekoppelte assoziative Wörterbuch (FAM), das man für den Decodiervorgang benötigt, wird unter Verwendung eines Zwischenspeichers und der Speicherstruktur von Bild 6.21 aufgebaut und ist in Bild 6.22 wiedergegeben. Die Adresse A_1 wird im Register gespeichert und an den Eingang A_i angelegt. Nachdem das erste Wort W_1 decodiert und der Schlüssel X_2 eingegeben worden ist, schreibt man die ausgegebene Adresse A_2 in das Register: Dieser Vorgang wiederholt sich periodisch für alle weiteren Schlüssel X_{i+1}.

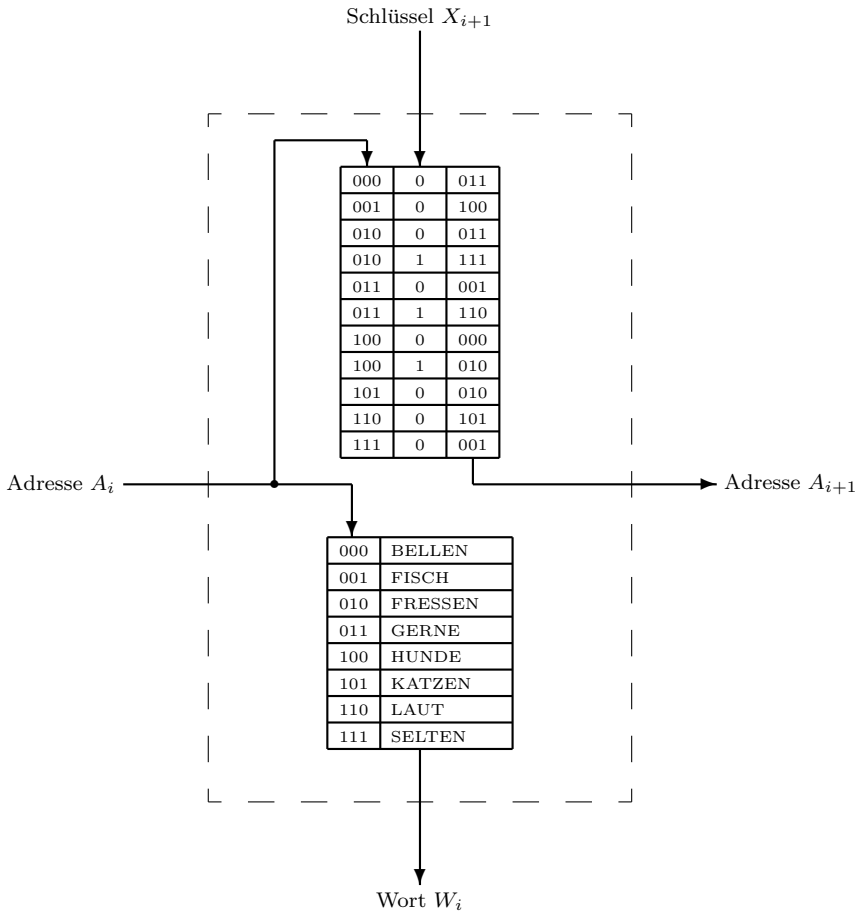

Bild 6.21: Speichermodul für das erweiterte assoziative Feld von Bild 6.18.

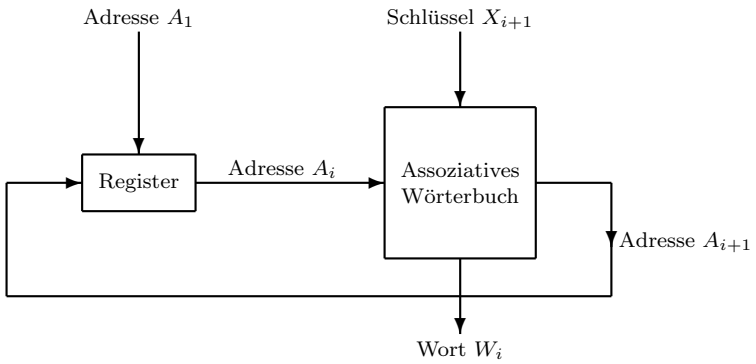

Bild 6.22: Blockschaltbild eines rückgekoppelten assoziativen Wörterbuches (FAM) für das Decodieren von Texten.

6.1 Aufwandsbetrachtungen für ein assoziatives Feld

Der Hardware-Aufwand für das beschriebene rückgekoppelte assoziative Wörterbuch (FAM), bei dem die beiden assoziativen Speicher, die in den Bildern 6.2 und 6.14 gezeigt sind, Verwendung finden, soll nun abgeschätzt werden:

Im Speicher von Bild 6.2 ist der gesamte Wortvorrat enthalten, d. h. man benötigt hier n_0 Speicherzeilen. Für das LIMAS-Korpus ($n_0 = 10^5$) ist die Adresse A_i dann $\lceil \mathrm{ld}\, 10^5 \rceil$ bit = 17 bit breit.* Wenn das Wort W_i mit ASCII-Zeichen (siehe Kapitel 4.1) dargestellt wird, ist bei einer maximalen Wortlänge von 37 Buchstaben eine Speicherspalte der Breite $37 \cdot 7$ bit = 259 bit erforderlich. Insgesamt muß dieser Speicher demnach eine Kapazität von $10^5 \cdot (17 + 259)$ bit = 3,45 Megabyte besitzen.

Der assoziative Speicher von Bild 6.14 enthält die Adressen aller Wortpaare, d. h. es sind n_{paar} Zeilen vorhanden, wobei für $n_0 = 10^5$ (LIMAS-Korpus) eine Zeile maximal $3 \cdot 17$ bit = 51 bit breit ist, da der Schlüssel X_{i+1} höchstens so viele Bitstellen wie eine Adresse besitzen kann. Nach Gleichung (3.28) beträgt die Anzahl der Wortpaare $n_{\mathrm{paar}} \approx 4{,}5 \cdot 10^5$: Die Kapazität dieses Speichers liegt deshalb maximal in der Größenordnung von $4{,}5 \cdot 10^5 \cdot 51$ bit $\approx$ 2,87 Megabyte.

Wegen dieser relativ großen Speicherkapazitäten kann das oben erläuterte rückgekoppelte assoziative Wörterbuch (FAM) heutzutage noch nicht aus assoziativen Speichern aufgebaut werden. Die Simulation dieser Speicherstruktur mit Hilfe der Streuspeicherung (siehe Kapitel 5.1.1) ist jedoch möglich und wurde erstmals von R. Burkard untersucht [109]: Bei der assoziativen Codierung von Texten mit diesem Speicher konnte eine mittlere Codelänge von 0,82 bit/Buchstabe erreicht werden, wobei der Wortvorrat aus $n_0 \approx 10^4$ Wörtern bestand. Damit scheint der Shannon'sche Grenzwert von 1 bit/Buchstabe schon etwas unterboten. Allerdings wurde bei dieser Untersuchung nur ein relativ kurzer Text verwendet. Die mittlere Codelänge bei größeren Wortvorräten läßt sich anhand des Verzweigungsgrades (siehe Bild 3.10 auf Seite 17) wie folgt abschätzen (vergleiche auch mit [82]).

6.2 Die Berechnung der mittleren Codelänge

Mit der Speicheranordnung von Bild 6.21 formt man den Schlüssel X_{i+1} in die Adresse A_{i+1} um. Bei dieser „Codewandlung" hat der Schlüssel X_{i+1} für alle Wörter die gleiche Bitbreite, die der mittleren Länge $\overline{C}_{\mathrm{ass}}$ der Assoziativcodes entspricht, da man die Adresse A_1 des ersten Wortes bei langen Texten vernachlässigen kann. Die maximale Anzahl $v(1)$ der Nachfolgerwörter bestimmt die mittlere Codelänge

$$\overline{C}_{\mathrm{ass}} = \left\lceil \mathrm{ld}\, v(1) \right\rceil \qquad \text{(in bit/Wort)}. \tag{6.3}$$

* Mit den unten offenen „Gaußklammern" wird bekanntlich das Aufrunden auf die nächste ganze Zahl zum Ausdruck gebracht, wie z. B. bei $\lceil \pi \rceil = 4$.

Mit Gleichung (3.23), $n_0 = 10^5$, $\psi = 0{,}8$ und der mittleren Wortlänge des LIMAS-Korpus von $\overline{\ell} = 5{,}7$ Buchstaben/Wort berechnet man unter Berücksichtigung der auf die Wörter folgenden Leerzeichen

$$\overline{C}_{\text{ass}} = \frac{\left\lceil \operatorname{ld} n_0^{\psi} \right\rceil}{\overline{\ell} + 1} = \frac{\left\lceil \psi \operatorname{ld} n_0 \right\rceil}{\overline{\ell} + 1} \approx 2{,}09 \text{ bit/Buchstabe.} \tag{6.4}$$

Dieser Wert liegt in der Größenordnung der Optimalcodes auf Wortebene (siehe Kapitel 5.1.4). Bessere Ergebnisse sind zu erzielen, wenn man die Schlüsselbreite eines Wortes seinem Verzweigungsgrad anpaßt: Der Code $C(n)$ des Wortes, das auf das n-te Wort folgt, hat dann die Länge

$$C(n) = \left\lceil \operatorname{ld} v(n) \right\rceil \qquad \text{(in bit/Wort).} \tag{6.5}$$

Bei der oben erwähnten Simulation [109] wurde dieses Verfahren realisiert. W. Hilberg entwarf unabhängig davon eine ähnliche Speicherkonfiguration, bei der die Anzahl der Nachfolger nicht kontinuierlich, sondern in Zweierpotenzen abgestuft ist [55]. Auf diese Schaltung soll hier jedoch nicht näher eingegangen werden.

Die mittlere Länge der Codes $C(n)$, die ebenfalls mit $\overline{C}_{\text{ass}}$ bezeichnet werden soll, berechnet man unter Berücksichtigung der Gleichungen (4.3), (5.7) und (6.5) zu

$$\overline{C}_{\text{ass}} = \sum_{n=1}^{n_0} \frac{n_0}{s \cdot n} \cdot \left\lceil \operatorname{ld} v(n) \right\rceil \qquad \text{(in bit/Wort).} \tag{6.6}$$

Die darin auftretende Gaußklammer-Funktion $g(u) = \lceil u \rceil$, wobei $u = \operatorname{ld} v(n)$, kann man durch $g^*(u) = u + 0{,}5$ ersetzen, da viele verschiedene Funktionswerte berechnet werden, die in erster Näherung gleichmäßig verteilt sind: Wie im folgenden Bild zu erkennen ist, heben sich die dadurch verursachten Abweichungen im Mittel gerade auf.

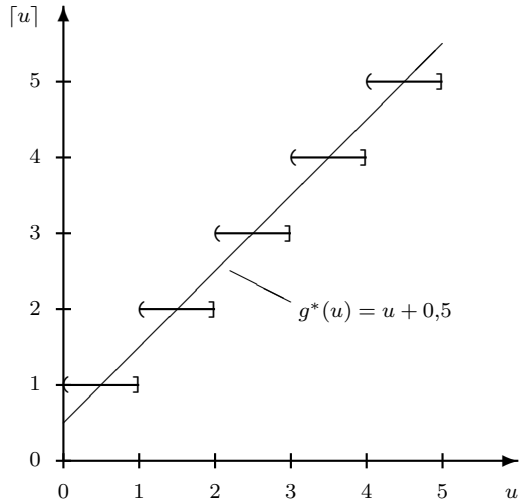

Bild 6.23: Näherung der Gaußklammer-Funktion $g(u) = \lceil u \rceil$ durch $g^*(u) = u + 0{,}5$.

Mit dieser Approximation ermittelt man nach Einsetzen von Gleichung (3.23) in Gleichung (6.6)

$$\begin{aligned}\overline{C}_{\text{ass}} &\approx \sum_{n=1}^{n_0} \frac{n_0}{s \cdot n} \cdot \left[\text{ld}\left(\frac{n_0}{n}\right)^{\psi} + 0{,}5 \right] \\ &= \left[\frac{\psi \cdot \text{ld}\, n_0 + 0{,}5}{s} \cdot \sum_{n=1}^{n_0} \frac{n_0}{n} - \frac{\psi \cdot n_0}{s} \cdot \sum_{n=1}^{n_0} \frac{\text{ld}\, n}{n} \right] \qquad \text{(in bit/Wort).} \end{aligned} \tag{6.7}$$

Daraus wird mit den Gleichungen (3.4), (3.6) und (5.9)

$$\begin{aligned}\overline{C}_{\text{ass}} &\approx \left[\psi \cdot \text{ld}\, n_0 + 0{,}5 - \frac{\psi \cdot \text{ld}\, n_0}{2} \right] \\ &= \frac{1 + \psi \cdot \text{ld}\, n_0}{2} \qquad \text{(in bit/Wort)} \qquad \text{für } n_0 \gg 1. \end{aligned} \tag{6.8}$$

Mit den Werten des LIMAS-Korpus ($n_0 = 10^5$ und $\psi = 0{,}8$) berechnet man damit die mittlere Länge der Assoziativcodes zu $\overline{C}_{\text{ass}} \approx 7{,}14$ bit/Wort. Bezieht man mit Hilfe der mittleren Wortlänge $\overline{\ell} = 5{,}7$ Buchstaben/Wort unter Berücksichtigung der auf die Wörter folgenden Leerzeichen diese Codelänge auf Buchstaben, so ergibt sich

$$\overline{C}_{\text{ass}} \approx \frac{7{,}14}{6{,}7} \text{ bit/Buchstabe} \approx 1{,}07 \text{ bit/Buchstabe} \qquad \text{für } n_0 = 10^5. \tag{6.9}$$

Dieser Wert stimmt nahezu mit dem Shannon'schen Grenzwert [11] überein und entspricht etwa der Hälfte der mit dem ungünstigeren Ansatz in Gleichung (6.4) ermittelten Codelänge.

Die in [109] anhand eines Ausschnittes aus dem LIMAS-Korpus gemessene mittlere Codelänge (siehe oben) und die mit Gleichung (6.8) für $n_0 = 10^4$, $\psi = 0{,}8$ und $\overline{\ell} = 5{,}7$ Buchstaben/Wort berechnete $\overline{C}_{\text{ass}} \approx 0{,}87$ bit/Buchstabe sind praktisch identisch. Zwar konnten mit der *Optimal*codierung der Nachfolgerwörter, die in Kapitel 5.3 beschrieben wurde, noch kürzere Codelängen erreicht werden [99], jedoch steht die dabei erzielte Redundanzreduktion in keinem Verhältnis zu dem dafür notwendigen Aufwand. Das Codierungssystem, das man mit dem oben erläuterten rückgekoppelten assoziativen Wörterbuch aufbaut, stellt deshalb einen vernünftigen Kompromiß zwischen Hardware-Aufwand und mittlerer Codelänge dar.

Im nächsten Kapitel wird beschrieben, wie man mit einem hierarchischen Aufbau, dessen Grundelement ein rückgekoppeltes assoziatives Wörterbuch ist, mittlere Codelängen erreicht, die um Größenordnungen kleiner sind als der „Shannon'sche Grenzwert". Dabei wird für die Berechnung der mittleren Codelängen stets vorausgesetzt, daß der Inhalt der FAMs nicht übertragen werden muß, sondern daß er ein fester Bestandteil des Decodierers ist.

7 Der semantische Speicher

Bei der Codierung eines Textes mit Hilfe des assoziativen Speichers von Bild 6.2 beträgt die mittlere Länge $\overline{C}_{\text{bin}}$ der Binärcodes A_i

$$\begin{aligned} \overline{C}_{\text{bin}} &= \lceil \operatorname{ld} n_0 \rceil && \text{(in bit/Wort)} \\ &= \frac{\lceil \operatorname{ld} n_0 \rceil}{\overline{\ell} + 1} && \text{(in bit/Buchstabe).} \end{aligned} \tag{7.1}$$

Mit n_0 ist dabei die Anzahl der verschiedenen Wörter und mit $\overline{\ell}$ die mittlere Wortlänge bezeichnet. Nach Gleichung (6.8) berechnet man die mittlere Codelänge $\overline{C}_{\text{ass}}$, die bei der Codierung mit dem assoziativen Feld aus Bild 6.16 erreicht wird. Die Abschätzung des Quotienten dieser beiden Codelängen ergibt

$$\frac{\overline{C}_{\text{ass}}}{\overline{C}_{\text{bin}}} \approx \frac{1 + \psi \cdot \operatorname{ld} n_0}{2 \cdot \lceil \operatorname{ld} n_0 \rceil} < \frac{1}{2} \qquad \text{für } \psi = 0{,}8 \text{ und } n_0 \gg 1. \tag{7.2}$$

Demnach halbiert sich die mittlere Codelänge in erster Näherung, wenn man zur Codierung eines Textes ein assoziatives Feld verwendet. Es müßte möglich sein, diesen Code mit Hilfe eines weiteren assoziativen Feldes zu verkürzen, wobei man den Code eines Codes, d. h. einen „Metacode“ erhält. Basierend auf diesem Gedanken entwarf W. Hilberg folgende Hierarchie aus assoziativen Feldern [45, 56]:

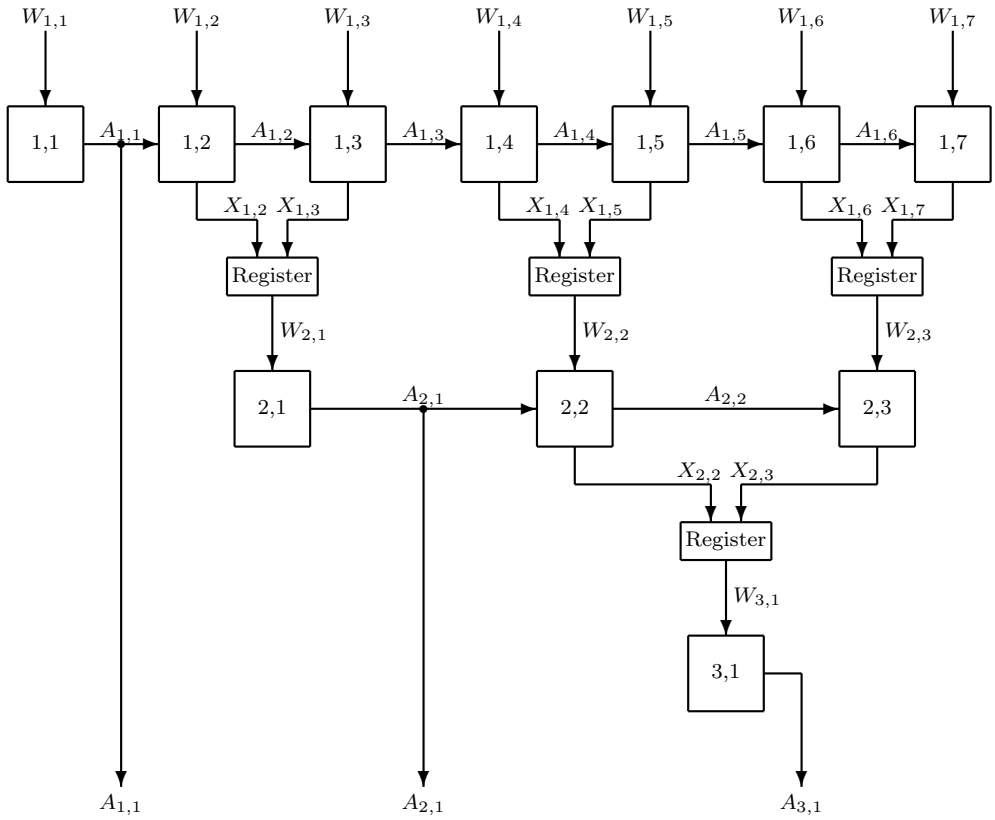

Bild 7.1: Eine Hierarchie aus assoziativen Feldern für das Codieren von Texten.

Zum besseren Verständnis ist die Speicherhierarchie in Bild 7.1 nicht aus rückgekoppelten Speichern (FAMs) aufgebaut, sondern aus kaskadierten assoziativen Speichermodulen von Bild 6.19, die hier mit „j, i“ gekennzeichnet sind: Die Module mit den Nummern „$1, i$“ gehören zur *ersten* Hierarchie-Ebene und entsprechen dem assoziativen Feld von Bild 6.16. Wie in Kapitel 6 beschrieben, gibt man die Wörter, die hier mit $W_{1,i}$ bezeichnet sind, in dieses Feld ein und erhält die Adresse $A_{1,1}$ des ersten Wortes sowie die Folge der externen Schlüssel $X_{1,2}$, $X_{1,3}$, $X_{1,4}$, ..., die nach Gleichung (7.2) im Mittel etwa halb so lang sind wie die Adressen $A_{1,i}$. Wenn man jeweils zwei Schlüssel $X_{1,2\cdot i}$ und $X_{1,2\cdot i+1}$ zu dem Codewort $W_{2,i}$ zusammenfaßt, ergibt sich ein „Metatext“, der aus den „Metaworten“ $W_{2,1}$, $W_{2,2}$, $W_{2,3}$, ... besteht. Dieser Text, der etwa halb so viele Wörter enthält wie der Originaltext, wird mit Hilfe eines zweiten assoziativen Feldes, das aus den Modulen mit den Nummern „$2, i$“ aufgebaut ist, in die Adresse $A_{2,1}$ und die Schlüsselfolge $X_{2,2}$, $X_{2,3}$, ... umgesetzt. Mit jeder weiteren Ebene dieser Speicherhierarchie wird die Anzahl der Metaworte ungefähr halbiert, bis schließlich nur noch die Adressen $A_{1,1}$, $A_{2,1}$, $A_{3,1}$, ... übrigbleiben, die dem codierten Text entsprechen. Zur Decodierung dieser Codes verwendet man eine zum Bild 7.1 komplementäre Speicherorganisation (siehe auch Bild 6.21):

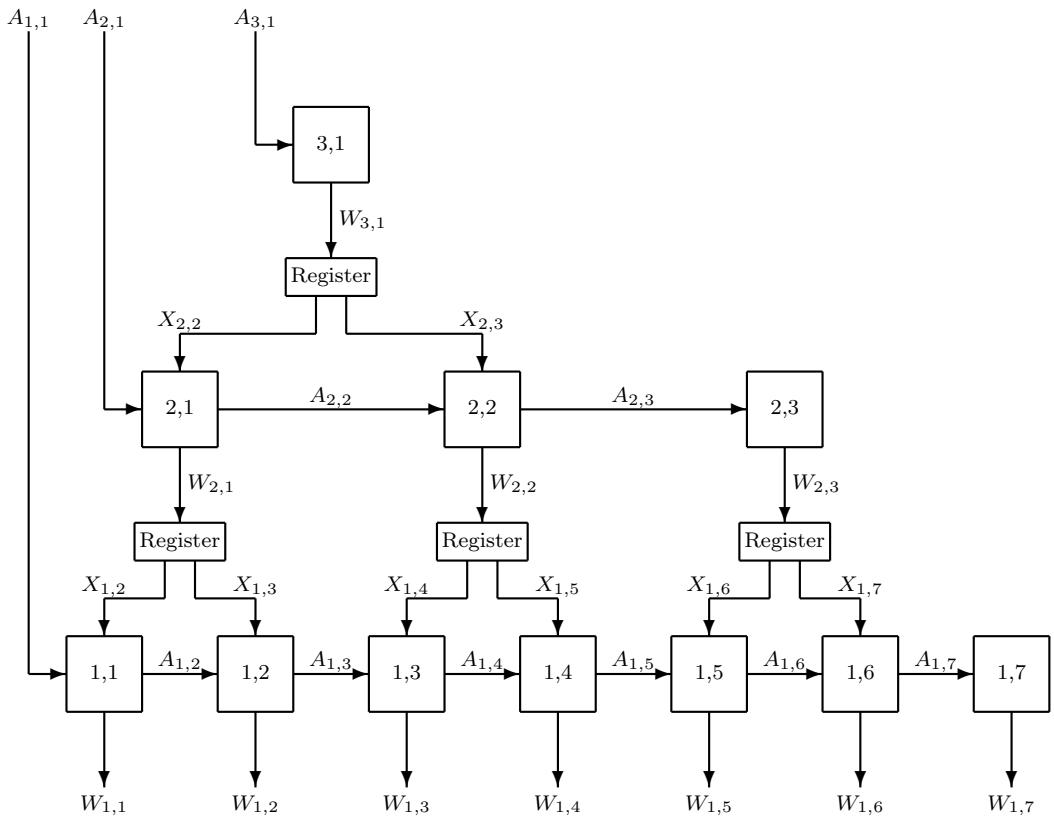

Bild 7.2: Eine Hierarchie aus assoziativen Feldern für das Decodieren von Texten.

Mit der Eingabe der Adresse $A_{1,1}$ in das Modul mit der Nummer „$1, 1$“ erhält man das Wort $W_{1,1}$. Danach kann die Adresse $A_{2,1}$ angelegt werden: Das ausgegebene Metawort $W_{2,1}$ zerlegt man in die Schlüssel $X_{1,2}$ und $X_{1,3}$, mit denen die Wörter $W_{1,2}$

und $W_{1,3}$ decodiert werden. Schließlich gibt man die Adresse $A_{3,1}$ ein, mit der das „Metametawort“ $W_{3,1}$ aufgerufen wird. Mit den daraus gewonnenen Schlüsseln $X_{2,2}$ und $X_{2,3}$ decodiert man die Metaworte $W_{2,2}$ und $W_{2,3}$, die in die Schlüssel $X_{1,4}$ bis $X_{1,7}$ zerlegt werden: Diese Schlüssel entsprechen den Wörtern $W_{1,4}$ bis $W_{1,7}$.

Selbstverständlich kann man die assoziativen Felder in den oben gezeigten Speicherhierarchien auch mit den rückgekoppelten assoziativen Wörterbüchern (FAMs) realisieren, die in den Bildern 6.20 und 6.22 wiedergegeben sind. Der Textcodierer von Bild 7.1 wird dann nach dem folgenden Prinzipschaltbild aufgebaut:

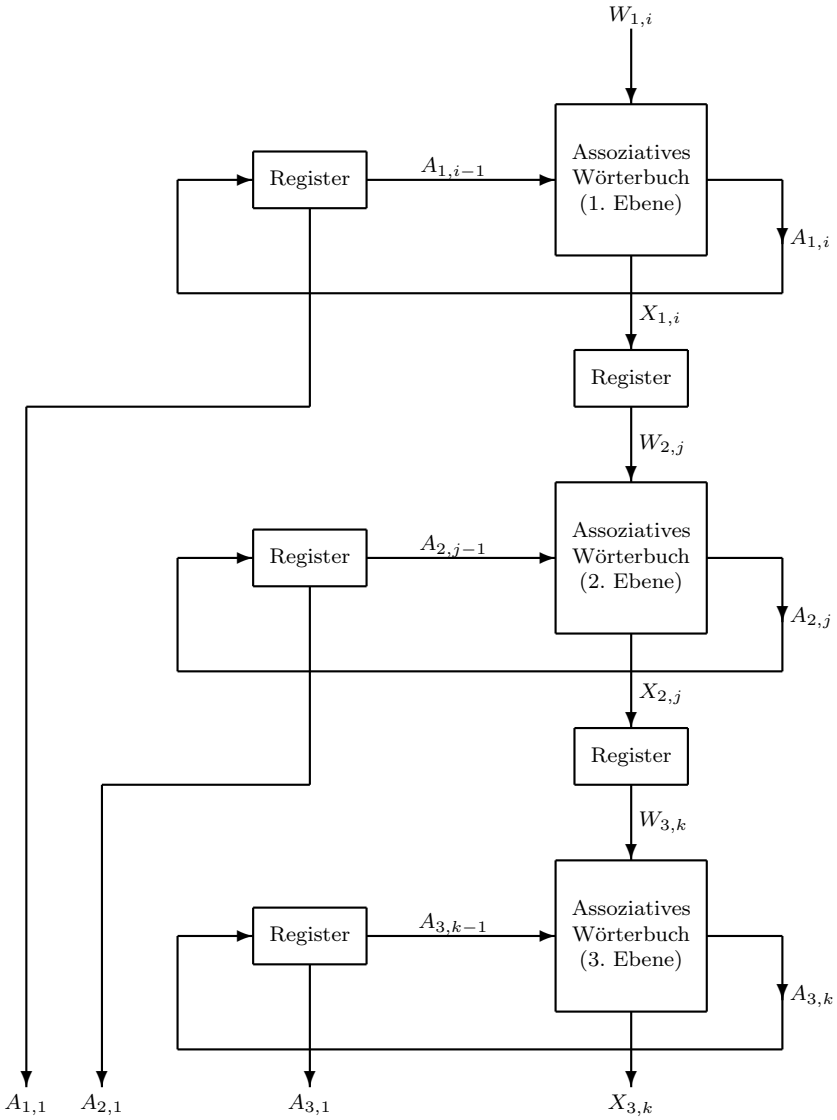

Bild 7.3: Blockschaltbild einer Hierarchie aus rückgekoppelten assoziativen Wörterbüchern (FAMs) für das Codieren von Texten (semantischer Speicher).

In den Eingang des rückgekoppelten assoziativen Wörterbuches der ersten Ebene gibt man die Wörter $W_{1,i}$ ein, die in die Adresse $A_{1,1}$ und die externen Schlüssel $X_{1,i}$ umgesetzt werden. In dem Register, das sich zwischen der ersten und der zweiten

Ebene befindet, werden diese Schlüssel jeweils paarweise zusammengesetzt und als Metaworte $W_{2,j}$ dem zweiten rückgekoppelten assoziativen Wörterbuch zugeführt. Entsprechend verfährt man auf allen weiteren Ebenen, bis schließlich bei einer Hierarchie, die aus insgesamt μ Ebenen besteht, die Adressen $A_{1,1}$, $A_{2,1}$, $A_{3,1}$, ..., $A_{\mu,1}$ und die externen Schlüssel $X_{\mu,2}$, $X_{\mu,3}$, $X_{\mu,4}$, ... übrigbleiben.

Der Decodierer wird mit rückgekoppelten assoziativen Wörterbüchern (FAMs) von Bild 6.22 aufgebaut, wobei man ein zu Bild 7.3 inverses Blockschaltbild verwendet:

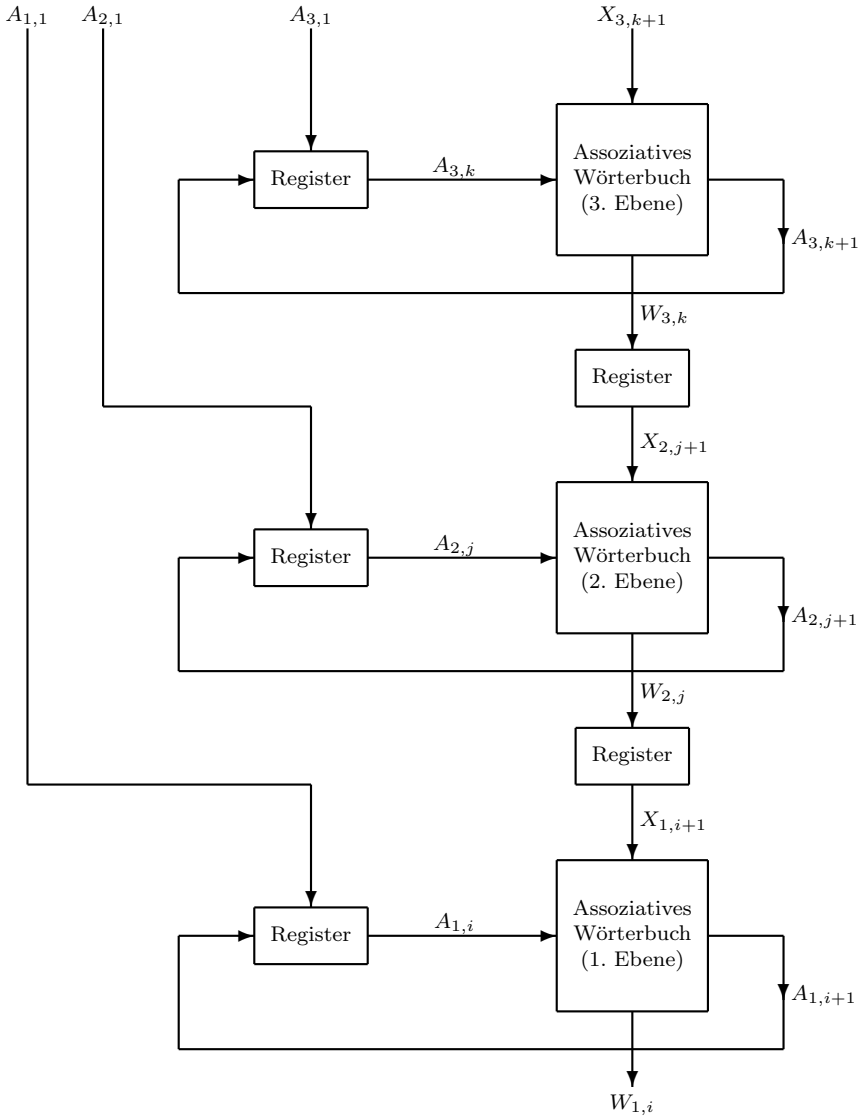

Bild 7.4: Blockschaltbild einer Hierarchie aus rückgekoppelten assoziativen Wörterbüchern (FAMs) für das Decodieren von Texten (semantischer Speicher).

Mit der Adresse $A_{1,1}$ initialisiert man das rückgekoppelte assoziative Wörterbuch der ersten Ebene: Das erste Wort $W_{1,1}$ des Textes wird ausgegeben. Entsprechend erhält man nach dem Anlegen der Adresse $A_{2,1}$ das Metawort $W_{2,1}$, das in dem Register

zwischen der ersten und der zweiten Ebene zwischengespeichert und in die externen Schlüssel $X_{1,2}$ und $X_{1,3}$ zerlegt wird: Das FAM der ersten Ebene liest jeweils nur eine Hälfte $X_{1,i+1}$ des Registerinhaltes aus und decodiert damit das nächste Wort $W_{1,i+1}$. Analog dazu gibt man alle weiteren Adressen $A_{3,1}$, ..., $A_{\mu,1}$ und externen Schlüssel $X_{\mu,2}$, $X_{\mu,3}$, $X_{\mu,4}$, ... in die rückgekoppelten assoziativen Wörterbücher der höheren Ebenen ein, bis schließlich der gesamte Text decodiert ist.

Die oben erläuterten Speicherhierarchien eignen sich zum Aufbau eines Codierungssystems für Texte, bei dem neben syntaktischen auch semantische Abhängigkeiten berücksichtigt werden, wie im folgenden gezeigt wird:

Die Adresse $A_{1,1}$ ist der Code des ersten Wortes $W_{1,1}$, das ein beliebiges Wort aus dem gesamten Wortvorrat sein kann; den beiden folgenden Wörtern $W_{1,2}$ und $W_{1,3}$ entspricht die Adresse $A_{2,1}$. Diese Wörter müssen zu dem ersten Wort grammatikalisch und inhaltlich passen und werden daher aus einer *Teilmenge* des Wortvorrates ausgewählt. Daraus resultiert eine kürzere mittlere Codelänge: *Eine* Adresse entspricht *zwei* Wörtern. Bei den folgenden vier Wörtern sind die jeweiligen Teilmengen noch kleiner, da diese Wörter die vorangegangenen, die den Anfang eines Satzes bilden, „sinnvoll“ ergänzen müssen: Jetzt codiert man mit einem einzigen Codewort (der Adresse $A_{3,1}$) schon vier Wörter. Allgemein gilt demnach, daß die Adresse $A_{j,1}$ insgesamt $2^{(j-1)}$ Wörtern entspricht.

Man stelle sich nun folgendes Gedankenexperiment vor, bei dem ein gegebener Text unter Verwendung einer Speicherhierarchie mit beispielsweise zehn Ebenen codiert werden soll: Mit den ersten neun Adressen $A_{1,1}$, ..., $A_{9,1}$ codiert man die ersten $1 + 2 + 4 + \cdots + 256 = 511$ Wörter des Textes. Die Teilmenge des Wortvorrates, aus der das 512te Wort stammt, ist relativ klein, da dieses Wort syntaktisch und semantisch zu den vorangegangenen passen muß. Aus diesem Grund ist die mittlere Codelänge, bezogen auf ein Wort, nun sehr klein geworden: Mit der Adresse $A_{10,1}$ codiert man insgesamt 512 Wörter, d. h. etwa eine ganze Buchseite!

Die obige Speicherorganisation wird wegen der Berücksichtigung der semantischen Abhängigkeiten bei der Codierung von Texten als „semantischer Speicher“ bezeichnet [45]. Im folgenden Abschnitt soll die mittlere Codelänge, die man mit einer „semantischen“ Codierung erreichen kann, abgeschätzt werden (vergleiche auch mit [81]).

7.1 Die Berechnung der mittleren Codelänge

Es sei ein Text gegeben, der aus insgesamt s Wörtern besteht und unter Verwendung eines semantischen Speichers mit μ Ebenen codiert werden soll:

Wie man leicht anhand von Bild 7.1 nachprüfen kann, besteht der Code dieses Textes aus μ Adressen ($A_{1,1}$, $A_{2,1}$, $A_{3,1}$, ..., $A_{\mu,1}$) und $2 \cdot ((s+1)/2^{\mu} - 1)$ externen Schlüsseln $X_{\mu,k}$, wobei $(s+1)$ durch 2^{μ} teilbar sein muß. Ist diese Bedingung

nicht erfüllt, so kann man den gegebenen Term zumindest als Näherungswert in die folgenden Gleichungen einsetzen.

Nach Gleichung (7.2) ist ein externer Schlüssel $X_{1,i}$ im Mittel etwa halb so lang wie eine Adresse $A_{1,i}$: Geht man von dem Idealfall aus, daß alle externen Schlüssel eine *konstante* Bitbreite haben, dann besteht ein Metawort $W_{2,j}$ aus genausovielen Bitstellen wie eine Adresse $A_{1,i}$ (siehe Bild 7.1), d. h. zur Adressierung eines Metawortes benötigt man eine Adresse $A_{2,j}$, die ebenfalls so lang ist wie eine Adresse $A_{1,i}$. Unter diesen Voraussetzungen haben alle Adressen $A_{j,i}$ in erster Näherung die gleiche Bitbreite: Die mittlere Länge $\overline{C}_{\text{sem}}$ der „semantischen“ Codes bestimmt man damit unter Berücksichtigung von Gleichung (7.1) näherungsweise zu

$$\begin{aligned} \overline{C}_{\text{sem}} &\approx \frac{\mu + \frac{1}{2} \cdot 2 \cdot \Big((s+1)/2^{\mu} - 1\Big)}{s} \cdot \overline{C}_{\text{bin}} \\ &= \frac{\mu + (s+1)/2^{\mu} - 1}{s} \cdot \frac{\lceil \operatorname{ld} n_0 \rceil}{\overline{\ell} + 1} \qquad \text{(in bit/Buchstabe).} \end{aligned} \tag{7.3}$$

Dabei ist die Anzahl der externen Schlüssel $X_{\mu,k}$ stets positiv:

$$2 \cdot \left(\frac{s+1}{2^{\mu}} - 1 \right) \geq 0 \qquad \Longrightarrow \qquad \mu \leq \operatorname{ld}(s+1). \tag{7.4}$$

Wegen dieser Bedingung kann man bei der Codierung eines Textes der Länge s maximal $\operatorname{ld}(s+1)$ Ebenen eines semantischen Speichers verwenden. Nach dem Einsetzen dieses Extremwertes in Gleichung (7.3) erhält man eine Abschätzung für die erreichbare *minimale* mittlere Codelänge

$$\overline{C}_{\text{sem}} \approx \frac{\operatorname{ld}(s+1)}{s} \cdot \overline{C}_{\text{bin}} \approx \frac{\operatorname{ld} s}{s} \cdot \frac{\lceil \operatorname{ld} n_0 \rceil}{\overline{\ell} + 1} \qquad \text{(in bit/Buchstabe)} \qquad \text{für } s \gg 1. \tag{7.5}$$

Bei dieser Gleichung fällt auf, daß die minimale mittlere Codelänge $\overline{C}_{\text{sem}}$ aus der mit dem Faktor $(\operatorname{ld} s)/s$ multiplizierten Länge $\overline{C}_{\text{bin}}$ der Adressen $A_{1,i}$ hervorgeht, d. h. mit wachsender Textlänge wird $\overline{C}_{\text{sem}}$ beliebig klein. (Selbstverständlich erreicht $\overline{C}_{\text{sem}}$ niemals den Wert Null!)

Die Gleichungen (7.3) und (7.5) sollen nun an zwei Diagrammen veranschaulicht werden: Für $s = 10^6$, $n_0 = 10^5$ und $\overline{\ell} = 5{,}7$ Buchstaben/Wort, d. h. für die Werte des LIMAS-Korpus [2], ist in Bild 7.5 die mittlere Codelänge $\overline{C}_{\text{sem}}$ nach Gleichung (7.3) über der Ebenenanzahl μ aufgetragen.

Anhand von Bild 7.5 ist deutlich zu erkennen, wie für größer werdendes μ die mittlere Codelänge $\overline{C}_{\text{sem}}$ bei den ersten Ebenen des semantischen Speichers jeweils um den Faktor 2 abnimmt, bis schließlich ein Minimalwert erreicht wird: Für kleine Werte von μ überwiegt die Anzahl der externen Schlüssel, auf den höheren Ebenen dagegen diejenige der Initialisierungsadressen (siehe auch Bild 7.1).

In Bild 7.6 sind die nach Gleichung (7.5) berechneten Minimalwerte der mittleren Codelänge $\overline{C}_{\text{sem}}$ für verschiedene Textlängen s wiedergegeben, wobei ein semantischer Speicher mit 20 Ebenen vorausgesetzt wurde, der für eine Codierung des gesamten LIMAS-Korpus ausgelegt ist.

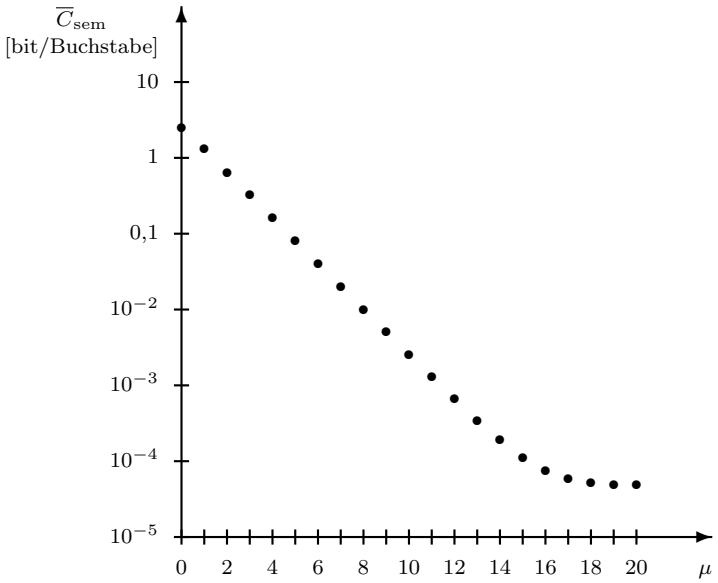

Bild 7.5: Die mit Gleichung (7.3) berechnete mittlere Codelänge $\overline{C}_{\text{sem}}$ in Abhängigkeit der Anzahl μ der Ebenen des semantischen Speichers für $s = 10^6$, $n_0 = 10^5$ und $\overline{\ell} = 5{,}7$ Buchstaben/Wort (Werte des LIMAS-Korpus).

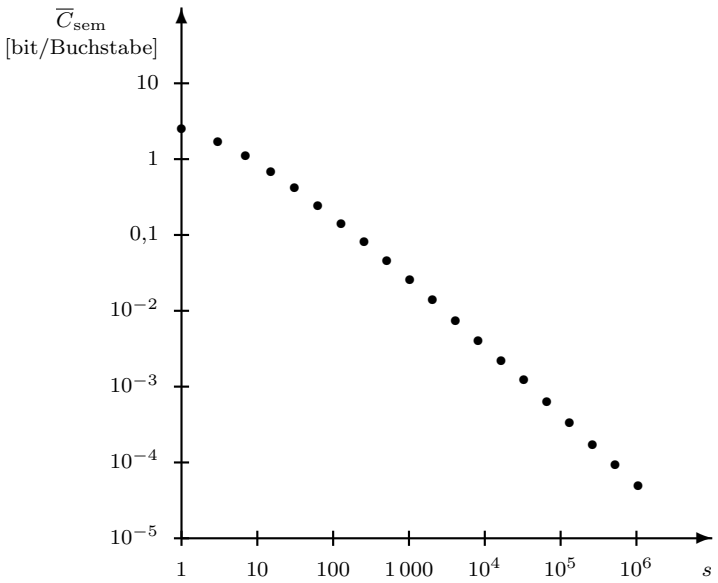

Bild 7.6: Die mit Gleichung (7.5) berechnete minimale mittlere Codelänge $\overline{C}_{\text{sem}}$ in Abhängigkeit der Textlänge s für $n_0 = 10^5$ und $\overline{\ell} = 5{,}7$ Buchstaben/Wort.

Aus Bild 7.6 geht hervor, daß für beliebig lange Texte im Prinzip beliebig kleine mittlere Codelängen $\overline{C}_{\text{sem}}$ erreichbar sind, die weit unterhalb des Shannon'schen Grenzwertes von 1 bit/Buchstabe liegen (siehe auch [57]). Daraus darf allerdings nicht der Trugschluß gezogen werden, daß ein Text umso weniger Information enthält, je länger er ist: Gegen Null strebt nur die *mittlere* Codelänge. Die minimale *absolute* Codelänge C_{sem} eines Textes, der mit einem semantischen Speicher codiert wurde, berechnet man mit Gleichung (7.5) zu

$$C_{\text{sem}} \approx \text{ld}(s+1) \cdot \lceil \text{ld}\, n_0 \rceil \approx \text{ld}\, s \cdot \lceil \text{ld}\, n_0 \rceil \qquad \text{(in bit)} \qquad \text{für } s \gg 1. \tag{7.6}$$

Diese Codelänge C_{sem} ist für wachsende Textlängen s streng monoton steigend, wie das folgende Bild zeigt.

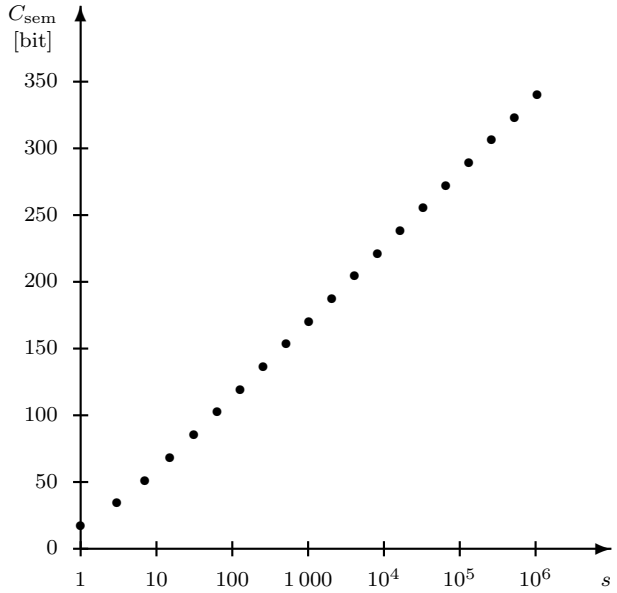

Bild 7.7: Die mit Gleichung (7.6) berechnete minimale absolute Codelänge C_{sem} in Abhängigkeit der Textlänge s für $n_0 = 10^5$.

Den in diesem Bild wiedergegebenen Zusammenhang erklärt man sich anschaulich mit Hilfe von Bild 7.1: Für die Codierung eines Textes der Länge s benötigt man einen semantischen Speicher mit $\mu \approx \text{ld}\, s$ Ebenen. Ein doppelt so langer Text ($2 \cdot s$ Wörter) wird mit $(\mu + 1)$ Ebenen codiert, d. h. bei einer *Verdopplung* der Textlänge vergrößert sich die absolute Codelänge lediglich um *eine* Adresse. Nach der obigen Näherung haben alle Initialisierungsadressen $A_{j,1}$ die gleiche Bitbreite: Für exponentiell anwachsende Textlängen s steigen daher die Codelängen C_{sem} linear an. Die im nächsten Kapitel diskutierten Ergebnisse der Simulation eines semantischen Speichers, mit dem man das gesamte LIMAS-Korpus codieren kann, zeigen, daß die obigen Abschätzungen der Codelängen mit den gemessenen Werten im großen und ganzen übereinstimmen.

7.2 Weiterführende Gedanken zum semantischen Speicher

W. Hilberg, der den oben skizzierten semantischen Speicher entwarf, um Texte redundanzarm codieren zu können, äußerte auch einige Vorschläge für andere Anwendungen dieser neuen Speicherorganisation [58, 59]:

Der Code eines Textes, d. h. die Folge der Adressen $A_{1,1}$, $A_{2,1}$, $A_{3,1}$, ..., $A_{\mu,1}$, ist eine sehr „kompakte“ Darstellung des Klartextes. Man kann die Codierung dieses Textes, die schrittweise von Ebene zu Ebene des semantischen Speichers voranschreitet, auch als eine Art *Abstraktionsprozeß* ansehen, bei dem von dem Klartext, d. h. den Buchstaben, Wörtern etc., abstrahiert wird. Der resultierende Code entspricht dann derjenigen Information, die in dem Text enthalten ist: Man könnte diesen Code auch als *Symbol* für die „Textaussage“ bezeichnen. Wird dieses Symbol in den semantischen Speicher eingegeben, dann erhält man den ursprünglichen Klartext zurück, d. h. in der Speicherhierarchie wird *Sprachwissen* gespeichert. Wenn nun *zwei* semantische Speicher verwendet werden, die das Sprachwissen zweier verschiedener Sprachen enthalten, ist es prinzipiell möglich, Texte von einer Sprache automatisch in eine andere zu übersetzen: Mit der ersten Speicherhierarchie erzeugt man den Code eines Textes, den man mit der zweiten Hierarchie in den Klartext der anderen Sprache umsetzt. Dazu müssen die Codes jeweils zweier Texte, die in den verschiedenen Sprachen dasselbe aussagen, identisch sein.

Man könnte den semantischen Speicher vielleicht auch für andere Aufgaben einsetzen, die bisher mit Hilfe der künstlichen Intelligenz (KI, [30]) gelöst werden sollen. Möglicherweise kann man damit die Organisation des menschlichen Gehirns verstehen [60]: Dieser Ansatz unterscheidet sich auch von demjenigen der neuronalen Netzwerke, da hierbei nicht von der Stufe der kleinsten „Bausteine“ des Gehirns, der Neuronen, ausgegangen wird, sondern von einer wesentlich höheren, abstrakteren Ebene. Zunächst ist dieser neue Gedanke reine Spekulation, da noch keine konkreten Vorschläge für die Realisierung einer Gehirnstruktur mit Hilfe eines semantischen Speichers vorliegen, jedoch scheint dieser Weg zumindest prinzipiell erfolgversprechend zu sein, wie im folgenden gezeigt wird:

Aufbau und Funktion einzelner Neuronen sind schon recht lange bekannt, so daß z. B. K. Küpfmüller und F. Jenik zu Anfang der sechziger Jahre Transistorschaltungen entwerfen konnten, die das elektrische Übertragungsverhalten natürlicher Nervenzellen nachbilden [31]. Auch wurden *qualitative* Verhaltensweisen realer Neuronensysteme (z. B. das Lernverhalten des Menschen) weitgehend erforscht [32]. Dagegen ist die *Verschaltung* der Neuronen nur für abgegrenzte, relativ kleine Nervennetze, wie z. B. den Sehapparat der Stubenfliege, hinreichend geklärt [33]. Anhand dieser Kenntnisse wurden Netzwerke aus nachgebildeten Neuronen aufgebaut, mit denen man primitive Gehirnfunktionen simulieren kann: R. Schneller stellte eine Auswahl derjenigen Literatur zusammen, die die wesentlichen Experimente, die hierzu durchgeführt wurden, beschreibt [110]. Die Neuronennetze, die bei diesen Versuchen simuliert wurden, bestanden in der Regel nur aus wenigen hundert bis tausend Einzelzellen [34, 35], so daß man noch weit davon entfernt ist, reali-

stische Gehirnstrukturen, die beim Menschen aus etwa 10^{10} Neuronen aufgebaut sind, nachbilden zu können. Das folgende Beispiel weist auf die Schwierigkeiten hin, die auftreten, wenn die Architektur eines Gehirns ausschließlich auf der Ebene der Nervenzellen erforscht werden soll: Falls man den inneren Aufbau eines Taschenrechners verstehen wollte, wäre es von vornherein aussichtslos, auf der Ebene der Transistoren das Gesamtsystem überblicken zu wollen; dagegen ist die Verwendung logischer, d. h. abstrakterer Begriffe, wie z. B. „Multiplizier-Einheit“, „Register“ etc. diesbezüglich von Vorteil. Analog dazu sind Ansätze, bei denen Gehirnstrukturen nicht aus Einzelzellen, sondern aus relativ komplexen Funktionseinheiten zusammengesetzt werden, vorzuziehen: Möglicherweise ist das Konzept des semantischen Speichers hierfür geeignet.

In dieser Arbeit sollen jedoch die obigen, rein spekulativen Aspekte bezüglich der Anwendungsbereiche eines semantischen Speichers nicht weiter verfolgt werden. Im Vordergrund steht zunächst die primäre und offensichtliche Anwendungsmöglichkeit der neuen Speicherhierarchie: Die Beschreibung der durchgeführten Simulationen eines semantischen Speichers, der für eine redundanzarme Textcodierung eingesetzt werden kann, erfolgt im nächsten Kapitel.

8 Die durchgeführten Simulationen

Ein semantischer Speicher läßt sich unter Verwendung der in den Bildern 7.3 und 7.4 gezeigten Hierarchie aus rückgekoppelten assoziativen Wörterbüchern (FAMs) realisieren, wobei ein FAM aus einem Register, einem assoziativen Wörterbuch und einem assoziativen Speicher nach Bild 6.14, in dem die Adressen aufeinanderfolgender Wörter sowie deren Codes X_{i+1} gespeichert sind, besteht (siehe die Bilder 6.19 bis 6.22): Das assoziative Wörterbuch von Bild 6.2 kann, wie in Kapitel 5.1.1 beschrieben, mit Hilfe der Streuspeicherung simuliert werden. Dagegen ist die Realisierung des assoziativen Speichers von Bild 6.14 zunächst problematisch, da sich die Bitbreite der Schlüssel X_{i+1}, die möglichst gering sein sollte, nach der maximalen Nachfolgeranzahl $v(1)$ richtet (siehe Gleichung (6.3)). Die Lösung dieser Schwierigkeiten ist z. B. mit folgender Methode möglich, bei der man die Anzahl der Bitstellen der Schlüssel X_{i+1} unabhängig von $v(1)$ auf beliebige Werte einstellen kann [83]:

Der Inhalt des assoziativen Speichers von Bild 6.14 läßt sich in Form eines gerichteten Graphen darstellen, dessen Knoten verschiedene Wörter enthalten und dessen Kanten den Vorgänger-Nachfolger-Beziehungen der jeweiligen Wörter entsprechen. Dieses Netzwerk, das auch „Sprachnetzwerk“ oder „Wortzellen-Netz“ genannt wird (vergleiche [46, 47, 62, 63, 64]), soll anhand von Bild 8.2 und dem Beispieltext aus Bild 8.1 veranschaulicht werden: Jedem Wort dieses Textes ist ein Knoten, d. h. eine „Wortzelle“ zugeordnet, die über gerichtete „Verbindungen“ (Kanten) mit allen Wortzellen verbunden ist, die die Nachfolgerwörter enthalten.

HUNDE JAGEN KATZEN UND KATZEN JAGEN MÄUSE .
JAGEN HUNDE MÄUSE ?
HUNDE BELLEN UND JAGEN KATZEN .
BELLEN KATZEN ?

Bild 8.1: Beispieltext zur Erläuterung des Wortzellen-Netzes.

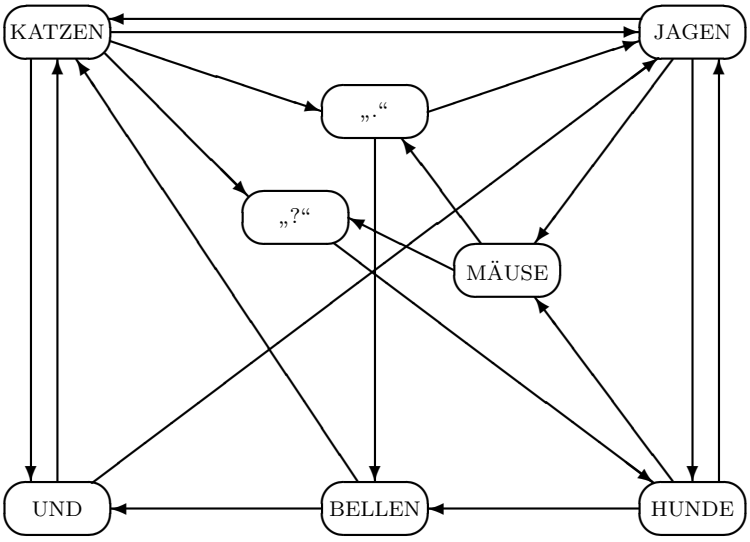

Bild 8.2: Das Wortzellen-Netz zu dem Text aus Bild 8.1.

Eine wichtige Kenngröße eines Wortzellen-Netzes ist die maximale Anzahl $v_{\max}$ der Verbindungen einer Zelle, durch die die Anzahl der binären Entscheidungen bei der Auswahl eines konkreten Nachfolgers festgelegt wird, d. h. die Größe $v_{\max}$ bestimmt die Bitbreite der Schlüssel X_{i+1} (siehe Bild 6.14), die nach den Voraussetzungen von Kapitel 7.1 im folgenden *konstant* sein soll. Zur Bestimmung von $v_{\max}$ wird in jede Wortzelle zusätzlich die Anzahl ihrer Verbindungen zu Nachfolgern aufgenommen: Das folgende Bild zeigt das auf diese Art erweiterte Wortzellen-Netz von Bild 8.2.

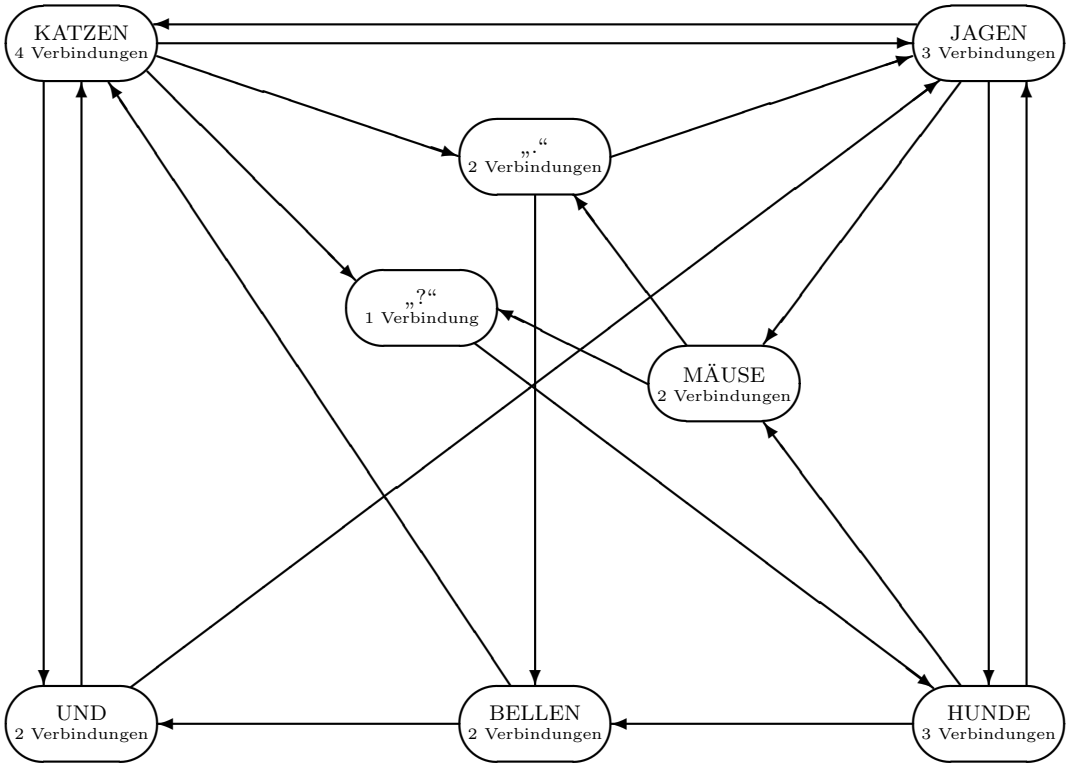

Bild 8.3: Mit der Anzahl der Verbindungen versehenes Wortzellen-Netz von Bild 8.2.

Nach der im Kapitel 3.4 ermittelten Verzweigungsgradverteilung $v(n)$ besitzen die meisten Wörter nur eine geringe Anzahl an Nachfolgern (siehe Bild 3.10), d. h. ein Wortzellen-Netz ist für reale Texte in der Regel schwach vernetzt. Allerdings existieren einige Wörter, die relativ viele Nachfolger haben und somit die Anzahl der Bitstellen der Schlüssel X_{i+1} unnötig vergrößern. Vorteilhaft wäre es, wenn man die maximale Verbindungsanzahl $v_{\max}$ auf einen bestimmten Wert begrenzen könnte, für das obige Netz z. B. auf $v_{\max} = 3$ Verbindungen: Beim Netzaufbau kann wegen dieser Bedingung dann nur eine einzige Verbindung nicht eingetragen werden, die Verbindung von „KATZEN“ nach „?“ (siehe Bild 8.4). Um diese Verbindung aufnehmen zu können, muß zuerst eine bestehende Verbindung gelöscht werden, beispielsweise die Verbindung zu dem *häufigsten* Nachfolger von „KATZEN“: Da in dem obigen Text alle Nachfolger von „KATZEN“ gleichhäufig vorkommen, kann man eine beliebige Verbindung abbauen, z. B. die Verbindung von „KATZEN“ nach „JAGEN“. Anstelle der gelöschten Verbindung erzeugt man eine neue Zelle, die die Wortfolge „KATZEN JAGEN“ enthält (siehe Bild 8.5).

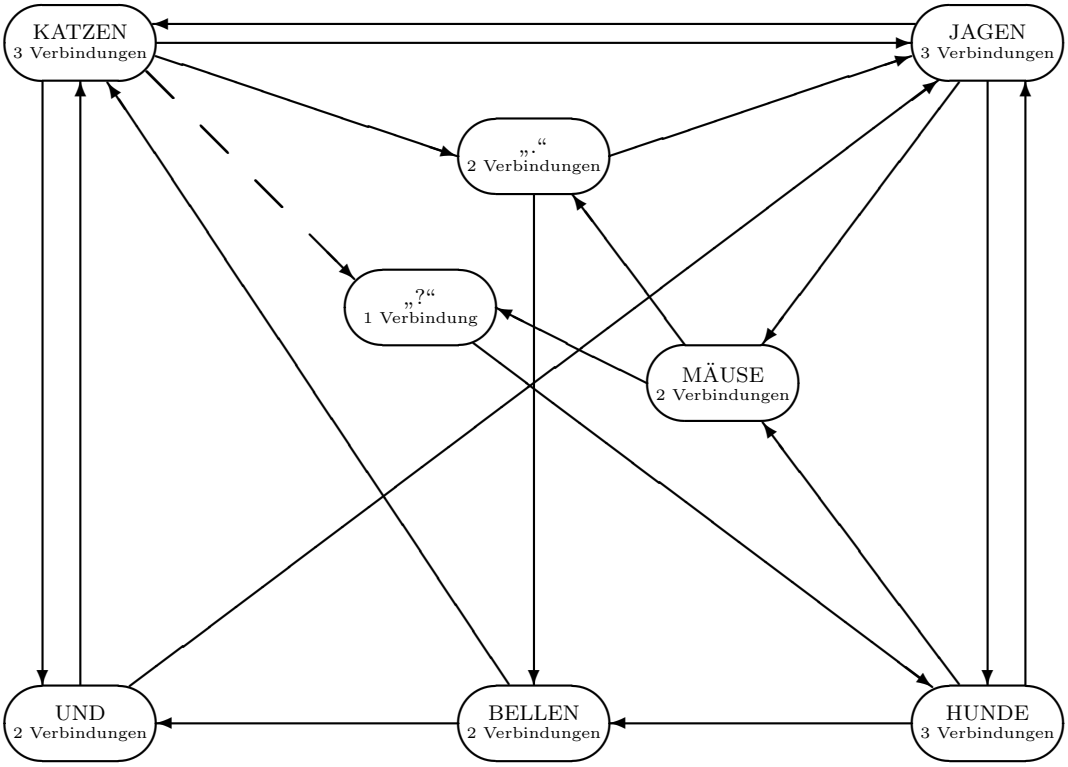

Bild 8.4: „Momentaufnahme“ vom Aufbau eines Wortzellen-Netzes.

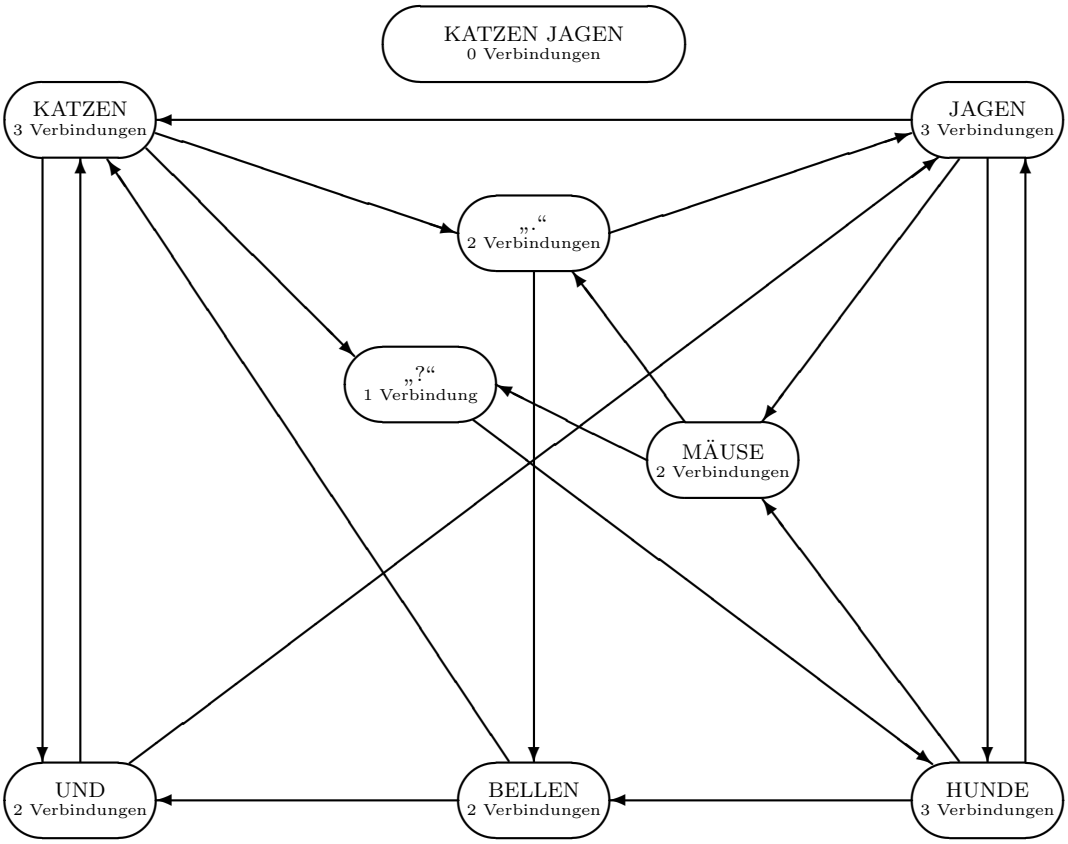

Bild 8.5: „Momentaufnahme“ vom Aufbau eines Wortzellen-Netzes.

Zur Unterscheidung sollen diejenigen Zellen, die Einzelwörter enthalten, von nun an als „Wortzellen“ bezeichnet werden und diejenigen, in denen Wortfolgen stehen, als „Wortfolgenzellen“. Die Wortfolgenzelle „KATZEN JAGEN“ in Bild 8.5 ist nach dem Durchlesen des Beispieltextes von Bild 8.1 isoliert von dem restlichen Netz: Der Lerntext muß ein zweites Mal durchgelesen werden, um das vollständige Netz, das im folgenden Bild gezeigt ist, zu erhalten.

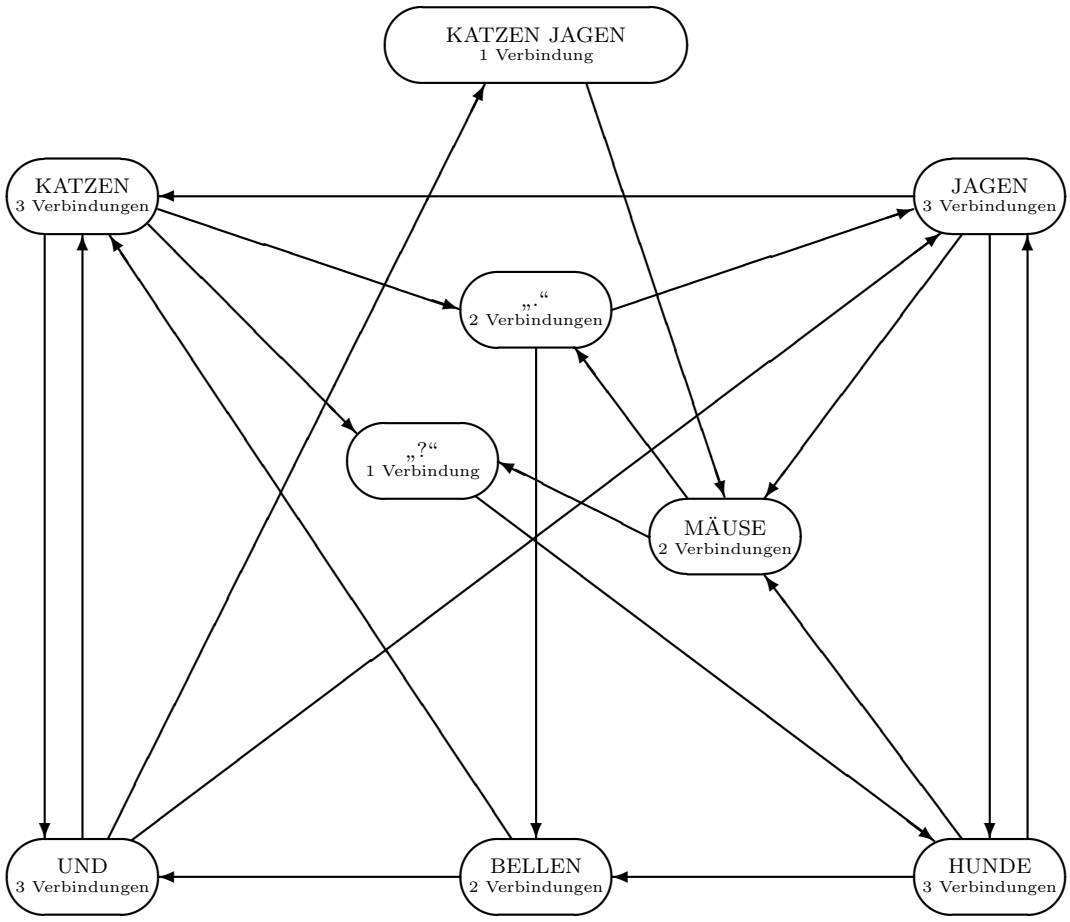

Bild 8.6: Wortzellen-Netz mit maximal drei Verbindungen pro Zelle ($v_{\max} = 3$). Überflüssige Verbindungen wurden nicht gelöscht.

Dieses Wortzellen-Netz enthält zwei überflüssige Verbindungen: Die Verbindungen von „UND“ nach „KATZEN“ und von „JAGEN“ nach „MÄUSE“ stammen aus dem ersten Lese-Durchgang und hätten zusammen mit der Verbindung von „KATZEN“ nach „JAGEN“ gelöscht und durch die Wortfolgenzelle und ihre Verbindungen ersetzt werden müssen. In realen Texten herrschen allerdings nicht so überschaubare Verhältnisse wie in dem gegebenen Beispieltext, so daß es sinnvoll erscheint, beim Überschreiten der maximalen Verbindungsanzahl $v_{\max}$ nur *eine* Verbindung zu löschen und eine isolierte Wortfolgenzelle zu erzeugen. Das Löschen der überflüssigen Verbindungen muß dann nachträglich erledigt werden. Wenn man bei dem obigen Netz alle überflüssigen Verbindungen löscht, dann ergibt sich das in Bild 8.7 gezeigte Wortzellen-Netz.

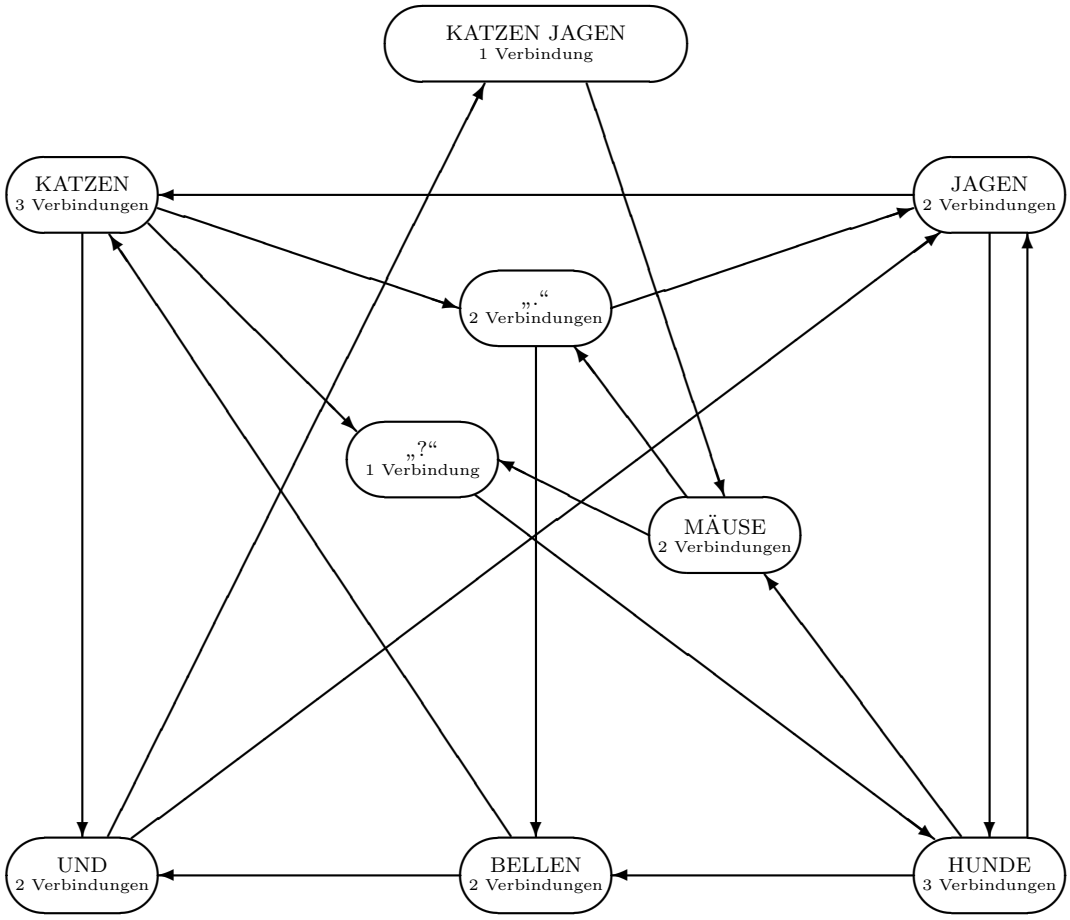

Bild 8.7: Wortzellen-Netz mit maximal drei Verbindungen pro Zelle ($v_{\max} = 3$). Überflüssige Verbindungen wurden gelöscht.

Für die Berechnung der mittleren Codelänge $\overline{C}_{\text{sem}}$ in Kapitel 7.1 war vorausgesetzt worden, daß alle Schlüssel $X_{j,i}$ genau die *halbe* Bitbreite der zugehörigen Adressen $A_{j,i}$ haben, d. h. bei einem Wortzellen-Netz, das aus n_0 Zellen besteht, darf dann jede Zelle maximal $v_{\max}$ Verbindungen besitzen, wobei

$$v_{\max} = n_0^{1/2} = \sqrt{n_0}. \tag{8.1}$$

Das obige Beispielnetz besteht aus 9 Zellen: Nach Gleichung (8.1) sind pro Zelle dann $v_{\max} = \sqrt{9} = 3$ Verbindungen zulässig. Die Wortzellen-Netze der Bilder 8.6 und 8.7 wären demnach für die Simulation eines semantischen Speichers geeignet.

Mit dem Verzweigungsgrad $v(n)$ aus Gleichung (3.23) bestimmt man die Anzahl n derjenigen Wörter, die mehr als $\sqrt{n_0}$ Nachfolger haben:

$$\min\Big(v(n)\Big) = \min\left(\left(\frac{n_0}{n}\right)^{\psi}\right) > \sqrt{n_0}. \tag{8.2}$$

Auflösen dieser Ungleichung nach n und Abrunden auf die nächstkleinere ganze Zahl (oben offene Gaußklammern) ergibt

$$n = \left\lfloor \sqrt{n_0^{(2-1/\psi)}} \right\rfloor. \tag{8.3}$$

Mit den Werten des LIMAS-Korpus ($n_0 = 10^5$ und $\psi = 0{,}8$) berechnet man, daß die häufigsten 74 Wörter mehr als $\sqrt{10^5} \approx 316$ Nachfolger haben, d. h. für 99,926 % aller Wörter ist die Bedingung $v(n) \leq v_{\max}$ erfüllt und für weniger als 0,1 % der Wörter müssen beim Aufbau des Wortzellen-Netzes Wortfolgenzellen angelegt werden. Bei dieser Abschätzung ist allerdings nicht berücksichtigt, daß Wortfolgenzellen ebenfalls nur $v_{\max}$ Verbindungen haben dürfen und man eventuell Zellen mit relativ langen Wortfolgen erzeugen muß: Wieviele Wortfolgenzellen man bei gegebenem $v_{\max}$ tatsächlich benötigt, wurde bei umfangreichen Messungen [83], deren Ergebnisse im nächsten Abschnitt präsentiert werden, mit verschieden langen Textausschnitten ermittelt. Für den Aufbau der Wortzellen-Netze, der bisher nur qualitativ anhand von Beispielen erläutert worden ist, wurde dabei folgender Algorithmus verwendet [83]:

1. Gegeben sei ein Text mit s Wörtern W_1, W_2, W_3, …, W_s und die maximal zugelassene Anzahl $v_{\max}$ der Verbindungen pro Zelle. Es existiert noch keine Zelle des aufzubauenden Wortzellen-Netzes. Setze $i = 1$, $m = 0$ und $n = 0$.

2. Suche diejenige Zelle, die das Wort W_i enthält. Wenn diese Wortzelle nicht existiert, dann erzeuge eine neue Zelle, in die das Wort W_i eingetragen wird. Die Zelle mit dem Wort W_i sei die „Vorgängerzelle“.

3. Wenn eine Wortfolgenzelle existiert, die die Wortfolge W_i, …, W_{i+j} enthält, und keine andere Zelle die Wortfolge W_i, …, W_{i+k} mit $k > j$ beinhaltet, dann setze $m = j$: Die Zelle mit W_i, …, W_{i+m} sei nun die Vorgängerzelle.

4. Für $i+m = s$ ist das Durchlesen des Textes beendet und es wird mit Schritt 16 fortgefahren.

5. Suche diejenige Zelle, die das Wort W_{i+m+1} enthält. Wenn diese Zelle nicht existiert, dann erzeuge eine neue Zelle, in die das Wort W_{i+m+1} eingetragen wird. Die Zelle mit dem Wort W_{i+m+1} sei die „Nachfolgerzelle“.

6. Wenn eine Wortfolgenzelle existiert, die die Wortfolge W_{i+m+1}, …, $W_{i+m+1+j}$ enthält, und keine andere Zelle die Wortfolge W_{i+m+1}, …, $W_{i+m+1+k}$ mit $k > j$ beinhaltet, dann setze $n = j$: Die Zelle mit W_{i+m+1}, …, $W_{i+m+1+n}$ sei nun die Nachfolgerzelle.

7. Wenn die Verbindung von der Vorgänger- zu der Nachfolgerzelle schon existiert, dann fahre mit Schritt 13 fort.

8. Wenn die Anzahl der Verbindungen der Vorgängerzelle kleiner ist als die zugelassene Maximalanzahl $v_{\max}$, dann fahre mit Schritt 12 fort.

9. Suche diejenige Verbindung der Vorgängerzelle, die die größte Häufigkeit hat. Die Zelle, auf die diese Verbindung zeigt, sei die „Verschmelzungszelle“.

10. Lösche die Verbindung von der Vorgänger- zu der Verschmelzungszelle.

11. Erzeuge eine neue Wortfolgenzelle, die die Wörter der Vorgänger- und der Verschmelzungszelle enthält.

12. Trage die Verbindung von der Vorgänger- zu der Nachfolgerzelle ein und setze den Häufigkeitszähler dieser Verbindung auf Null.

13. Inkrementiere den Häufigkeitszähler der Verbindung von der Vorgänger- zu der Nachfolgerzelle um 1.

14. Für $i + m + 1 + n = s$ ist das Durchlesen des Textes beendet und es wird mit Schritt 16 fortgefahren.

15. Die Vorgängerzelle sei die bisherige Nachfolgerzelle. Inkrementiere i um $m+1$, setze $m = n$, $n = 0$ und fahre mit Schritt 5 fort.

16. Wenn bei diesem Lese-Durchgang keine neuen Wortfolgenzellen erzeugt worden sind, dann fahre mit Schritt 18 fort.

17. Lösche alle Verbindungen, setze $i = 1$, $m = 0$, $n = 0$ und fahre mit Schritt 2 fort.

18. Das Wortzellen-Netz ist vollständig aufgebaut und keine Zelle besitzt mehr als $v_{\max}$ Verbindungen.

8.1 Die Simulation eines Wortzellen-Netzes

Bei einer Untersuchung, deren Ergebnisse im Institutsbericht [83] zusammengefaßt sind, wurde der oben erläuterte Algorithmus für den Aufbau eines Wortzellen-Netzes eingesetzt, wobei folgende Messungen durchgeführt wurden: Zur Bestimmung des Aufwands, den man beim Netzaufbau treiben muß, wurde die Anzahl N_{w} der Wortzellen und die Anzahl N_{f} der Wortfolgenzellen ermittelt. Andere Kenngrößen eines Netzes sind z. B. die mittlere Anzahl $\overline{v}$ der Verbindungen pro Zelle, die maximale Anzahl $f_{\max}$ der Wörter pro Zelle und die mittlere Anzahl $\overline{f}$ sowie die Anzahl D der Lese-Durchgänge. Alle genannten Parameter sind abhängig von der gewählten Textlänge s und der maximal zugelassenen Anzahl $v_{\max}$ der Verbindungen pro Zelle. Für die Untersuchungen wurden Teile des LIMAS-Korpus [2] verwendet, wobei die Textlänge s im Intervall zwischen 64 und 16 384 Wörtern und die maximale Verbindungsanzahl $v_{\max}$ zwischen 2 und 512 Verbindungen variiert wurde.

Das Bild 8.8 zeigt den Quotienten $N_{\mathrm{f}}/N_{\mathrm{w}}$ der Anzahl der Wortfolgenzellen zu derjenigen der Wortzellen in Abhängigkeit der Textlänge s, wenn pro Zelle zwei Verbindungen zugelassen sind ($v_{\max} = 2$). Die mit „×“ gekennzeichneten Meßwerte wurden unter Verwendung des obigen Algorithmus ermittelt, die Meßwerte mit „○“ ebenfalls mit dieser Methode, wobei jedoch die Verbindungen im Schritt 17 *nicht* gelöscht wurden, um die Auswirkungen der überflüssigen Verbindungen demonstrieren zu können. Bei beiden Meßreihen ist ein starkes Anwachsen der Anzahl der Wortfolgenzellen für länger werdende Texte zu beobachten: Beispielsweise benötigt man für $s = 16\,384$ mehr als doppelt so viele Wortfolgen- wie Wortzellen, wenn die überflüssigen Verbindungen nicht gelöscht werden; beim Löschen dieser Verbindungen beträgt das Verhältnis $N_{\mathrm{f}}/N_{\mathrm{w}}$ nur etwa 1,66. Ein derartiges Ergebnis war zu

erwarten, da überflüssige Verbindungen zum frühzeitigen Überschreiten der zugelassenen Verbindungsanzahl und damit zu einer zu großen Anzahl an Wortfolgenzellen führen: Überflüssige Verbindungen sollten demnach gelöscht werden.

Die Bilder 8.9 bis 8.11 geben die entsprechenden Meßergebnisse für andere Werte von $v_{\max}$ wieder.* Je mehr Verbindungen pro Zelle erlaubt sind, desto weniger Wortfolgenzellen müssen erzeugt werden: Für $v_{\max} = 128$ und $s = 16\,384$ kommt auf etwa fünf Wortzellen lediglich eine Wortfolgenzelle (siehe Bild 8.11).

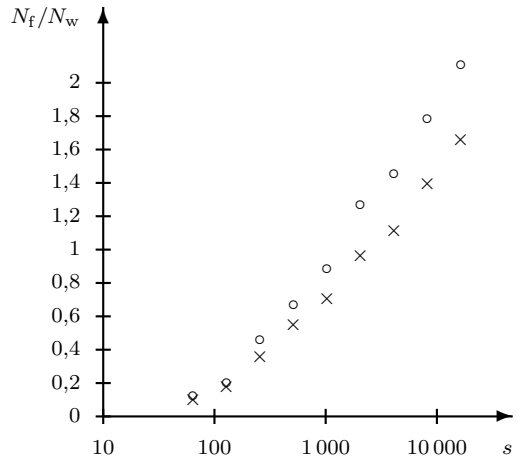

Bild 8.8: Das Verhältnis N_f/N_w über der Textlänge s für $v_{\max} = 2$ (Erläuterung von „×“ und „○“ siehe Text).

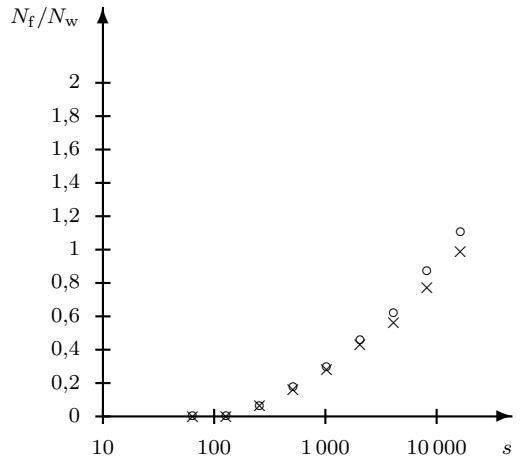

Bild 8.9: Das Verhältnis N_f/N_w über der Textlänge s für $v_{\max} = 8$ (Erläuterung von „×“ und „○“ siehe Text).

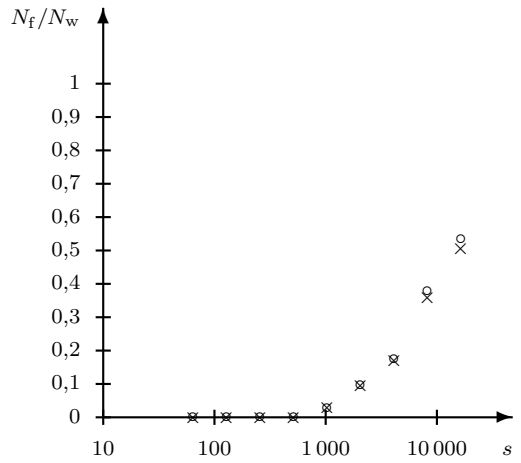

Bild 8.10: Das Verhältnis N_f/N_w über der Textlänge s für $v_{\max} = 32$ (Erläuterung von „×“ und „○“ siehe Text).

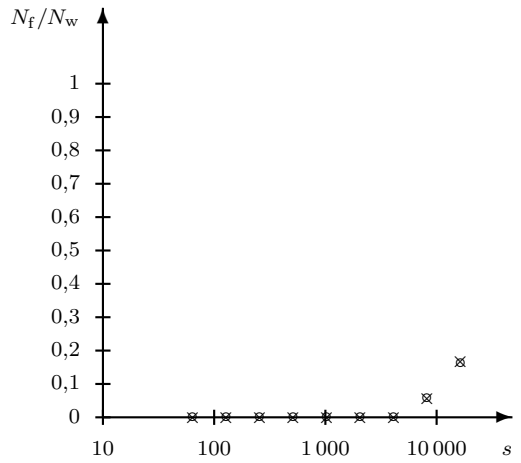

Bild 8.11: Das Verhältnis N_f/N_w über der Textlänge s für $v_{\max} = 128$ (Erläuterung von „×“ und „○“ siehe Text).

* Der Maßstab der Bilder 8.8 und 8.9 unterscheidet sich von demjenigen der Bilder 8.10 und 8.11!

In Bild 8.12 ist der Quotient $N_\mathrm{f}/N_\mathrm{w}$ in Abhängigkeit der maximal zugelassenen Anzahl v_max der Verbindungen pro Zelle für die konstante Textlänge $s = 16\,384$ wiedergegeben. Wie man schon anhand der Bilder 8.8 bis 8.11 erkennen konnte, ist eine starke Zunahme der Wortfolgenzellen-Anzahl festzustellen, wenn man die Anzahl der zugelassenen Verbindungen pro Zelle verkleinert.

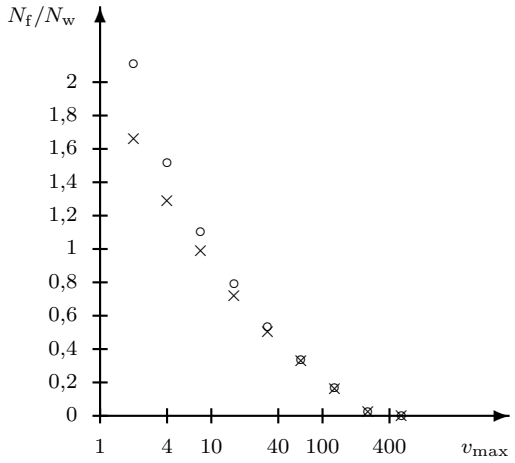

Bild 8.12: Der Quotient $N_\mathrm{f}/N_\mathrm{w}$ in Abhängigkeit der maximal zugelassenen Anzahl v_max der Verbindungen pro Zelle für $s = 16\,384$ („×“ ≙ überflüssige Verbindungen wurden gelöscht, „○“ ≙ überflüssige Verbindungen wurden *nicht* gelöscht).

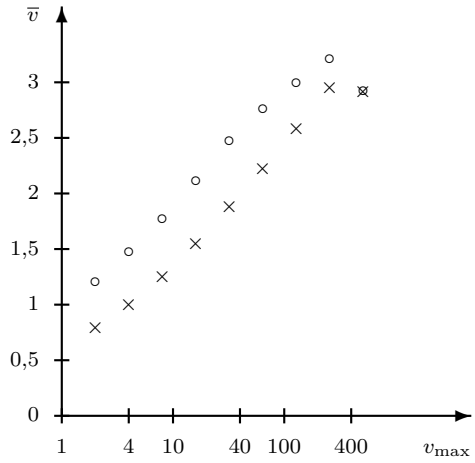

Bild 8.13: Mittlere Anzahl $\overline{v}$ der Verbindungen pro Zelle in Abhängigkeit der Maximalanzahl v_max für die Textlänge $s = 16\,384$ („×“ ≙ überflüssige Verbindungen wurden gelöscht, „○“ ≙ überflüssige Verbindungen wurden *nicht* gelöscht).

Ebenfalls für die Textlänge $s = 16\,384$ ist in Bild 8.13 die mittlere Anzahl $\overline{v}$ der Verbindungen pro Zelle über der Maximalanzahl v_max aufgetragen. Zunächst ist erstaunlich, daß nach diesem Bild für alle Werte von v_max eine Zelle im Mittel weniger als vier Verbindungen hat. Diese Meßwerte sind durchaus realistisch, wie die Berechnung von $\overline{v}$ zeigt, für die der Quotient aus der Anzahl n_paar der Wortpaare und dem Wortvorrat n_0 gebildet wird: Mit $n_0 = 10^5$ und dem Näherungswert aus Gleichung (3.28) bestimmt man die mittlere Nachfolgeranzahl $\overline{v}$ der Wörter des LIMAS-Korpus zu

$$\overline{v} = \frac{n_\mathrm{paar}}{n_0} \approx \frac{450\,000}{10^5} = 4{,}5 \qquad \text{für } n_0 = 10^5. \tag{8.4}$$

Außerdem stellt man anhand von Bild 8.13 fest, daß für $v_\mathrm{max} = 2$ der mit „×“ gekennzeichnete Meßwert (überflüssige Verbindungen wurden gelöscht) unter 1 liegt, d. h. eine Zelle im Mittel *weniger* als eine Verbindung hat: Wegen der sehr kleinen zulässigen Verbindungsanzahl müssen hier viele Wortfolgenzellen erzeugt werden, von denen man einige nur temporär benötigt. Diese Zellen sind im endgültigen Netz überflüssig und haben keine Verbindungen zu anderen Zellen, wodurch der Mittelwert $\overline{v}$ entsprechend gesenkt wird.

In den folgenden beiden Bildern ist die mittlere Anzahl $\overline{f}$ der Wörter pro Zelle (mit „•“ dargestellt) der Maximalanzahl $f_{\max}$ (mit „∗“ bezeichnet) gegenübergestellt. Im linken Bild ist $\overline{f}(s)$ und $f_{\max}(s)$ für $v_{\max} = 2$ aufgetragen, im rechten $\overline{f}(v_{\max})$ und $f_{\max}(v_{\max})$ für $s = 16\,384$.

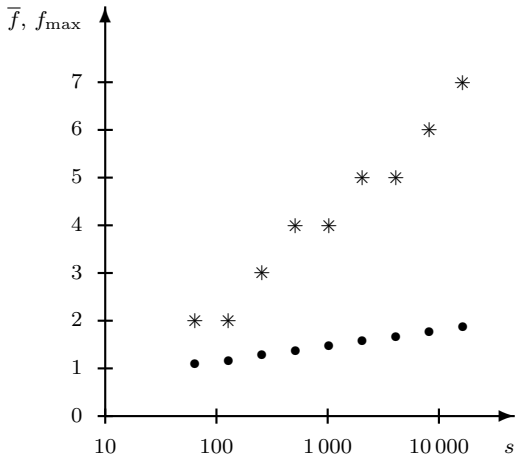

Bild 8.14: Die mittlere Anzahl $\overline{f}$ (≙ „•“) und die maximale Anzahl $f_{\max}$ (≙ „∗“) der Wörter pro Zelle in Abhängigkeit von s für $v_{\max} = 2$.

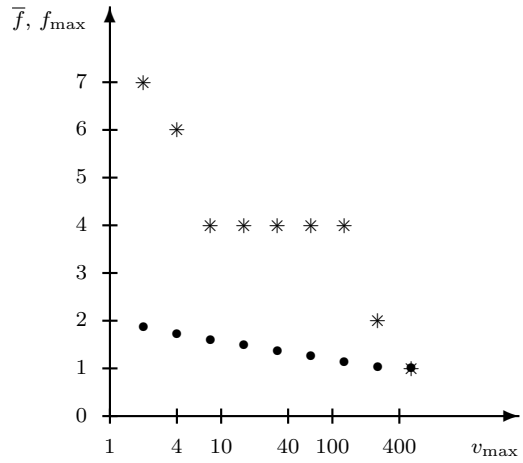

Bild 8.15: Die mittlere Anzahl $\overline{f}$ (≙ „•“) und die maximale Anzahl $f_{\max}$ (≙ „∗“) der Wörter pro Zelle in Abhängigkeit von $v_{\max}$ für $s = 16\,384$.

Der Mittelwert $\overline{f}$ liegt in der Größenordnung von eins und reagiert nur wenig auf Änderungen von s und $v_{\max}$: Für alle untersuchten Parameterkombinationen enthält eine Zelle im Mittel stets weniger als zwei Wörter. Ein Vergleich der Bilder 8.13 und 8.15 zeigt, daß für kleiner werdendes $v_{\max}$ die mittlere Anzahl $\overline{v}$ der Verbindungen pro Zelle etwa im gleichen Maße ab- wie die mittlere Anzahl $\overline{f}$ der Wörter pro Zelle zunimmt: Setzt man voraus, daß bei der Simulation eines Wortzellen-Netzes der Speicherplatzbedarf einer Verbindung mit demjenigen eines Wortes übereinstimmt, dann ist der Speicherplatz, den man im Mittel für eine Zelle benötigt, in erster Näherung unabhängig von $v_{\max}$. Das Verhältnis $N_{\mathrm{f}}/N_{\mathrm{w}}$ (siehe Bild 8.12) gibt somit Aufschluß über den *Mehr*aufwand, den man sich durch die Begrenzung der Verbindungsanzahl und die Erzeugung von Wortfolgenzellen bei einer konstanten Textlänge s einhandelt. Wählt man beispielsweise $v_{\max}$ so, daß genausoviele Wortfolgen- wie Wortzellen angelegt werden, dann hat man etwa mit dem doppelten Speicherbedarf zu rechnen (siehe Bild 8.12).

Die maximale Anzahl $f_{\max}$ der Wörter pro Zelle liegt für den extremen Fall $v_{\max} = 2$ und $s = 16\,384$ zwar bei sieben Wörtern, für realistischere Werte der zugelassenen Verbindungsanzahl (z. B. $v_{\max} = 256$ Verbindungen pro Zelle) enthält eine Zelle allerdings maximal nur zwei Wörter (siehe Bild 8.15): Offensichtlich tritt der oben angesprochene Fall, daß eine Wortfolgenzelle mehr als $v_{\max}$ Verbindungen besitzt, hier nicht ein, da dann mindestens eine Zelle existieren müßte, die mehr als zwei Wörter beinhaltet.

Wenn man die überflüssigen Verbindungen nicht löscht, dann ergeben sich für $\overline{f}(s)$, $f_{\max}(s)$, $\overline{f}(v_{\max})$ und $f_{\max}(v_{\max})$ praktisch die gleichen Diagramme wie in den Bildern 8.14 und 8.15 (siehe [83]): Das Löschen der überflüssigen Verbindungen wirkt sich demnach kaum auf $\overline{f}$ und $f_{\max}$ aus. Auch in den Bildern 8.8 bis 8.13 liegen die Ergebnisse beider Meßreihen relativ dicht beieinander, so daß der Löschvorgang im Schritt 17 des obigen Algorithmus scheinbar keinerlei Vorteile bringt. Deutliche Unterschiede der Meßergebnisse wurden jedoch bei der Anzahl D der Lese-Durchgänge festgestellt: In den folgenden beiden Bildern ist wiedergegeben, wie oft man den Lerntext durchlesen muß, bis keine neuen Wortfolgenzellen mehr entstehen (siehe Schritt 16 des oben erläuterten Algorithmus); links ist $D(s)$ für $v_{\max} = 2$ und rechts $D(v_{\max})$ für $s = 16\,384$ aufgetragen, wobei die mit „×“ bezeichneten Werte *mit* Löschen und die mit „○“ eingetragenen Werte *ohne* Löschen der überflüssigen Verbindungen ermittelt wurden.

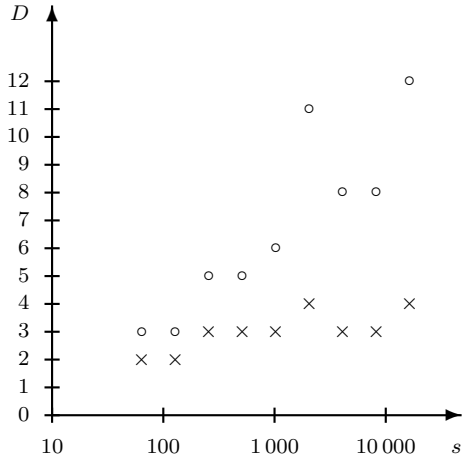

Bild 8.16: Die Anzahl D der Lese-Durchgänge in Abhängigkeit von s für $v_{\max} = 2$ („×“ ≙ überflüssige Verbindungen wurden gelöscht, „○“ ≙ überflüssige Verbindungen wurden *nicht* gelöscht).

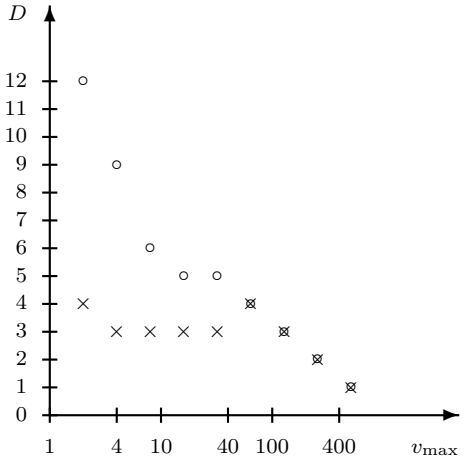

Bild 8.17: Die Anzahl D der Lese-Durchgänge in Abhängigkeit von $v_{\max}$ für $s = 16\,384$ („×“ ≙ überflüssige Verbindungen wurden gelöscht, „○“ ≙ überflüssige Verbindungen wurden *nicht* gelöscht).

Offensichtlich verhindern die überflüssigen Verbindungen eine schnelle Konvergenz beim Netzaufbau: Werden diese Verbindungen gelöscht, dann ist die Anzahl D der Lese-Durchgänge praktisch unabhängig von der Textlänge s. Nach einer groben Abschätzung anhand von Bild 8.16 muß ein gegebener Lerntext etwa dreimal eingelesen werden, d. h. wegen der Begrenzung der Verbindungsanzahl hat man ungefähr mit einer Verdreifachung der Rechenzeit zu rechnen.

Nachdem der Speicher- und Rechenzeitaufwand für den Aufbau eines Netzes mit dem beschriebenen Algorithmus nun abschätzbar ist, soll die „Struktur“ eines solchen Wortzellen-Netzes untersucht werden: In Bild 8.18 ist die Verzweigungsgradverteilung $v(n)$ der Wörter eines Ausschnittes aus dem LIMAS-Korpus, der $s = 16\,384$ Wörter umfaßt, wiedergegeben, wobei diese Meßwerte mit „•“ bezeichnet sind. In

Bild 8.19 sind die absoluten Häufigkeiten $a(n)$ dieser Wörter aufgetragen, die ebenfalls mit „•“ gekennzeichnet sind und sich nach dem „Zipf'schen Gesetz“ verhalten (siehe Kapitel 3.1). Mit diesem Text wurde ein Wortzellen-Netz aufgebaut, dessen maximale Verbindungsanzahl auf $v_{\max} = 32$ eingestellt wurde: Bild 8.18 zeigt die Anzahl $v^*(n)$ der Verbindungen der Zellen und Bild 8.19 die Häufigkeiten $a^*(n)$ der Zellen dieses Netzes, die in beiden Bildern mit „○“ dargestellt sind.

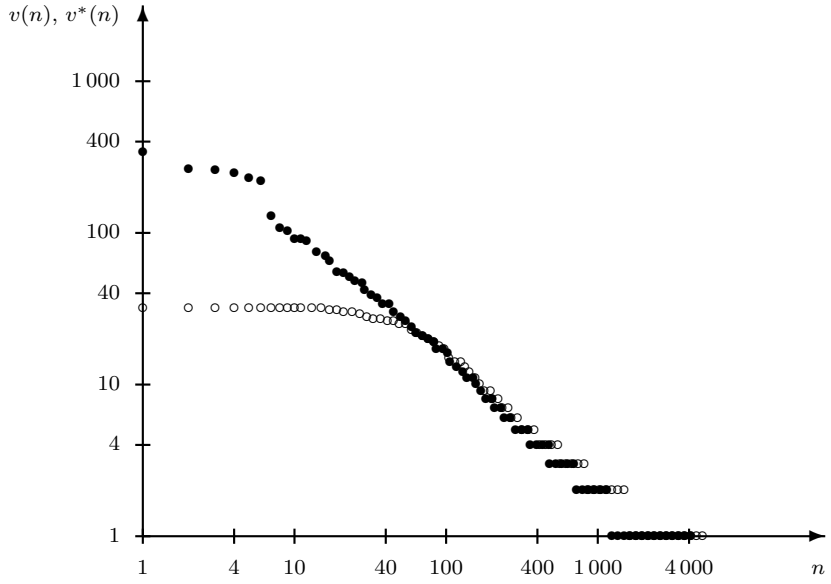

Bild 8.18: Die Anzahl $v(n)$ der Nachfolger der Wörter ($\widehat{=}$ „•“) und die Anzahl $v^*(n)$ der Verbindungen der Zellen ($\widehat{=}$ „○“) in Abhängigkeit der Ordnungszahlen n für $s = 16\,384$ und $v_{\max} = 32$.

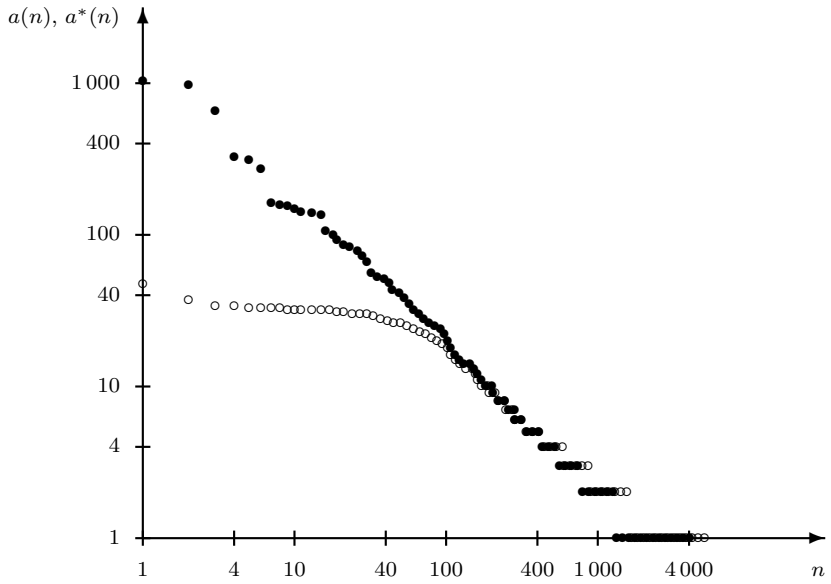

Bild 8.19: Die absoluten Häufigkeiten $a(n)$ der Wörter ($\widehat{=}$ „•“) und $a^*(n)$ der Zellen ($\widehat{=}$ „○“) in Abhängigkeit der Ordnungszahlen n für $s = 16\,384$ und $v_{\max} = 32$.

Bild 8.18 zeigt, wie der obige Algorithmus die Verzweigungsgradverteilung $v(n)$ „verbiegt“, so daß keine Zelle mehr als $v_{\max}$ Verbindungen besitzt. Wegen dieser Begrenzung mußte der Lerntext dreimal eingelesen und insgesamt $N_{\mathrm{f}} = 2\,084$ Wortfolgenzellen angelegt werden, wobei die Anzahl der Wortzellen $N_{\mathrm{w}} = 4\,118$ beträgt. Während des Netzaufbaus wurden 880 temporär benötigte Wortfolgenzellen erzeugt, so daß das endgültige Netz nur aus 5 322 Zellen besteht. Die angegebenen Zahlenwerte verdeutlichen den Aufwand für den Aufbau dieses Netzes: Wegen der maximalen Verbindungsanzahl $v_{\max} = 32$ waren etwa 50 % mehr Speicherplatz und ungefähr die dreifache Rechenzeit gegenüber $v_{\max} = \infty$ nötig.

Mit dem aufgebauten Wortzellen-Netz kann man den assoziativen Speicher von Bild 6.14 simulieren: Jeder Zelle wird eine Adresse A_i zugeordnet, die eine Bitbreite von $\lceil \mathrm{ld}(N_{\mathrm{w}} + N_{\mathrm{f}}) \rceil$ bit hat. Mit diesen Adressen, die mit den oben gegebenen Werten 13 bit breit sind, kann man die einzelnen Zellen auswählen. Außerdem werden die Verbindungen jeder Zelle binär durchnumeriert: Die Nummer einer Verbindung entspricht dem Schlüssel X_{i+1} in Bild 6.14 und dient zur Auswahl der Nachfolgerzelle. Der Schlüssel X_{i+1} hat hierbei eine Bitbreite von $\lceil \mathrm{ld}\, v_{\max} \rceil$ bit, d. h. mit dem obigen Wert ist X_{i+1} ein 5-bit-Code. Der Vorteil des beschriebenen Algorithmus zum Netzaufbau liegt darin, daß sich die Anzahl der Bitstellen von X_{i+1} mit Hilfe von $v_{\max}$ auf beliebige Werte einstellen läßt.

Das assoziative Wörterbuch von Bild 6.2 ist implizit in der Netzstruktur enthalten, da mit einer Adresse A_i diejenige Zelle ausgewählt wird, die das Wort W_i, das in diesem Fall auch eine Wort*folge* sein kann, beinhaltet.

Die Simulation eines rückgekoppelten assoziativen Wörterbuches (FAM) (siehe die Bilder 6.19 bis 6.22) bereitet nun keine Schwierigkeiten mehr: Nach Eingabe einer Wortfolge W_1, W_2, W_3, ..., W_s baut man zuerst ein Wortzellen-Netz auf, mit dem dann die Codefolge A_1, X_2, X_3, ... erzeugt wird. Diese Codefolge beschreibt den Weg durch das Netz: Mit der Adresse A_1 wählt man die erste Zelle aus und mit dem Schlüssel X_2 die Verbindung zur nächsten Zelle. Von dieser zweiten Zelle gelangt man über die Verbindung, die durch X_3 festgelegt wird, zu der dritten Zelle. Dieses Fortschreiten von Zelle zu Zelle wird solange fortgesetzt bis die gesamte Codefolge abgearbeitet ist und der komplett decodierte Text vorliegt.

Im folgenden Abschnitt wird der Aufbau einer Hierarchie aus Wortzellen-Netzen diskutiert, mit der man einen semantischen Speicher (siehe Kapitel 7) simulieren kann.

8.2 Die Simulation eines semantischen Speichers

Der semantische Speicher, der in den Bildern 7.1 und 7.2 wiedergegeben ist, besteht aus hierarchisch angeordneten assoziativen Feldern, die man mit rückgekoppelten assoziativen Wörterbüchern (FAMs) realisieren kann (siehe die Bilder 7.3 und 7.4). Die Simulation eines FAM ist z. B. mit dem oben beschriebenen Wortzellen-Netz möglich.

Da sich die FAMs der Ebenen eines semantischen Speichers nur in ihrem Inhalt, nicht aber in ihrem Aufbau unterscheiden, genügt es, das FAM einer einzigen Ebene zu realisieren, das nach dem Einlesen des gespeicherten Netzes der μ-ten Ebene die (Meta-)Wortfolge $W_{\mu,1}$, $W_{\mu,2}$, $W_{\mu,3}$, ..., $W_{\mu,s}$ in die (Meta-)Codefolge $A_{\mu,1}$, $W_{\mu+1,1}$, $W_{\mu+1,2}$, $W_{\mu+1,3}$, ... umwandelt und beim Decodieren den umgekehrten Vorgang durchführt.

Die Adressen $A_{\mu,i}$ sind ganze Zahlen aus der Menge $\{0, 1, 2, 3, \ldots, (N_{\mathrm{w},\mu} + N_{\mathrm{f},\mu} - 1)\}$, wobei $(N_{\mathrm{w},\mu} + N_{\mathrm{f},\mu})$ die Anzahl der Zellen auf der μ-ten Ebene angibt. Ein Metawort $W_{\mu+1,k}$ wird aus zwei Schlüsseln $X_{\mu,i}$ und $X_{\mu,i+1}$ zusammengesetzt: Mit dem Schlüssel $X_{\mu,i}$ wählt man einen bestimmten Nachfolger des Wortes $W_{\mu,i-1}$ aus, d. h. $X_{\mu,i} \in \{0, 1, 2, 3, \ldots, (v_{\max,\mu} - 1)\}$, wobei $v_{\max,\mu}$ die maximale Verbindungsanzahl des Netzes auf der μ-ten Ebene ist. Das Metawort $W_{\mu+1,k}$ wird dann nach folgender Vorschrift gebildet:

$$W_{\mu+1,k} = X_{\mu,i} + v_{\max,\mu} \cdot X_{\mu,i+1}. \tag{8.5}$$

Bei gegebenem Metawort bestimmt man die Schlüssel $X_{\mu,i}$ und $X_{\mu,i+1}$ wie folgt:

$$X_{\mu,i} = W_{\mu+1,k} \bmod v_{\max,\mu}. \tag{8.6}$$

$$X_{\mu,i+1} = \frac{W_{\mu+1,k} - X_{\mu,i}}{v_{\max,\mu}}. \tag{8.7}$$

Nach Gleichung (8.5) ist das Metawort $W_{\mu+1,k}$ eine ganze Zahl aus der Menge $\{0, 1, 2, 3, \ldots, (v^2_{\max,\mu} - 1)\}$. Da für alle Ebenen des semantischen Speichers dieselben Randbedingungen gelten sollen, müssen auch die Wörter $W_{1,i}$ ganze Zahlen sein: Man benötigt deshalb ein „Codebuch“, das jedem Wort eineindeutig einen Code $W_{1,i}$ zuweist, wobei $W_{1,i} \in \{0, 1, 2, 3, \ldots, (n_0 - 1)\}$. Dieses Codebuch kann ähnlich wie das assoziative Wörterbuch von Kapitel 5.1.1 mit Hilfe der Streuspeicherung realisiert werden.

Mit dem erwähnten Codebuch soll das Codieren beliebiger Texte möglich sein: Diese Forderung kann allerdings nicht erfüllt werden, solange das Codebuch nur die Wörter des Lerntextes enthält und andere Wörter nicht codierbar sind: Wie in Kapitel 5.5 werden daher Silben und ASCII-Zeichen in das Codebuch aufgenommen, so daß ein unbekanntes Wort aus Silben oder, falls auch fremde Silben auftreten, aus Buchstaben zusammengesetzt werden kann. Ein gegebener Text wird auf diese Art vor der Eingabe in den semantischen Speicher in eine Zahlenfolge $W_{1,1}$, $W_{1,2}$, $W_{1,3}$, ... umgewandelt.

Für die Simulation eines semantischen Speichers mit dem beschriebenen Konzept bietet sich eine weitere Maßnahme an: Wie Bild 8.18 zeigt, haben die meisten Zellen (hier: 69,7 %) nur eine einzige Verbindung; deshalb kommt der Schlüssel „0“ besonders häufig vor. Diesen Effekt kann man verstärken, indem man die Verbindungen jeder Zelle nach ihren Häufigkeiten sortiert und in dieser Reihenfolge durchnumeriert: Der Schlüssel „$(v_{\max,\mu} - 1)$“ hat dann die kleinste, der Schlüssel „0“ die größte Häufigkeit. Wie S. Bolz und M. Spieß zeigten, verhalten sich die Häufigkeiten der Metaworte annähernd nach dem „Zipf'schen Gesetz“ (siehe Kapitel 3.1), wenn man

die Verbindungen der Zellen sortiert [112]. Würde man die Verbindungen *nicht* sortieren, könnte man feststellen, daß sich die Häufigkeitsverteilungen der Metaworte auf den „höheren“ Ebenen des semantischen Speichers egalisieren: Dieser Effekt ist allerdings unerwünscht, da die Entropie gleichverteilter Symbole größer ist als diejenige ungleichmäßig verteilter [20]. Mit dem semantischen Speicher nutzt man letztendlich die Tatsache, daß die Häufigkeiten sowohl der (Meta-)Worte als auch ihrer Nachfolger sehr ungleichmäßig verteilt sind, für eine redundanzarme Codierung aus. Das Sortieren der Verbindungen wirkt einer Egalisierung der Häufigkeitsverteilungen entgegen und steigert somit die erreichbare Redundanzreduktion.

Nach der beschriebenen Methode wurde mit dem gesamten LIMAS-Korpus [2] ein semantischer Speicher aufgebaut, der aus 17 Ebenen besteht. Das dabei verwendete Codebuch (siehe oben) enthält die 30 627 häufigsten Wörter und die 8 234 häufigsten Silben dieses Standardtextes. Bei der Codierung waren damit 92,0 % aller Wörter im Codebuch enthalten; die restlichen Wörter konnten in 356 717 Silben und 78 049 ASCII-Zeichen zerlegt und codiert werden: Die 1 288 120 Wörter des LIMAS-Korpus wurden somit in 1 620 338 Zahlencodes $W_{1,i}$ umgesetzt. Der Aufbau des ersten Wortzellen-Netzes mit diesem „Zahlentext“ dauerte insgesamt etwa 19 CPU-Stunden, wobei eine VAX 8530 verwendet wurde. Mit dem Metatext, der nur aus 650 888 Metaworten besteht, konnte das zweite Netz in etwas mehr als 9 CPU-Stunden aufgebaut werden. Da die „Metametatexte“ mit jeder weiteren Ebene kürzer werden (siehe Kapitel 7), dauerte der Aufbau der dritten Ebene nur noch 1,5 CPU-Stunden und der Aufbau von der vierten bis zur 17ten Ebene zusammengenommen nur 29 CPU-Minuten.

Die folgenden beiden Bilder geben die Anzahl $(N_{\mathrm{w},\mu} + N_{\mathrm{f},\mu})$ der Zellen sowie die maximale Verbindungsanzahl $v_{\max,\mu}$ der Netze aller Ebenen des semantischen Speichers wieder, der mit dem LIMAS-Korpus aufgebaut wurde.

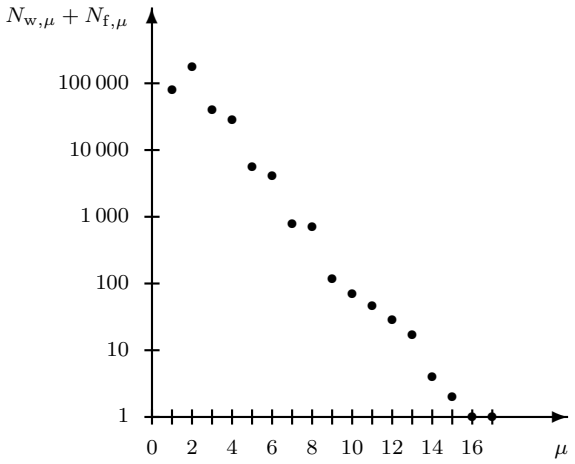

Bild 8.20: Die Anzahl $(N_{\mathrm{w},\mu} + N_{\mathrm{f},\mu})$ der Zellen des Wortzellen-Netzes auf der μ-ten Ebene des semantischen Speichers.

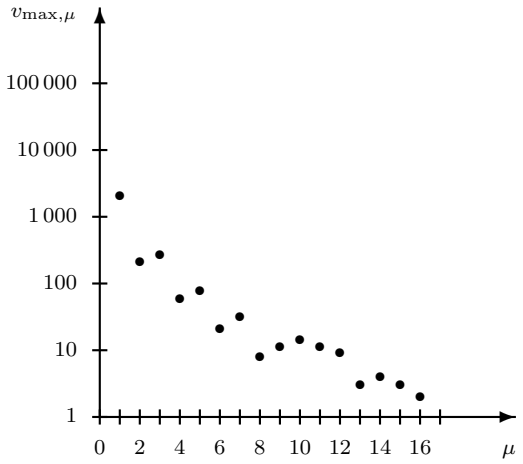

Bild 8.21: Die maximale Anzahl $v_{\max,\mu}$ der Verbindungen pro Zelle für das Netz auf der μ-ten Ebene des semantischen Speichers.

Bei der beschriebenen Simulation wurde die maximale Verbindungsanzahl $v_{\max,\mu}$ jeder Ebene so gewählt, daß entweder die Anzahl der Wortfolgenzellen mit derjenigen der Wortzellen ungefähr übereinstimmt oder überhaupt keine Wortfolgenzellen angelegt werden müssen. Bild 8.22 zeigt das Verhältnis $N_{\mathrm{f},\mu}/N_{\mathrm{w},\mu}$ aller Ebenen des semantischen Speichers und Bild 8.23 die mittlere Verbindungsanzahl $\overline{v}_\mu$.

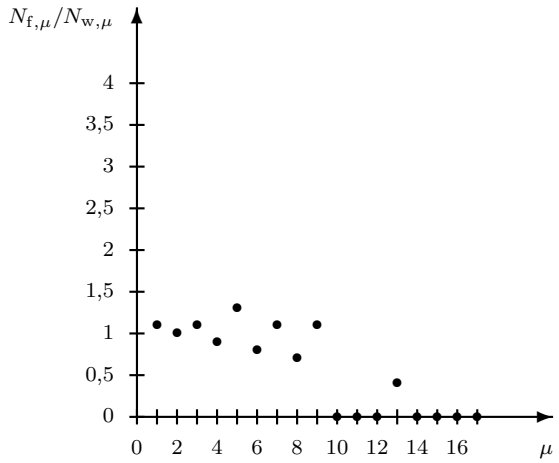

Bild 8.22: Der Quotient $N_{\mathrm{f},\mu}/N_{\mathrm{w},\mu}$ des Netzes auf der μ-ten Ebene des semantischen Speichers.

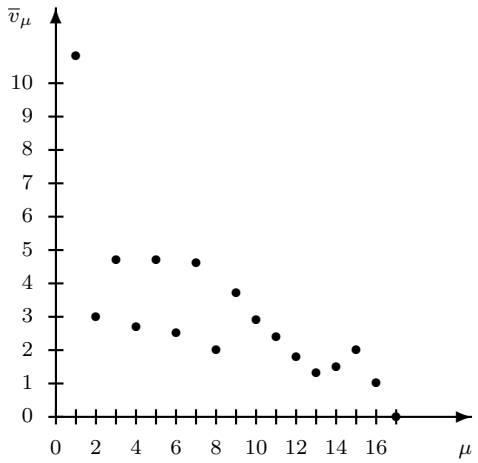

Bild 8.23: Die mittlere Anzahl $\overline{v}_\mu$ der Verbindungen pro Zelle für das Netz auf der μ-ten Ebene des semantischen Speichers.

Auffallend ist, daß die in den Bildern 8.20 bis 8.23 gezeigten Zusammenhänge zwischen zwei Einhüllenden hin- und herpendeln. Diese Beobachtung läßt sich folgendermaßen erklären: Wenn ein Netz sehr viele verschiedene Zellen hat, dann ist die mittlere Verbindungsanzahl notwendigerweise relativ klein; aus diesem Grund kommt man mit einer kleinen Anzahl verschiedener Schlüssel und Metaworte aus. Das Netz der nächsten Ebene besteht dann aus wenigen Zellen, die im Mittel vergleichsweise viele Verbindungen haben: Die Anzahl der verschiedenen Schlüssel und Metametaworte ist nun wieder relativ groß, so daß das Netz der nächsten Ebene viele Zellen umfaßt.

Schließlich geben die Bilder 8.24 und 8.25 noch die mittlere Anzahl $\overline{f}_\mu$ und die Maximalanzahl $f_{\max,\mu}$ der Wörter pro Zelle sowie die Anzahl D_μ der Lese-Durchgänge für den Aufbau des Netzes auf der μ-ten Ebene wieder.

Die Bilder 8.20 bis 8.24 vermitteln einen Eindruck von der inneren Struktur des simulierten semantischen Speichers, mit dem das gesamte LIMAS-Korpus codiert wurde: Die Tabelle 8.1 zeigt die Initialisierungsadressen $A_{\mu,1}$, die dem Code dieses Standardtextes entsprechen. Initialisiert man einen Decodierer, dem ebenfalls die Netze aller 17 Ebenen zur Verfügung stehen, mit dieser Codefolge, dann wird das LIMAS-Korpus vollständig und fehlerfrei decodiert. Da die Netze der 16ten und 17ten Ebene jeweils nur aus einer einzigen Zelle bestehen (siehe Bild 8.20), benötigt man für diese „Netze“ keine Startadressen.

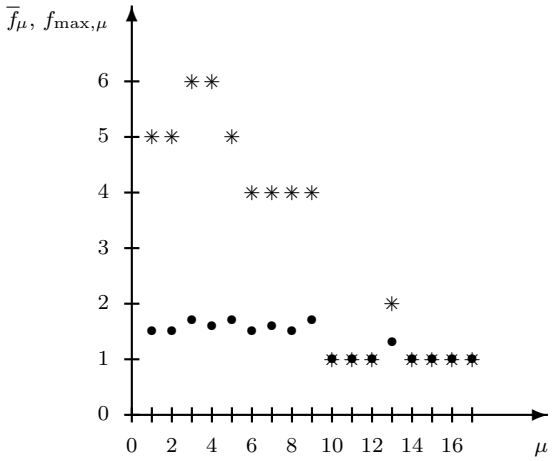

Bild 8.24: Die mittlere Anzahl $\overline{f}_\mu$ ($\widehat{=}$ „•“) und die maximale Anzahl $f_{\max,\mu}$ ($\widehat{=}$ „$*$“) der Wörter pro Zelle für das Netz auf der μ-ten Ebene des semantischen Speichers.

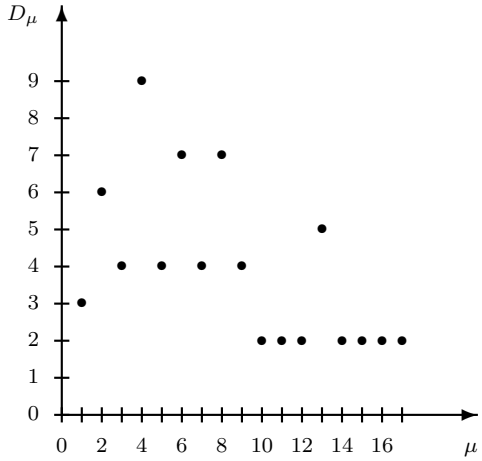

Bild 8.25: Die Anzahl D_μ der Lese-Durchgänge für den Aufbau des Netzes auf der μ-ten Ebene des semantischen Speichers.

Ebene μ	Initialisierungsadresse $A_{\mu,1}$
1	00111011101001110
2	101000101010000011
3	0001000101100010
4	000010100001001
5	0000001000011
6	0000000010110
7	0000101011
8	0100011101
9	0010000
10	0000010
11	011001
12	01111
13	00001
14	00
15	0
16	—
17	—

Tabelle 8.1: Die Initialisierungsadressen $A_{\mu,1}$ des semantischen Speichers für das Decodieren des gesamten LIMAS-Korpus.

Der Decodierer kennt die Anzahl der Bitstellen jeder Initialisierungsadresse $A_{\mu,1}$ (siehe Tabelle 8.1), so daß man eine kontinuierliche Bitfolge übertragen kann: Im folgenden Bild ist die Codefolge des LIMAS-Korpus wiedergegeben, die aus 145 bit besteht.

00111011101001110101000101010
00001100010001011000100000101
00001001000000100001100000000
10110000010101101000111010010
00000000100110010111100001000

Bild 8.26: Der Code des LIMAS-Korpus.

Da die Netze der 17 Ebenen Teil des Decodierers sind, muß nur die obige Codefolge übertragen werden, d. h. die Netze gehen nicht in die Berechnung der mittleren Codelänge ein: Das LIMAS-Korpus besteht aus 8 088 120 ASCII-Zeichen; damit bestimmt man eine mittlere Codelänge von $\overline{C}_{\text{sem}} = 17{,}9$ μbit/Buchstabe (vergleiche auch mit dem Schätzwert $\overline{C}_{\text{sem}}$ für $s = 10^6$ in Bild 7.6 auf Seite 72), ein Wert, der weit unterhalb des Shannon'schen Grenzwertes von 1 bit/Buchstabe liegt [11]. Die angegebene mittlere Codelänge ist allerdings nicht sehr anschaulich; der Kehrwert vermittelt einen besseren Eindruck: Mit einem einzigen Bit decodiert man im Mittel 55 780 Buchstaben, d. h. etwa 15–20 Seiten Text.

Insgesamt existieren $2^{145} \approx 4{,}5 \cdot 10^{43}$ verschiedene Bitfolgen, die 145 bit lang sind. Jede dieser Codefolgen entspricht einem Text, den man mit dem semantischen Speicher decodieren kann und der etwa die Länge eines LIMAS-Korpus besitzt. Wenn der „semantische“ Code redundanzfrei wäre, dann müßten alle decodierbaren Texte sinnvoll, d. h. grammatikalisch und inhaltlich korrekt sein: Wie in Kapitel 4.3.2 beschrieben, kann man mit Hilfe zufällig erzeugter Codes die „Güte“ einer Codierung stichprobenartig prüfen.

8.2.1 Die Erzeugung von Zufallstexten

Zur Erzeugung zufälliger Initialisierungsadressen $A_{\mu,1}$ wurde nach [36] ein einfacher Zufallszahlengenerator programmiert, mit dem verschiedene „Zufallstexte“ erzeugt wurden:

Bild 8.27 zeigt den Ausschnitt einer zufälligen Wortfolge, die durch simples Auswürfeln der Wörter eines Wörterbuches entstand. Dieser „Text“ enthält weder eine korrekte grammatikalische Konstruktion noch eine inhaltliche Aussage.

Der Zufallstext von Bild 8.28 wurde dagegen mit einem semantischen Speicher generiert, wobei die Startadressen $A_{\mu,1}$ Zufallszahlen waren. Zum Aufbau dieses semantischen Speichers wurde das Lesebuch [4] verwendet, so daß alle decodierbaren Texte aus dem Wortschatz dieses Kinderbuches bestehen und entsprechend Bild 8.28 etwas „surrealistisch“ anmuten.

Großmutter kugelrund wenn versuche Tier pfiffiger englisches tragen Japan deinen weitergeklettert lieben geschlüpft versuchen ich gepackt bewegte Stein Gleich Unsinn wachsam schlüpfte vergiß ein erhob würde verwechsle zum Zuhause diesem langsam letzten diese Absprache Bär nennt ...

Bild 8.27: Durch einfaches Auswürfeln der Wörter entstandener Zufallstext.

Ich trinke am Himmel. Und dann — sah er auch nicht in Afrika, der Spatz, so kennt er doch hier, wie bunt er ist und wie schön du singst“, sagte Sokrates. „Wie machst du das?“ „Ich entfliehe der Erde“, tirilierte die Lerche auf einem roten Luftballon hoch, ganz hoch, ganz gewöhnliche Blätter ...

Bild 8.28: Durch zufällig erzeugte Initialisierungsadressen eines semantischen Speichers generierter Zufallstext.

Obwohl die Qualität der mit dem semantischen Speicher erzeugten Zufallstexte durchaus akzeptabel ist,* hat das oben erläuterte Codierungssystem einen gravierenden Nachteil: Mit dem simulierten semantischen Speicher kann man nicht jeden beliebigen Text codieren, da in dem Netz der ersten Ebene nur die Wort-Übergänge des Lerntextes enthalten sind. Bei der Codierung fremder Texte kommt es vor, daß man Wort-Übergänge codieren will, die im Lerntext nicht auftreten. Im folgenden Abschnitt wird ein System beschrieben, mit dem man *beliebige* Texte codieren kann, indem nichtvorhandene Verbindungen des Wortzellen-Netzes nach einer besonderen Methode codiert werden.

8.3 Der universelle semantische Speicher

Nach den beschriebenen Voruntersuchungen soll nun ein praktisch einsetzbares Codierungssystem für Texte aufgebaut werden, mit dem man z. B. folgende Aufgabe lösen kann: Beliebige Texte sollen möglichst redundanzarm codiert werden, damit eine schnelle Übertragung von einem Sendeort zu einem (entfernten) Empfangsort möglich ist.

Ein Modell für eine derartige Übertragung ist in Bild 8.29 gezeigt: Wenn man z. B. das gesamte LIMAS-Korpus ($\approx$ 8 Megabyte) *ohne* eine spezielle redundanzarme Codierung über eine Telefonleitung schicken will, bei der man pro Sekunde 2 400 bit übertragen kann, so dauert die Übertragung des gesamten Textmaterials über sieben Stunden. Dagegen benötigt man für die Übertragung des redundanzarm codierten Textes, dessen Code z. B. wie oben 145 bit umfaßt, nur Bruchteile einer Sekunde. Selbstverständlich darf hierbei kein einziges Bit durch einen Übertragungsfehler verfälscht werden, da man sonst einen anderen Text decodieren würde. Aus diesem Grund wird man auf den redundanzarmen Code die üblichen Verfahren anwenden, mit denen eine gesicherte Übertragung erreicht wird. Auf diese fehlerkorrigierenden Codes soll hier jedoch nicht näher eingegangen werden.

* Der Autor füllte einen semantischen Speicher mit vier Institutsberichten [76, 81, 78, 83] und erzeugte damit einen „Zufalls-Institutsbericht“: Ein Leser dieses Zufallsberichtes nahm an, daß es sich um einen „echten“ Forschungsbericht handelt, und bemerkte dazu, das sei ihm zu „hoch“.

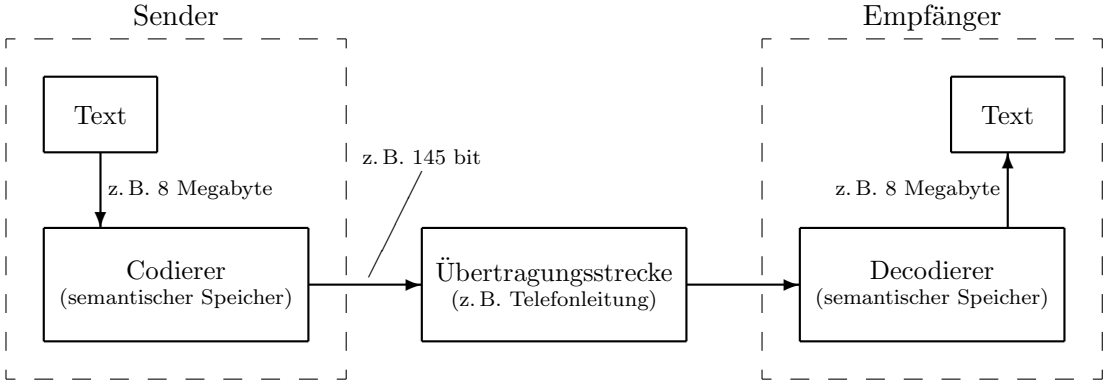

Bild 8.29: Modell für die Übertragung eines redundanzarm codierten Textes unter Verwendung von zwei komplementären semantischen Speichern.

Das in Bild 8.29 gezeigte Codierungssystem kann folgendermaßen realisiert werden: Anhand eines Lerntextes baut man zwei komplementäre semantische Speicher auf (siehe die Bilder 7.3 und 7.4 sowie die Bilder 6.19 und 6.21), die am Sende- und Empfangsort installiert werden. Mit diesem System sollen *beliebige* Texte redundanzarm codiert werden können: Dazu muß die im letzten Abschnitt erläuterte semantische Codierung entsprechend erweitert werden.

8.3.1 Das Prinzip der universellen semantischen Codierung

Bei der Codierung unbekannter Texte treten verschiedene Schwierigkeiten auf: Das erste Problem wurde bei dem oben beschriebenen semantischen Speicher schon gelöst, da zur Codierung fremder Wörter ein Codebuch verwendet wird, in dem häufige Wörter und Silben sowie alle ASCII-Zeichen enthalten sind. Unbekannte Wörter zerlegt man mit Hilfe des Trennalgorithmus von Kapitel 5.5.1 in Silben und unbekannte Silben in Buchstaben. Mit dem Codebuch wird der Text dann in eine Zahlenfolge umgesetzt, die der semantische Speicher verarbeiten und redundanzarm codieren kann.

Während des Codiervorganges mit den Wortzellen-Netzen können die folgenden drei Fälle auftreten:

1. Sowohl die Vorgänger- als auch die Nachfolgerzelle existiert. Außerdem ist die Verbindung von der Vorgänger- zu der Nachfolgerzelle vorhanden.
2. Die Nachfolgerzelle existiert. Entweder fehlt die Vorgängerzelle oder die Verbindung von der Vorgänger- zu der Nachfolgerzelle.
3. Die Nachfolgerzelle existiert nicht.

Der erste Fall entspricht der „normalen“ Codierung, die oben erläutert wurde. Zur Lösung der beiden anderen Fälle wurden folgende Vereinbarungen getroffen: Die Wörter $W_{\mu,i}$ und die Adressen $A_{\mu,i}$ aller Ebenen haben eine konstante Bitbreite,

wobei alle Schlüssel $X_{\mu,i}$ die halbe Anzahl der Bitstellen einer Adresse aufweisen. Mit diesen Konventionen wurde eine „16-bit-Version“ eines semantischen Speichers simuliert, d. h. alle Wörter und Adressen sind jeweils 16 bit und alle Schlüssel 8 bit breit. Das Netz jeder Ebene kann damit maximal $(N_{\mathrm{w},\mu} + N_{\mathrm{f},\mu}) = 65\,536$ Zellen enthalten. Jede Zelle darf höchstens $v_{\max,\mu} = 256$ Verbindungen besitzen. Unter diesen Voraussetzungen werden bei der Codierung eines fremden Textes die obigen drei Fälle (in derselben Reihenfolge) folgendermaßen behandelt, wobei die Schlüssel „00000000“ und „00000001“ eine besondere Bedeutung haben:

1. Da die Verbindung von der Vorgänger- zu der Nachfolgerzelle existiert, codiert man diesen Übergang mit dem entsprechenden Schlüssel $X_{\mu,i}$, d. h. mit einem 8-bit-Code, und sendet diesen Code zur Meta-Ebene.

2. Da entweder die Vorgängerzelle oder die Verbindung von der Vorgänger- zu der Nachfolgerzelle fehlt, codiert man die Nachfolgerzelle mit ihrer Adresse $A_{\mu,i}$ unter Verwendung eines „Fluchtsymbols“ „0“, das dem Schlüssel „00000000“ entspricht: Dieses Fluchtsymbol und die Adresse $A_{\mu,i}$ werden zur Meta-Ebene gesendet. Beim Decodieren kündigt der Schlüssel „00000000“ dem Decodierer an, daß das nächste Codewort die Adresse $A_{\mu,i}$ der Nachfolgerzelle ist. In diesem Fall benötigt man zur Codierung eines Übergangs insgesamt 24 bit.

3. Da die Zelle, die das nächste Wort $W_{\mu,i}$ des Textes beinhaltet, nicht existiert, sendet man das Fluchtsymbol „1“, d. h. den Schlüssel „00000001“, sowie das Codewort $W_{\mu,i}$ in die Meta-Ebene. Der Schlüssel „00000001“ kündigt dann dem Decodierer an, daß das nächste Codewort das Wort $W_{\mu,i}$ ist. Hier benötigt man zur Codierung eines 16-bit-Wortes insgesamt 24 bit.

Bei der Codierung von Texten hofft man, daß die letzten beiden Fälle nur relativ selten auftreten und der erste Fall überwiegt. Prinzipiell läßt sich mit dieser Methode allerdings *jeder* Text codieren: Die erreichte minimale mittlere Codelänge $\overline{C}_{\mathrm{sem}}$ ist dann ein Maß dafür, wie „unähnlich“ der codierte Text dem Lerntext ist.

8.3.2 Der Aufbau des universellen semantischen Speichers

Bei dem im letzten Abschnitt beschriebenen semantischen Speicher wurde das Verhältnis $N_{\mathrm{f},\mu}/N_{\mathrm{w},\mu}$ durch geeignete Wahl von $v_{\max,\mu}$ so eingestellt, daß die Anzahl der Wortfolgenzellen mit derjenigen der Wortzellen etwa übereinstimmt. Für den Aufbau des neuen semantischen Speichers gelten andere Voraussetzungen, da die maximale Verbindungsanzahl $v_{\max,\mu} = 256$ fest vorgegeben ist: Die Anzahl $(N_{\mathrm{w},\mu} + N_{\mathrm{f},\mu})$ der Zellen eines Netzes stellt man nun mit der Textlänge s ein, wobei der gewählte Maximalwert von 65 536 Zellen möglichst erreicht werden sollte.

Anhand des LIMAS-Korpus wurde ein „universeller“ semantischer Speicher simuliert, der aus zwölf Ebenen besteht und mit dem beliebige Texte codiert werden können. Das hierbei verwendete Codebuch besteht aus den 2 176 häufigsten Wörtern und den 2 363 häufigsten Silben des LIMAS-Korpus sowie allen ASCII-Zeichen. Die

Untersuchungen zeigten, daß damit rund 73 % aller Wörter eines zu codierenden Textes im Codebuch enthalten sind, d. h. etwa jedes vierte Wort in Silben zerlegt werden muß. Damit man bei der Codierung fremder Texte möglichst kleine Codelängen erreichen kann, müssen die Zellen jeder Ebene die erwähnten Fluchtsymbole enthalten. Deshalb wurde jede Ebene dieses semantischen Speichers mit einem anderen LIMAS-Ausschnitt aufgebaut: Nachdem die erste Ebene mit einem ersten Textausschnitt aufgebaut worden war, wurde zum Aufbau der zweiten Ebene ein anderer Ausschnitt verwendet, der jedoch erst einmal mit Hilfe der ersten Ebene codiert werden mußte, so daß die Fluchtsymbole „0“ und „1“, die man für das Codieren fremder Texte benötigt (siehe oben), in die Zellen des zweiten Netzes eingebaut worden sind. Fremde Texte, die man auf der ersten Ebene ungünstig codiert, können damit auf der zweiten Ebene möglicherweise wieder günstiger codiert werden. Alle weiteren Ebenen wurden nach demselben Schema aufgebaut.

Die folgenden vier Bilder zeigen die Meßwerte vom Aufbau dieses semantischen Speichers (vergleiche auch mit den Bildern 8.22 bis 8.25): Bild 8.30 gibt das Verhältnis $N_{\mathrm{f},\mu}/N_{\mathrm{w},\mu}$, Bild 8.31 die mittlere Verbindungsanzahl $\overline{v}_\mu$, Bild 8.32 die mittlere Anzahl $\overline{f}_\mu$ und die Maximalanzahl $f_{\max,\mu}$ der Wörter pro Zelle und Bild 8.33 die Anzahl D_μ der Lese-Durchgänge wieder.

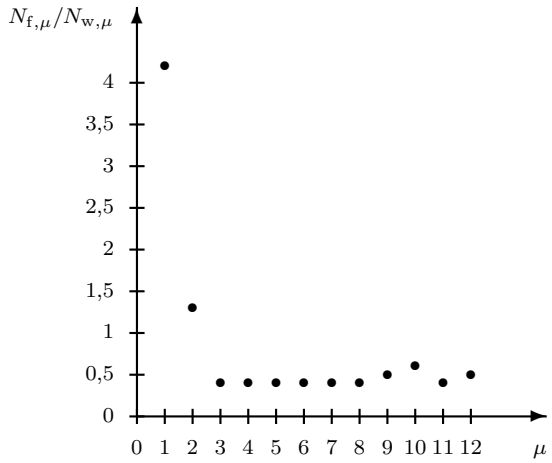

Bild 8.30: Der Quotient $N_{\mathrm{f},\mu}/N_{\mathrm{w},\mu}$ des Netzes auf der μ-ten Ebene des semantischen Speichers.

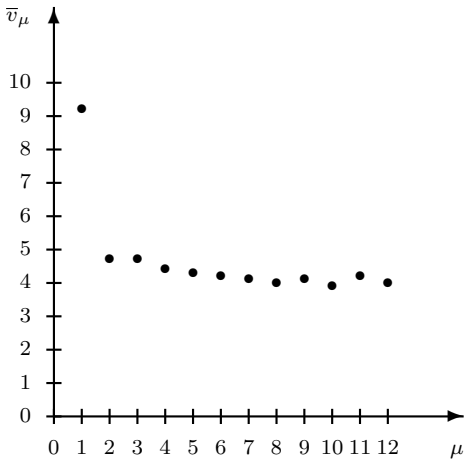

Bild 8.31: Die mittlere Anzahl $\overline{v}_\mu$ der Verbindungen pro Zelle für das Netz auf der μ-ten Ebene des semantischen Speichers.

Da die Netze aller Ebenen des semantischen Speichers nach denselben Vorgaben aufgebaut wurden, stimmen die Parameter aller Netze ungefähr überein, wobei die erste Ebene eine Ausnahme bildet (siehe die Bilder 8.30 bis 8.33): Das Netz der ersten Ebene, das die Übergänge der Wörter, Silben und ASCII-Zeichen enthält, ist offenbar so stark vernetzt, daß es überwiegend aus Wortfolgenzellen besteht und die mittlere Verbindungsanzahl $\overline{v}_1$ einer Zelle deutlich über den entsprechenden Werten der anderen Ebenen liegt. Bei den anderen Netzen wird die maximale Verbindungsanzahl $v_{\max,\mu} = 256$ relativ selten überschritten, wie Bild 8.30 zeigt.

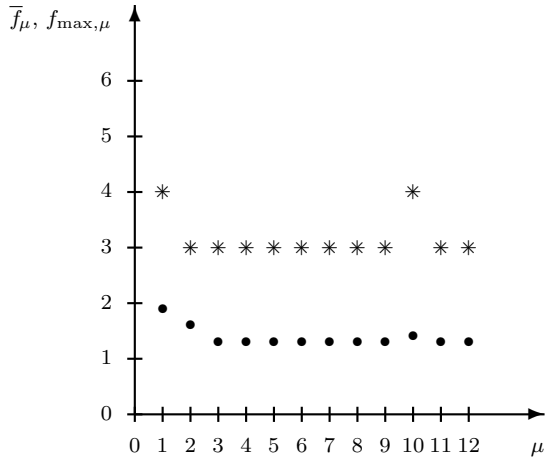

Bild 8.32: Die mittlere Anzahl $\overline{f}_\mu$ ($\widehat{=}$ „•“) und die maximale Anzahl $f_{\max,\mu}$ ($\widehat{=}$ „$*$“) der Wörter pro Zelle für das Netz auf der μ-ten Ebene des semantischen Speichers.

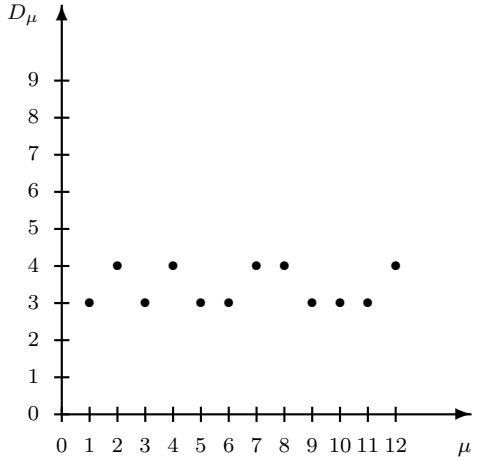

Bild 8.33: Die Anzahl D_μ der Lese-Durchgänge für den Aufbau des Netzes auf der μ-ten Ebene des semantischen Speichers.

Nach dem Aufbau eines Wortzellen-Netzes kann man die Wahrscheinlichkeit, daß die Verbindung von einer Zelle zu einer anderen existiert, durch folgende Erweiterung vergrößern, d. h. die erreichbare minimale mittlere Codelänge senken: Eine Zelle hat nach Bild 8.31 im Mittel weniger als zehn Verbindungen, so daß von den 254 potentiell codierbaren Nachfolgern nur ein geringer Bruchteil erreicht werden kann. Aus diesem Grund wurde für jede Zelle die maximale Verbindungsanzahl ausgenutzt, indem nachträglich Verbindungen zu den am häufigsten benutzten Zellen eingebaut wurden.

8.3.3 Die gemessenen Codelängen

Mit dem beschriebenen semantischen Speicher können, wie erwähnt, beliebige Texte codiert werden. Die mittleren Codelängen $\overline{C}_{\text{sem}}$, die dabei erreichbar sind, wurden anhand von fünf unterschiedlichen Texten gemessen:

Die ersten drei Texte sind verschieden lange Teile desjenigen Lerntext-Ausschnittes, der den unterschiedlichen Lerntexten *aller* zwölf Ebenen gemeinsam ist. Der erste Text enthält 257 487 Wörter, der zweite 53 541 Wörter und der dritte, bei dem es sich um das Lesebuch [4] handelt, 4 809 Wörter.

Die beiden anderen Texte stellen für den semantischen Speicher unbekanntes Textmaterial dar: Der vierte Text ist dem Lerntext stilistisch ähnlich, da er aus dem Lesebuch (siehe oben) durch Vertauschen einiger Redewendungen gewonnen wurde, so daß sich unter Beibehaltung der Textaussage eine neue Wortfolge ergab. Dieser Text besteht wie das Lesebuch aus 4 809 Wörtern. Der fünfte Text schließlich ist dem semantischen Speicher vollständig fremd: Es handelt sich hierbei um einen 1 485 Wörter langen Ausschnitt aus einer Erzählung von H. Hesse [37].

In Bild 8.34 sind die absoluten Codelängen C_{sem} der drei Lerntext-Ausschnitte über der Anzahl μ der Ebenen des semantischen Speichers aufgetragen.

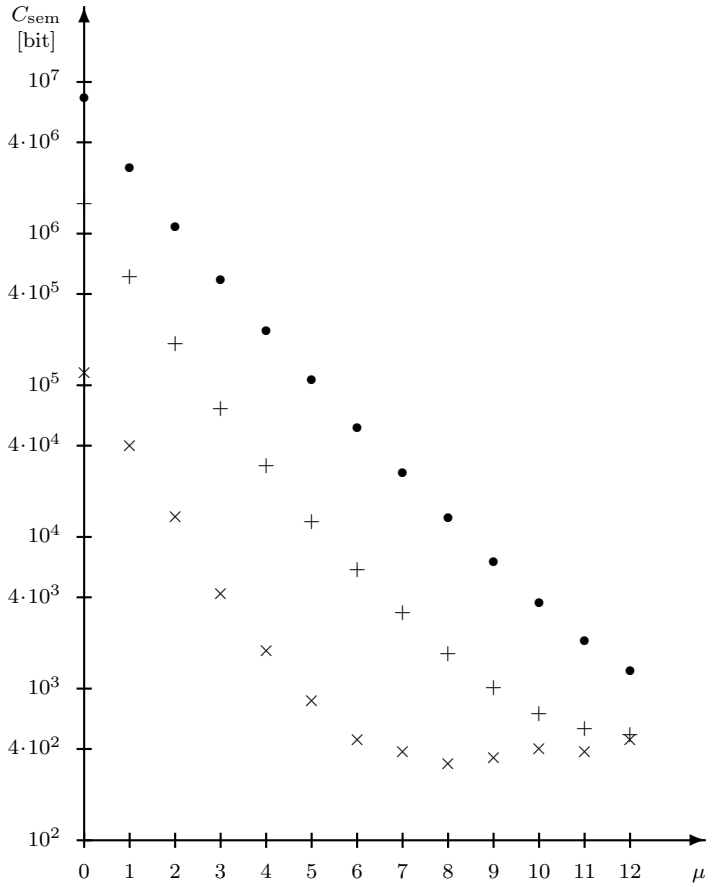

Bild 8.34: Vergleich der absoluten Codelängen C_{sem} eines langen ($\widehat{=}$ „•“), mittleren ($\widehat{=}$ „+“) und kurzen ($\widehat{=}$ „×“) Lerntext-Ausschnittes in Abhängigkeit der Anzahl μ der Ebenen, aus denen der semantische Speicher aufgebaut ist.

Entsprechend den Abschätzungen aus Kapitel 7.1 halbieren sich die Codelängen bei der von Ebene zu Ebene voranschreitenden Codierung, bis ein Minimalwert erreicht ist, der für alle drei Lerntext-Ausschnitte in der Größenordnung von 500 bit liegt. Bei dem kurzen Text ($\widehat{=}$ „×“) ergibt sich schon nach der achten Ebene die minimale Codelänge: Wegen der speziellen Codierung (siehe oben) steigt die Codelänge auf den höheren Ebenen dann wieder an. Für den langen Text ($\widehat{=}$ „•“) benötigt man mehr als zwölf Ebenen, da der Endwert hier offenbar noch nicht erreicht ist.

Bild 8.35 zeigt die mittleren Codelängen $\overline{C}_{\text{sem}}$ der drei Lerntext-Ausschnitte (vergleiche auch mit Bild 7.5 auf Seite 72). Auf den ersten Ebenen sind die mittleren Codelängen praktisch unabhängig von der Textlänge: Beispielsweise unterschreitet man schon nach der zweiten Ebene für alle Texte den Shannon'schen Grenzwert von 1 bit/Buchstabe. Erst auf den höheren Ebenen weichen die Codelängen verschieden langer Texte voneinander ab. Je nach Textlänge werden verschiedene Minima erreicht: Für das Lesebuch ($\widehat{=}$ „×“), das aus 23 138 ASCII-Zeichen be-

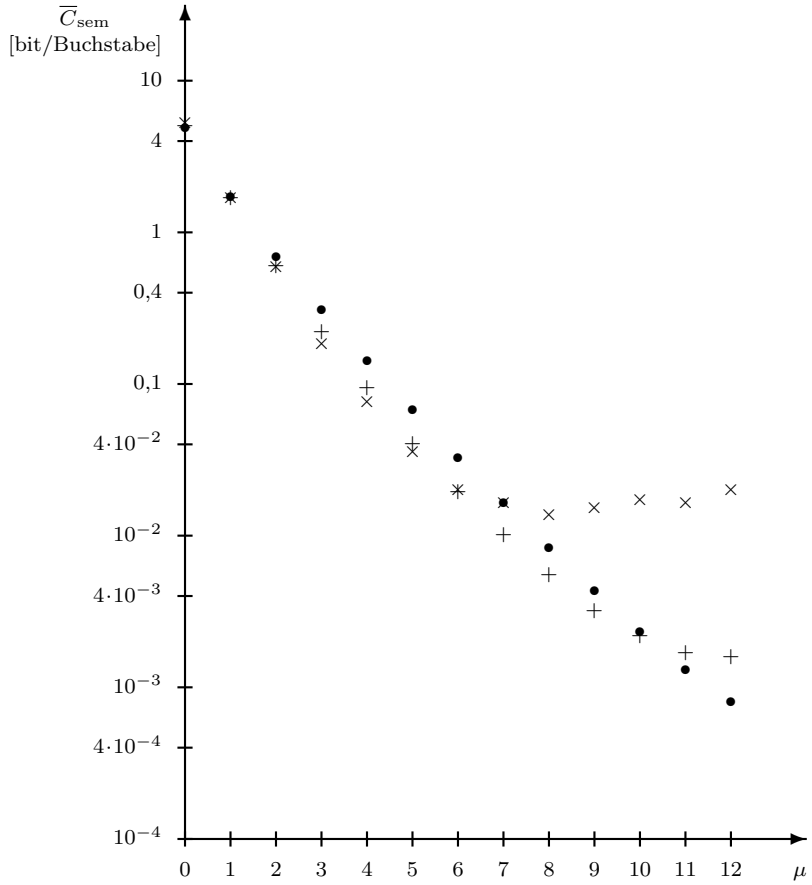

Bild 8.35: Vergleich der mittleren Codelängen $\overline{C}_{\text{sem}}$ eines langen ($\widehat{=}$ „•“), mittleren ($\widehat{=}$ „+“) und kurzen ($\widehat{=}$ „×“) Lerntext-Ausschnittes in Abhängigkeit der Anzahl μ der Ebenen, aus denen der semantische Speicher aufgebaut ist.

steht, d. h. einen Umfang von etwa acht Seiten besitzt, benötigt man mindestens 13,8 mbit/Buchstabe. Mit dem längsten der drei Lerntext-Ausschnitte ($\widehat{=}$ „•“), der etwa 500 Seiten umfaßt, ergibt sich eine mittlere Codelänge von weniger als einem Millibit pro Buchstabe, so daß der Einsatz eines semantischen Speichers für eine redundanzarme Codierung besonders bei langen Texten vorteilhaft erscheint.

In Bild 8.36 sind die mittleren Codelängen $\overline{C}_{\text{sem}}$ des kurzen Lerntext-Ausschnittes, d. h. des Lesebuchs [4] ($\widehat{=}$ „×“), des dem Lesebuch stilistisch ähnlichen Textes ($\widehat{=}$ „+“) und des fremden Textes [37] ($\widehat{=}$ „•“) gegenübergestellt:

Bei der Codierung des stilistisch ähnlichen Textes erreicht man zwar nicht die minimale mittlere Codelänge des Lesebuchs, liegt mit 139,7 mbit/Buchstabe aber dennoch deutlich unter dem Shannon'schen Grenzwert.

Dagegen versagt der semantische Speicher, wenn Texte, die zu dem Lerntext weder formal noch inhaltlich eine Beziehung haben, redundanzarm codiert werden sollen: Die Erzählung von H. Hesse [37] hätte man besser mit der universellen Optimalcodierung auf Wortebene codiert, da sich dann anstelle einer mittleren Codelänge von 4 bit/Buchstabe etwa der halbe Wert ergeben hätte (siehe Kapitel 5.5).

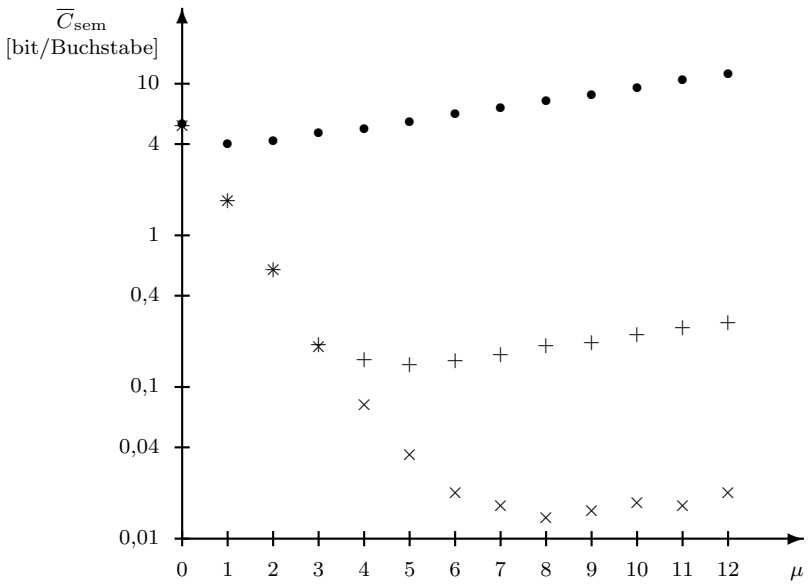

Bild 8.36: Vergleich der mittleren Codelängen $\overline{C}_{\text{sem}}$ eines bekannten ($\hat{=}$ „×“), eines unbekannten, aber stilistisch ähnlichen ($\hat{=}$ „+“) und eines völlig unbekannten ($\hat{=}$ „•“) Textes in Abhängigkeit der Anzahl μ der Ebenen des semantischen Speichers.

Nach den erfolgreich durchgeführten Simulationen konnte die Hardware eines semantischen Speichers aufgebaut werden. Der nächste Abschnitt skizziert den Aufbau und den Test des realisierten Prototyps.

8.4 Der Hardware-Aufbau eines semantischen Speichers

G. Müller und V. Nachtwey bauten ein Demonstrationsmodell eines semantischen Speichers auf, wobei sie zunächst nur die Hardware für ein FAM entwarfen [113]. Die wichtigsten Komponenten des realisierten FAM sind ein Prozessor des Typs „MC 68 000“, der alle anfallenden Steueraufgaben erledigt, Ein- und Ausgabebausteine sowie die notwendigen Speicher (EPROMs): Bei der oben beschriebenen 16-bit-Version mußten pro Wortzellen-Netz insgesamt 2 Megabyte Speicherplatz reserviert werden; aus Speicherplatzgründen wurde nun eine 14-bit-Version projektiert, bei der für das Speichern eines Wortzellen-Netzes maximal 500 Kilobyte vorgesehen sind, ein Wort und eine Adresse jeweils 14 bit und ein Schlüssel 7 bit breit ist. Der schließlich realisierte semantische Speicher besteht aus fünf Ebenen: Dazu wurden fünf FAMs verwendet.

Die Codierungsergebnisse der 16-bit- und der 14-bit-Version des semantischen Speichers wurden anhand von drei verschiedenen Texten verglichen: Der erste Text ist der längste Ausschnitt, der den Lerntexten aller Ebenen sowohl der 14-bit- als auch der 16-bit-Version gemeinsam ist. Dieser Text umfaßt 53 541 Wörter (siehe oben). Bei dem zweiten Text handelt es sich um den schon erwähnten, dem Lesebuch [4] stilistisch ähnlichen Text und bei dem dritten wieder um die Erzählung von H. Hesse [37].

Im folgenden Bild sind die mittleren Codelängen $\overline{C}_{\text{sem}}$ des Lerntext-Ausschnittes für die 16-bit- und die 14-bit-Version des semantischen Speichers gegenübergestellt.

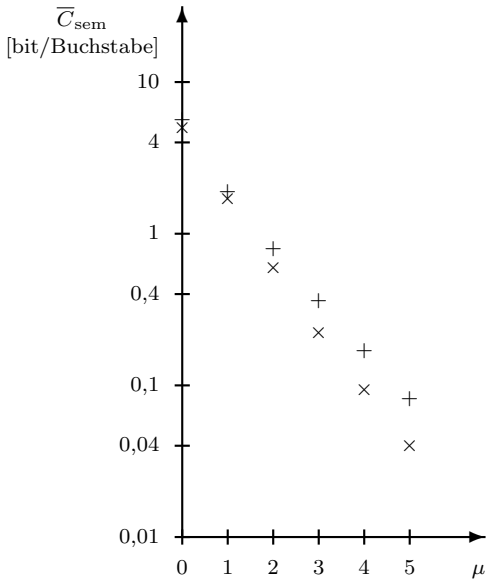

Bild 8.37: Vergleich der mittleren Codelängen $\overline{C}_{\text{sem}}$ eines Lerntext-Ausschnittes für die 16-bit-Version ($\widehat{=}$ „×“) und die 14-bit-Version ($\widehat{=}$ „+“) eines semantischen Speichers.

Wie im obigen Bild schneidet die 16-bit-Version auch in den nächsten beiden Bildern besser ab: Links sind die mittleren Codelängen des stilistisch ähnlichen Textes und rechts diejenigen des fremden Textes aufgetragen.

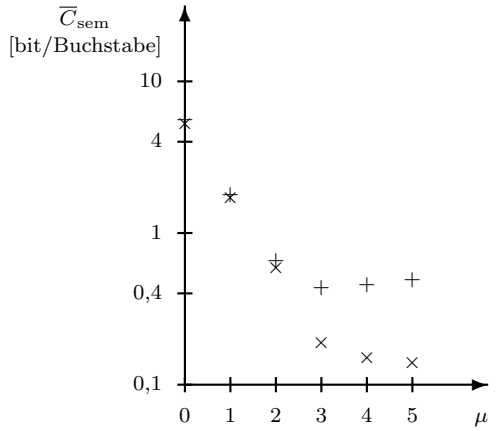

Bild 8.38: Vergleich der mittleren Codelängen $\overline{C}_{\text{sem}}$ eines dem Lerntext stilistisch ähnlichen Textes für die 16-bit-Version ($\widehat{=}$ „×“) und die 14-bit-Version ($\widehat{=}$ „+“) eines semantischen Speichers.

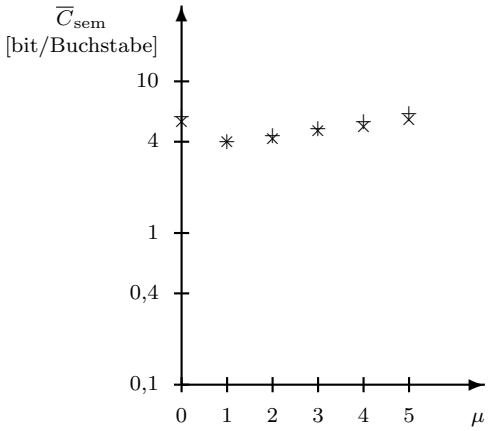

Bild 8.39: Vergleich der mittleren Codelängen $\overline{C}_{\text{sem}}$ eines fremden Textes [37] für die 16-bit-Version ($\widehat{=}$ „×“) und die 14-bit-Version ($\widehat{=}$ „+“) eines semantischen Speichers.

Besonders deutlich sind die Unterschiede der mittleren Codelängen bei dem stilistisch ähnlichen Text zu erkennen (siehe Bild 8.38): Hier bringt die Erweiterung eines semantischen Speichers von „14 bit“ auf „16 bit“ einen wesentlich redundanzärmeren Code, da wegen der größeren Anzahl der Zellen und Verbindungen seltener auf die oben erläuterte Fluchtsymbol-Codierung zurückgegriffen werden muß. Die Codierung eines dem Lerntext stilistisch ähnlichen Textes stellt obendrein den realistischen Anwendungsfall für einen semantischen Speicher dar: In der Praxis wird man weder nur Teile des Lerntextes noch dem Lerntext vollständig fremde Texte codieren, da im ersten Fall die direkte Übertragung des gesamten Lerntextes und im zweiten Fall der Einsatz von Universal- oder Optimalcodes angemessener wäre.

Bei den durchgeführten Simulationen wurden die Wortzellen-Netze während des Aufbaus und der Codierung im Arbeitsspeicher des verwendeten Rechners gehalten: Da die Netze nach der Aufbauphase nicht mehr verändert werden müssen, ist es günstiger, die Netze in einem Massenspeicher mit relativ schnellem Zugriff abzuspeichern, d. h. den Arbeitsspeicher des Codierers zu entlasten. Bei dem realisierten Demonstrationsmodell des semantischen Speichers wurden dazu verhältnismäßig teure EPROMs verwendet: Billigere Massenspeicher sind z. B. CD-ROMs, die jedoch längere Zugriffszeiten haben. Für den Aufbau großer semantischer Speicher muß man aus Kostengründen auf diese langsamen Speicher zurückgreifen.

Plant man beispielsweise die „20-bit-Version“ eines semantischen Speichers mit insgesamt 15 Ebenen, dann kann man alle Wortzellen-Netze auf einem CD-ROM speichern, wie folgende Abschätzung zeigt: Der Speicherbedarf für ein Netz vervierfacht sich ungefähr, wenn man von der 14-bit- zu der 16-bit-Version übergeht (siehe oben); das Extrapolieren dieser Werte ergibt einen Gesamtspeicherbedarf von $15 \cdot 16 \cdot 2$ Megabyte = 480 Megabyte. Bei dieser 20-bit-Version enthält ein Netz maximal $2^{20} \approx 10^6$ Zellen, die jeweils $2^{10} = 1\,024$ Verbindungen besitzen dürfen. Für den Aufbau eines solchen Codierungssystems benötigt man umfangreiche Lerntexte, die in bestimmten (in künftigen Untersuchungen noch festzulegenden) stilistischen Eigenschaften mit den später zu codierenden Texten übereinstimmen müssen. Die Tendenz, die im Bild 8.38 abzulesen ist, läßt vermuten, daß mit diesem semantischen Speicher unter realistischen Bedingungen (Codierung stilistisch ähnlicher Texte) mittlere Codelängen im Bereich von zehn Millibit pro Buchstabe erreichbar sind, d. h. der Shannon'sche Grenzwert um zwei Größenordnungen unterschritten wird (siehe dazu auch [65]).

Der in diesem Abschnitt beschriebene Prototyp eines semantischen Speichers konnte im wesentlichen wegen der oben erläuterten Strategie zum Netzaufbau realisiert werden. Neben dieser Methode existiert noch eine Reihe von Vorschlägen zur alternativen Strukturierung eines semantischen Speichers [66, 67, 68, 69, 70]: Zukünftige Untersuchungen müssen zeigen, welche dieser Alternativen zu einer Optimierung des vorgestellten Prototyps führen.

9 Zusammenfassung und Ausblick

Im Rahmen dieser Arbeit wurden zwei neue Methoden der redundanzarmen Textcodierung entwickelt und untersucht:

1. In Kapitel 5.5 wird die *universelle Optimalcodierung auf Wortebene* beschrieben, die eine Weiterentwicklung des bekannten Huffman-Codes ist, mit dem bei den klassischen Anwendungen nur Einzelbuchstaben codiert wurden.

2. In Kapitel 8.3 wird ein grundsätzlich neues Codierverfahren erläutert, das mit Hilfe eines sogenannten *semantischen Speichers* realisiert worden ist.

Bei den Untersuchungen stellte sich heraus, daß mit der universellen Optimalcodierung von Wörtern mittlere Codelängen von etwa 2 bit/Buchstabe erreicht werden können, d. h. im Vergleich zu der üblichen Textcodierung mit ASCII-Zeichen (7 bit/Buchstabe) wird hier eine Verkürzung der mittleren Codelänge um den Faktor 3,5 erzielt. Selbst wenn man den Aufwand bei der Codierung steigert (z. B. indem man ganze *Wortfolgen* optimal codiert), läßt sich mit diesem erweiterten klassischen Verfahren keine wesentlich stärkere Datenkompression realisieren.

Eine um Größenordnungen höhere Redundanzreduktion ist möglich, wenn man den von W. Hilberg vorgeschlagenen *semantischen Speicher* für eine Textcodierung einsetzt. Mit diesem neuen Speicher, der im Rahmen der vorliegenden Arbeit erstmalig simuliert wurde, sind bei der Codierung von Texten mittlere Codelängen in der Größenordnung von einigen Millibit pro Buchstabe erreichbar. Dies ermöglicht z. B. eine sehr schnelle Übertragung großer Textmengen. Der semantische Speicher wurde mit Hilfe einer Hierarchie aus Sprachnetzwerken aufgebaut, in der das für eine Textcodierung notwendige „Sprachwissen“ gespeichert ist. Nach den erfolgreich durchgeführten Simulationen konnte schließlich ein Demonstrationsmodell des semantischen Speichers aufgebaut werden, das die technische Realisierbarkeit dieses neuen Speichertyps verdeutlicht.

Neben der hier gewählten Anwendung könnte ein semantischer Speicher, wie anfangs erwähnt, auch für andere Zwecke eingesetzt werden, da die neue Speicherstruktur nicht nur speziell für Texte, sondern allgemein für sequentielle Signale mit starken inneren Bindungen entworfen wurde: Weitere Untersuchungen werden zeigen, inwieweit sich semantische Speicher beispielsweise bei der Spracherkennung oder der Bildverarbeitung durchsetzen. Auf der anderen Seite kann die neue Speicherorganisation möglicherweise auch für Aufgaben eingesetzt werden, die bisher mit Hilfe der künstlichen Intelligenz gelöst werden sollen: Zum Beispiel könnte man damit das Problem angehen, Texte automatisch von einer Sprache in eine andere zu übersetzen.

Anhang: Der Code der vorliegenden Arbeit

Anhand des Textmaterials dieser Dissertation wurde nach Kapitel 8.2 ein semantischer Speicher aufgebaut, der aus 12 Ebenen besteht. Mit dieser Speicherhierarchie konnte die gesamte Arbeit einschließlich der Formatierungskommandos des Textverarbeitungssystems sowie der Gleichungen und Bilder mit 65 bit codiert werden. In der folgenden Tabelle sind die Ergebnisse der Codierung wiedergegeben:

Textlänge (Klartext)	:	503 627	ASCII-Zeichen
Textlänge (Klartext)	:	38 548	Wörter
Codelänge	:	65	bit
Mittlere Codelänge	:	129,1	μbit/Buchstabe
Mittlere Codelänge	:	1 686,2	μbit/Wort
Mittlere Codelänge	:	7 748,1	Buchstaben/bit
Mittlere Codelänge	:	593,0	Wörter/bit
Mittlere Codelänge	:	1,8	Seiten/bit

Der Code des gesamten vorliegenden Werkes entspricht der Bitfolge

$$01000010100110100101110110001000010000000100001000000000000000000.$$

Insgesamt existieren $2^{65} \approx 3{,}7 \cdot 10^{19}$ verschiedene Bitfolgen, die 65 bit lang sind, d. h. man könnte 36 893 488 147 419 103 231 „Zufalls-Dissertationen“ generieren. Nur mit dem obigen Code wird die vorliegende, „echte“ Arbeit decodiert.*

* Auch *diese* Seite wird mit der angegebenen Bitfolge erzeugt!

Verzeichnis der verwendeten Formelzeichen

$a(n)$ Absolute Häufigkeit des n-ten Wortes

$a(m|n)$ Absolute Häufigkeit, mit der das m-te Wort im Text neben dem n-ten steht

$a^*(n)$ Absolute Häufigkeit der n-ten Zelle

$a_f(n)$ Absolute Häufigkeit der n-ten Folge aus f Wörtern

$a_\mathrm{j}(n)$ Joos'sche Näherung der Häufigkeitsverteilung von Wörtern

$a_\mathrm{m}(n)$ Mandelbrot'sche Näherung der Häufigkeitsverteilung von Wörtern

$a_\mathrm{n}(m|n)$ Absolute Häufigkeit, mit der das m-te Wort im Text nach dem n-ten steht

$a_\mathrm{paar}(n)$ Absolute Häufigkeit des n-ten Wortpaares

$a_\mathrm{v}(m|n)$ Absolute Häufigkeit, mit der das m-te Wort im Text vor dem n-ten steht

$a_\mathrm{vier}(n)$ Absolute Häufigkeit der n-ten Folge aus vier Wörtern

A Adresse

$\mathcal{A}$ Alphabet

$A^*(W)$ Hash-Code des Wortes W

$C(n)$ Länge des Codes für das Zeichen Z_n

C_sem Absolute Codelänge bei der semantischen Codierung

$\overline{C}$ Mittlere Codelänge

$\overline{C}_\mathrm{ass}$ Mittlere Länge der Assoziativcodes

$\overline{C}_\mathrm{bin}$ Mittlere Länge der Binärcodes

$\overline{C}_\mathrm{sem}$ Mittlere Codelänge bei der semantischen Codierung

$\overline{C}_\mathrm{opt}$ Mittlere Länge der Optimalcodes auf Wortebene

$\overline{C}_\mathrm{uni}$ Mittlere Länge der universellen Optimalcodes auf Wortebene

D Anzahl der Lese-Durchgänge

f Länge einer Wortfolge

$\overline{f}$ Mittlere Anzahl der Wörter pro Zelle

f_max Maximale Anzahl der Wörter pro Zelle

$g(u)$ Funktion von u

$g^*(u)$ Funktion von u

H Mittlerer Informationsgehalt (Entropie)

H_g Von C. E. Shannon geschätzter, unterer Grenzwert für die Text-Entropie

H_h Von W. Hilberg berechneter, unterer Grenzwert für die Text-Entropie

H_k Entropie von Texten, bei der man alle Bindungen innerhalb von Ketten aus k Buchstaben berücksichtigt

H_w Entropie der Wörter

H_0 Entropie der Buchstaben ohne Berücksichtigung ihrer Häufigkeitsverteilung

H_1 Entropie der Buchstaben mit Berücksichtigung ihrer Häufigkeitsverteilung

i Index

I Nutzinformation

$I(n)$ Informationsgehalt des Zeichens Z_n

j Index

k Index

$\mathcal{K}$ Menge der Konsonanten

ℓ Wortlänge

$\overline{\ell}$ Mittlere Wortlänge

m Ordnungszahl

m_k Index für die m-te Kette aus k Symbolen

M Anzahl der verschiedenen Symbolketten

n Ordnungszahl

n_f Anzahl der verschiedenen Folgen aus f Wörtern

n_{paar} Anzahl der verschiedenen Wortpaare

n_0 Anzahl der verschiedenen Wörter eines Textes (Wortvorrat)

N Anzahl der Symbole eines Alphabets

N_{f} Anzahl der Wortfolgenzellen

N_{w} Anzahl der Wortzellen

$p(n)$ Relative Häufigkeit des n-ten Wortes oder Buchstabens

$p(m,n)$ Relative Häufigkeit, mit der die Kombination des m-ten und des n-ten Symbols auftritt

$p(n|m)$ Bedingte Wahrscheinlichkeit, mit der das n-te Symbol auf das m-te folgt

P Größe des Adreßbereichs (Primzahl)

$\mathcal{P}$ Auftrittshäufigkeiten eines Alphabets

Q Nachrichtenquelle

R Redundanz

s Anzahl der Wörter eines Textes (Textlänge)

S Schlüssel für die Streuspeicherung

$\mathcal{S}$ Menge der Sonderfallkonsonanten

u Variable

$\ddot{U}(W)$ Platznummer des Wortes W in der Überlaufliste

$v(n)$ Verzweigungsgrad des n-ten Wortes

$v^*(n)$ Verbindungsanzahl der n-ten Zelle

$\overline{v}$ Mittlere Verbindungsanzahl pro Zelle

$v_f(n)$ Verzweigungsgrad der n-ten Folge aus f Wörtern

$v_i(n)$ Die i-te Näherung des Verzweigungsgrades $v(n)$

$v_{\max}$ Maximal zugelassene Anzahl der Verbindungen pro Zelle

$v_{\text{n}}(n)$ Nachfolgeranzahl des n-ten Wortes

$v_{\text{v}}(n)$ Vorgängeranzahl des n-ten Wortes

$\mathcal{V}$ Menge der Vokale

W Wort

$W[i]$ Der i-te Buchstabe des Wortes W

x Schloßwort

X Externes Schlüsselwort

y Schloßwort

Y Verkettungsschlüsselwort

Z_n Zeichen eines Alphabets

$\alpha(n)$ Negative Geradensteigung der Nachfolgerverteilung des n-ten Wortes

$\overline{\alpha}$ Negativer Mittelwert der Geradensteigungen der Nachfolgerverteilungen

$\beta(n)$ Negative Geradensteigung der Vorgängerverteilung des n-ten Wortes

$\overline{\beta}$ Negativer Mittelwert der Geradensteigungen der Vorgängerverteilungen

δ Konstante für die Mandelbrot'sche Näherung

ε Konstante für die Joos'sche Näherung

ϑ Negative Geradensteigung der Häufigkeitsverteilung der Wortpaare

μ Anzahl der Ebenen eines semantischen Speichers

σ Füllungsgrad einer Übergangsmatrix

$\varphi(f)$ Negative Geradensteigung der Häufigkeitsverteilung der Folgen aus f Wörtern

ψ Negative Geradensteigung der Verzweigungsgradverteilung

Literaturverzeichnis

[1] Schaeder, B.:
Maschinenlesbare Textkorpora des Deutschen und des Englischen.
Deutsche Sprache (1976) 4, S. 356–370.

[2] LIMAS-Korpus. Deutscher Standardtext (ca. 1 Mio. Wortformen).
Universität Bonn, Institut für Kommunikationsforschung und Phonetik, 1970.

[3] Lenders, W.; Willée, G.:
Linguistische Datenverarbeitung.
Opladen: Westdeutscher Verlag, 1986.

[4] Herrndobler, I.; Gick, G.:
Ich und Du — Du und Ich. Lesefibel für die zweite Hälfte des ersten Schuljahres.
List Verlag, 1976.

[5] DUDEN Band 1, Rechtschreibung der deutschen Sprache und der Fremdwörter.
Mannheim · Wien · Zürich: Bibliographisches Institut, 1980.

[6] Zipf, G. K.:
Human Behavior and the Principle of Least Effort — An Introduction to Human Ecology.
Reading, Massachusetts: Addison-Wesley Press, 1949;
New York: Hafner Publishing Company, 1972 (Facsimile).

[7] Wagner, S. W.:
Automatische Stichwortanalyse nach dem Rangkriterienverfahren.
TH Karlsruhe, Dissertation (1966).

[8] Guiter, H.; Arapov, M. V.:
Studies on Zipf's Law.
Bochum: Studienverlag Brockmeyer, 1982.

[9] Shannon, C. E.; Weaver, W.:
Mathematische Grundlagen der Informationstheorie.
München · Wien: R. Oldenbourg Verlag, 1976.

[10] Shannon, C. E.:
A Mathematical Theory of Communication.
The Bell System Technical Journal 27 (1948) 3, S. 379–423;
The Bell System Technical Journal 27 (1948) 4, S. 623–656.

[11] Shannon, C. E.:
Prediction and Entropy of Printed English.
The Bell System Technical Journal 30 (1951) 1, S. 50–64.

[12] Endres, W.:
A Comparison of Redundancy in Written and Spoken Language.
Tsakadsor, Armenien: 2nd International Symposium on Information Theorie, 1971, S. 53–59.

[13] Küpfmüller, K.:
Die Entropie der deutschen Sprache.
Fernmeldetechnische Zeitschrift 7 (1954) 6, S. 265–272.

[14] Bronstein, I. N.; Semendjajew, K. A.:
Taschenbuch der Mathematik.
Zürich · Frankfurt/Main · Thun: Verlag Harri Deutsch, 1977.

[15] Steinbuch, K.:
Taschenbuch der Nachrichtenverarbeitung.
Berlin · Göttingen · Heidelberg: Springer-Verlag, 1962.

[16] Siemens:
BS1000 — BS2000 — TRANSDATA PDN — Systemkonventionen.
München: Siemens AG, 1984.

[17] Digital Equipment Corporation:
VAX Text Processing — Utility Reference Manual.
Maynard, Massachusetts: DEC, 1985.

[18] Huffman, D. A.:
A Method for the Construction of Minimum-Redundancy Codes.
Proceedings of the I.R.E. 40 (1952) 9, S. 1098–1101.

[19] Fano, R. M.:
Informationsübertragung — Eine statistische Theorie der Nachrichtenübertragung.
München · Wien: R. Oldenbourg Verlag, 1966.

[20] Meyer-Eppler, W.:
Grundlagen und Anwendungen der Informationstheorie.
Berlin · Heidelberg · New York: Springer-Verlag, 1969.

[21] Wirth, N.:
Algorithmen und Datenstrukturen.
Stuttgart: B. G. Teubner, 1975.

[22] Herschel, R.; Pieper, F.:
PASCAL und PASCAL-Systeme — Systematische Darstellung für den Anwender.
München · Wien: R. Oldenbourg Verlag, 1985.

[23] Barth, W.; Nirschl, H.:
Sichere sinnentsprechende Silbentrennung für die deutsche Sprache.
Angewandte Informatik (1985) 4, S. 152–159.

[24] DUDEN Band 5, Fremdwörterbuch.
Mannheim · Wien · Zürich: Bibliographisches Institut, 1982.

[25] Knuth, D. E.:
The TEXbook.
Reading, Massachusetts etc.: Addison-Wesley Publishing Company, 1984.

[26] Lamport, L.:
LATEX — A Document Preparation System.
Reading, Massachusetts etc.: Addison-Wesley Publishing Company, 1986.

[27] Jessen, E.:
Assoziative Speicherung.
Braunschweig: F. Vieweg & Sohn, 1965.

[28] Kohonen, T.:
Content-Adressable Memories.
Berlin · Heidelberg · New York: Springer-Verlag, 1980.

[29] Gall, R.; Nagl, M.:
Software-Implementation assoziativer Speicher.
Elektronische Rechenanlagen 23 (1981) 2, S. 61–71.

[30] Stevens, L.:
Auf der Suche nach der künstlichen Intelligenz — Wege zur perfekten Maschine.
Landsberg am Lech: moderne verlagsgesellschaft, 1985.

[31] Küpfmüller, K.; Jenik, F.:
Über die Nachrichtenverarbeitung in der Nervenzelle.
Kybernetik 1 (1961) 1, S. 1–6.

[32] Vester, F.:
Denken, Lernen, Vergessen.
München: Deutscher Taschenbuch Verlag, Sachbuch Nr. 1327, 1975.

[33] Braitenberg, V.:
Gehirngespinste — Neuroanatomie für kybernetisch Interessierte.
Berlin · Heidelberg · New York: Springer-Verlag, 1973.

[34] Willwacher, G.:
Fähigkeiten eines assoziativen Speichersystems im Vergleich zu Gehirnfunktionen.
Biological Cybernetics 24 (1976) 1, S. 181–198.

[35] Palm, G.:
Assoziative Informationsspeicherung in Nervennetzen.
In: Marhold, G.: Künstliche Intelligenz.
Düsseldorf: VDI Verlag, S. 163–177.

[36] Zieliński, R.:
Erzeugung von Zufallszahlen — Programmierung und Test auf Digitalrechnern.
Thun · Frankfurt/Main: Verlag Harri Deutsch, 1978.

[37] Hesse, H.:
Peter Camenzind.
Suhrkamp Taschenbuch Verlag, Taschenbuch Nr. 161, 1953.

[38] Hughes, J. K.; Michtom, J. I.:
Strukturierte Software-Herstellung — Anleitung für Programmierer und Manager.
München · Wien: R. Oldenbourg Verlag, 1980.

[39] Hilberg, W.:
Elektronische digitale Speicher.
München · Wien: R. Oldenbourg Verlag, 1975.

[40] Hilberg, W.:
Digitale Speicher 1.
München · Wien: R. Oldenbourg Verlag, 1987.

[41] Hilberg, W.:
Assoziative Gedächtnisstrukturen, funktionale Komplexität — Neue Forschungen auf dem Gebiet der digitalen Systeme.
München · Wien: R. Oldenbourg Verlag, 1984.

[42] Hilberg, W.:
Ist die Entropie als Informationsmaß schon der Weisheit letzter Schluß?
Elektronische Rechenanlagen 26 (1984) 6, S. 281–289.

[43] Hilberg, W.:
Information ohne Redundanz — Von den Markoffschen Ketten zu vollständigen Beziehungsnetzen.
FREQUENZ 40 (1986) 1, S. 18–24.

[44] Hilberg, W.:
Das assoziative Feld — Eine neue Schaltungsstruktur mit überaus guten Speichereigenschaften.
Elektronische Rechenanlagen 26 (1984) 4, S. 167–178.

[45] Hilberg, W.:
Semantische Speicherung von Texten.
AEÜ 40 (1986) 4, S. 225–228.

[46] Hilberg, W.:
Das Netzwerk der menschlichen Sprache und Grundzüge einer entsprechend gebauten Sprachmaschine.
ntzArchiv 10 (1988) 6, S. 133–146.

[47] Hilberg, W.; Meyer, J.:
Zur effizienten Speicherung von Sprache.
In: Digitale Speicher — ITG-Fachbericht 102.
Berlin · Offenbach: vde-verlag, 1988, S. 421–430.

[48] Hilberg, W.:
Die Äquivalenz von Information und Komplexität.
TH Darmstadt, Institut für Datentechnik, Institutsbericht Nr. 39 (1985).

[49] Hilberg, W.:
Ist eine deterministische Informationstheorie vorstellbar?
TH Darmstadt, Institut für Datentechnik, Institutsbericht Nr. 45 (1986).

[50] Hilberg, W.:
Über die Notwendigkeit und Möglichkeit einer Anpassung der Speicherstruktur an die Datenstruktur.
TH Darmstadt, Institut für Datentechnik, Institutsbericht Nr. 56 (1986).

[51] Hilberg, W.:
Verkettungen in assoziativen Feldern I.
TH Darmstadt, Institut für Datentechnik, Institutsbericht Nr. 29 (1984).

[52] Hilberg, W.:
Verkettungen in assoziativen Feldern II.
TH Darmstadt, Institut für Datentechnik, Institutsbericht Nr. 30 (1985).

[53] Hilberg, W.:
Die Speicherung der Wortverbindungen in der Übergangsmatrix durch verkettete Wörterbücher.
TH Darmstadt, Institut für Datentechnik, Institutsbericht Nr. 38 (1985).

[54] Hilberg, W.:
Möglichkeiten der Kontextfestlegung, dargestellt in der Übergangsmatrix.
TH Darmstadt, Institut für Datentechnik, Institutsbericht Nr. 58 (1987).

[55] Hilberg, W.:
Verkettung mit Codewandlung.
TH Darmstadt, Institut für Datentechnik, Institutsbericht Nr. 89 (1988).

[56] Hilberg, W.:
Semantische Speicher.
TH Darmstadt, Institut für Datentechnik, Institutsbericht Nr. 54 (1986).

[57] Hilberg, W.:
Gelten auch für die semantische Verdichtung von Texten die Grenzwerte von Shannon und Küpfmüller?
TH Darmstadt, Institut für Datentechnik, Institutsbericht Nr. 50 (1986).

[58] Hilberg, W.:
Die Eignung des semantischen Speichers für wissensbasierte Systeme.
TH Darmstadt, Institut für Datentechnik, Institutsbericht Nr. 52 (1986).

[59] Hilberg, W.:
Die Sprachmaschine aus Netzwerken kann mehr als nur effizient speichern!
TH Darmstadt, Institut für Datentechnik, Institutsbericht Nr. 100 (1988).

[60] Hilberg, W.:
Der semantische Speicher als Schlüssel zum Verständnis von Denkvorgängen?
TH Darmstadt, Institut für Datentechnik, Institutsbericht Nr. 41 (1986).

[61] Hilberg, W.:
Zur effizienten Speicherung von Sprache.
TH Darmstadt, Institut für Datentechnik, Institutsbericht Nr. 91 (1988).

[62] Hilberg, W.:
Assoziativ gesteuerte Verbindungen in einem Zellen-Netz. (Erweiterungen des assoziativen Feldes).
TH Darmstadt, Institut für Datentechnik, Institutsbericht Nr. 70 (1987).

[63] Hilberg, W.:
Ein technisches Netzmodell für die Erzeugung und Speicherung menschlicher Sprache.
TH Darmstadt, Institut für Datentechnik, Institutsbericht Nr. 73 (1987).

[64] Hilberg, W.:
Was ist eigentlich Denken? — Das Durchschreiten eines simplen Sprachnetzwerkes?
TH Darmstadt, Institut für Datentechnik, Institutsbericht Nr. 81 (1987).

[65] Hilberg, W.:
Ist die semantische Komprimierung von Sprache ein realistisches Konzept?
TH Darmstadt, Institut für Datentechnik, Institutsbericht Nr. 95 (1988).

[66] Hilberg, W.:
Assoziatives Feld mit variabler Verkettung.
TH Darmstadt, Institut für Datentechnik, Institutsbericht Nr. 36 (1985).

[67] Hilberg, W.:
Zur redundanzfreien Kodierung von Sprache.
TH Darmstadt, Institut für Datentechnik, Institutsbericht Nr. 64 (1987).

[68] Hilberg, W.:
Sprachnetzwerk mit Zwischenspeicherung und zentraler Steuerung.
TH Darmstadt, Institut für Datentechnik, Institutsbericht Nr. 94 (1988).

[69] Hilberg, W.:
Strukturierung des semantischen Speichers.
TH Darmstadt, Institut für Datentechnik, Institutsbericht Nr. 92 (1988).

[70] Hilberg, W.:
Syntaktische und semantische Speicher.
TH Darmstadt, Institut für Datentechnik, Institutsbericht Nr. 97 (1988).

[71] Meyer, J.:
Gilt das Zipfsche Gesetz auch für die chinesische Schriftsprache?
ntzArchiv 11 (1989) 1, S. 13–16.

[72] Meyer, J.:
Codierung eines großen Wortschatzes mit Hilfe des Huffman-Code.
FREQUENZ 41 (1987) 11/12, S. 305–309.

[73] Meyer, J.:
Zusammenfassung der statistischen Untersuchungen des LIMAS-Korpus.
TH Darmstadt, Institut für Datentechnik, Institutsbericht Nr. 43 (1986).

[74] Meyer, J.:
Gilt das Zipf'sche Gesetz auch im Chinesischen?
TH Darmstadt, Institut für Datentechnik, Institutsbericht Nr. 88 (1988).

[75] Meyer, J.:
Die Codelänge des Huffman-Codes für sehr große Vokabulare.
TH Darmstadt, Institut für Datentechnik, Institutsbericht Nr. 61 (1987).

[76] Meyer, J.:
Das Zipf'sche Gesetz für die direkten Vorgänger- und Nachfolger-Wörter.
TH Darmstadt, Institut für Datentechnik, Institutsbericht Nr. 69 (1987).

[77] Meyer, J.:
Untersuchungen der statistischen Verteilung von häufigen Wortfolgen anhand eines deutschen Standardtextes.
TH Darmstadt, Institut für Datentechnik, Institutsbericht Nr. 48 (1986).

[78] Meyer, J.:
Die Häufigkeitsverteilungen von Wortfolgen.
TH Darmstadt, Institut für Datentechnik, Institutsbericht Nr. 79 (1987).

[79] Meyer, J.:
Redundanzarme Textspeicherung mit Hilfe von assoziativen Wörterbüchern.
TH Darmstadt, Institut für Datentechnik, Institutsbericht Nr. 37 (1985).

[80] Meyer, J.:
Optimalcodierung von Texten auf Wortebene.
TH Darmstadt, Institut für Datentechnik, Institutsbericht Nr. 47 (1986).

[81] Meyer, J.:
Erste praktische Erfahrungen mit einem semantischen Speicher.
TH Darmstadt, Institut für Datentechnik, Institutsbericht Nr. 72 (1987).

[82] Meyer, J.:
Berechnungen zur assoziativen Text-Codierung.
TH Darmstadt, Institut für Datentechnik, Institutsbericht Nr. 63 (1987).

[83] Meyer, J.:
Simulation eines Wortzellen-Netzes.
TH Darmstadt, Institut für Datentechnik, Institutsbericht Nr. 80 (1987).

[84] Meyer, J.:
Zur effizienten Speicherung von Sprache.
TH Darmstadt, Institut für Datentechnik, Institutsbericht Nr. 90 (1988).

[85] Meyer, J.:
Die Programme zu der Dissertation „Die Verwendung hierarchisch strukturierter Sprachnetzwerke zur redundanzarmen Codierung von Texten“.
TH Darmstadt, Institut für Datentechnik, Institutsbericht Nr. 110 (1989).

[86] Färber, H.-J.:
Installation eines Programmsystems auf verschiedenen Rechnern zur Erforschung von assoziativen Speichern.
TH Darmstadt, Institut für Datentechnik, Diplomarbeit DS 331 (1988).

[87] Herrigel, R.; Kottenbrink, J.:
Entwicklung eines komfortablen Programm-Pakets zur Klausurdatenverarbeitung.
TH Darmstadt, Institut für Datentechnik, Studienarbeit DS 289 (1987).

[88] Böhnke, M.; Helbig, J.:
Untersuchung von selbstorganisierenden Systemen zur Flugbahnbestimmung von schnellbewegten Objekten.
TH Darmstadt, Institut für Datentechnik, Studienarbeit DS 330 (1988).

[89] Höfling, T.:
Untersuchung von Markoff-Ketten auf Wortebene.
TH Darmstadt, Institut für Datentechnik, Diplomarbeit DS 303 (1987).

[90] Erler, T.:
Untersuchung formaler grammatikalischer Zusammenhänge für die assoziative Abspeicherung großer Texte.
TH Darmstadt, Institut für Datentechnik, Diplomarbeit DS 244 (1985).

[91] Schmitt, D.:
Auswahl eines geeigneten Abstandsmaßes für Wortformen.
TH Darmstadt, Institut für Datentechnik, Diplomarbeit DS 304 (1987).

[92] Lauth, M.:
Auswahl eines Abstandsmaßes für Wortformen unter Verwendung von Silben.
TH Darmstadt, Institut für Datentechnik, Diplomarbeit DS 336 (1988).

[93] Rüdiger, W.:
Entwurf eines baumstrukturierten Wörterbuchs mit schnellem Zugriff.
TH Darmstadt, Institut für Datentechnik, Diplomarbeit DS 246 (1985).

[94] Herrmann, C.:
Entwurf und Untersuchung verschiedener Hash-Funktionen für die Verwendung in einem elektronischen Wörterbuch.
TH Darmstadt, Institut für Datentechnik, Diplomarbeit DS 258 (1986).

[95] Müller, H.; Passinger, B.:
Entwicklung der Hardware für ein elektronisches Wörterbuch.
TH Darmstadt, Institut für Datentechnik, Studienarbeit DS 283 (1988).

[96] Heidemann, S.:
Redundanzreduktion von Texten mittels Optimalcodierung der Worte.
TH Darmstadt, Institut für Datentechnik, Diplomarbeit DS 253 (1986).

[97] Sauerbrey, J.:
Entwicklung einer Methode zur dynamischen Optimalcodierung.
TH Darmstadt, Institut für Datentechnik, Diplomarbeit DS 282 (1987).

[98] Lüssem, A.:
Simulation eines Text-Codierungs-Systems unter Verwendung von Optimalcodes und assoziativen Verknüpfungen sehr häufiger Wörter.
TH Darmstadt, Institut für Datentechnik, Diplomarbeit DS 299 (1987).

[99] Böttger, M.:
Text-Codierung unter Ausnutzung der Vorgänger- und Nachfolger-Verteilungen von Wörtern.
TH Darmstadt, Institut für Datentechnik, Diplomarbeit DS 308 (1988).

[100] Langer, U.:
Entwicklung und Untersuchung eines Verfahrens zur Optimalcodierung ganzer Wortketten.
TH Darmstadt, Institut für Datentechnik, Diplomarbeit DS 284 (1987).

[101] Gümpel, N.:
Behandlung von Spezialfällen bei der Optimal-Codierung von Texten.
TH Darmstadt, Institut für Datentechnik, Diplomarbeit DS 267 (1987).

[102] Sacher, R.:
Untersuchung von Verfahren zur Redundanz-Reduktion von Texten mit Hilfe der Codierung von Silben.
TH Darmstadt, Institut für Datentechnik, Diplomarbeit DS 286 (1987).

[103] Müller, H.:
Untersuchung eines Sprachnetzes unter Verwendung von Silben.
TH Darmstadt, Institut für Datentechnik, Diplomarbeit DS 340 (1988).

[104] Sallanz, J.:
Implementierung eines universellen Codierungssystems für Texte mit Hilfe der Codierung von Silben.
TH Darmstadt, Institut für Datentechnik, Diplomarbeit DS 285 (1988).

[105] Sturm, M.; Engel, K.:
Untersuchung assoziativer Strukturen für die redundanzarme Abspeicherung von großen Textmengen.
TH Darmstadt, Institut für Datentechnik, Studienarbeit DS 208 (1986).

[106] El-Chami, M.; Schmitt, D.:
Realisierung eines Textspeichers unter Verwendung von Optimal-Codes und Assoziations-Listen seltener Wörter.
TH Darmstadt, Institut für Datentechnik, Studienarbeit DS 263 (1987).

[107] Hantschel, A.; Küpper, M.:
Entwicklung einer Methode zur assoziativen Textcodierung mit Hilfe von Übergangsmatrizen.
TH Darmstadt, Institut für Datentechnik, Studienarbeit DS 149 (1988).

[108] Anastasiu, C.:
Realisierung eines assoziativen Feldes mit Hilfe eines Wortzellen-Netzes.
TH Darmstadt, Institut für Datentechnik, Diplomarbeit DS 337 (1988).

[109] Burkard, R.:
Realisierung eines Textspeichers unter Verwendung eines rückgekoppelten assoziativen Wörterbuches.
TH Darmstadt, Institut für Datentechnik, Diplomarbeit DS 300 (1987).

[110] Schneller, R.:
Elektronische Modelle für die Informationsverarbeitung im Gehirn.
TH Darmstadt, Institut für Datentechnik, Studienarbeit DS 295 (1988).

[111] Beyer, S.; Lüth, T.:
Simulation von Gehirnstrukturen anhand von einfachen Modellen.
TH Darmstadt, Institut für Datentechnik, Studienarbeit DS 301 (1988).

[112] Bolz, S.; Spieß, M.:
Simulation eines semantischen Speichers.
TH Darmstadt, Institut für Datentechnik, Studienarbeit DS 317 (1988).

[113] Müller, G.; Nachtwey, V.:
Hardware-Aufbau eines semantischen Speichers.
TH Darmstadt, Institut für Datentechnik, Studienarbeit DS 302 (1989).

[114] Zanella, R.:
Untersuchung von Verfahren der assoziativen Text-Codierung unter Berücksichtigung mehrerer Vorgängerwörter.
TH Darmstadt, Institut für Datentechnik, Diplomarbeit DS 368 (1989).

Jochen Meyer

Schloßgartenstr. 47
D-6100 Darmstadt

Lebenslauf

Persönliche Daten:

19. 2. 1960	Geboren in Berlin als Sohn der Eheleute Rudolf Meyer, Dipl.-Ing., und Roswitha Meyer, geb. Schreier, Ingenieur-Assistentin

Ausbildung und beruflicher Werdegang:

19. 4. 1966 – 16. 7. 1969	Grundschule (Carl-Ulrich-Schule in Darmstadt-Arheilgen)
2. 9. 1969 – 12. 6. 1978	Altsprachliches Gymnasium (Ludwig-Georgs-Gymnasium in Darmstadt)
12. 6. 1978	Abitur
1. 10. 1978 – 1. 8. 1984	Studium der Elektrotechnik an der Technischen Hochschule Darmstadt, Fachrichtung Nachrichtentechnik
12. 6. 1978 – 21. 12. 1982	Insgesamt 26 Wochen Industriepraktikum bei – Carl Schenck AG Darmstadt, – GSI mbH Darmstadt, – Siemens AG, ZN Frankfurt
11. 4. 1983 – 31. 7. 1984	Studien- und Diplomarbeit am Forschungsinstitut der Deutschen Bundespost in Darmstadt
1. 8. 1984	Diplom
Seit 1. 11. 1984	Wissenschaftlicher Mitarbeiter am Institut für Datentechnik, Fachgebiet Digitaltechnik, der Technischen Hochschule Darmstadt