Verlustfreie Kompression

Tim Rolff

Arbeitsbereich Wissenschaftliches Rechnen
Fachbereich Informatik
Fakultät für Mathematik, Informatik und Naturwissenschaften
Universität Hamburg

8. Juni 2016

Gliederung

1 Was ist Kompression?

2 Die Mathematik dahinter

3 Algorithmen

4 Kompression von Wetterformaten

5 Probleme

Warum überhaupt komprimieren?

- Festplattenspeicher ist auch heute noch nicht unbegrenzt.
- Zusätzliche Hardware kostet Geld und nimmt Platz.
- Bandbreite ist leider auch begrenzt.
- Das Übertragen von Daten kostet Geld (Beispiel Handytarif).
- Erhöhung der Bandbreite und Leistung.

Zwei Wege zur Kompression

- Verlustfreie Kompression
  - Originaldaten können exakt wiederhergestellt werden.
- Verlustbehaftete Kompression
  - Originaldaten können nicht wieder hergestellt werden
  - Ein Teil der Information geht verloren.
  - Das Design ist durch das Anwendungsgebiet bestimmt.
    - Bilder $\rightarrow$ JPEG 2000
    - Musik $\rightarrow$ MP2, MP3, Ogg Theora
    - Video $\rightarrow$ MPEG-1, MPEG-2, MPEG-3

---

http://www.thinkstockphotos.de/image/stock-illustration-yellow-metal-sign/483393490/popup?sq=undefined

Verlustfreie Kompression

- Nutzt die Redundanz der Daten aus.
- Arbeitet sehr häufig mit Wörterbüchern.
- Typische Algorithmen sind meist Verbesserungen des Lempel–Ziv Algorithmus.
  - Lempel–Ziv (LZ) Bsp. LZ77 oder LZX
  - Deflate Bsp. GZip oder PNG

Verlustfreie Kompression: Grundidee für Textkompression

- Dies ist ein langer Text, welcher ganz oft das Wort Text enthält, da Text ein tolles Wort ist.
- Dies **ist** ein langer **Text**, welcher ganz oft das **Wort Text** enthält, da **Text** ein tolles **Wort ist**.
- Dies **ist** ein langer **Text**, welcher ganz oft das **Wort -6** enthält, da **-9** ein tolles **-7 -16**.

Verlustfreie Kompression: Grundidee für Textkompression

- Dies ist ein langer Text, welcher ganz oft das Wort Text enthält, da Text ein tolles Wort ist.
- Dies **ist** ein langer **Text**, welcher ganz oft das **Wort Text** enthält, da **Text** ein tolles **Wort ist**.
- Dies **ist** ein langer **Text**, welcher ganz oft das **Wort -6** enthält, da **-9** ein tolles **-7 -16**.

Verlustfreie Kompression: Grundidee für Textkompression

- Dies ist ein langer Text, welcher ganz oft das Wort Text enthält, da Text ein tolles Wort ist.
- Dies **ist** ein langer **Text**, welcher ganz oft das **Wort Text** enthält, da **Text** ein tolles **Wort ist**.
- Dies **ist** ein langer **Text**, welcher ganz oft das **Wort -6** enthält, da **-9** ein tolles **-7 -16**.

Verlustfreie Kompression: Grundidee für Tokenkompression

- Dies ist ein langer Text, welcher ganz oft das Wort Text enthält, da Text ein tolles Wort ist.
- Dies **ist** ein langer **Text**, welcher ganz oft das **Wort Text** enthält, da **Text** ein tolles **Wort ist**.
- Wörterbuch: {ist: #0}, {Text: #1}, {Wort: #2}

  Dies **#0** ein langer **#1**, welcher ganz oft das **#2 #1** enthält, da **#1** ein tolles **#2 #0**.

Verlustfreie Kompression: Grundidee für Tokenkompression

- Dies ist ein langer Text, welcher ganz oft das Wort Text enthält, da Text ein tolles Wort ist.
- Dies **ist** ein langer **Text**, welcher ganz oft das **Wort Text** enthält, da **Text** ein tolles **Wort ist**.
- Wörterbuch: {ist: #0}, {Text: #1}, {Wort: #2}

  Dies **#0** ein langer **#1**, welcher ganz oft das **#2 #1** enthält, da **#1** ein tolles **#2 #0**.

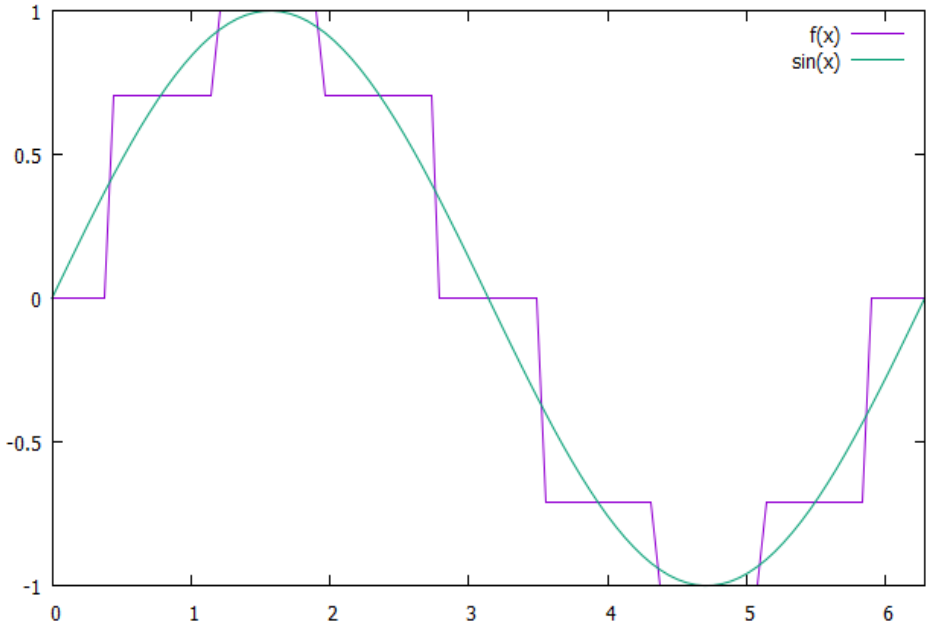
Verlustfreie Kompression: Grundidee für Tokenkompression

- Dies ist ein langer Text, welcher ganz oft das Wort Text enthält, da Text ein tolles Wort ist.
- Dies **ist** ein langer **Text**, welcher ganz oft das **Wort Text** enthält, da **Text** ein tolles **Wort ist**.
- Wörterbuch: {ist: #0}, {Text: #1}, {Wort: #2}

  Dies **#0** ein langer **#1**, welcher ganz oft das **#2 #1** enthält, da **#1** ein tolles **#2 #0**.

Verlustbehaftete Kompression

- Grundidee ist eine Beschreibung der Daten durch eine endliche Menge von Codewörtern.
- Beispiel: Messung einer Sinuswelle mit 3-Bit.



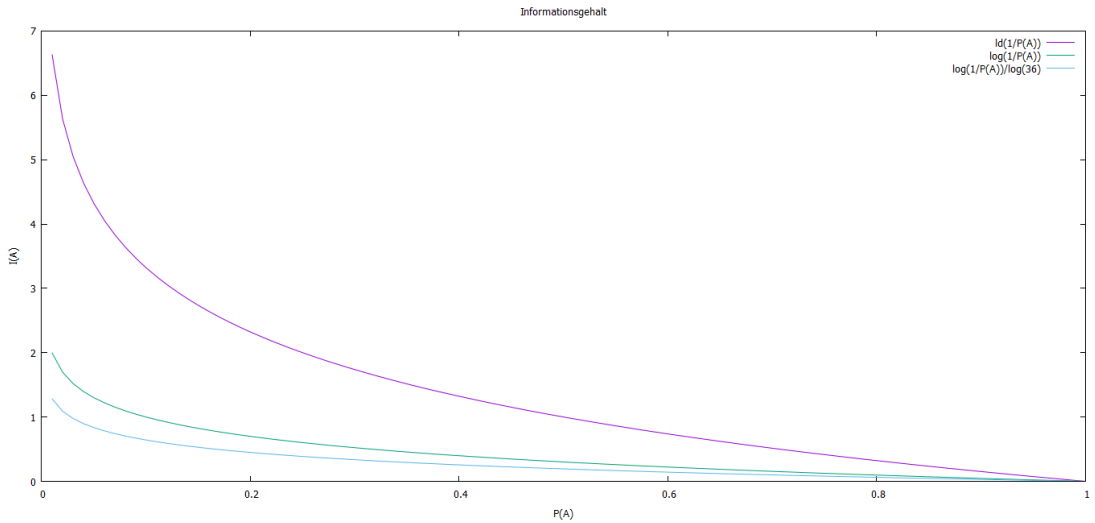
$$f(x) = \begin{cases} sin(0) & x \in \left[0, \frac{1\pi}{8}\right) \\ sin(\frac{2*pi}{8}) & x \in \left[\frac{1\pi}{8}, \frac{3\pi}{8}\right) \\ sin(\frac{4*pi}{8}) & x \in \left[\frac{3\pi}{8}, \frac{5\pi}{8}\right) \\ sin(\frac{6*pi}{8}) & x \in \left[\frac{5\pi}{8}, \frac{7\pi}{8}\right) \\ sin(\frac{8*pi}{8}) & x \in \left[\frac{7\pi}{8}, \frac{9\pi}{8}\right) \\ sin(\frac{10*pi}{8}) & x \in \left[\frac{9\pi}{8}, \frac{11\pi}{8}\right) \\ sin(\frac{12*pi}{8}) & x \in \left[\frac{11\pi}{8}, \frac{13\pi}{8}\right) \\ sin(\frac{14*pi}{8}) & x \in \left[\frac{13\pi}{8}, \frac{15\pi}{8}\right) \\ sin(0) & sonst \end{cases}$$

Bild mit gnuplot erstellt

Wie können Informationen beschrieben werden?

- Der Informationsgehalt[1] eines Ereignisses

$$I(A) = log_a\left(\frac{1}{P(A)}\right)$$



1 de.wikipedia.org/wiki/Informationsgehalt

Bild mit gnuplot erstellt

Wie können Informationen beschrieben werden?

- Der mittlere Informationsgehalt (Entropie) einer Nachricht S

$$H(A) = \sum_{A \in S} P(A) \cdot I(A)$$

  Beispiel deutsche Sprache binär codiert[2]: $H \approx 4.09$

- Die ist das Maß für die Qualität eines Codes und kann aus der Redundanz bestimmt werden.

$$R(A) = L(A) - H(A)$$

2 de.wikipedia.org/wiki/Buchstabenhäufigkeit

Warum das Ganze?

- Zur effizienten Erzeugung von Codes (Entropiekodierung).

Live Demo

Wie funktioniert das Ganze?

- Mittels der Fanobedingung

  Kein Code ist Prefix eines anderen Codes.
- Codes sind damit eindeutig decodierbar
- Ebenfalls ausschlaggebend die Kraftungleichung

  Bedingung für die Existenz eines eindeutig dekodierbaren Codes

$$\sum_{i=1}^{s} \frac{1}{q^{l_i}} \leq 1$$

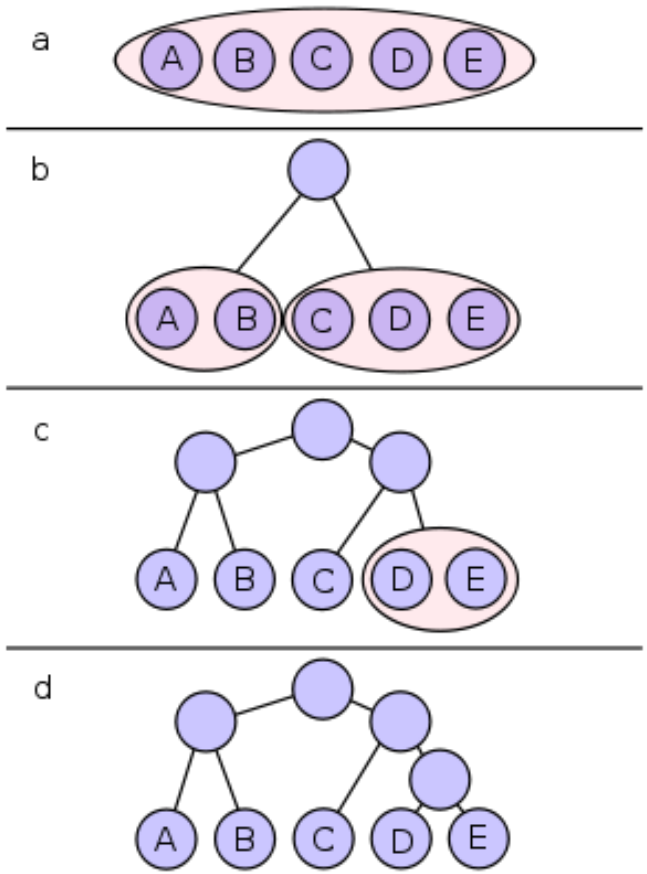
Shannon-Fano Codierung

- Eingabe von Grundwörtern a_i mit einer Auftrittswahrscheinlichkeit $P(a_i)$.

```
def shannon(alphabet, codes, currentCode):
  sort alphabet descending by there probability
  if (length of alphabet = 1):
    codes append (alphabet.word, currentCode)
  else:
    index = divide array into two parts with approximately the same
        ↪ probability
    shannon(alphabet[start to index], codes, currentCode + '0') +
    shannon(alphabet[index+1 to end], codes, currentCode + '1')
```

Algorithmus in anlehnung an [1]

Shannon-Fano Codierung



https://de.wikipedia.org/wiki/Shannon-Fano-Kodierung

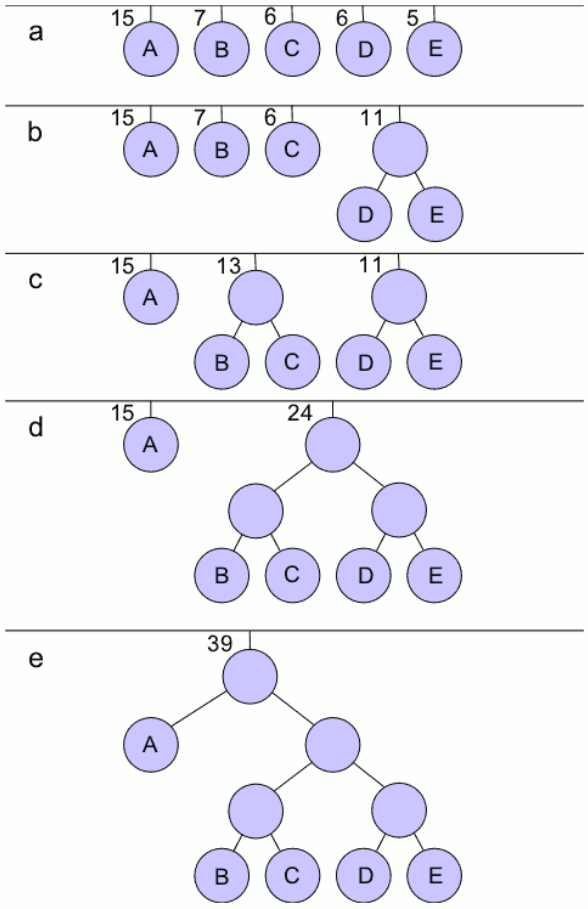
Huffman Codierung

- Eingabe von Grundwörtern a_i mit einer Auftrittswahrscheinlichkeit $P(a_i)$.

```
def huffman(alphabet):
  nodes = create node for every (word, prob) in alphabet

  # build tree
  while (nodes count > 1)
    sort nodes ascending by there probability
    parent = (nodes[0], nodes[1])
    remove nodes[0] and nodes[1] from nodes
    add parent to nodes

  codes = empty list
  traverseTree(nodes, codes, '')
  return codes

def traverseTree(nodes, codes, currentCode):
  if leaf node:
    codes append (nodes.word, currentCode)
  else:
    traverseTree(nodes.child[0], codes, currentCode + '1') +
    traverseTree(nodes.child[1], codes, currentCode + '0')
```

Algorithmus in Anlehnung an [1]

Shannon-Fano Codierung



---

https://en.wikipedia.org/wiki/Huffman_coding

Was ist mit verlustbehafteter Kompression?

- Verlustbehaftete Kompression kann nicht mittels vollständigen Codes beschrieben werden.
- Damit ist die Qualität auch nicht auf gleiche Weise messbar.
- Somit wird, wenn X das Original ist und Y das komprimierte Objekt, dann ist der Fehler mittels quadratischer Abweichung:

$$\sigma^2 = \frac{1}{N} \sum_{i=1}^{N} (x_n - y_n)^2$$

Lauflängencodierung

- Sehr einfaches sehr schnelles Verfahren, da nur einmal über die Daten iteriert werden muss.
- Sehr gute Kompression bei sich häufig gleichbleibenden Daten. Bsp: Boolsche Daten.
- Kann wieder mit einer Huffmancodierung verbunden werden.
- Bsp: 000011111111110 wird zu 4 0, 9 1, 1 0

Die Wörterbuchmethode LZ77 / Codierung

- Arbeitet auf den bereits verarbeiteten Daten, nutzt aber ein '"Gleitfenster'üm den Suchoverhead zu minimieren.
- Das Fenster teilt sich in einen Suchpuffer und Codierpuffer.
- Suchpuffer ist eine Lookup der bereits kodierten Wörter.
- Codierpuffer ist der zu kodierende Code.
- gzip arbeitet mit diesem Prinzip.

Der LZ77 Algorithmus

```
set view to the begining of the data and fill it
while view is not empty:
  find longest prefix of the view in the coded
      ↪ part of the window.
  if found:
    i = position of the pointer in the coded part
    j = length of the pointer
    n = next character in the view
    yield tuple (i, j, K(n))
  else:
    j = 0
    yield tuple (0, j, K(n))

  slide window by j + 1
```

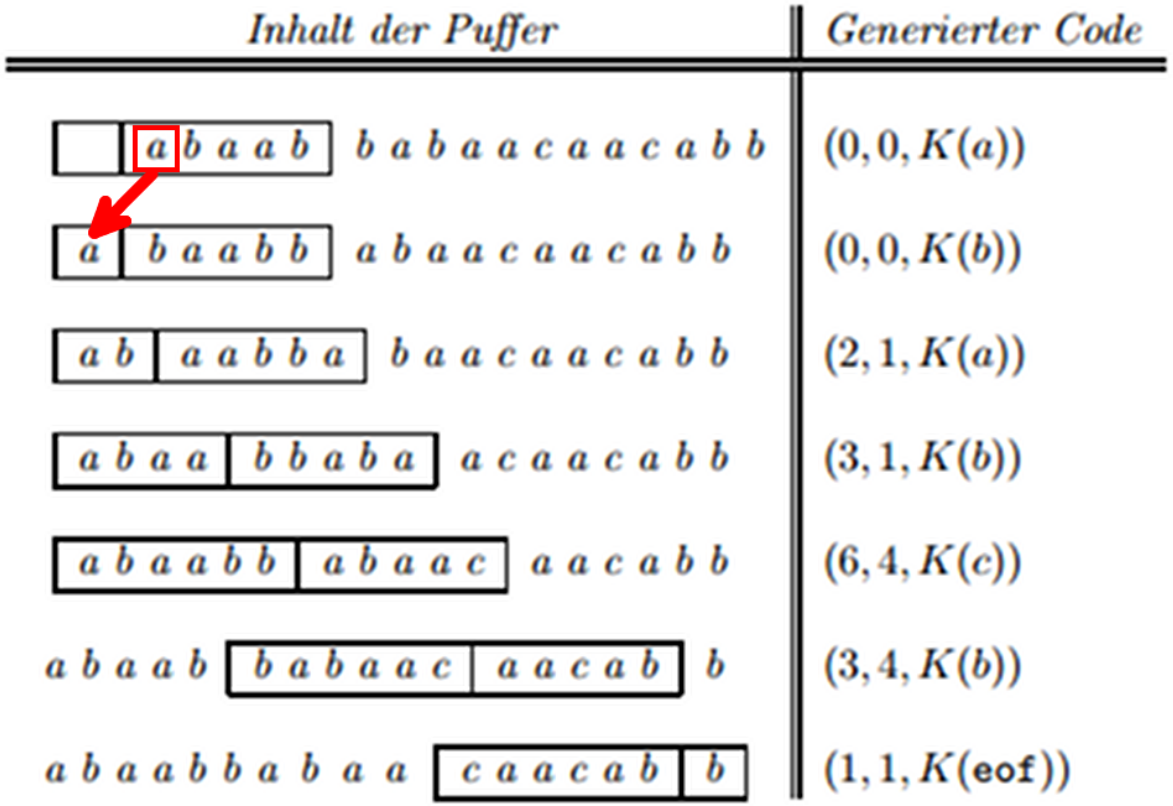
Algorithmus in Anlehnung an https://en.wikipedia.org/wiki/LZ77_and_LZ78

Eine Darstellung des LZ77 Algorithmus

Inhalt der Puffer	*Generierter Code*
$\square \mid a\ b\ a\ a\ b \mid b\ a\ b\ a\ a\ c\ a\ a\ c\ a\ b\ b$	$(0, 0, K(a))$
$a \mid b\ a\ a\ b\ b \mid a\ b\ a\ a\ c\ a\ a\ c\ a\ b\ b$	$(0, 0, K(b))$
$a\ b \mid a\ a\ b\ b\ a \mid b\ a\ a\ c\ a\ a\ c\ a\ b\ b$	$(2, 1, K(a))$
$a\ b\ a\ a \mid b\ b\ a\ b\ a \mid a\ c\ a\ a\ c\ a\ b\ b$	$(3, 1, K(b))$
$a\ b\ a\ a\ b\ b \mid a\ b\ a\ a\ c \mid a\ a\ c\ a\ b\ b$	$(6, 4, K(c))$
$a\ b\ a\ a\ b \mid b\ a\ b\ a\ a\ c \mid a\ a\ c\ a\ b \mid b$	$(3, 4, K(b))$
$a\ b\ a\ a\ b\ b\ a\ b\ a\ a \mid c\ a\ a\ c\ a\ b \mid b$	$(1, 1, K(\texttt{eof}))$

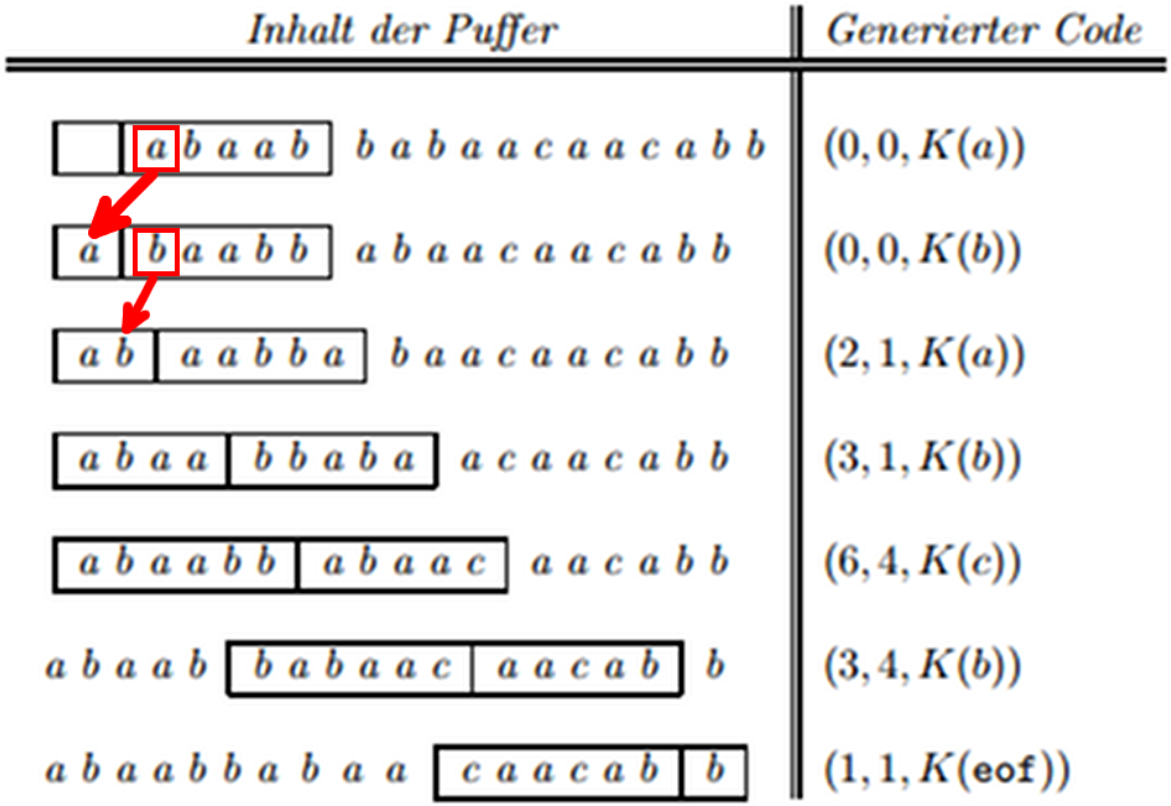
Bilder in Anlehnung an [2]

Eine Darstellung des LZ77 Algorithmus



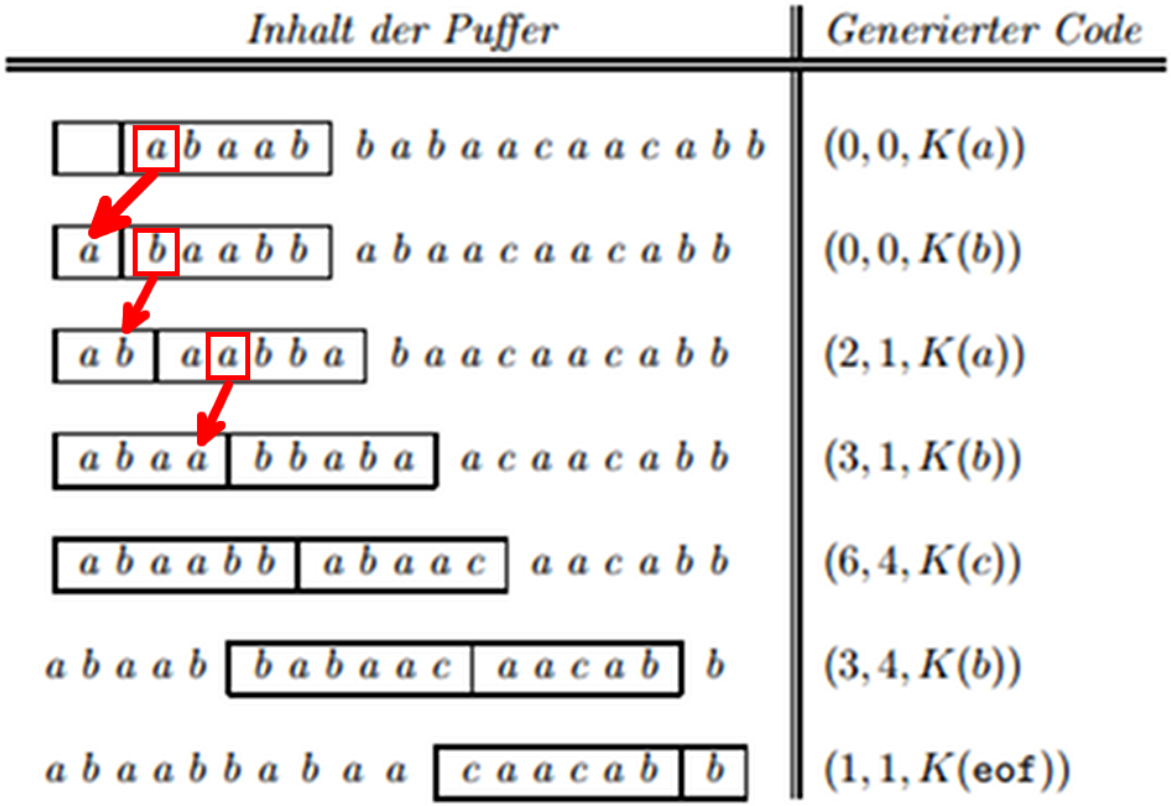
Bilder in Anlehnung an [2]

Eine Darstellung des LZ77 Algorithmus



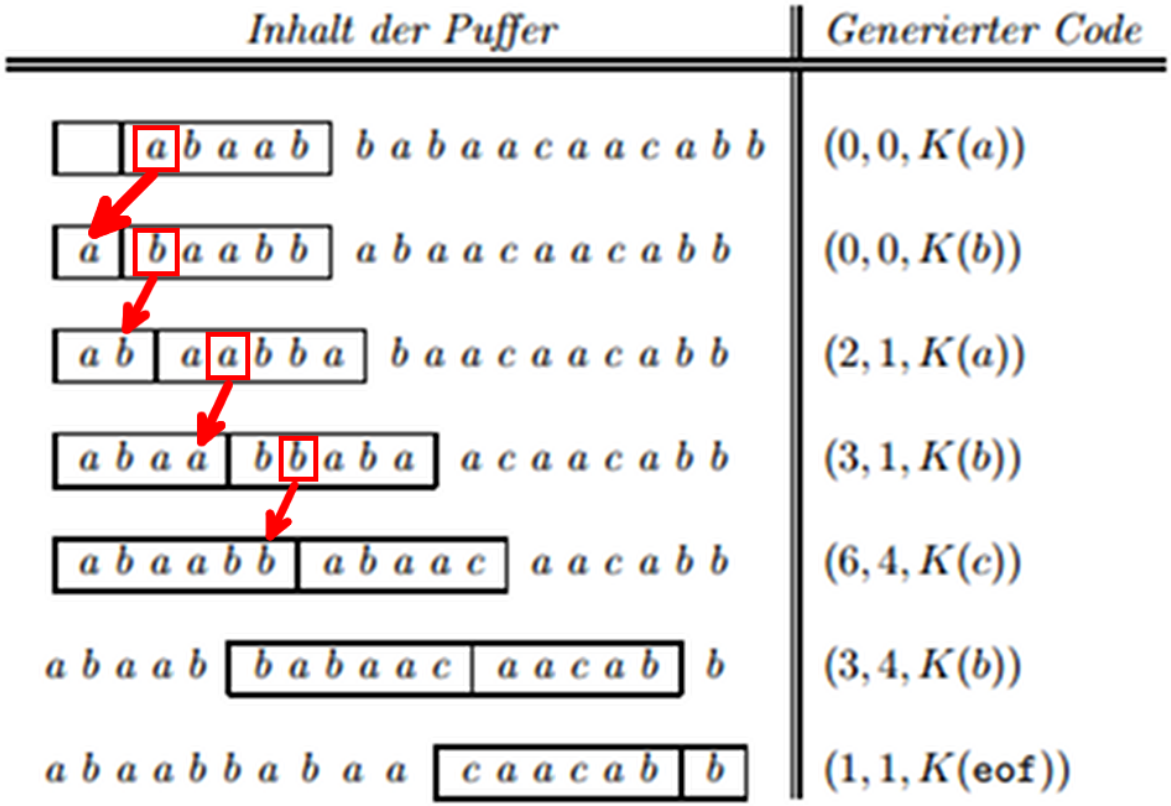
Bilder in Anlehnung an [2]

Eine Darstellung des LZ77 Algorithmus



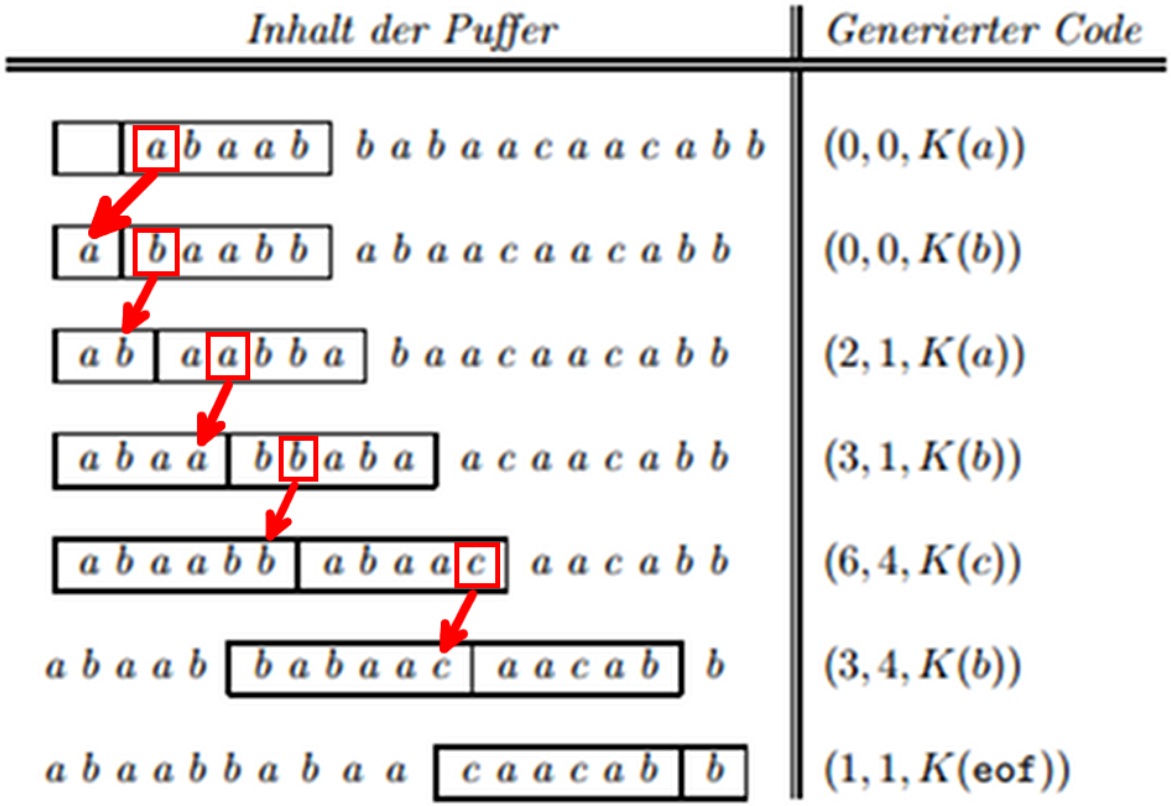
Bilder in Anlehnung an [2]

Eine Darstellung des LZ77 Algorithmus



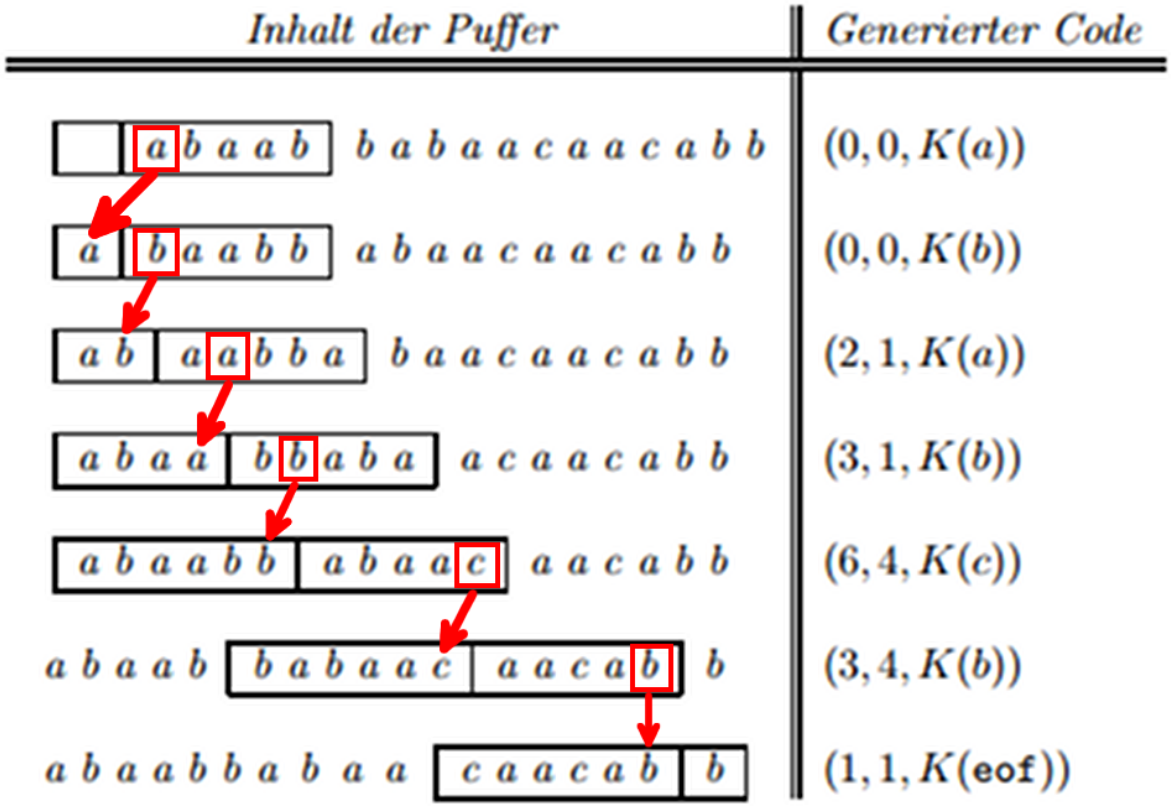
Bilder in Anlehnung an [2]

Eine Darstellung des LZ77 Algorithmus



Bilder in Anlehnung an [2]

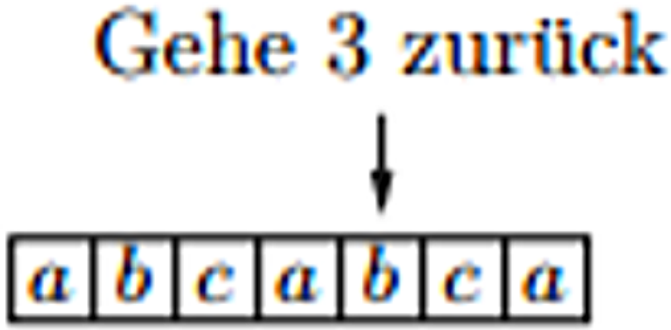
Eine Darstellung des LZ77 Algorithmus



Bilder in Anlehnung an [2]

Die Wörterbuchmethode LZ77 / Dekodierung

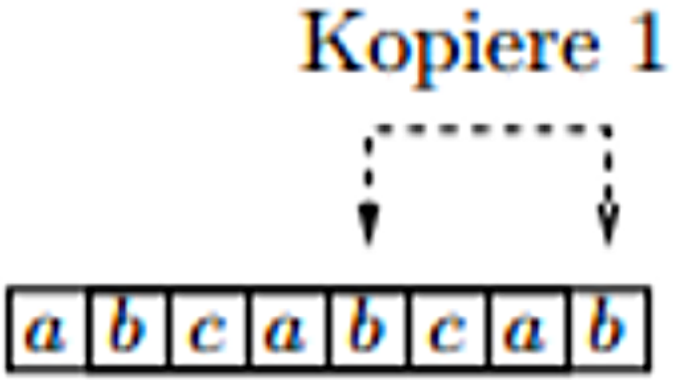
- Nur noch der Suchpuffer nötig und der in der Kodierung generierte Code.
- Beispiel für die Zeichenfolge *abcabca* und dem Tuple $(i = 3, j = 5, K(d))$



Bilder in Anlehnung an [2]

Die Wörterbuchmethode LZ77 / Dekodierung

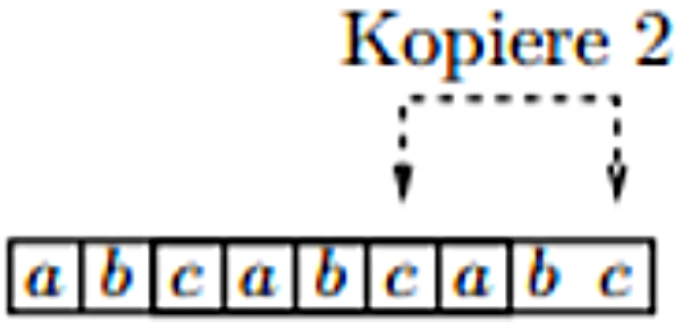
- Nur noch der Suchpuffer nötig und der in der Kodierung generierte Code.
- Beispiel für die Zeichenfolge *abcabca* und dem Tuple $(i = 3, j = 5, K(d))$



Bilder in Anlehnung an [2]

Die Wörterbuchmethode LZ77 / Dekodierung

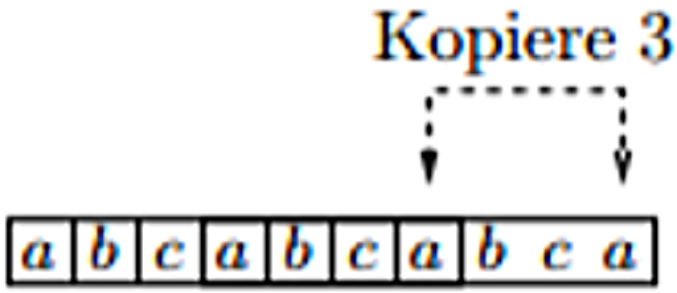
- Nur noch der Suchpuffer nötig und der in der Kodierung generierte Code.
- Beispiel für die Zeichenfolge *abcabca* und dem Tuple $(i = 3, j = 5, K(d))$



Bilder in Anlehnung an [2]

Die Wörterbuchmethode LZ77 / Dekodierung

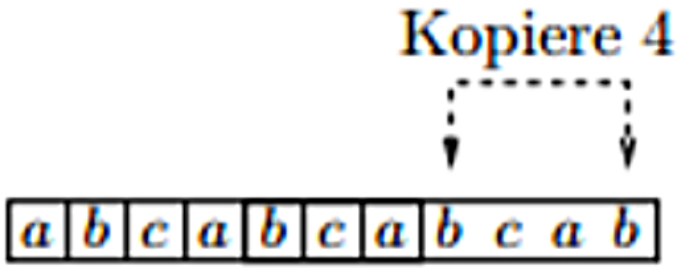
- Nur noch der Suchpuffer nötig und der in der Kodierung generierte Code.
- Beispiel für die Zeichenfolge *abcabca* und dem Tuple $(i = 3, j = 5, K(d))$



Bilder in Anlehnung an [2]

Die Wörterbuchmethode LZ77 / Dekodierung

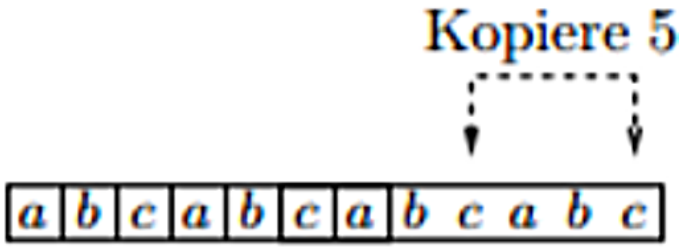
- Nur noch der Suchpuffer nötig und der in der Kodierung generierte Code.
- Beispiel für die Zeichenfolge *abcabca* und dem Tuple $(i = 3, j = 5, K(d))$



Bilder in Anlehnung an [2]

Die Wörterbuchmethode LZ77 / Dekodierung

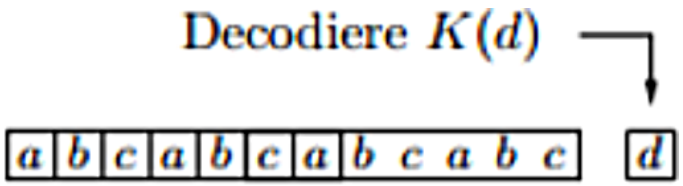
- Nur noch der Suchpuffer nötig und der in der Kodierung generierte Code.
- Beispiel für die Zeichenfolge *abcabca* und dem Tuple $(i = 3, j = 5, K(d))$



Bilder in Anlehnung an [2]

Die Wörterbuchmethode LZ77 / Dekodierung

- Nur noch der Suchpuffer nötig und der in der Kodierung generierte Code.
- Beispiel für die Zeichenfolge *abcabca* und dem Tuple $(i = 3, j = 5, K(d))$



Bilder in Anlehnung an [2]

Der LZ78 Algorithmus[3]

- Führt ein globales Dictionary ein.
- Ein Dictionaryeintrag für einen Code besteht aus $(\text{index}, \text{code})$.
- Ein Dictionaryeintrag für bereits bestehende Codes aus $(\text{index}_1, \text{index}_2)$.
- Wenn kein Eintrag in einem Dictionary gefunden wird, wird ein neues erstellt.
- Leichte Abwandlung als LZW, dieser fügt die Character direkt statt des Codes ein.

3 **https://en.wikipedia.org/wiki/LZ77_and_LZ78**.

Der LZ78 Algorithmus ein Beispiel

Input: 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			

Bilder in Anlehnung an [2]

Kompression des Grib Formats

- Umwandeln: int value = round(scale * (float value - offset))
- Kompression:
  - Simple Packing
    - berechnet die Bitbreite zum schreiben der Daten.
  - Complex Packing
    - Teilt Daten in Gruppen auf.
    - Berechnet die Bitbreite von max Wert - min Wert.
    - Schreibt alle Daten mit der berechneten Bitbreite.
  - Complex Packing (spatial differencing)
    - Ähnlich wie Complex Packing
    - Nutzt die Differenz von $\text{Wert}_i - \text{Wert}_{i-1}$ aus
  - JPEG 2000
  - PNG

Kompression des NetCDF4 / HDF5 Formats

- Verlustfreie Kompression:
  - Nutzt den LZ77 Algorithmus mit gzip
  - HDF5 kann alternativ noch SZip (proprietär), dies nutzt Rice Compression[4]

4 https://www.hdfgroup.org/doc_resource/SZIP/

Vergleich der Algorithmen

Table 6: Saving percentages of all selected algorithms

Original File						
File	File Size	RLE	LZW	Adaptive	Huffman	Shannon
1	22,094	-0.71	38.24	39.21	37.42	36.06
2	44,355	1.25	43.78	39.32	38.32	37.81
3	11,252	-0.13	30.70	35.88	32.60	31.99
4	15,370	11.39	47.98	44.15	41.70	40.91
5	78,144	11.79	69.03	42.53	41.94	40.82
6	39,494	3.91	44.35	42.11	41.07	40.72
7	118,223	-0.40	50.39	37.82	37.38	36.24
8	180,395	0.54	55.85	42.51	42.24	40.51
9	242,679	0.11		39.38	39.15	37.85
10	71,575	0.53	49.31	38.38	37.71	37.40

In Anlehnung an `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.300.9031&rep=rep1&type=pdf`

Big Data

- Kompression großer Datenmengen ist immer noch langsam, da Komprimieren ein serieller Prozess ist.
- So ist beispielsweise die englische Wikipedia unkomprimiert mit der gesamten Historie 10TB groß und komprimiert 100GB[5]
- Dies kann je nach System mehrere Stunden bis Tage in Anspruch nehmen.

5 https://en.wikipedia.org/wiki/Wikipedia:Size_of_Wikipedia

Big Data

- Geschwindigkeit der Algorithmen ist ebenfalls durch die Prozessorgeschwindigkeit begrenzt, da die meisten Algorithmen seriell sind.

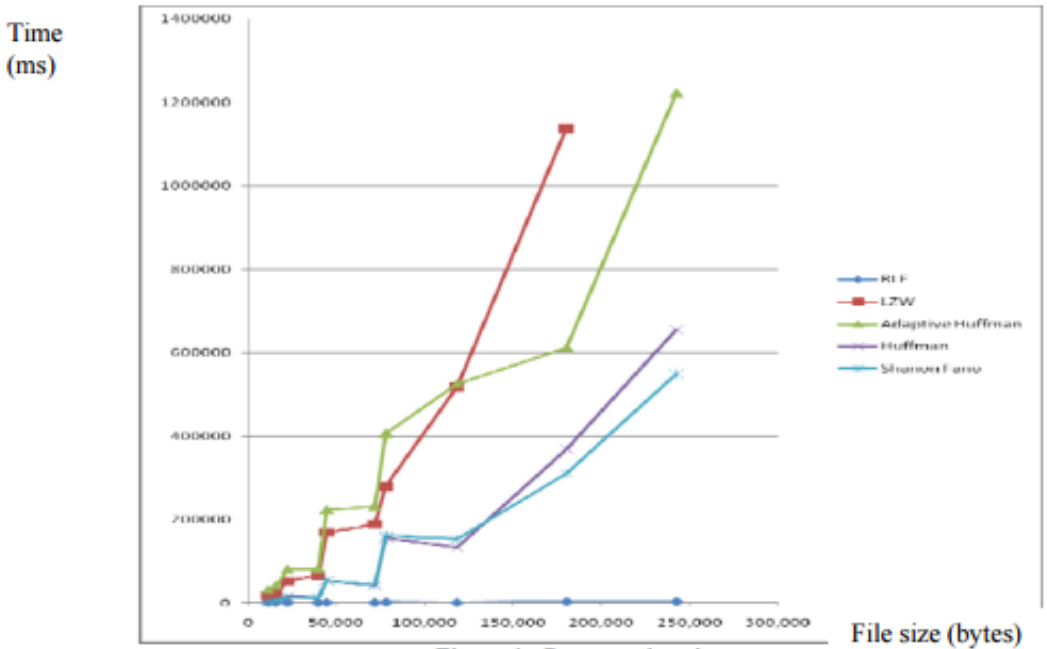


Figure 1: Compression times

---

http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.300.

Parallele Kompression als Lösung?

- Es existieren kaum parallele Implementierungen und Algorithmen.
- Meiste Implementierungen nutzen einfache Zerteilung der Daten Bsp: **plzip** (`http://www.nongnu.org/lzip/plzip.html`)
- Ein Programm für parallele Kompression ist **pigz** (`http://zlib.net/pigz/`).
- Komprimiert nach dem gzip format.

Parallele Kompression als Lösung?

- Aufteilen des Inputs in N Blöcke und dem Ausführung des Kompressionsalgorithmus auf jedem Block parallel.

  Vorteile:

  - Die Methode ist sehr einfach zu implementieren
  - Sehr hohe Parallelisierung da die Daten von einander unabhängig sind.

  Nachteile:

  - Komprimiert bei kurzen Eingaben sehr schlecht

`http://people.engr.ncsu.edu/dzbaron/research/parallel.html`

Parallele Kompression als Lösung?

- Lesen von B Datensätzen und dann das parallele Komprimieren

  Vorteile:

  - Die Methode ist sehr einfach zu implementieren
  - Sehr hohe Parallelisierung da die Daten von einander unabhängig sind.
  - Ergebnis entspricht dem einer sequenziellen Kompression.

  Nachteile:

  - Braucht aber B mal mehr Ressourcen.

`http://people.engr.ncsu.edu/dzbaron/research/parallel.html`

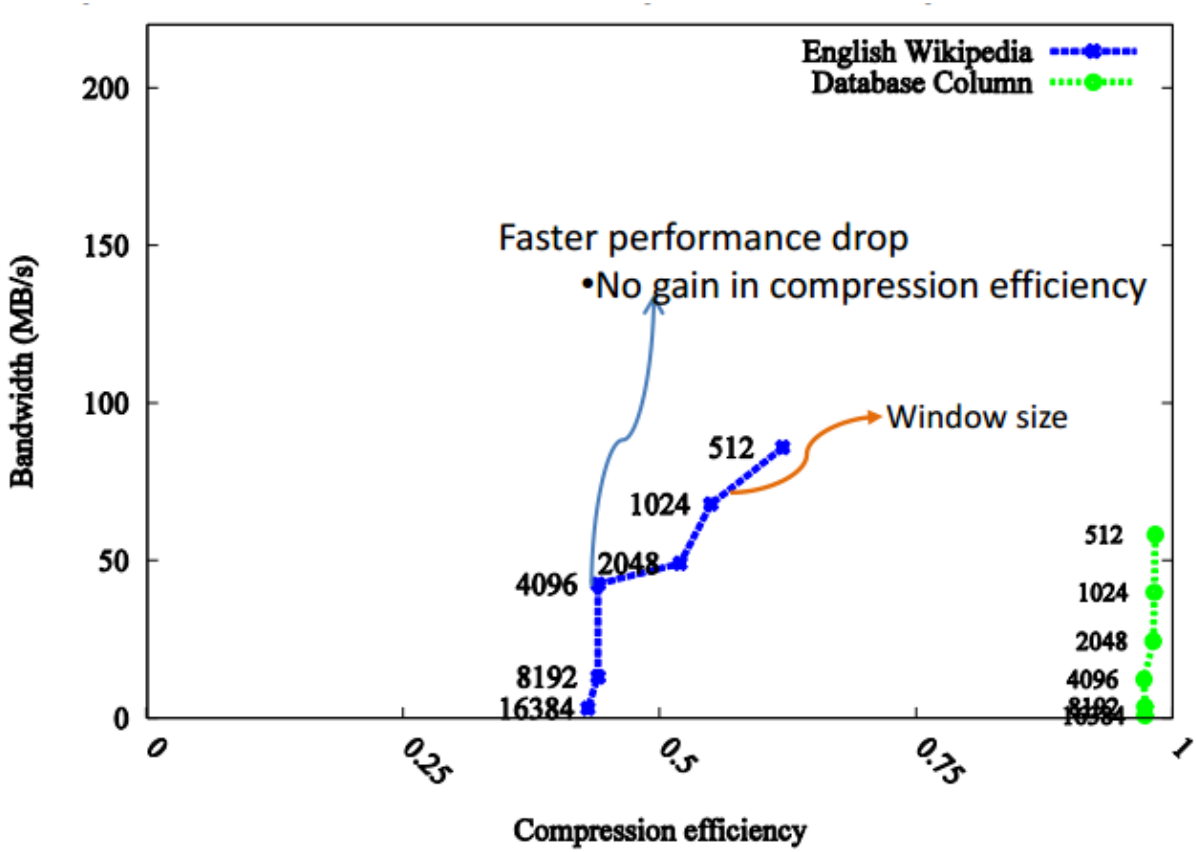
GPU Kompression als Lösung?

- Das Gebiet der GPU Kompression ist noch relativ neu und daher existieren kaum etablierte Lösungen.
- Versuch der Parallelisierung von LZSS, besonders:
  - Das Vergleichen von Eingabesequenzen mit bereits gefundenen.
  - Das Suchen von Einträgen im Dictionary.
  - Huffman encoding.

`http://www2.imm.dtu.dk/pubdb/views/edoc_download.php/6642/pdf/imm6642.pdf`

`http://on-demand.gputechconf.com/gtc/2014/presentations/S4459-parallel-lossless-compression-using-gpus.pdf`

GPU Kompression als Lösung?



---

http://on-demand.gputechconf.com/gtc/2014/presentations/S4459-parallel-lossless-compression-using-gpus.pdf

Big Data

- Hutter Prize schreibt bis zu 50,000 Euro aus wenn die ersten 100MB der englischen Wikipedia auf unter 16MB (stand 10.5.2016) komprimiert werden.
- Die Organisatoren des Wettbewerbs glauben, dass es sich bei Kompression um Problem äquivalent zum Machine Learning handelt.
- Ein idealer Kompressor muss die beste Menge der Symbole und Textketten sowie das nächste Symbol bestimmen welches mit größter Wahrscheinlichkeit kommt.

`https://en.wikipedia.org/wiki/Hutter_Prize`

`http://prize.hutter1.net/`

Literatur I

[1] Norman Hendrich. "64-040 Modul InfB-RS: Rechnerstrukturen, Kapitel 7". URL: https://tams.informatik.uni-hamburg.de/lectures/2015ws/vorlesung/rs/doc/rs-07.pdf.

[2] Henning Fernau Maciej Li'skiewicz. "Datenkompression". URL: https://www.uni-trier.de/fileadmin/fb4/prof/INF/TIN/Folien/DK/script.pdf.

[3] Steve Sullivan. *Comparison of Netcdf4 (HDF4) and Grib2 compression methods for meteorological data*. Accessed: 24.05.2016. Juni 2011. URL: https://wiki.ucar.edu/download/attachments/23364539/gridCompressionStudy-v1.pdf%3Fversion%3D1%26modificationDate%3D1317412688000.

Literatur II

[4] Wikipedia. *Data compression*. Accessed: 29-04-2016. Apr. 2016. URL: https://en.wikipedia.org/wiki/Data_compression.

[5] Wikipedia. *Data compression*. Accessed: 29-04-2016. Apr. 2016. URL: https://simple.wikipedia.org/wiki/Data_compression.