The Hutter Prize: Getting Paid for Compression

Alex Fraser

January 8, 2026

Overview

What is the Hutter Prize?

- A competition for data compression on a fixed dataset
- Aims to advance general intelligence via compression
- Named after Marcus Hutter

Why Compression?

- Compression and prediction are closely linked
- Good compressors find patterns in data
- Hutter Prize motivates progress on universal compression

Origins of the Prize

- Proposed by Marcus Hutter in 2006, 50.000 Euros, 500 Euros per one percent compression
- Inspired by Solomonoff Induction and Hutter's AIXI
  - Solomonoff's research program is related to the Kolgorov complexity, the complexity of the shortest program to output a string
- Dataset chosen: English Wikipedia dump (enwik8)
  - A 100MB dataset from Wikipedia
  - Reasonably clean, real-world text

Prize Rules (Very Simplified!)

- Compress enwik8 to smallest size
  - Compression must be reasonably fast (on old hardware). No GPUs
  - Compression can only use limited RAM, limited disk space
- Single file created must be a decompressor binary (windows or linux) which can reconstruct the exact original text
  - No cheating (access network, access anything other than input/output, etc)
- Prize awarded as better results appear
- Updated later to be enwik9 (1 GB), prize is now based on 500.000 Euros
- See results here: `http://prize.hutter1.net/`

Basics of Compression

- Remove redundancy in data
- Two types: Lossless and Lossy
- Lossy compression you use: JPEG
- Hutter Prize requires lossless compression (as does zip, gzip, etc)

Entropy and Information

- Shannon entropy measures minimum coding length
- Lower entropy $\rightarrow$ higher compression potential
- Random strings cannot be compressed!
- Real text exhibits patterns to exploit

Common Techniques in Compression

- Statistical modeling (e.g., PPM, CTW)
- Dictionary methods (LZ77, LZ78 variants)
- Transform + entropy coding (BWT + arithmetic)

Modeling vs Coding

- Coding: Arithmetic / range coding
- Modeling: Predict next symbol distribution
- Better models $\rightarrow$ better compression
- IMPORTANT: coding is a *solved* problem. We are getting extremely close to Shannon entropy
  - Programming it well can be a bit tricky though

The Challenge of General Text

- Natural language has complex structure
- Requires capturing long dependencies
- Standard compressors struggle compared with machine learning models

Machine Learning in Compression

- Neural networks can learn text patterns
- RNNs / LSTMs / Transformers used as predictors
- Ideally, combine these with arithmetic coding
- But think about the Hutter Prize rules: we need a small model + payload

Modern Hutter Compressors

- Use an ensemble of predictors
- Looks a bit like a standard NN and/or a mixture of experts, but simpler
- *Adaptive* context models are needed to win!

ML-based Predictive Coding

- Train model to predict next bytes
- Use predicted distribution for arithmetic encoder
- Better predictions = fewer bits

Adaptive Models

- Combine classical models with sinple machine learning models
- Switcher frameworks to pick best predictor per context
- Meta-models adjust weights on the fly

Practical Hacks in Winning Solutions, somewhat OK

- Adaptive context mixing
- Byte-level tokenization

Practical Hacks in Winning Solutions, UGLY!

- Cache and memory optimizations
- Preprocessing that reorders data to get better compression
- Heavy reliance on very many different predictors
- Removal of SOA compression code for, e.g., compressing images
  - Images are very different from text
  - One big problem is (even simple) noise

Limitations and Challenges

- Computationally expensive
- Memory constraints
- Diminishing returns as models grow

Conclusions, Positive

- Hutter Prize is both practical and philosophical
  - Pushes compression research forward, very quantitative
  - Encourages generalizable AI techniques
  - Linked to theoretical ideas of intelligence but extremely quantitative (versus, e.g., improving ChatGPT, Llama, Apertus or whatever)
- Compression is central to learning and prediction
- Best solutions currently use simple ML + clever engineering
- It is very possible that better solutions will feedback to better machine learning
- But...

fx-cmix2, currently best solution

Mods over fx-cmix, 4 items taken directly from
`https://github.com/kaitz/fx2-cmix/blob/main/README.md`

- NLP (Natural language processing)
- Online reverse dictionary transform
- Single pass wikipedia transform
- Updated order of articles

NLP involves simple part-of-speech, stemming, stop words

Conclusions, Negative

- Deep semantic understanding costs model complexity, not paid back in bits saved (think of discourse structure)
- Wikipedia is mostly fairly predictable (just as news articles are fairly predictable after the headline and first sentence)
- Winning systems often exploit XML structure, other formatting regularities, English Wikipedia conventions
- We don't know how winning systems are really winning. Are they learning syntax? Semantics? Or just formatting?
- It must be possible to engineer systems that are heavily overfit to enwik9 (note that PAQ8 does not appear to have this problem, but more recent entries probably do)
- Overall: is a system that does better really more "intelligent"?

Further reading

- Hutter Prize: `http://prize.hutter1.net/`
- Matt Mahoney (2013). Data Compression Explained. Free online book (html and epub)
- Byron Knoll (2011). A Machine Learning Perspective on Predictive Coding with PAQ8 and New Applications. Masters Thesis UBC (supervised by Nando de Freitas, see also joint conference paper)
- Grégoire Delétang et al. (2024). Language modeling is compression. ICLR
- Kaido Orav and Byron Knoll (2024), fx2-cmix (best current Hutter solution): `https://github.com/kaitz/fx2-cmix`
- See also my simple example at the very end of this slide set

Questions?

- Questions?

Thank you!

- Thank you!

Coding with Huffman vs Arithmetic

Alex Fraser*, TUM

Encoding "aaaaaaaaaab<EOS>" - Uniform vs High p(a) Distribution

Uniform Distribution

Probabilities:

- p(a) = 1/3 ≈ 0.333
- p(b) = 1/3 ≈ 0.333
- p(<EOS>) = 1/3 ≈ 0.333

Huffman Encoding

Code assignment:

- 'a': `0` (1 bit)
- 'b': `10` (2 bits)
- '<EOS>': `11` (2 bits)

Encoding "aaaaaaaaaab<EOS>":

0000000000 10 11

Result: 14 bits total

Arithmetic Encoding

Interval refinement:

- Start: [0, 1)
- After 10 a's: [0, (1/3)^10) = [0, 1/59049) ≈ [0, 0.0000169...)
- After 'b': [(1/3)^10, 2×(1/3)^11) ≈ [0.0000169..., 0.0000226...)
- After '<EOS>': [(1/3)^10 + (1/3)^11, (1/3)^10 + 2×(1/3)^11) ≈ [0.0000188..., 0.0000226...)

We need about 16-17 bits to specify this narrow interval.

Result: ~17 bits total

Theoretical entropy: $12 \times \log_2(1/3) \approx$ **19.0 bits**

High p(a) Distribution

Probabilities:

- p(a) = 0.7
- p(b) = 0.2
- p(<EOS>) = 0.1

Huffman Encoding

Code assignment:

- 'a': `0` (1 bit)
- 'b': `10` (2 bits)
- '<EOS>': `11` (2 bits)

Encoding "aaaaaaaaaab<EOS>":

0000000000 10 11

Result: 14 bits total

Arithmetic Encoding

Interval refinement:

- Start: [0, 1)
- After 10 a's: [0, 0.7^10) = [0, 0.0282475...)
- After 'b': [0.0197732..., 0.0254207...)
- After '<EOS>': [0.0249553..., 0.0254207...)

Using `000001101` = 0.0253906... (9 bits)

Result: 9 bits total

Theoretical entropy: $-[10 \times \log_2(0.7) + \log_2(0.2) + \log_2(0.1)] \approx$ **8.77 bits**

Summary

Distribution	Huffman	Arithmetic	Entropy
Uniform	14 bits	17 bits	19.0 bits
High p(a)	14 bits	9 bits	8.77 bits

Key insight: When 'a' is highly probable, arithmetic encoding achieves much better compression (9 vs 14 bits) by efficiently encoding the frequent symbol with fractional bits.

Important: arithmetic coding can sometimes appear to "beat" the limit for a specific, highly repetitive string because it treats the entire message as one unit, whereas the Shannon limit is an average expectation for all possible messages from that source!

(*Use of AI: this was mostly computed using Claude since ChatGPT completely failed, but I actually had to discuss the last point in italics with Gemini because Claude failed. Remaining errors are of course my responsibility!)