Problem

- Compress
  - Text
  - Images, video, sound, ...
- Reduce space, efficient communication, etc...
- Exact compression/decompression
- Lossy compression

Links

- [http://datacompression.info/](http://datacompression.info/)
- **Data Compression** Debra A. Lelewer and Daniel S. Hirschberg  
- **Information Theory Primer With an Appendix on Logarithms** by Tom Schneider  

What it’s about?

- Elimination of redundancy
- Being able to predict...
- Compression and decompression
  - Represent data in a more compact way
  - Decompression - restore original form
- Lossy and lossless compression
  - Lossless - restore in exact copy
  - Lossy - restore almost the same information
    - Useful when no 100% accuracy needed
  - voice, image, movies, ...
  - Decompression is deterministic (lossy compression phase)
  - Can achieve much more effective results
Methods covered:

- Code words (Huffman coding)
- Run-length encoding
- Arithmetic coding
- Lempel-Ziv family (compress, gzip, zip, pkzip, ...)
- Burrows-Wheeler family (bzip2)
- Other methods, including images
- Kolmogorov complexity
- Search from compressed texts

Model

- Let $p_S$ be a probability of message $S$
- The information content can be represented in terms of bits
  - $I(S) = -\log(p_S)$ bits
- If the $p=1$ then the information content is $0$ (no new information)
  - If $P(r)=1$ then $I(S) = 0$.
  - In other words, $I$(death)=$I$(taxes)=0
- $I$(heads or tails) = 1 --- if the coin is fair
- Entropy $H(S)$ is the average information content of $S$
  - $H(S) = \sum_{x} p(x) \cdot I(x) = -\sum_{x} p(x) \cdot \log(p(x))$ bits

Shannon’s experiments with human predictors show an information rate of between 5 and 1.3 bits per character, depending on the experimental setup; the PPM compression algorithm can achieve a compression ratio of 1.5 bits per character.
No compression can on average achieve better compression than the entropy. Entropy depends on the model (or choice of symbols). Let M(s_1, s_2, ..., s_n) be a set of symbols of the model A and let p(m) be the probability of the symbol m. The entropy of the model A, H(M) is \( -\sum_{i=1}^{n} p(m_i) \cdot \log(p(m_i)) \) bits. Let the message S = s_1s_2..., s_n, and every symbol s_i be in the model M. The information content of model A is \( -\sum_{i=1}^{n} \log p(s_i) \). Every symbol has to have a probability, otherwise it cannot be coded if it is present in the data.

**Static or adaptive**
- Static model does not change during the compression.
- Adaptive model can be updated during the process.
- Symbols not in message cannot have 0-probability.
- Semi-adaptive model works in two stages, off-line.
- First create the code table, then encode the message with the code table.

**How to compare compression techniques?**
- Ratio (t/p): original message length
- p: compressed message length
- In texts - bits per symbol
- The time and memory used for compression
- The time and memory used for decompression
- Error tolerance (e.g. self-correcting code)

**Run-length encoding**
- The string: “aaaabccccdeeefgggggggg”
- Alphabet of 8
- Length = 40 symbols
- Equal length codewords
- 3-bit a 000 b 001 c 010 d 011 e 100 f 101 g 110 space 110
- S compressed - 3*40 = 120 bits

\[ S = 'aa bbb ccccc ddddd eeeeee fffffffffffffffffggggggggg' \]

- Alphabet of 8
- Length = 40 symbols
- Equal length codewords
- 3-bit a 000 b 001 c 010 d 011 e 100 f 101 g 110 space 110
- S compressed - 3*40 = 120 bits
### Coding the alphabetically ordered word-lists

| resume | 0resume |
| retail | 2tail   |
| retain | 5n      |
| retard | 4rd     |
| retire | 3ire    |

### Coding techniques

- Coding refers to techniques used to encode tokens or symbols.
- Two of the best known coding algorithms are **Huffman Coding** and **Arithmetic Coding**.
- Coding algorithms are effective at compressing data when they use fewer bits for high probability symbols and more bits for low probability symbols.

### Variable length encoders

- How to use codes of variable length?
- Decoder needs to know how long is the symbol

- **Prefix-free code**: no code can be a prefix of another code

### Calculate the frequencies and probabilities of symbols:

<table>
<thead>
<tr>
<th>symbol</th>
<th>freq</th>
<th>ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>2</td>
<td>0.05</td>
</tr>
<tr>
<td>b</td>
<td>3</td>
<td>0.075</td>
</tr>
<tr>
<td>c</td>
<td>4</td>
<td>0.1</td>
</tr>
<tr>
<td>d</td>
<td>5</td>
<td>0.125</td>
</tr>
<tr>
<td>space</td>
<td>5</td>
<td>0.125</td>
</tr>
<tr>
<td>e</td>
<td>6</td>
<td>0.15</td>
</tr>
<tr>
<td>f</td>
<td>7</td>
<td>0.175</td>
</tr>
<tr>
<td>g</td>
<td>8</td>
<td>0.2</td>
</tr>
</tbody>
</table>

### Algorithm Shannon-Fano

- Input: probabilities of symbols
- Output: Codewords in prefix free coding

1. Sort symbols by frequency
2. Divide to **two almost probable groups**
3. First group gets prefix 0, other 1
4. Repeat recursively in each group until 1 symbol remains

### Example 1

<table>
<thead>
<tr>
<th>symbol</th>
<th>probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1/2</td>
</tr>
<tr>
<td>b</td>
<td>1/4</td>
</tr>
<tr>
<td>c</td>
<td>1/8</td>
</tr>
<tr>
<td>d</td>
<td>1/16</td>
</tr>
<tr>
<td>e</td>
<td>1/32</td>
</tr>
<tr>
<td>f</td>
<td>1/32</td>
</tr>
</tbody>
</table>
Example 1

Code:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Frequency</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1/2</td>
<td>0</td>
</tr>
<tr>
<td>b</td>
<td>1/4</td>
<td>10</td>
</tr>
<tr>
<td>c</td>
<td>1/8</td>
<td>110</td>
</tr>
<tr>
<td>d</td>
<td>1/16</td>
<td>1110</td>
</tr>
<tr>
<td>e</td>
<td>1/32</td>
<td>11110</td>
</tr>
<tr>
<td>f</td>
<td>1/32</td>
<td>11111</td>
</tr>
</tbody>
</table>

Shannon-Fano

\[ S = 'aa bbb cccc dddd eeeeee fffffffggggggg' \]

\[
\begin{array}{c|c|c}
\text{symbol} & p(s) & \text{code} \\
\hline
g & 0.2 & 00 \\
f & 0.175 & 010 \\
e & 0.15 & 011 \\
d & 0.125 & 100 \\
\text{space} & 0.125 & 101 \\
c & 0.1 & 110 \\
b & 0.075 & 1110 \\
a & 0.05 & 1111 \\
\end{array}
\]

Shannon-Fano

- \( S = 'aa bbb cccc dddd eeeeee fffffffggggggg' \)
- \( S \) in compressed is 117 bits
- \( 2^4 + 3^4 + 4^3 + 5^3 + 5^3 + 6^3 + 7^3 + 8^2 = 117 \)
- Shannon-Fano not always optimal
- Sometimes 2 equal probable groups cannot be achieved
- Usually better than H+1 bits per symbol, when H is entropy.

Huffman code

- Works the opposite way.
- Start from least probable symbols and separate them with 0 and 1 (suffix)
- Add probabilities to form a "new symbol" with the new probability
- Prepend new bits in front of old ones.

Huffman example

- Huffman coding is optimal when the frequencies of input characters are powers of two. Arithmetic coding produces slight gains over Huffman coding, but in practice these gains have not been large enough to offset arithmetic coding’s higher computational complexity and patent royalties
- (as of November 2001/Jul2006, IBM owns patents on the core concepts of arithmetic coding in several jurisdictions).

Properties of Huffman coding

- **Error tolerance** quite good
- In case of the loss, adding or change of a single bit, the differences remain local to the place of the error
- Error usually remains quite local (proof?)
- Has been shown, the code is optimal
- Can be shown the average result is \( H + p + 0.086 \), where \( H \) is the entropy and \( p \) is the probability of the most probable symbol

Move to Front

- Move to Front (MTF), Least recently used (LRU)
- Keep a list of last \( k \) symbols of \( S \)
- Code
  - use the code for symbol.
  - if in codebook, move to front
  - if not in codebook, move to first, remove the last
- c.f. the handling of memory paging
- Other heuristics ...

Arithmetic (en)coding

- Arithmetic coding is a method for lossless data compression.
- It is a form of entropy encoding, but where other entropy encoding techniques separate the input message into its component symbols and replace each symbol with a code word, arithmetic coding encodes the entire message into a single number, a fraction \( n \) where \( (0.0 < n < 1.0) \).
- Huffman coding is optimal for character encoding (one character one code word) and simple to program. Arithmetic coding is better still, since it can allocate fractional bits, but more complicated.
- Every symbol gets a probability based on the model
- Probabilities represent non-intersecting intervals
- Every text is such an interval

Let \( P(A)=0.1, P(B)=0.4, P(C)=0.5 \)

- A \([0.1, 0.2)\)
- AA \([0.0, 0.01)\)
- AB \([0.01, 0.05)\)
- AC \([0.05, 0.1)\)
- B \([0.1, 0.5)\)
- BA \([0.1, 0.14)\)
- BB \([0.14, 0.3)\)
- BC \([0.3, 0.5)\)
- C \([0.5, 1)\)
- CA \([0.5, 0.55)\)
- CB \([0.55, 0.75)\)
- CC \([0.75, 1)\)

- Add a EOF symbol.
- Problem with infinite precision arithmetics
- Alternative - blockwise, use integer-arithmetic
- Works, if smallest \( p \) not too small
- Best ratio
- Problem - the speed, and error tolerance, small change has catastrophic effect
• *Arithmetic coding revisited* by Alistair Moffat, Radford M. Neal, Ian H. Witten - [http://portal.acm.org/citation.cfm?id=874788](http://portal.acm.org/citation.cfm?id=874788)

• Models for arithmetic coding
• HMM Hidden Markov Models
• ...
• Context methods: Abrahamson dependency model
• Use the context to maximum, to predict the next symbol
• PPM - Prediction by Partial Matching
• Several contexts, choose best
• Variations

• Dictionary based
  • Dictionary (symbol table), list codes
  • If not in dictionary, use escape
  • Usual heuristics searches for longest repeat
  • With fixed table one can search for optimal code
  • With adaptive dictionary the optimal coding is NP-complete
  • Quite good for English language texts, for example

**Lempel-Ziv, LZ, LZ-77**

• Use the dictionary to memorise the previously compressed parts
• LZ-77 pure
• Sliding window of previous text; and text to be compressed
  /bbaaabbbababbaa ... abbaabab ...
• **Lookahead** - longest prefix that begins within the moving window, is encoded with [position,length]
  • In example, [5,6]
• **Fast!**  (Commercial software, e.g. PKZIP, Stacker, DoubleSpace, )
• Several alternative codes for same string (alternative substrings will match)
• LZ78 compression from McGill Univ.

**Original LZ77**

• **Triples** [position,length,next char]
• If output [a,b,c], advance by b+1 positions
• For each part of the triple the nr of bits is reserved depending on window length
  \[ \log(n-f) + |\log(f)| + 8 \]
  where n is window size, and f is lookahead size
• Example: abbbabbc [0,0,a] [0,0,b] [1,3,a] [3,2,c]
• In example the match actually overlaps with lookahead window

**LZ-78**

• **Dictionary**
  • Store strings from processed part of the message
  • Next symbol is the longest match from dictionary, that matches the text to be processed
• LZ78 (Ziv and Lempel 1978)
• First, dictionary is empty, with index 0
• Code [i,c] - refers to dictionary (word u at pos. i) and c is the next symbol
• Add uc to dictionary
• Example:  abababc \(\rightarrow [0,a][0,b][1,b][0,c][3,c]\)

**LZW**

• Code consists of indices only!
  • First, dictionary has every symbol /alphabet/
  • Update dictionary like LZ78
  • In decoding there is a danger: See ababca
    • if abc is in dictionary
      • add abc to dictionary
    • next is ab, output that code
  • But when decoding, after abc it is not known that abc is in the dictionary
• Solution: if the dictionary entry is used immediately after its creation, the 1st and last characters have to match
• Many representations for the dictionary
  • List, hash, sorted list, combination, binary tree, trie, suffix tree, ...
**LZJ**

- Coding - search for longest prefix.
- **Code - address of the trie node**
- From the root of the trie, there is a transition on every symbol (like in LZW).
- If out of memory, remove these nodes/branches that have been used only once
- In practice, $h=6$, dictionary has 8192 nodes

**LZFG**

- Effective LZ method
- From LZJ
- Create a suffix tree for the window
- Code - node address plus nr opf characters from teh edge.
- The internal and leaf nodes with different codes
- small match directly… (?)

**Burrows-Wheeler**

- The method described in the original paper is really a composite of three different algorithms:
  - the block sorting main engine (a lossy, very slightly expansive preprocessor)
  - the move-to-front coder (a byte-per-byte simple, fast, locally adaptive noncompressive codec)
  - a simple statistical compressor (first order Huffman is mentioned as a candidate) eventually doing the compression.
- Of these three methods only the first two are discussed here as they are what constitutes the heart of the algorithm. These two algorithms combined form a completely reversible (lossless) transformation that - with typical input - slants the first-order symbol distributions to make the data more compressible with simple methods. Intuitively speaking, the method transforms slack in the higher order probabilities of the input block (thus making them more even, whitening them) to slack in the lower order statistics. This effect is what is seen in the histogram of the resulting symbol data.
- Please, read the article by Mark Nelson:
CODE:
- hat acts like this:<13><10><1
- hat buffer to the constructor
- hat corrupted the heap, or wo
- W: hat goes up must come down:<13><1
- t: hat happens, it isn’t. Likely
- w: hat if you want to dynamical
- t: hat indicates an error.<13><1
- t: hat it removes arguments from
- t: hat looks like this:<13><10><1
- t: hat looks something like this
- t: hat looks something like this
- t: hat once I detect the mangled

Syntactic compression
- Context Free Grammar for presenting the syntax tree
- Usually for source code
- Assumption - program is syntactically correct
- Comments
- Features, constants - group by group

Image compression
- Many images, photos, sound, video, ...

Fax group 3
- Fax/group 3
- Black/white, 0/1 code
- Run-length: 000111001000 → 3,3,2,1,3
- Variable-length codes for representing run-lengths.
• Joint Photographic Experts Group JPEG 2000
  http://www.jpeg.org/jpeg2000/
• Color image, 8 or 12 bits per pixel per color.
• Four modes Sequential Mode
• Lossless Mode
• Progressive Mode
• Hierarchical Mode
• DCT (Discrete Cosine Transform)
• from http://www.utdallas.edu/~wrl/mcl/post/
• Lossy signal compression works on the basis of transmitting the "important" signal content, while omitting other parts (Quantization). To perform this quantization effectively, a linear de-correlating transform is applied to the signal prior to quantization. All existing image and video coding standards use this approach. The most commonly used transform is the Discrete Cosine Transform (DCT) used in JPEG, MPEG-1, MPEG-2, H.261 and H.263 and its descendants. For a detailed discussion of the theory behind quantization and justification of the usage of linear transforms, see reference [1] below.
• A brief overview of JPEG compression is as follows. The JPEG encoder partitions the image into 8x8 blocks of pixels. To each of these blocks it applies a 2-dimensional DCT. The transform matrix is normalized (element-wise) by a 8x8 quantization matrix and then rounded to the nearest integer. This operation is equivalent to applying different uniform quantizers to different frequency bands of the image. The high-frequency image content can be quantized more coarsely than the low-frequency content, due to two factors.
• 19. Compression/lena/

Vector quantization
• Vector quantization
• Dictionary-meetod
• 2-dimensional blocks

Discrete cosine transform
• A discrete cosine transform (DCT) expresses a sequence of finitely many data points in terms of a sum of cosine functions oscillating at different frequencies. DCTs are important to numerous applications in science and engineering, from lossy compression of audio and images (where small high-frequency components can be discarded), to spectral methods for the numerical solution of partial differential equations. The use of cosine rather than sine functions is critical in these applications: for compression, it turns out that cosine functions are much more efficient (as explained below, fewer are needed to approximate a typical signal), whereas for differential equations the cosines express a particular choice of boundary conditions.
• http://en.wikipedia.org/wiki/Discrete_cosine_transform

2d DCT (type II) compared to the DFT
For both transforms, there is the magnitude of the spectrum on left and the histogram on right; both spectra are cropped to 1/4, to zoom the behaviour in the lower frequencies. The DCT concentrates most of the power on the lower frequencies.
• In particular, a DCT is a Fourier-related transform similar to the discrete Fourier transform (DFT), but using only real numbers. DCTs are equivalent to DFTs of roughly twice the length, operating on real data with even symmetry (since the Fourier transform of a real and even function is real and even), where in some variants the input and/or output data are shifted by half a sample. There are eight standard DCT variants, of which four are common.

• Digital Image Processing: [http://www.ee.uta.edu/dip/](http://www.ee.uta.edu/dip/)
**JPEG Compression (Q=75%)**

45 KB, compression ratio ~ 4:1

**JPEG Compression (Q=75% & 30%)**

45 KB 22 KB

From Liu's EE330 (Princeton)

**1.4-billion-pixel digital camera**

- **Monday, November 24, 2008**
  - [http://www.technologyreview.com/computing/21705/page1/](http://www.technologyreview.com/computing/21705/page1/)
- **Giant Camera Tracks Asteroids**
  - The camera will offer sharper, broader views of the sky.
  - The focal plane of each camera contains an almost complete 64 x 64 array of CCD devices, each containing approximately 600 x 600 pixels, for a total of about 1.4-gigapixels. The CCDs themselves employ the innovative technology called "orthogonal transfer", which is described below.

**Fractal compression**

- Fractal Compression group at Waterloo
  - [http://links.uwaterloo.ca/fractals.home.html](http://links.uwaterloo.ca/fractals.home.html)
- A "Hitchhiker's Guide to Fractal Compression" For Beginners
  - [ftp://links.uwaterloo.ca/pub/Fractals/Papers/Waterloo/vr95.pdf](ftp://links.uwaterloo.ca/pub/Fractals/Papers/Waterloo/vr95.pdf)
- Encode using fractals.
- Search for regions that with a simple transformation can be similar to each other.
- Compression ratio 20:80

- Moving Pictures Experts Group (http://www.chiariglione.org/mpeg/)
- MPEG Compression:
  - [http://www.cs.cf.ac.uk/Dave/Multimedia/node255.html](http://www.cs.cf.ac.uk/Dave/Multimedia/node255.html)
- Screen divided into 256 blocks, where the changes and movements are tracked
- Only differences from previous frame are shown
- Compression ratio 50-100
**Kolmogorov (or Algorithmic) complexity**

- Kolmogorov, Chaitin, ...
- What is the compressed version of sequence
  '1234567891011121314151617181920212223242526...'?
- Every symbol appears almost equally frequently, almost "random" by entropy
- for i=1 to n do print i;
- **Algorithmic complexity** (or Kolmogorov complexity) for string
  S is the length of the shortest program that reproduces S,
  often noted K(S)
- Conditional complexity : K(S|T). Reproduce S given T.

---

**G J Chaitin**

- [http://www.cs.umaine.edu/~chaitin/](http://www.cs.umaine.edu/~chaitin/)

---

**Distance using K.**

- $d(S,T) = \left( K(S|T) + K(T|S) \right) / \left( K(S) + K(T) \right)$
- We cannot calculate $K$, but we can approximate it
- E.g. by compression LZ, BWT, etc
- $d(S,T) = \left( C(ST) + C(TS) \right) / \left( C(S) + C(T) \right)$

---

**Algorithmic information theory** is a field of study which attempts to capture the concept of complexity using tools from theoretical computer science. The chief idea is to define the complexity (or Kolmogorov complexity) of a string as the length of the shortest program which, when run without any input, outputs that string. Strings that can be produced by short programs are considered to be not very complex. This notion is surprisingly deep and can be used to state and prove impossibility results akin to Gödel's incompleteness theorem and Turing's halting problem.

- The field was developed by Andrey Kolmogorov and Gregory Chaitin starting in the late 1960s.

---

**Use of Kolmogorov Distance Identification of Web Page Authorship, Topic and Domain**

David Parry (PPT) in [OSWIR 2005](http://en.wikipedia.org/wiki/OSWIR), 2005 workshop on Open Source Web Information Retrieval

- [Informatstionikaagus](http://www.cs.technion.ac.il/~mperry1/2003/OSWIR/seminar/04main.pdf) by Mart Sõmermaa, Fall 2003 (in Data Mining Research seminar)

---

*When one compresses files, can we still use fast search techniques without decompressing first?*

- Sometimes, yes
- *e.g.* Udi Manber has developed a method
- Approximate Matching of Run-Length Compressed Strings
  Vel Mäkinen, Gonzalo Navarro, Esko Ukkonen.

* We focus on the problem of approximate matching of strings that have been compressed using run-length encoding. Previous studies have concentrated on the problem of computing the longest common subsequence (LCS) between two strings of length $m$ and $n$, compressed to $m^2 \ln n$ and $n^2 \ln m$ runs. We extend an existing algorithm for the LCS to the Levenshtein distance, achieving $O(m \ln m)$ complexity. Furthermore, we extend this algorithm to a weighted edit distance model, where the weights of the three basic edit operations can be chosen arbitrarily. This approach also gives an algorithm for approximate searching of a pattern of $m$ letters in a text of $n$ letters in $O(m \ln m)$ time. Then we propose improvements for a greedy algorithm for the LCS, and conjecture that the improved algorithm has $O(m^2)$ expected case complexity. Experimental results are provided to support the conjecture.