Updates and Corrections to

Huffman Coding

ACM Computing Surveys, 52(4):85.1–85.35, June 2019

Page 1: Typo

The tenth line of the introduction should say "more than 7,500 citations", rather than "more the 7,500 citations". Thanks to Yufei Zhang (University of Melbourne) for spotting that one.

Page 31,32: ANS flexibility

I wrote:

> *The ANS approach cannot be used where adaptive probability estimates and multi-context models are in operation, because the symbols are regenerated by the decoder in the reverse order that they are processed by the encoder.*

The statement and corresponding entries in Table 7 are correct if the encoder and decoder pair being implemented must operate in an strictly on-line manner, that is, if the encoder must start emitting output bits/bytes just as soon as it has seen sufficient of the input to allow bits/bytes to be generated; and if similarly the decoder, upon receipt of sufficient input bits/bytes, must start emitting decoded source symbols just as soon as those bytes/bits have been received. For example, this level of responsiveness may be required if the encoder and decoder are to operate in some kind of latency-guaranteed stream mode, one at each end of a communications channel.

But in practice, even in a dynamic system designed to be used for data streams, the implementation is likely to operate over blocks/buffers of source data, and also over blocks/buffers of compressed data; and in that case the following operational arrangement is possible with an ANS coder:

- The encoder reads and processes the source data as a sequence of blocks.
- The encoder processes each block from left to right, building an adaptive/multi-context model, as required.
- But instead of updating the coding *state* as each "coding decision" is made, and thereby causing output bits/bytes to be generated, the encoder stacks the coding parameters for each and every decision (there may be multiple decisions per symbol), each requiring a current value of m, a current $base[s]$ value, and a current $W[s]$ value) into a FILO queue (a stack), and defers the actual coding operations.
- After each symbol's coding decisions have been added to the FILO queue, the encoder's model structure can be altered (new contexts created, or a new context selected as the starting point for the next input symbol), and/or updates made to $W[s]$ and hence $base[\cdot]$ and m, in much the same way as is done for arithmetic coding, and with the same algorithmic overheads in terms of tracking evolving symbol frequency counts.

- At the end of that block of data, the coding operations that have been stacked are replayed in FILO order, and committed to the output stream via the ANS arithmetic computation. This results in the coding decisions – whatever they may have been – associated with first symbol in the block being the last ones committed to the compressed bit/byte-stream.

- Upon receipt of that entire block's worth of compressed data, the decoder processes it from right to left, decoding in the usual ANS manner.

- The first coding decisions processed by the decoder are the first that were made by the encoder, and hence the first symbol decoded is the first symbol of the block.

- As each coding decision is recovered by the decoder, it can be used to mimic the model changes already enacted in the encoder. That is, after decoding each symbol, the decoder is able to update its view of $W[s]$ and hence of the *base*$[\cdot]$ and m values being used, and/or alter the structure of the coding model (create another state, or switch to another state, for example) in synchrony with what the encoder did while it was processing the block.

- The final model that has been constructed by the encoder as a result of processing the k th input block can be used as the initial model when commencing processing the $k+1$ th input block, provided that the decoder knows to do likewise. That is, long-term adaptivity and model evolution can be maintained across block boundaries.

- Because the ANS coding state must be flushed at the end of each block, and restarted for the next block, the blocks should be large enough that the per-block ANS overheads don't have a notable impact on overall compression effectiveness. That is, when used for stream compression, there may be a tension introduced between latency (that is, the time from when a symbol arrives at the encoder, and when it is successfully emitted by the decoder after being transmitted through the communications channel) and effectiveness.

Thanks to Jarosław Duda for drawing this approach to my attention. Jarosław has also pointed at a list of ANS-based compression implementations[1]; and a site at which compression effectiveness and efficiency are compared across a range of entropy coder implementations[2].

Page 32: People

I have somewhat carelessly given an incorrect name, and "Franz Giesen" should be "Fabian Giesen". Apologies to Fabian for this mistake, and thanks to Jake Taylor for communicating it.

Page 32: Implementations using Huffman coding

The recent Brotli compressor employs Huffman coding as one of its many components, see Alakuijala et al. (2019) for details.

References

J. Alakuijala, A. Farruggia, P. Ferragina, E. Kliuchnikov, R. Obryk, Z. Szabadka, and L. Vandevenne. Brotli: A general-purpose data compressor. *ACM Transactions on Information Systems*, 37(1):4:1–4:30, 2019.

[1] `https://encode.su/threads/2078-List-of-Asymmetric-Numeral-Systems-implementations`

[2] `https://sites.google.com/site/powturbo/entropy-coder`

Alistair Moffat,
ammoffat@unimelb.edu.au,
June 10, 2022