9th International Workshop on
ON-BOARD PAYLOAD DATA COMPRESSION
2-4 October 2024
Las Palmas de Gran Canaria - SPAIN

Tabled Asymmetric Numeral Systems compression for the Laser Altimeters on ESA's Bepi-Colombo and JUICE missions

Christian Hüttig[(1)], Alexander Stark[(1)], Hauke Hußmann[(1)]

[(1)]German Aerospace Center (DLR)
Institute of Planetary Research
Rutherfordstr. 2
12489 Berlin - Germany
Email:christian.huettig@dlr.de, alexander.stark@dlr.de , hauke.hussmann@dlr.de

INTRODUCTION

The Laser Altimeters BELA [3] and GALA [2] (BEpi-colombo Laser Altimeter and GAnymede Laser Altimeter) are currently en-route to Mercury and Ganymede, respectively. They measure distance to the surface by emitting short laser pulses at frequencies between 10 Hz and 48 Hz, which are subsequently sampled upon return using a single APD (Avalanche Photodiode) operating at 200 MHz (GALA) and 80 MHz (BELA). The resulting time-of-flight measurements enable precise distance calculations. On-board, a Field-Programmable Gate Array (FPGA) selects four brief windows of 64 or 42 (GALA / BELA) 12-bit samples from an Analog-to-Digital Converter (ADC) per shot from a range window as potential candidates and transmits these to the central processing unit for further analysis. Subsequently, a Gaussian fit is applied, and only the fitted center position, pulse width, and amplitude are transmitted back to Earth, minimizing data volume. This approach enables high detection accuracy in the decimeter range despite the relatively low sampling frequency together with a low data-rate, required for deep-space missions.
Unfortunately, secondary science objectives such as surface roughness and slope are lost when only transmitting fit information [1]. Moreover, even higher accuracy can be achieved by accounting for the non-Gaussian nature of the outgoing laser pulses. It became evident from examining raw data that these secondary objectives, along with improved precision, are highly desirable goals. However, downlink bandwidth will not magically increase at our request, so maintaining a low data rate is crucial. Consequently, implementing a compression method that our central processor can handle has become a priority.
When dealing with noisy sensor data, entropy coding emerges as the method of choice. To leverage the existing Gaussian fit, we subtracted it from the signal to decrease the remaining symbol count. The remaining samples could then be compressed by a factor of ~3.5 using an implementation of the Tabled Asymmetric Numeral Systems (TANS) encoder. Remarkably, the compression ratio is close to the Shannon entropy under all circumstances. We provide a comprehensive comparison with Huffman coding as well as detailed implementation specifics and performance metrics for our C version on both the Gaisler GR712RC in-flight CPU and Commercial Off-The-Shelf hardware.

THEORY OF (TABLED) ASYMMETRIC NUMERAL SYSTEMS

Entropy Coding

Entropy coding is a lossless compression technique that aims to reduce the number of bits required to represent data, basically an array of symbols, by exploiting the statistical properties of that data. The core idea behind entropy coding is to use shorter codes for more frequent symbols and longer codes for less frequent ones, thereby reducing the overall size of the encoded data [6].
Entropy coding is widely used in various fields, including multimedia compression (e.g., JPEG, MP3, H.264), data transmission, and storage systems. It's a powerful tool for reducing data size without losing any information, making it crucial for efficient data handling in many technologies.
What entropy coding does not do is positional coding, i.e. exploiting positional relationships between symbols like run-length encoding, delta coding, Fourier transforms or wavelets. For best compression ratios a positional coder specific to the type of data is usually combined with an entropy coder [6].

Arithmetic Coding on Symmetric Numeral Systems

Arithmetic coding [7,8] is a sophisticated form of entropy encoding used for lossless data compression. Unlike other methods such as Huffman coding [6,5], which replace input symbols with specific codes, arithmetic coding maps the entire input message to a single number that can be represented using fewer bits than the original message.
A prominent example is our own way of writing numbers. We use 10 symbols (0-9) in a Base-10 (or radix-10) system. If these 10 symbols have an equal probability of occurring in the message (i.e., the number itself), then this is the optimal way to represent numbers. When seeking a binary representation to store this number inside computers, we simply transform the number to Base-2, thereby creating an optimal system to store a 10-symbol alphabet without being restricted to a $2n$ alphabet size that would limit us in a naïve fixed-width binary representation. Our 10-symbol alphabet requires only $\log2(10)\approx3.322$ bits per symbol. Using 4 bits would waste almost 0.7 bits per symbol.
The power of symmetric numeral systems lies in their ability to represent any alphabet size N as an optimal bit-stream through pure Base (or radix) transformation. The system can accurately retain fractions of bits without wasting space, making it highly efficient for data compression purposes.

Asymmetric Numeral Systems (ANS)

ANS builds upon symmetric numeral systems and arithmetic coding. It generalizes this process for arbitrary sets of symbols $s \in S$ with an accompanying probability distribution $(p_s)_{s\in S}$ [10, 11, 12]. This is best understood with the binary example, also called the uniform asymmetric binary system from [11]:

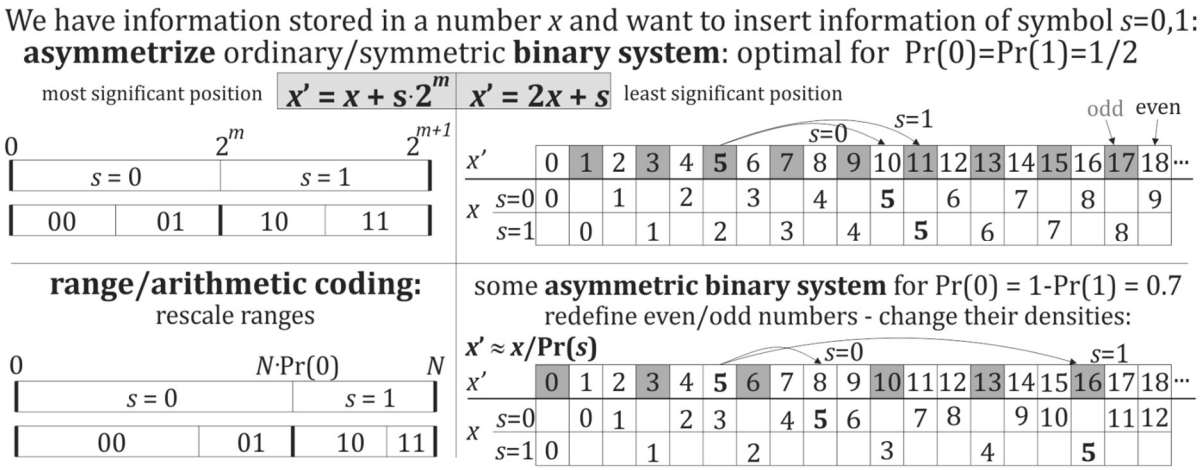

Figure 1 Two ways to asymmetrize binary numeral system. Having some information stored in a natural number x, to attach information from 0/1 symbol s, we can add it in the most significant position (x 0 = x + s2 m), where s chooses between ranges, or in the least significant (x 0 = 2x + s) position, where s chooses between even and odd numbers. The former asymmetrizes to AC by changing range proportions. The latter asymmetrizes to ABS by redefining even/odd numbers, such that they are still uniformly distributed, but with different density. Now x 0 is x-th element of the s-th subset – from [11]

The theory of arithmetic coding would require to store an ever-growing numeral state which becomes quickly impracticable on computers. ANS handles that problem efficiently by a re-normalization of the state. A shift is applied to the state and the lowest bits that are shifted out go into the bit-stream. [10] describes the ANS system to manage arbitrary message lengths with arithmetic coding and [11] goes into detail of a highly performant encoder/decoder that even exceeds Huffman Coding in compression ratio and speed [12]. Besides the ABS described above, two main variants are described:

- The ranged ANS (rANS) can be applied to a symbol stream where an approximated probability is sufficient. The symbol probability table is updated as new symbols arrive.
- The tabled ANS (tANS) can be described as a finite-state machine with pre-computed tables. Ideally the entire symbol block must be available to construct accurate tables. Practically this is split into convenient blocks of finite symbols.

The rANS version has the advantage of operating on a stream with the shortcoming that it uses multiplication operations and is slightly sub-optimal by design. tANS will mostly outperform rANS in compression ratio but tANS requires a longer

setup phase as tables need to be created from symbol probabilities (with higher memory footprint as well) but without the need of multiplications.

IMPLEMENTATION AND PERFORMANCE DETAILS FOR BELA AND GALA

The incoming raw data from the sensor is in a worst-case scenario (GALA@48Hz, 4 windows) at 19200 byte/s. While BELA and GALA have 128Mbyte of RAM [2,3], it is possible to store the required samples for an entire orbit in RAM and compress everything after continuous shooting. With this option we relaxed the speed requirements on the compressor which at the end was not even necessary. One hour of continuous worst-case science operation on BELA (10Hz, 4+1 windows) produces 11.3Mbyte data. This scenario is ideal for the tANS encoder that requires accurate symbol tables and avoids potentially slow multiplication.
The tabled ANS consists of two primary phases: initialization and encoding. During the initialization phase, various encoding tables are constructed based on the symbol probability distribution within the alphabet. The compression ratio improves as the table size increases, approaching the Shannon Entropy (1), expressed in average bits / symbol [6, 14]:

$$H(X) := -\sum_{x \in X} p(x) \log_2 p(x) \tag{1}$$

To reduce execution time in the encoding phase, three variables for each symbol are generated: $k[s]$, $start[s]$, and $bound[s]$. During the encoding phase, each symbol contributes either $k[s]$ or $k[s]-1$ bits to the output stream. This is determined by comparing the current state value with $bound[s]$. Symbols are encoded using the encoding tables with the three precomputed variables from the initialization phase.
We follow the optimized algorithm [12, 13] that yields identical results with fewer mathematical operations. Notably, for each symbol s and state x, there exists a unique value and a unique new state that can be precomputed. The key is to create two new tables, $nbBitsTable$ and $stateTable$, which store all possible combinations of input symbols and current states after the initialization phase. Two loops generate all possible combinations of s and x. The address within the new tables is formed by concatenating the input symbol s and the current state x. Once $nbBits$ is calculated, the stream function transfers $nbBits$ least significant bits of x to the output stream.
To test and compare this method, we first developed a reference algorithm in Python without dependencies (pure Python) as a proof-of-concept. Subsequently, a C version was created that can compress arbitrary data on the Linux command line. Data were divided into blocks of 64k symbols with 16-bit state tables, representative of BELA/GALA data chunks. The performance is detailed in Table 1. For GALA, we did not test the encoder yet; however, since the hardware is identical except for the clock speed (GALA at 50 MHz and BELA at 30 MHz), we anticipate a linear influence of clock speed on encoding speed, potentially reaching up to 140k symbols/second. Unfortunately, we had to use a relatively old compiler for BELA & GALA (GCC 4), which means many cache optimizations available in recent compilers were not utilized. The decoder was not implemented within the central software of BELA/GALA as it will not be used. Decoding with the C version was slightly slower than encoding, but only by around 5-10%. A significant difference in coding speed is observed if the additional lookup tables fit into the CPU cache.
For comparison with different compression mechanisms and to test the algorithm on real data, we selected five datasets as detailed in Table 1. The datasets included original BELA/GALA data and, for comparison, raw image data from the JANUS instrument, the camera system of JUICE, of a recent moon flyby (LEGA). For GALA, two datasets were used: one with original data including the pulse and another with the subtracted fit. The corresponding probability distributions are shown in Figures 2, 3, and 4. Table 2 presents the results of the compression, while Table 3 shows the encoding speed for the various implementations.

Number	Description	Size (Symbols)	Bits / Symbol	Bits / Symbol – Shannon
1	GALA samples from APD with expected pulse shape	65536	12	4.885
2	GALA samples with removed pulse	65536	12	3.493
3	BELA samples from APD with typical noise	65535	12	3.187
4	JANUS raw image A from LEGA flyby	3008000	14	12.522
5	JANUS raw image B from LEGA flyby – High Contrast	3008000	14	12.290

Table 1 Description and details of the 5 test data-sets

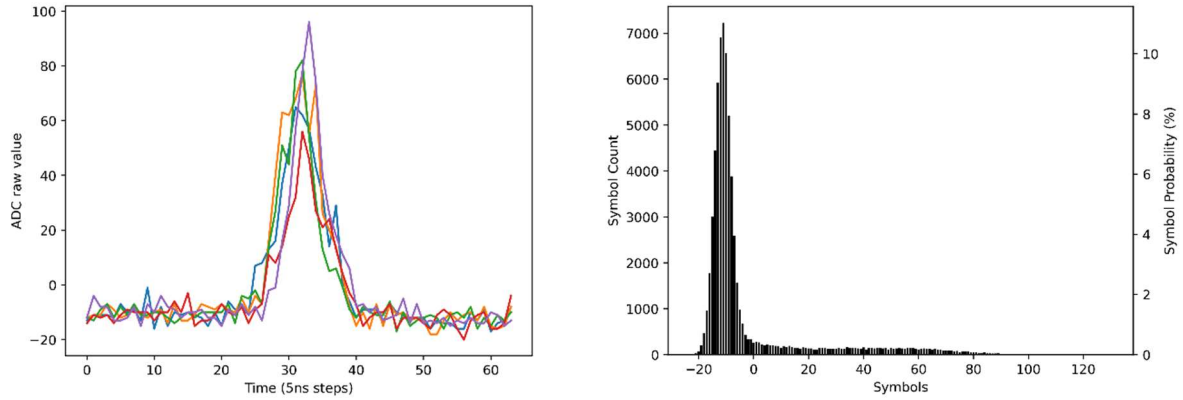

Figure 2: Left: GALA raw data with received pulses (stacked), right: distribution of symbols, both of dataset 1

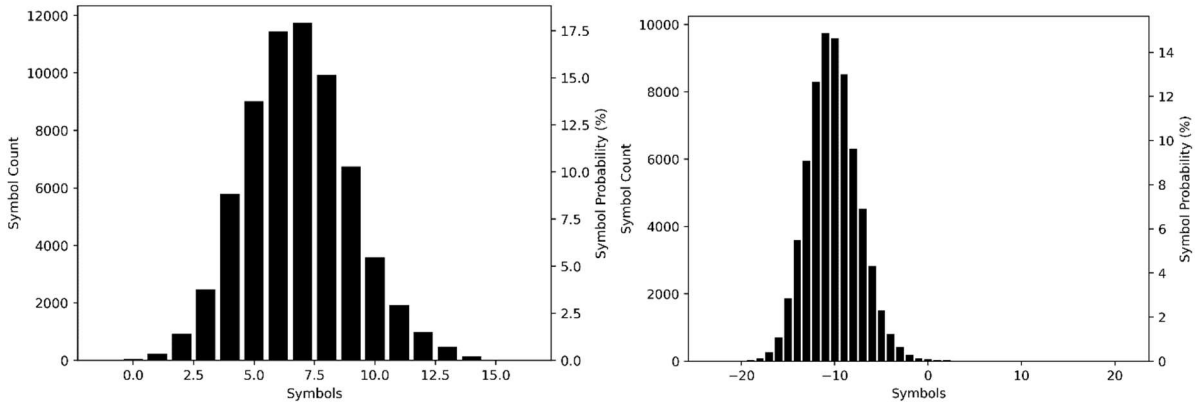

Figure 3: Left: BELA noise distribution (dataset 3), right: GALA removed pulse distribution (dataset 2)

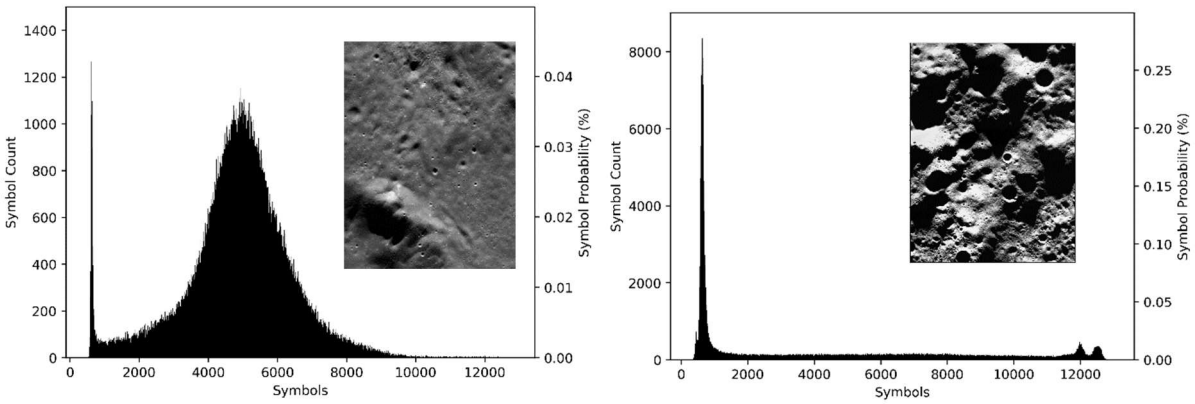

Figure 4 Left: JANUS image from the moon fly-by and its symbol distribution (dataset 4), right: high-contrast moon fly-by image (dataset 5).

System	Description	Block Size (Symbols)	Bits / Symbol	Encoding speed
Intel 3.5GHz i7-1370P	C-encoder	65536	8	140MS/s
Intel 3.5GHz i7-1370P	Python Reference encoder	65536	8	532kS/s
Gaisler SPARC GR712RC 30MHz, incl. SpaceWire delivery	C-encoder (BELA)	65535	12	87kS/s

Table 2 Encoding speeds on different hardware and for different implementations

Dataset	Huffman – Stream	TANS – Stream	TANS – with Symbol Table	Bzip2 linux	Theoretical Shannon	Delta Bytes / MByte Huffman vs TANS Stream
1	2.441	2.456	2.407	2.544	2.456	2624
2	3.410	3.435	3.410	3.151	3.435	450
3	3.703	3.764	3.750	3.707	3.765	4589
4	1.1146	1.1174	1.1013	1.326	1.1179	2357
5	1.1356	1.1387	1.1211	1.351	1.1391	2513

Table 3 Compression ratios for the 5 different datasets. The Bzip2 compressor is a combination of a general-purpose positional and entropy coder, which is why it can exceed the Shannon limit.

OUTLOOK AND CONCLUSION

Although CPU based compression is quite slow compared to FPGA [4, 9] based, we found it more than sufficient for the low data-production rates of BELA and GALA. Because we subtract the signal fit, the remaining quasi-noise has excellent compression ratios and enables us to transmit the raw samples without increasing the promised transmission budget. As with most compression, corruption within a compressed block is fatal. On the other hand, the fit information is transmitted separately in the science stream so primary science goals would remain intact and the block-size of 64kS would leave us with only around 780 lost shots or 78s operation time (assuming 2 windows transmitted per shot).
For an outlook, the perspective of using the symbol table to improve compression ratio and turn the method into a hybrid positional / entropy coder is tempting but also not quite easy. Simple approaches by varying distributions on the symbols have been tried with different symbol spreading techniques but achieving notable results is tough. Nevertheless, a symbol spread permutation could be used to fit any kind of signal, but it remains to be seen that the transmission overhead of a custom (and varying) symbol spread function, which must match precisely within the encoder and decoder, does in the end yield better results.

REFERENCES

[1] Nishiyama, G., Stark, A., Hüttig, C., Hussmann, H., Gwinner, K., Hauber, E., M. Lara, L., and Thomas, N.: Simulation of laser pulse shapes received by the BepiColombo Laser Altimeter (BELA): Implications for future constraints on surficial properties of Mercury, Europlanet Science Congress 2022, Granada, Spain, 18–23 Sep 2022, EPSC2022-326, https://doi.org/10.5194/epsc2022-326, 2022
[2] Hussmann, H., Lingenauber, K., Kallenbach, R. et al. (2019). The Ganymede Laser Altimeter (GALA): key objectives, instrument design, and performance. CEAS space journal, 11, 381–390.906
[3] Thomas, N., Hussmann, H., Spohn, T. et al. (2021). The BpiColombo Laser Altimeter. Space science reviews, 217(1), 1–62. 990
[4] S. Rigler, W. Bishop, and A. Kennings, "Fpga-based lossless data compression using huffman and lz77 algorithms," in Electrical and Computer Engineering, 2007. CCECE 2007, April 2007, pp. 1235–1238.
[5] T. Lee and J. Park, "Design and implementation of static huffman encoding hardware using a parallel shifting algorithm," Nuclear Science, IEEE Transactions on, vol. 51, no. 5, pp. 2073–2080, Oct 2004.
[6] K. Sayyod, Introduction to Data Compression. Morgen Kaufmann, 1996.

[7] A. Said, “Comparative analysis of arithmetic coding computational complexity,” in Data Compression Conference, 2004. Proceedings. DCC 2004, March 2004, pp. 562–.
[8] J. Supol and B. Melichar, “Arithmetic coding in parallel.” in Stringology. Department of Computer Science and Engineering, Faculty of Electrical Engineering, Czech Technical University, 2004, pp. 168–187.
[9] H.-Y. Lee, L.-S. Lan, M. hwa Sheu, and C.-H. Wu, “A parallel architecture for arithmetic coding and its vlsi implementation,” in Circuits and Systems, 1996., IEEE 39th Midwest symposium on, vol. 3, Aug 1996, pp. 1309–1312 vol.3.
[10] J. Duda, “Asymmetric numeral systems,” arXiv:0902.0271, Feb. 2009.
[11] J. Duda, “Asymmetric numeral systems: entropy coding combining speed of huffman coding with compression rate of arithmetic coding,” arXiv:1311.2540, Jan. 2014.
[12] J. Duda, K. Tahboub, N. J. Gadgil, and E. J. Delp, “The use of asymmetric numeral systems as an accurate replacement for huffman coding,” 31st Picture Coding Symposium, May 2015.
[13] Y. Collet, https://github.com/Cyan4973/FiniteStateEntropy.
[14] C. Shannon, “A mathematical theory of communication,” in The Bell System Technical Journal, vol. 27, Jul. 1948, pp. 379–42