Random Access Archives for Efficient Compression of Many Small Files

or

Avoiding the void

Robert Jan Hensing

July 8, 2011

Overview

Compression

Finding a more space-efficient way to store data.

Lempel, Ziv '77 Replace strings of symbols by references to earlier occurrences (LZ77)

Huffman Coding Use fewer bits for frequent symbols

Information Theory: compression can be optimal in a sense

Adaptive Compression

- LZ77 adapts to its input
- Huffman Coding does not: mapping of bits to symbols does not change.
  Large files: store the mapping
- Adaptive Huffman is: modify the mapping while encoding/decoding

Adaptive Compression

Advantages

- Compress any type of file
- Mixed files can be compressed too

Adaptive algorithms are great!

Are they?

- LZ77 can not refer to anything until some symbols are encoded.
- Adaptive Huffman does not know the distribution of probabilities until something is read.

Obvious solution

"Solid compression"

... and your small files are gone

Data

- Small text files
  - newspaper articles
  - source code
  - book of law
  - tweets
  - ...
- Storage efficiency
- Random access

Current Methods

Choice:

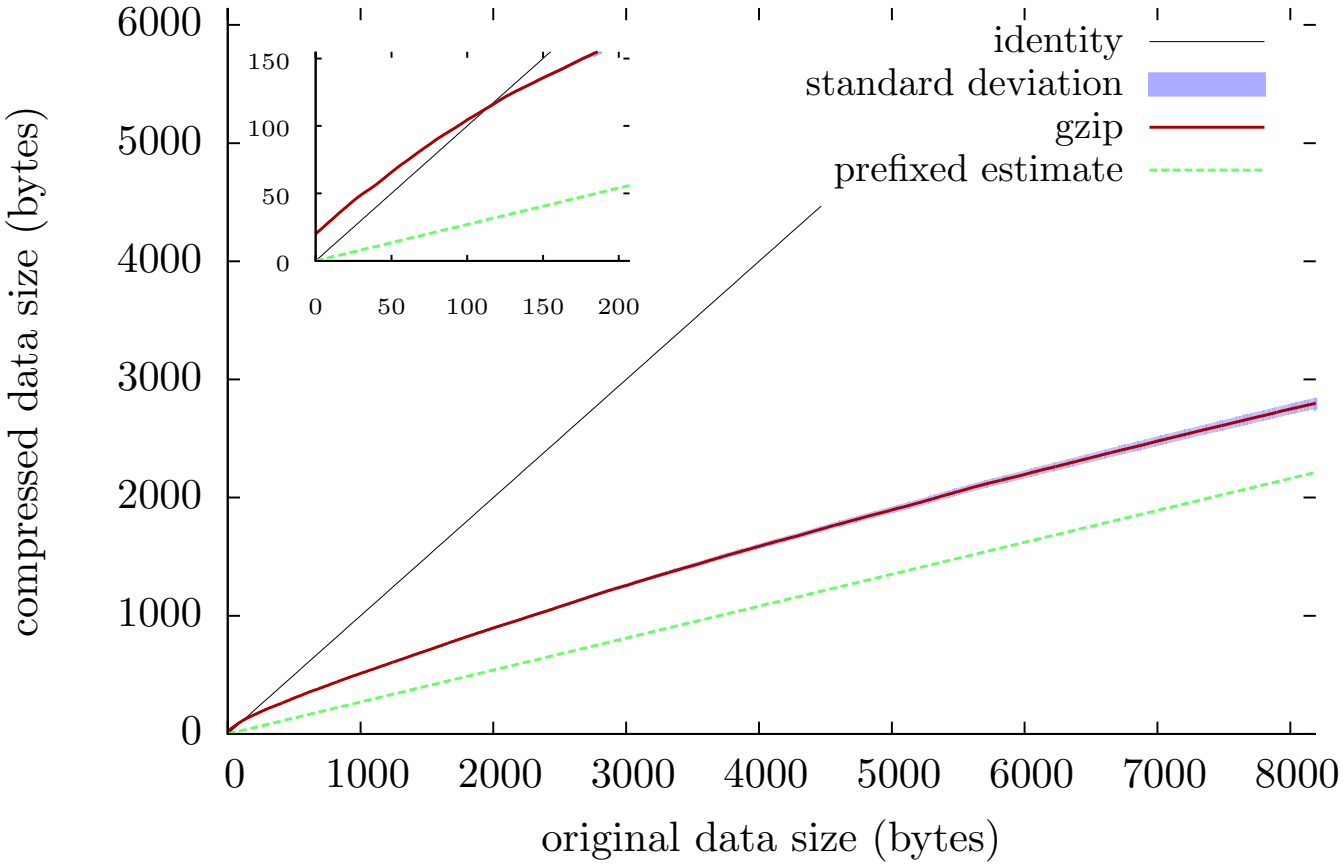
Solid compression Efficient coding, no random access

Example: back-ups and software distribution: `.tar.gz`

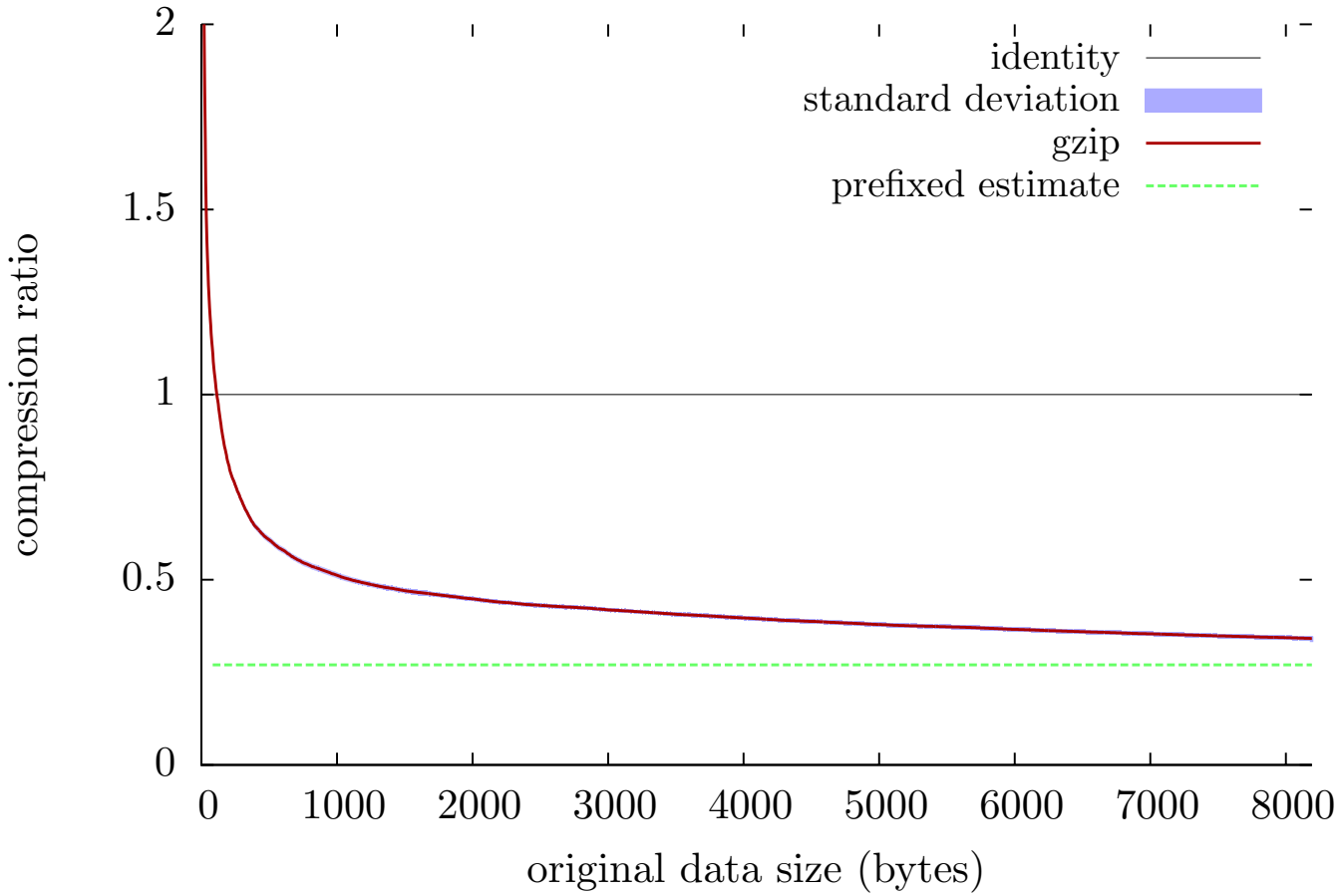
Individual compression Random access, inefficient coding of small files

Example: ZIP, used in `.zip`, `.jar`, OpenDocument, EPUB e-books.

How bad is it?


compressed data size (bytes)
original data size (bytes)
identity
standard deviation
gzip
prefixed estimate
0
1000
2000
3000
4000
5000
6000
7000
8000
0
50
100
150
200

How bad is it?


identity
standard deviation
gzip
prefixed estimate
compression ratio
original data size (bytes)
0
0.5
1
1.5
2
0
1000
2000
3000
4000
5000
6000
7000
8000

New Archiving Method

Make sure the model of the algorithm is in a suitable state.
Bad solution: group the files.

- Compression improves.
- Access time is linear in file group size; average is half group decoding time.
- Writing always requires rewriting the whole group.

New Archiving Method

Instead: generate a redundant file.

- Compression improves.
- Access time is linear in generated file size.
- Generated file can be more dense and can be size adjusted.

Deduplicating or trimming

How does increasing the input size help decrease the output size?

Deduplicating or trimming

How does increasing the input size help decrease the output size?
Condition:
At any point during compression, the output does depend on future data.
Or, more formally:

> *There exists a small constant k,*
> *such that for any soup s and file f,*
> *$c(s)$ equals $c(sf)$ up to $\#c(s) - k$ bytes.*

Archiving

- Generate Soup
- Prepend
- Compress
- Trim
- Store

Soup Generator

Counting: Consider only the number of files, not total occurrences

Locality: Most implementations favor references at a short distance

Frequency: The soup should contain all substrings that occur in at least two files.

Unicity: Any substring should be in the soup only once

Utility: Any short substring in the soup should occur in a least two input files.

Approximating Longest Common Substrings

Considering the input strings `AaaaBbbbCccc` and `BbbAaCcccc`, the longest common substrings are:

1. `Cccc`
2. `Bbb`
3. `Aa`

Approximating Longest Common Substrings

Data structure: limited depth trie

- Easily record unique substrings in a file (Counting)
- Can define merge operation
- Redundant substrings stored only once

Putting it together

- For all files: Read all fixed-length substrings of file into trie.
- Merge all tries
- For all frequent substrings (merged ≥ 1)
  - Prediction
  - Cut off
  - Reverse prediction
- Concatenate

Complexity

Time complexity $O(kn + n \log n)$.

Space complexity is $O(kn)$ in worst case.

where n is the number of input bytes and k is the depth limit
Worst space complexity: random files

Implementation

- Command line tool for archiving
- User space filesystem for access

Demo

Conclusion

- Adaptive algorithms need space to adapt to small files
- Redundancy between files can be significant and can be taken advantage of
- Redundancy between files can be modeled with a soup

Slides, paper and source code available later today
`http://roberthensing.nl/har/news.html`