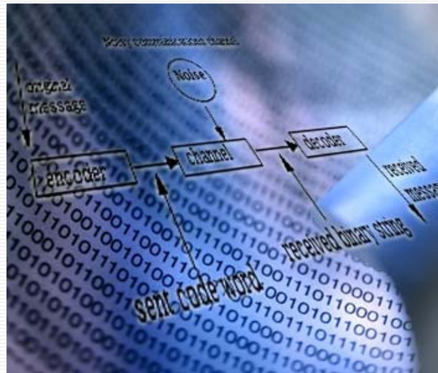
Kapitel 10: Optimalcodierung IV



Ziele des Kapitels

ETH

- Lempel-Ziv Coding
- Cover, pp. 319ff

Copyright M. Gross, ETH Zürich 2006, 2007

Lempel-Ziv Coding

ETH

- Wurde 1977 zum ersten Mal vorgestellt
- Benötigt keine Quellenstatistik
- Wesentliches Charakteristikum ist die Redundanzreduktion während des Codierungsvorganges
- Grundidee: Suche wiederholt auftretende Zeichenketten im Quelltext
- Speichere diese in einem dynamischen Wörterbuch ab
- Bilde möglichst viele Quellzeichen als Zeichenkette auf ein Codewort ab

Copyright M. Gross, ETH Zürich 2006, 2007

Lempel-Ziv Coding

ETH

- Die Quellenstatistik wird also dynamisch aufgebaut und aktualisiert
- Im Wörterbuch werden sowohl Zeichenwahrscheinlichkeiten, als auch Uebergangswahrscheinlichkeiten berücksichtigt
- Ferner können zeitlich veränderliche Wahrscheinlichkeiten berücksichtigt werden
- Die Codierung wird automatisch der Quellenstatistik angepasst
- Lempel-Ziv Codierung steht für eine ganze Klasse von Varianten
- Das Verfahren erreicht asymptotisch die Entropie

Copyright M. Gross, ETH Zürich 2006, 2007

Lempel-Ziv Coding

ETH

- Wir betrachten eine binäre Quelle mit $\chi = \{0,1\}$
- Das Original-Verfahren:
  Gegeben sei ein Quellstring $x_1 x_2 \dots x_n$ (1011010100010)
  1. Parse den String und zerlege ihn in disjunkte Teilstrings durch Einsetzen von Kommata
     - Beispiel: 1,0,11,01,010,00,10
  2. Nach jedem Komma, suche den nächsten Teilstring, der noch nicht aufgetaucht ist
  3. Da dies jeweils der kürzeste, bisher noch nicht aufgetauchte String ist, müssen alle seine Präfixe bereits aufgetaucht sein
  4. D.h. der gesamte vordere Substring bis auf das letzte Bit, ist bereits aufgetaucht

Copyright M. Gross, ETH Zürich 2006, 2007

Lempel-Ziv Coding

ETH

5. Wir codieren diesen String durch die Position (Pointer) des Präfixes sowie des letzten Bits
   - Beispiel: 4. String „01" würde zu (010,1)
6. Wiederhole dieses Verfahren, bis der String zu Ende ist

- Sei $c(n)$ die Anzahl gefundener Substrings in der Eingabesequenz der Länge n
- Dann benötigen wir $\log_2(c(n))$ Bit
- zur Codierung der Position des Präfix sowie 1 Bit für das letzte Bit des Strings
- Beispiel: (Präfix, letztes Bit)
  (000,1)(000,0)(001,1)(010,1)(100,0)(010,0)(001,0)

Copyright M. Gross, ETH Zürich 2006, 2007

Lempel-Ziv Coding

ETH

- Der Algorithmus benötigt 2 Durchgänge:
  1. Parse den Eingabestring, identifiziere alle Substrings, und berechne $c(n)$ sowie $\log(c(n))$
  2. Codiere den String mit Hilfe der ermittelten Substringpositionen
- Verbesserungsmöglichkeiten:
  1. Anzahl der benötigten Bits zur Positionscodierung ist nicht überall gleich gross (Anfang des Strings)
  2. Nur ein Durchlauf benötigen

Das Verfahren wird vor allem bei grossen Eingabestrings effizient, denn die Position des Präfixes lässt sich durch weniger Bits komprimieren, als der Präfix selbst.

Copyright M. Gross, ETH Zürich 2006, 2007

Lempel - Ziv

ETH

- Gegeben ein binärer String: 1011010100010
- Schritt 1: Parsing und Wörterbuch
  1,0,11,01,010,00,10

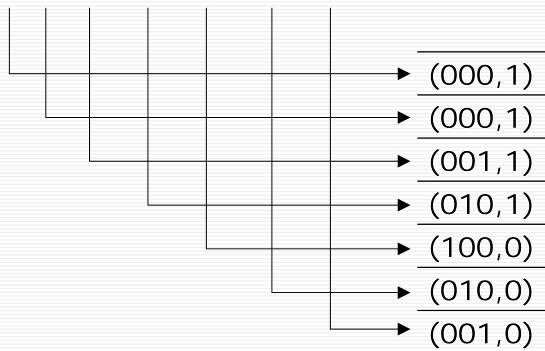
Positon	Substring
000	-
001	1
010	0
011	11
100	01
101	010
110	00
111	10

Copyright M. Gross, ETH Zürich 2006, 2007

Lempel - Ziv

ETH

- Schritt 2: Codierung (Präfix-Position, letztes Bit)
- 1, 0, 11, 01, 010, 00, 10



Beweis der Optimalität

ETH

- Der Beweis ist recht aufwändig, daher nur als Skizze dargestellt
- Wir betrachten eine binäre Quelle mit $\chi = \{0,1\}$
- Definition: Ein Parsing S eines Strings $x_1 x_2 \ldots x_n$ ist eine Zerlegung des Strings in Teilstrings, geteilt durch Kommata.
- Bei einem eindeutigen Parsing sind alle Teilstrings paarweise disjunkt
  - 0,111,1 ist eindeutiges Parsing von 01111
  - 0,11,11 ist nicht eindeutig
- Der Ziv-Lempel Algorithmus stellt ein eindeutiges Parsing dar

Beweis der Optimalität

ETH

- $c(n)$ sei die Anzahl der Teilstrings im Ziv-Lempel Parsing eines Strings der Länge n
- Die Quelle sei durch einen stationären, ergodischen Prozess der Form $X = X_1, X_2, \ldots$ beschrieben
- Jeder Positionspointer benötigt $\log(c(n))$ Bit
- Die Gesamtsequenz benötigt also

$$c(n)(\log_2(c(n))+1) \quad \text{Bit}$$

- Zu zeigen ist also, dass

$$\frac{c(n)(\log_2(c(n))+1)}{n} \to H(X)$$

- Wir benötigen einige Lemmas

Lemma 1

ETH

- Lemma: Für die Anzahl von Substrings in einem Lempel-Ziv Parsing einer binären Eingabesequenz $X_1 X_2 \ldots X_n$ gilt

$$c(n) \le \frac{n}{(1-\varepsilon_n)\log n} \quad \text{mit} \quad \varepsilon_n \to 0 \quad \text{und} \quad n \to \infty$$

- Beweis: Siehe Cover pp. 320-321

Die wichtige Aussage ist, dass wir eine nichttriviale obere Grenze für die Anzahl der Substrings finden können

Lemma 2 und Lemma 3

ETH

- Lemma: Sei Z eine Zufallsvariable, welche positive, ganzzahlige Werte mit Mittelwert μ besitzt. Die Entropie dieser Zufallsvariablen ist wie folgt begrenzt:

$$H(Z) \leq (\mu+1) \cdot \log(\mu+1) - \mu \cdot \log \mu$$

- Wir benötigen nun eine Markov-Approximation Q_k der Ordnung k des Prozesses
- Lemma: Für jedes eindeutige Parsing des Strings $x_1 x_2 \ldots x_n$ erhalten wir

$$\log Q_k(x_1, x_2, \ldots, x_n \mid s_1) \leq -\sum_{l,s} c_{ls} \log c_{ls}$$

- Die rechte Seite ist unabhängig von Q_k

Theorem 1

ETH

- Die Entropierate der Markov-Approximation konvergiert gegen die Prozessentropie für grosse k
- c_{ls} ist die Anzahl der Substrings y_i der Länge l und Präfix s
- Theorem: Sei $\{X_n\}$ ein stationärer, ergodischer Prozess mit Entropierate $H(X_n)$ und sei $c(n)$ die Anzahl von Substrings in einem eindeutigen Parsing eines Samples der Länge n des Prozesses. Dann gilt

$$\limsup_{n\to\infty} \frac{c(n)\log_2 c(n)}{n} \leq H(X)$$

- Beweis: Wir verwenden das vorherige Lemma

Theorem 2

ETH

- Theorem: Sei $\{X_n\}$ ein stationärer, ergodischer Prozess. Sei $l(X_1 X_2 \ldots X_n)$ die Lempel-Ziv Codewortlänge assoziiert mit der Sequenz $X_1 X_2 \ldots X_n$
- Aufgrund des vorherigen Lemmas gilt

$$\limsup_{n\to\infty} \frac{c(n)}{n} = 0$$

- Aus diesem Grund gilt

$$\limsup_{n\to\infty} \frac{l(X_1, X_2, \ldots, X_n)}{n} = \limsup_{n\to\infty} \left(\frac{c(n)\log_2 c(n)}{n} + \frac{c(n)}{n} \right) \leq H(X)$$

- mit einer Wahrscheinlichkeit von 1

Erweiterungen (LZW)

ETH

- Wir betrachten eine weit verbreitete Erweiterung des Verfahrens nach Welch (1984)
- Bis zu 2^{12} Einträge ins Wörterbuch möglich
- Die ersten 2^8 Einträge sind für ASCII-Zeichen des Quellalphabets vorinstalliert
- Jedem Eintrag ist ein Index i der Länge 12 Bit zugeordnet
- Der LZW-Algorithmus sucht bei sequentiellen Zeicheneingaben nach bekannten Mustern
- Längster Substring, der im Wörterbuch eingetragen ist

Erweiterungen (LZW)

- Suchvorgang erfolgt Zeichen für Zeichen
- Jedes Einzelzeichen ist bereits im Wörterbuch
- Entscheidung, ob Substring der maximalen Länge entspricht, erfolgt erst beim Einlesen des folgenden Zeichens
- Aktueller Substring *w* ist in diesem Falle ein Präfix von $w' = w*z$ (Konkatenation)
  - Lese aktuelles Zeichen *z*
  - Konkateniere *z* mit *w*
  - Prüfe, ob *w'* im Wörterbuch

Copyright M. Gross, ETH Zürich 2006, 2007

LZW

- Der Eingabestring baacbacbaacba soll kodiert werden. Das zu erstellende Wörterbuch ist bereits mit a=1, b=2 und c=3 vorinstalliert.

Eingabe *z*	Konkat. *w*z*	Wörterbuch-eintrag	Ausgabe <*i*>	Präfix *w*
		a=1		
		b=2		
		c=3		
b				b
a	b*a	ba=4	<2>	a
a	a*a	aa=5	<1>	a
c	a*c	ac=6	<1>	c
b	c*b	cb=7	<3>	b

Copyright M. Gross, ETH Zürich 2006, 2007

LZW

Eingabe *z*	Konkat. *w*z*	Wörterbuch-eintrag	Ausgabe <*i*>	Präfix *w*
a	b*a			ba
c	ba*c	bac=8	<4>	c
b	c*b			cb
a	cb*a	cba=9	<7>	a
a	a*a			aa
c	aa*c	aac=10	<5>	c
b	c*b			cb
a	cb*a			cba
-			<9>	

- Ausgabe: <2><1><1><3><4><7><5><9>

Copyright M. Gross, ETH Zürich 2006, 2007

Algorithmus

1. Lies erstes Zeichen *z* und setze Präfix *w:=z*
2. Lies nächstes Zeichen *z* und konkateniere *w*z*
3. Ist *wz* bereits im Wörterbuch?
   Ja: Setze *w:=wz*
   Nein: -Trage *wz* ins Wörterbuch ein
   -Gib Position (Pointer) von *w* aus
   -Setze *w:=z*
4. Ist aktuelles *z* letztes Zeichen des Strings?
   Nein: Gehe nach 2.
   Ja: Gib *w* aus und stop

Copyright M. Gross, ETH Zürich 2006, 2007

Decodierung

ETH

- Dazu muss Decodierer mit gleichem Wörterbuch, wie Codierer arbeiten
- Dazu wird die Systematik des Codierers genutzt
  - An jedes erkannte Muster wurde das nächstfolgende Zeichen angehängt
  - Daher ist das letzte Zeichen eines Eintrages immer identisch zum ersten des folgenden Eintrags
- Bei Dekodierung wird jedes empfangene Codewort übersetzt und als $(z(1),z(2),\ldots)$ ausgegeben
- Zwischenspeichern als Präfix w
- Verknüpfung $wz(1)$ mit ersten übersetzten Zeichen

LZW Decodierung

ETH

- Eingabe: <2><1><1><3><4><7><5><9>

Eingabe <i>	Ausgabe zk	Konkat. $w{*}z(1)$	Wörterbuch-eintrag	Präfix w
			a=1	
			b=2	
			c=3	
<2>	b			b
<1>	a	b*a	ba=4	a
<1>	a	a*a	aa=5	a
<3>	c	a*c	ac=6	c

LZW Decodierung

ETH

- <2><1><1><3><4><7><5><9>

Eingabe <i>	Ausgabe zk	Konkat. $w{*}z(1)$	Wörterbuch-eintrag	Präfix w
<4>	ba	c*b	cb=7	ba
<7>	cb	ba*c	bac=8	cb
<5>	aa	cb*a	cba=9	aa
<9>	cba	aa*c	aac=10	

- Ausgabe: baacbacbaacba

Anmerkungen

ETH

- Wird vor allem für verlustfreie Textkompression verwendet
- Man erreicht ca. 50% Kompression in praktischen Anwendungen
- Effektiv, wenn sich eine kleine Anzahl von Substrings im Text oft wiederholt
- Verbesserung durch zusätzliche Huffman-Codierung zu erreichen
- Generell ein sehr bedeutsames Verfahren