Datenkompression:

Kontext- und Wörterbuchtechniken

H. Fernau

email: fernau@uni-trier.de

SoSe 2013

Universität Trier

Die bedingte Entropie

- Seien A und B zwei Ereignisse und $P(A|B)$ die bedingte Wahrscheinlichkeit, dass A eintritt unter der Bedingung, dass auch B eintritt.
- Bedingter Informationsgehalt $i(A|B) = \log_2 \frac{1}{P(A|B)}$.
- Sei S eine fixierte Informationsquelle und T sei ein Kontext, in dem die Nachrichten aus S vorkommen.
- Die bedingte Entropie $H(S|T)$: der *mittlere Informationsgehalt für Nachrichten aus S mit dem Kontext T*:

$$H(S|T) = \sum_{t \in T} P(t) H(S|t) = \sum_{t \in T} P(t) \sum_{s \in S} P(s|t) \log_2 \frac{1}{P(s|t)}$$

- Wenn S *unabhängig* vom Kontext T ist, gilt $H(S|T) = H(S)$;
- ansonsten haben wir: $H(S|T) < H(S)$
  Die Kenntnis des Kontextes für S reduziert also die Entropie.

Markov-Modell am Beispiel: Schwarz-Weiß-Bild

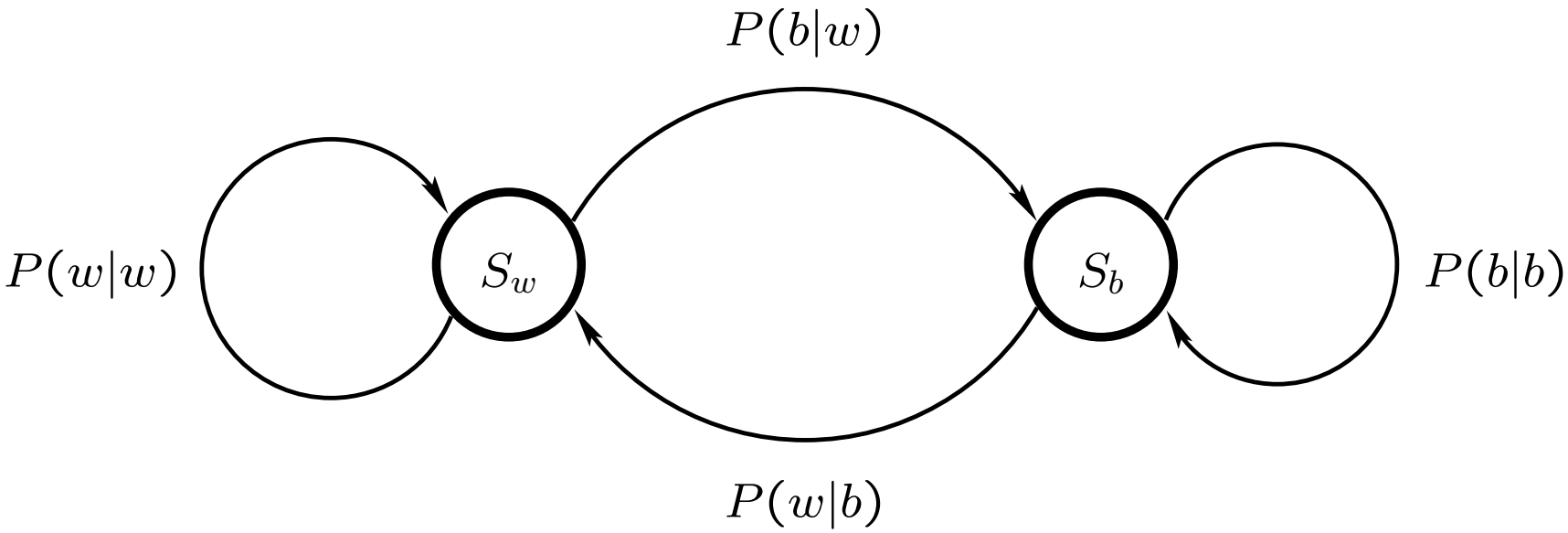
$S = \Sigma = \{w, b\}$. Sei $T = \{w, b\}$.

Entsprechende Wahrscheinlichkeiten:

$P(w|w)$ = W., ein weißes Pixel zu erhalten, wenn vorheriges Pixel weiß;
$P(w|b)$ = W., ein weißes Pixel zu erhalten, wenn vorheriges Pixel schwarz;
$P(b|b)$ = W., ein schwarzes Pixel zu erhalten, wenn vorheriges Pixel schwarz;
$P(b|w)$ = W., ein schwarzes Pixel zu erhalten, wenn vorheriges Pixel weiß.

Markov-Modell:

S_w, bzw. S_b, der Zustand, dass wir ein weißes, bzw. schwarzes Pixel lesen:



Markov-Modell – Beispiel

Nehmen wir die folgenden Wahrscheinlichkeiten an

$$\begin{array}{ll} P(w) = P(S_w) = 0.8 & P(b) = P(S_b) = 0.2 \\ P(w|w) = 0.95 & P(b|w) = 0.05 \\ P(w|b) = 0.3 & P(b|b) = 0.7 \end{array}$$

Dann ist die Entropie für $\Sigma = \{w, b\}$ gleich

$$H(\Sigma) \;=\; -0.8 \times \log 0.8 \;-\; 0.2 \times \log 0.2 \;=\; 0.7219$$

und die entsprechende bedingte Entropie:

$$\begin{aligned} H(\Sigma|\Sigma) \;&=\; 0.8 \times (-0.95 \times \log 0.95 \;-\; 0.05 \times \log 0.05) \;+ \\ &\quad\; 0.2 \times (-0.3 \times \log 0.3 \;-\; 0.7 \times \log 0.7) \\ &=\; 0.8 \times 0.2864 + 0.2 \times 0.8813 \;=\; 0.4054 \end{aligned}$$

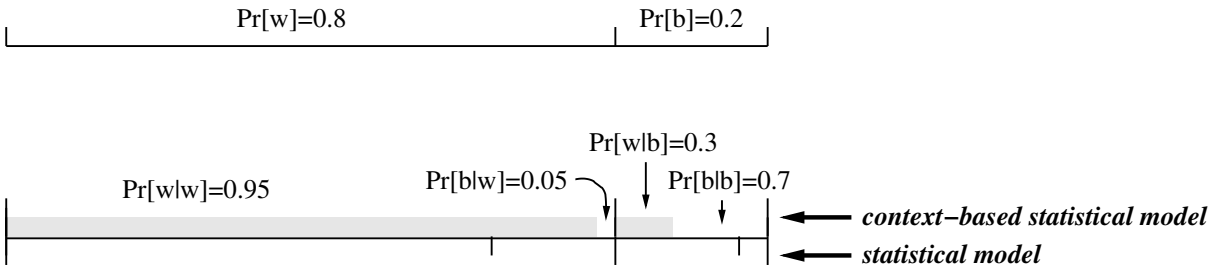
um etwa 44% besser als früher.

Markov-Modell Die Codierung/Decodierung

- Für jeden Kontext $w \in \Sigma^m$ konstruieren wir jeweils eine Codierung K_w für Σ im Kontext w.
- Dann benutzen wir K_w, um $a \in \Sigma$ im Kontext w zu codieren und zu dekodieren.
- Huffman-Codierung: ineffizient in dem Szenario.
- Frage: Wie funktioniert dieser *Ansatz* für die die arithmetische Codierung?
- Antwort: Prediction by Partial Match (ppm) (Cleary/Witten).

Prediction by Partial Match (ppm) (Cleary/Witten, 1984)

Idee: kontextabhängiges statistisches Modell.



Wir haben gesehen, dass:

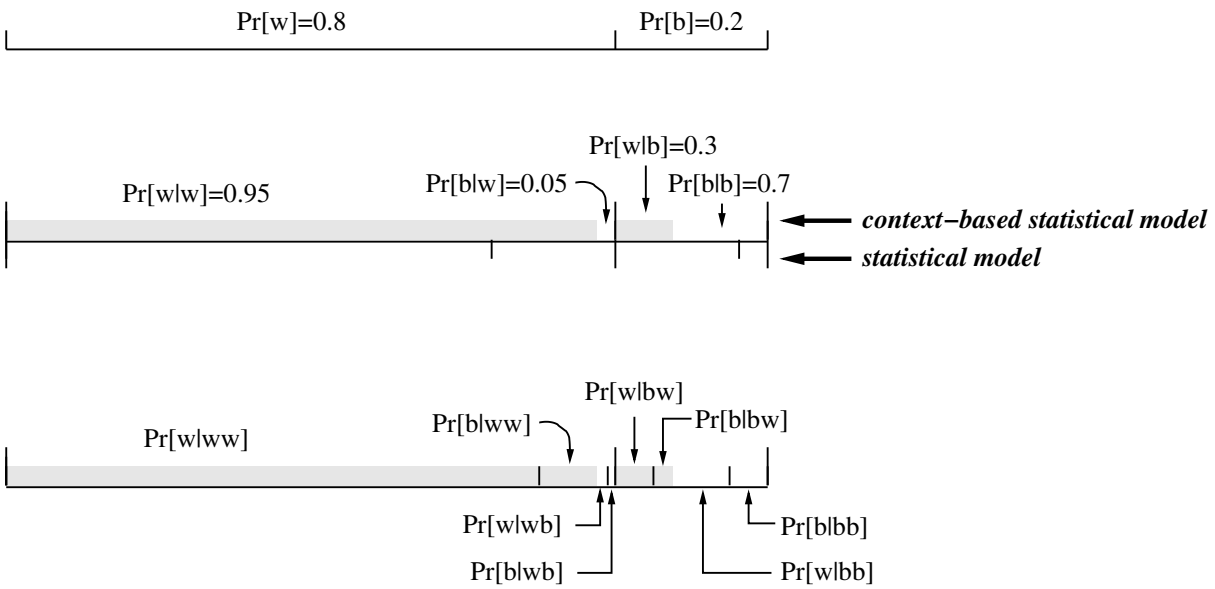
$$H(\Sigma) = 0.7219, \quad H(\Sigma|w) = 0.2864, \quad H(\Sigma|b) = 0.8813$$

und

$$\mathbb{E}(L) = 0.8 \times H(\Sigma|w) + 0.2 \times H(\Sigma|b) = 0.4054$$

Prediction by Partial Match (ppm)

Beispiel: Kontexte der Länge $0, 1, 2$.



Prediction with partial match (ppm)

ppm-Verfahren: eine adaptive arithmetische Codierung, die nicht nur einzelne Zeichen, sondern Zeichen in ihrem *Kontext* verwendet.

Adaption: bisher beobachtete relative Häufigkeiten werden als Grundlage für den Entwurf des *Wahrscheinlichkeitsmodells* genommen.

Problem Zeichen $a \in \Sigma$ im Kontext w mit bisher beobachtete relative Häufigkeit gleich *Null*!

Codierung

- Zunächst versuche, möglichst lange Kontexte anzunehmen (ein Wahrscheinlichkeitsmodell höchster Ordnung).

- Schlägt diese Annahme fehl, geht der Codierer auf das Wahrscheinlichkeitsmodell nächstniederer Ordnung über. Diesen Übergang signalisiert er dem Empfänger durch Übermittlung eines `Escape`-Zeichens.

- Grundmodell (der Ordnung -1): Zeichen des Grundalphabets wird dieselbe Wahrscheinlichkeit zugesprochen.

Beispiel 1 *Betrachten wir die Codierung des folgenden Satzes über dem Alphabet* $\{a, d, l, r, s, \_\}$*:*

$$lara\_lass\_das\_da$$

Das Wahrscheinlichkeitsmodell (-1)*-ter Ordnung (***nicht-adaptiv***):*

$$\forall x \in \{a, d, l, r, s, \_\} \text{ nimm } P(x) = 1/6.$$

Das W-keitsmodell nullter und erster Ordnung nach dem Einlesen von $lara\_lass$

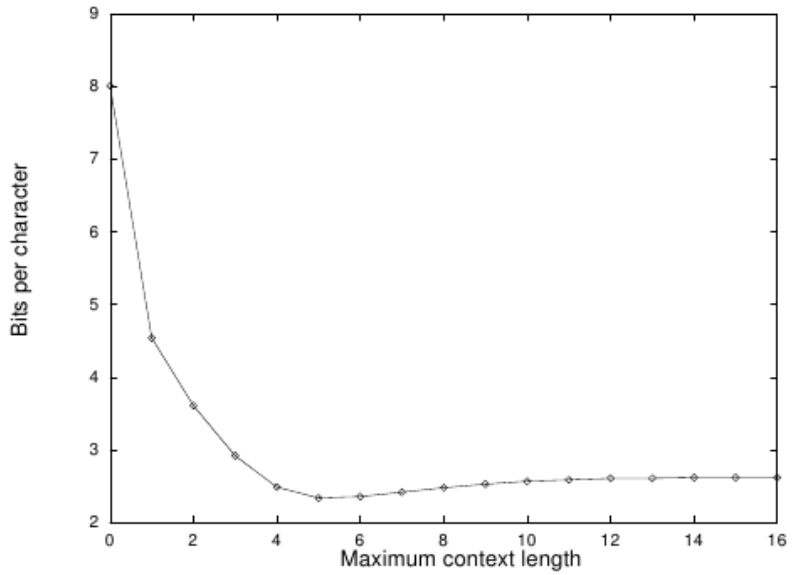
Buchstabe	Anzahl	kumulierte Anzahl	rel. Häufigkeit
a	3	3	3/10
l	2	5	2/10
r	1	6	1/10
s	2	8	2/10
_	1	9	2/10
ESC	1	10	1/10

Kontext	Buchstabe	Anzahl	kumulierte Anzahl	rel. Häufigkeit
l	a	2	2	2/3
l	ESC	1	3	1/3
a	r	1	1	1/4
a	s	1	2	1/4
a	_	1	3	1/4
a	ESC	1	4	1/4
r	a	1	1	1/2
r	ESC	1	2	1/2
_	l	1	1	1/2
_	ESC	1	2	1/2
s	s	1	1	1/2
s	ESC	1	2	1/2

Das W-keitsmodell nullter und erster Ordnung nach dem Einlesen von $lara\_lass\_$

Buchstabe	Anzahl	kumulierte Anzahl	rel. Häufigkeit
a	3	3	3/11
l	2	5	2/11
r	1	6	1/11
s	2	8	2/11
_	2	10	2/11
ESC	1	11	1/11

Kontext	Buchstabe	Anzahl	kumulierte Anzahl	rel. Häufigkeit
l	a	2	2	2/3
l	ESC	1	3	1/3
a	r	1	1	1/4
a	s	1	2	1/4
a	_	1	3	1/4
a	ESC	1	4	1/4
r	a	1	1	1/2
r	ESC	1	2	1/2
_	l	1	1	1/2
_	ESC	1	2	1/2
s	s	1	1	1/3
s	_	1	2	1/3
s	ESC	1	3	1/3



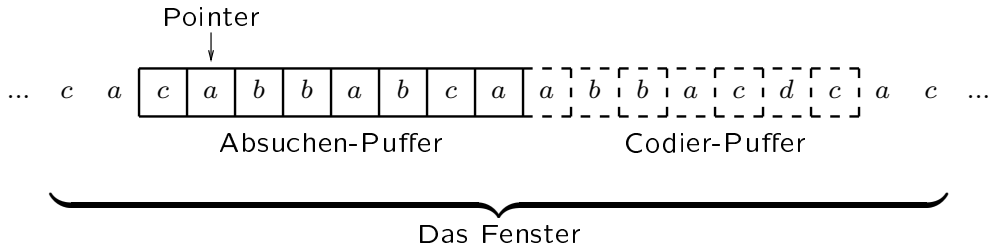
Das Verhalten des Kompressionsfaktors bei Veränderung der maximalen Kontextlänge bei PPM

Literatur: J. G. Cleary, W. Teahan, I. H. Witten, *Unbounded length contexts for PPM*, The Computer Journal, 40(1997), 67–75.

Dynamisches Wörterbuch

Alle wichtigen Verfahren dieser Familie basieren auf zwei Algorithmen, die von Jacob Ziv und Abraham Lempel 1977 bzw. 1978 beschrieben wurden, nämlich **LZ77** und **LZ78**.

- LZSS – Lempel-Ziv-Storer-Szymanski (`gzip`, `ZIP` und andere),
- LZW – Lempel-Ziv-Welch (u.a. `GIF` Format),
- LZC – Lempel-Ziv Unix `compress` (Anwendung von LZW),
- LZMW – Modifikation von LZW (Miller/Wegman),
- ....



LZ77 Gleitfenster-Algorithmus

1. Verschiebe den Pointer auf das erste Symbol von rechts im Absuch-Puffer. Dann sei
   `Startsymbol, S :=` "das erste Symbol im Codier-Puffer"; `offset, L := 0`.

2. Verschiebe den Pointer im Absuch-Puffer so weit nach links bis er das Zeichen findet, das gleich dem `Startsymbol` ist, oder die linke Grenze des Puffers erreicht hat. Wenn die Grenze des Puffers erreicht ist, dann gib als Code

   `(offset, L, K(S))`

   aus, wobei `K(S)` der Code des Zeichens `S` ist; sonst gehe zum nächsten Schritt.

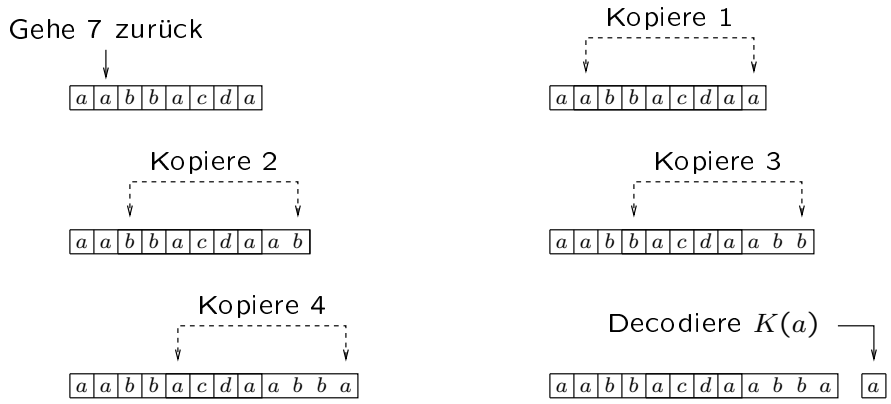
3. Lies Zeichen für Zeichen gleichzeitig vom `Startsymbol` und Pointer so lange, bis die Zeichen gleich sind. Wenn die Länge der durchgelesenen Zeichenfolgen größer als `L` ist, dann nimm diese Länge als neuen Wert für `L` und den Abstand des Pointers zur rechten Grenze des Absuch-Puffers als neuen Wert für `offset`. Weiterhin nimm das `L+1`-ste Symbol des Codier-Puffers als `S` und beginne so wieder mit Schritt 2.

Beispiel 2 *Codieren mit LZ77:* `abaabbabaacaacabb`

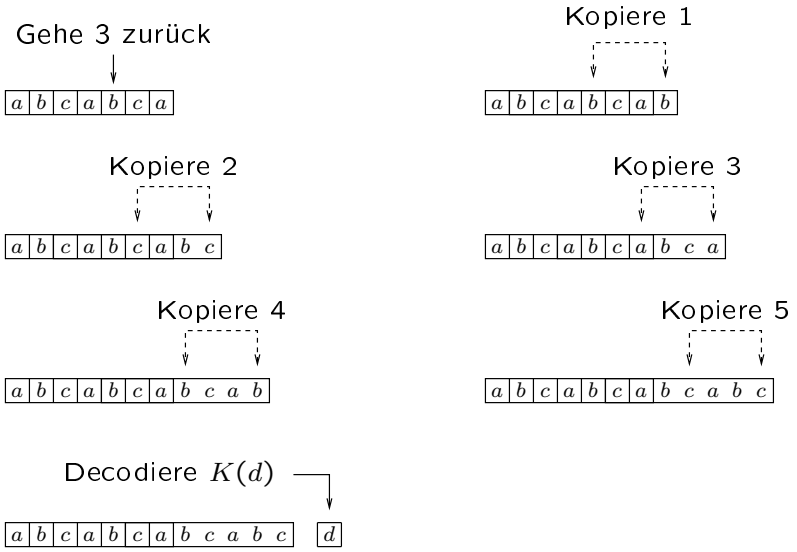
Inhalt der Puffer	Generierter Code
` | a b a a b` b a b a a c a a c a b b	(0,0,K(a))
a | b a a b b a b a a c a a c a b b	(0,0,K(b))
a b | a a b b a b a a c a a c a b b	(2,1,K(a))
a b a a | b b a b a a c a a c a b b	(3,1,K(b))
a b a a b b | a b a a c a a c a b b	(6,4,K(c))
a b a a b | b a b a a c | a a c a b b	(3,4,K(b))
a b a a b b a b a a | c a a c a b | b	(1,1,K(eof))

LZ77-Decodierer: simuliere Codier-Algorithmus.

Beispiel 3 *Der Inhalt des Absuch-Puffers ist: aabbacda und der Code:* $(7, 4, K(a))$.



Beispiel 4 *Der Inhalt des Absuch-Puffers ist: abcabca und der Code:* $(3, 5, K(d))$ *(die Decodierung für eine 'überlappende' Zeichenfolge).*



LZSS — eine Variante von LZ77, die im Ausgabecode keinen Code für das nächste Zeichen angibt. Jetzt hat jeder Ausgabecode eins von zwei Formaten:

(0, offset, Länge),

(1, Zeichen).

Typischerweise konstruiert LZSS den Code des ersten Formats, wenn die Länge größer als 2 ist, und dann benutzt der Algorithmus die Huffman-Codierung, um die Reihenfolge von 'Ausgabecode-Symbolen' zu codieren. In der Praxis ist die Länge des Fensters gleich 32.

Beispiel 5 *Eingabe:* `a a a a a a a`
Ausgabecode: `(1,a) (0,1,6) (1,eof)`

Eingabe: `a a a b a b a b a`
Ausgabecode: `(1,a) (1,a) (1,a) (1,b) (0, 2, 5) (1,eof)`

Nachteile

LZ77 (und seine Variante) bearbeitet besonders schlecht Eingaben, in denen sich Zeichenfolgen befinden, die sich im Eingabetext in größerem Abstand wiederholen, als die Länge des Fensters erfassen kann.

Ein weiteres Problem — Effiziente Implementierung des Wörterbuches.

LZ78 Komprimierverfahren

- Am Anfang ist die Matrize `Wörterbuch` leer.
- Jeder Code hat das Format:

  $$(i, k)$$

  wobei i der Index zum `Wörterbuch` ist und k der Code eines einzelnen Zeichens.
- Um den neuen Code zu generieren:
  1. finde im Eingabetext die längste Zeichenfolge $w \in$ `Wörterbuch`; nimm als i den Index für w (wenn $w = \epsilon$ dann $i = 0$).
  2. $Z :=$ "das nächste Zeichen".
  3. Füge wZ dem `Wörterbuch` hinzu und gib $(i, K(Z))$ aus.

  Für $K(Z)$ nehmen wir den Code für Z an so lange, bis der Algorithmus das erste Mal die Zeichenfolge wZ betrachtet, wobei $w = \epsilon$. Dann ist $K(Z)$ gleich dem Index im `Wörterbuch`.

Beispiel 6 *Codieren mit LZ78* `abababcabac`

Text	Wörterbuch Index	Wörterbuch Eintrag	Ausgang Code
a	1	a	(0,K(a))
b	2	b	(0,K(b))
a			
b	3	ab	(1,2)
a			
b			
c	4	abc	(3,K(c))
a			
b			
a	5	aba	(3,1)
c	6	c	(0,K(c))
eof			

LZW Komprimierverfahren (Terry Welch 1984) (eine Variante von LZ78)

LZW gibt im Ausgabecode keinen Code für die Zeichen an.
Als Codewörter benutzt LZW direkt die Indizes des Wörterbuches.

Der *LZW-Codierer* startet mit dem Wörterbuch, das die einzelnen Standardzeichen enthält.

```
w = NIL
while (Z = readchar())
{
  if wZ in Wörterbuch
    w = wZ
  else {
    gib Code für w aus
    gib wZ zum Wörterbuch
    w = Z }
}
gib Code für w aus
```

Beispiel 7

Anfangszustand:

Wörterbuch	
0	NULL
1	SOH
	...
97	a
98	b
99	c
100	d
	...
121	y
122	z
	...
255	255

Text	w	Z	Neuer Eintrag im Wörterbuch Eintrag	Index	Ausgang Code
a		a			
b	a	b	ab	256	97
a	b	a	ba	257	98
b	a	b			
a	ab	a	aba	258	256
b	a	b			
c	ab	c	abc	259	256
a	c	a	ca	260	99
b	a	b			
a	ab	a			
c	aba	c	abac	261	258
eof	c				99

Anfangszustand:

Wörterbuch	
0	NULL
1	SOH
...	
97	a
98	b
99	c
100	d
...	
121	y
122	z
...	
255	255

Text	w	Z	Neuer Eintrag im Wörterbuch Eintrag	Index	Ausgabe Code
a		a			
a	a	a	aa	256	97
a	a	a			
a	aa	a	aaa	257	256
a	a	a			
a	aa	a			
a	aaa	a	aaaa	258	257
eof	a				97

LZW Decodierer

```
v = Wörterbuch(readcode())
output v
while (C = readcode()) {
  if C in Wörterbuch
    w = Wörterbuch(C)
  else {
    K = v[0]  /* v[0] ist das erste Zeichen in v
    w = vK
  }
  output w
  Z = w[0]
  gib vZ zum Wörterbuch
  v = w
}
```

Kompressiongüte (in bpc) am Beispiel des Canterbury Corpus

File Set	File	pack	compress	DMC	gzip	bred	PPMC
English text	alice29.txt	4.62	3.27	2.38	2.86	2.55	2.30
Fax images	ptt5	1.66	0.97	0.82	0.88	0.82	0.98
C source code	fields.c	5.12	3.56	2.40	2.25	2.17	2.14
Spreadsheet Files	kennedy.xls	3.60	2.41	1.43	1.61	1.21	1.01
SPARC Executables	sum	5.42	4.21	3.03	2.70	2.77	2.71
Technical documents	lcet10.txt	4.70	3.06	2.31	2.72	2.47	2.18
English Poetry	plrabn12.txt	4.58	3.38	2.66	3.24	2.89	2.46
HTML	cp.html	5.30	3.68	2.69	2.60	2.50	2.36
lisp source code	grammar.lsp	4.87	3.90	2.84	2.65	2.69	2.41
GNU Manual pages	xargs.1	5.10	4.43	3.51	3.31	3.26	2.98
Plays	asyoulik.txt	4.85	3.51	2.64	3.13	2.84	2.52
Average		4.53	3.31	2.43	2.54	2.38	2.19

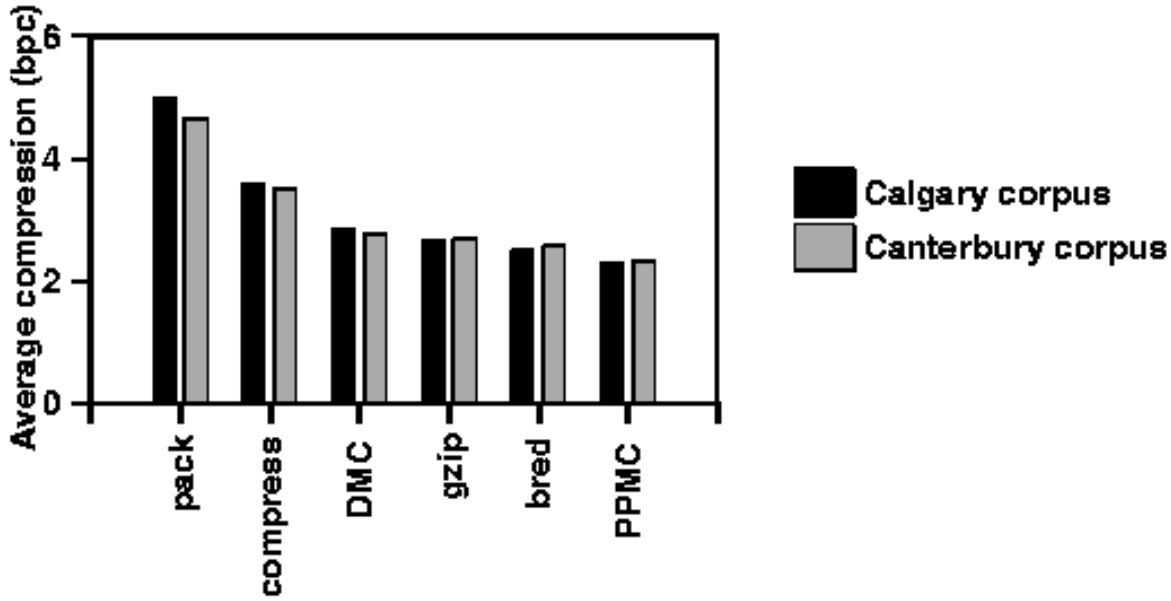
siehe: R. Arnold and T. Bell, *A corpus for the evaluation of lossless compression algorithms*, Proc. of the Data Compression Conference, 1997, pp. 201-210. `http://corpus.canterbury.ac.nz`.

Methoden: *pack* (Huffman coding), *compress* (LZW),

DMC (Dynamic Markov Compression [Cormack/Horspool 87]),

gzip (LZ77), *bred* (Burrows/Wheeler 94), und *PPMC* ([Moffat 90]).

Canterbury vs. Calgary corpora



siehe R. Arnold and T. Bell, Proc. DCC, 1997.

Weitere Literatur

1. K. Sayood. *Introduction to Data Compression,* Morgan Kaufmann Publishers, Inc., 3. Auflage, 2006.

2. J.G. Cleary and I.H. Witten, *Data compression using adaptive coding and partial string matching*, IEEE Transactions on Communications 32(4): 396–402, 1984.

3. A. Moffat, *Implementing the PPM data compression scheme*, IEEE Transactions on Communications 38(11): 1917–1921, 1990.

4. J. Ziv and A. Lempel, *A Universal Algorithm for Sequential Data Compression*, IEEE Transactions on Information Theory 23(3): 337–343, 1977.

5. J. Ziv and A. Lempel, *Compression of individual sequences via variable-rate coding*, IEEE Transactions on Information Theory 24(5): 530–536, 1978.