Data Compression
LZ77

Jens Müller
Universität Stuttgart
2008-11-25

Outline

- Introduction – Principle of dictionary methods
- LZ77 Sliding window
- Examples
- Optimization
- Performance comparison
- Applications/Patents

Principle of dictionary methods

- Compressing multiple strings can be more efficient than compressing single symbols only (e.g. Huffman encoding).
- Strings of symbols are added to a dictionary. Later occurrences are referenced.

- Static dictionary: Entries are predefined and constant according to the application of the text
- Adaptive dictionary: Entries are taken from the text itself and created on-the-fly

LZ77

- First paper by Ziv and Lempel in 1977 about lossless compression with an adaptive dictionary.
- Goes through the text in a **sliding window** consisting of a *search buffer* and a *look ahead buffer*.

Search buffer | Look-ahead buffer

```
…this | is a text that is being | read through | the window…
```

- The search buffer is used as dictionary
- Sizes of these buffers are parameters of the implementation. Assumption: Patterns in text occur within range of the search buffer.

LZ77 – Example (Encoding)

Encoding of the string:
`abracadabrad`

output tuple: (offset, length, symbol)

	7	6	5	4	3	2	1							output
								a	b	r	a	c	ada…	(0,0,a)
							a	b	r	a	c	a	dab…	(0,0,b)
						a	b	r	a	c	a	d	abr…	(0,0,r)
					a	b	r	a	c	a	d	a	bra…	(3,1,c)
			a	b	r	a	c	a	d	a	b	r	ad	(2,1,d)
	a	b	r	a	c	a	d	a	b	r	a	d		(7,4,d)
…ac	a	d	a	b	r	a	d							

Search buffer | Look-ahead buffer

12 characters compressed into 6 tuples

Compression rate: (12*8)/(6*(5+2+3))=96/60=1,6=60%.

Size of output

- Size for each output tuple (offset, length, symbol) when using fixed-length storage:

$$\lceil \log_2 S \rceil + \lceil \log_2 (S+L) \rceil + \lceil \log_2 A \rceil$$

where S is the length of the search buffer, L the length of the look ahead window, A the size of the alphabet.

- Why S+L and not only S? See next slide.
- Worst case if no symbol repeats in the search buffer:

$$\text{Blow up of } n\left(\lceil \log_2 S \rceil + \lceil \log_2 (S+L) \rceil + \lceil \log_2 A \rceil\right)$$

$$\text{instead of } \quad n \lceil \log_2 A \rceil$$

Encoding reaches into look-ahead buffer

Special case

	7	6	5	4	3	2	1							output
he		s	a	i	d	:		H	A	H	A	H	AHA!	(0,0,H)
he	s	a	i	d	:		H	A	H	A	H	A	HH!	(0,0,A)
…e s	a	i	d	:		H	A	H	A	H	A	H	A!	(2,4,H)
…d:	H	A	H	A	H	A	H	A	!					(2,1,!)
… HA	H	A	H	A	H	A	!							

Search buffer	Look-ahead buffer

Encoding – Pseudo code algorithm

```
while look-ahead buffer is not empty
   go backwards in search buffer to find longest match of the look-ahead buffer

   if match found
        print: (offset from window boundary, length of match, next symbol in look-
        ahead buffer);
        shift window by length+1;
   else
        print: (0, 0, first symbol in look-ahead buffer);
        shift window by 1;
   fi
end while
```

Example (Decoding)

input		7	6	5	4	3	2	1
(0,0,a)								a
(0,0,b)							a	b
(0,0,r)						a	b	r
(3,1,c)				a	b	r	a	c
(2,1,d)		a	b	r	a	c	a	d
(7,4,d)	abrac	a	d	a	b	r	a	d

Decoding – Pseudo code algorithm

```
for each token (offset, length, symbol)
   if offset = 0 then
        print symbol;
   else
        go reverse in previous output by offset characters and copy
        character wise for length symbols;
        print symbol;
   fi
next
```

LZ77 is asymmetric, encoding is more difficult than decoding as it needs to find the longest match.

Optimizations

Successors following LZ77 used different optimizations:

• Use variable size offset and length fields in the tuples instead of fixed-length. Better if small offsets and sizes prevail.

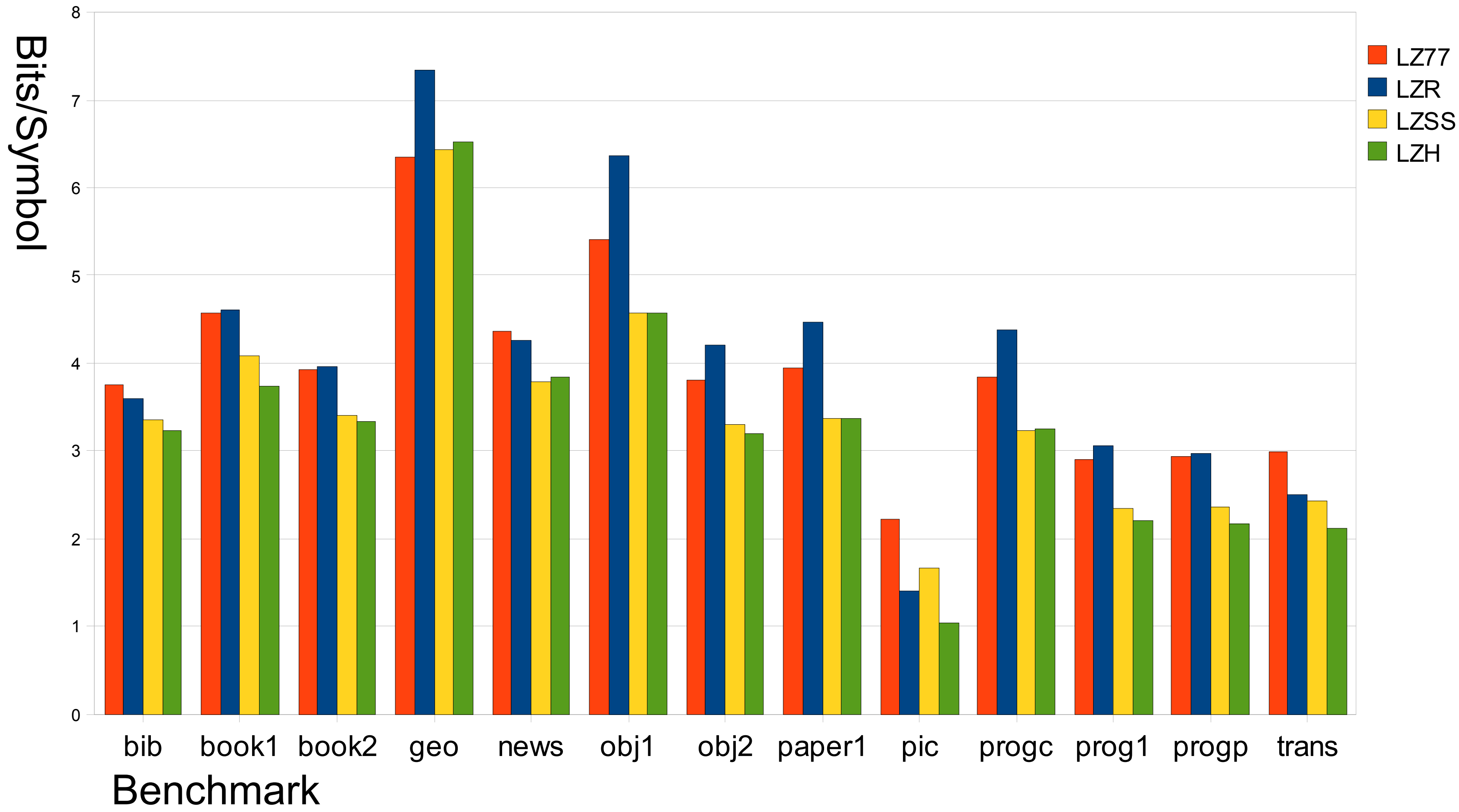
• Don‘t output a (0,0,x) token when character is not found but instead differentiate using a flag-bit: 0|x or 1|o,l

• Use better suited data structure (e.g. tree, hash set) for the buffers. This allows faster search and/or larger buffers.

• Additional Huffman coding of tuples/references.

-> LZSS, LZB, LZH, LZR, LZFG, LZMA, Deflate, …

Performance



(From Bell/Cleary/Witten: Text Compression)

Applications, Patents

Unlike **LZ78**, **LZ77** has not been patented. This may be a reason why its successors basing on LZ77 are so widely used:

Deflate is a combination of LZSS together with Huffman encoding and uses a window size of 32kB.

This algorithm is open source and used in what is widely known as ZIP compression (although the ZIP format itself is only a container format, like AVI and can be used with several algorithms), and by the formats PNG, TIFF, PDF and many others.

References

SOLOMON, D.: Data Compression, The Complete Reference., Springer, New York, 1998

BELL, T. C., CLEARY, J. G., WITTEN, I. H.: Text Compression, Prentice Hall Advanced Reference Series, 1990

SAYOOD, K.: Introduction to Data Compression, Academic Press, San Diego, CA,1996, 2000.

ZIV, J., LEMPEL, A.: A universal algorithm for sequential data compression. IEEE Transactions on Information Theory 23 (1977), 337–343.