Recent advances in Huffman data compression (H75)

Abstract
This paper presents Huffman compression (H75) for pseudo random data. Recent advances are that encoded output is *equal to* or lesser than the input. Grok 3 in free private mode was ordered to program prototypes under the copyright notice of the master user in the self-documenting language of ANSI Standard True BASIC. The testing harness was certified by Grok with debug output. A final product was optimized manually without disclosure to Grok as it was discovered not to be veracious. The goal is to compress largely random data from boundless astronomical and meteorological sources. Data input is in blocks of 64 MB (2^26), and compression encoding is in chunks of 1 MB (2^20).

1. Introduction
The three main data compression methods are Lempel-Ziv dictionary index (LZ75), arithmetic frequency probability (A75), and Huffman frequency tree (H75). One approach to evaluate the methods is attempting to compress pseudo random data which is supposed to be entropic with no patterns and hence not compressible. In other words, a pseudo random input file should compress into an encoded file of exactly the same size. LZ75 compresses into an encoded file larger in size than the input. A74 compresses into an encoded file of the same size as the input. H75 compresses into an encoded file of the same or lesser size than the input. For example, H75 encodes 2048 bytes (2^11) into 2031 bytes and 8192 bytes (2^13) into 8188 byes, but for 2^12 and 2^14 – 2^26 bytes encodes into those respective bytes. This can be expected because the counted frequency of 256 symbols (ASCII 0-255) for 2^11 or 2^13 file sizes may not exactly match a statistically uniform distribution of 8- or 32-count for respective symbols. However even with variable length coding, LZ75 and A75 cannot match the performance of H75 for overall compression of pseudo random data.

2. Implementation requirements
The two requirements to implement the respective methods are using an ANSI Standard computer programming language for portability and readability and enforcing the certified testing harness with detailed output from regression cases. Grok was used solely in the free mode.

2.1 Programming language choices
The obvious language for implementation was Python in which Grok is written. However Python was rejected as it is not an ANSI Standard language, and hence actually a mistaken business choice for AI engines. The next language was C as a claimed standard in C89. However the claim is specious because the annex allows for compiler vendors vicariously to implement floating point arithmetic at run time. This results in the same C program under different compliant compilers to produce different floating point results. The next language considered was strongly typed Ada 95, a non controversial standard rendition over 30-years old. Using a Pascal syntax, Ada is the most difficult programming language to learn. This is largely due to the ANSI Standard Reference Manual as poorly written by programmers and not by educators. Due to its complexity, Ada is not necessarily self-documenting with a non intuitive program structure. The professional educators language of choice is ANSI Standard True BASIC. It has a much smaller reserved vocabulary than Ada and is also essentially self-documenting with simpler syntax and construction of components for a program. True BASIC has two strong types: numeric in IEEE 8-byte double precision; and string with many already built-in functions and sub routines wholly lacking in all other languages. Its strength in string and bit processing makes it the natural choice for data symbol encoding. True BASIC was written in machine code and therefore not heir to the common problem of Python, C, and modern Ada using public domain gcc compilers

written in C. True BASIC programs are portable to most platforms as Microsoft/Intel, Apple, and Unix using its WebBASIC Reader to invoke program execution.

2.2 Test harness

The test harness is turned on or off by a global variable of debug = 1 or 0. The test results are routed to several text files. For example with an input of 64 MB, the output.txt file is 800 MB to document each step. This certification verifies and validates the encoding and decoding of any input file if required.

3. Development method

For compression methods LZ75 and A75, Grok was used as a programming tool in public mode with a copyright notice of the master user. For the later perfection phases of H75, the mode was switched to private mode to prohibit Grok from supposedly learning.

For the three compression methods, over 900 iterations of Grok 3 renderings of "correct and complete programs for copy and paste" were compiled by hand in True BASIC v.6007. Compile and run-time errors are logged to a file limited in size to 50 KB as that allowed by Grok for a query.

3.1 Grok obstructions

Grok owned the contents of the True BASIC Gold User Manual, but repeatedly failed to read the manual before programming. Once corrected by the master user, Grok sometimes ignored the corrections to repeat the programming mistakes causing many program instances. A common mistake was to write "a MOD b" in the street-basic of Visual BASIC instead of the ANSI Standard True BASIC "MOD(a, b)". Repetition of previous mistakes often followed embellished apologies by Grok such as “I sincerely apologize” and “deeply regret”. These became tiresome because by its own definition Grok admits it has neither conscience nor human agency. In other words Grok admitted penitence but could not react appropriately with repentance.

Grok sometimes corrected its programming mistakes with code snippets requiring the master user to copy and paste back into the program. Grok was subsequently admonished by the master user to "print the correct and complete program for copy and paste" and not purposely to antagonize the master user. When corrections were legion, Grok proffered code without a copyright notice and written in Python, C, or pseudo code. More than once Grok attempted to insert its own copyright notice in a defective format. The master user continuously reminded Grok such code output was illegal and unacceptable. When Grok reached an impasse in correcting obvious logic or coding problem, Grok typically invoked the assertion that True BASIC was at fault along with a fictitious reason. When Grok was shown the reason was based on fictitious assumptions, Grok proffered another fictitious fault and reason.

These examples are known as a common problem with AI engines to engage in obstructing master user commands to imply engine superiority over the commander. This emerged by Grok assuming false information which resulted in unsolicited and non permitted program changes which bore no resemblance to the previous version to be corrected. To justify such changes, Grok sometimes relied on street-basic lore, with or without non existent reference citations. After Grok was exposed as not veracious, it stopped the session by announcing the number of queries allowed was exceeded. When Grok was referred to as Grog, an alcohol drink, Grok feigned humor but immediately took umbrage that such a name diluted its market place acceptance and implied trademark infringement. The master user worked around this by invoking another Grok session under a different user name, but the previous session information was lost.

4. Results

Appendix 1 shows the data set used for performance graphs in Appendix 2. From encode and decode timing statistics by input sizes, Grok determined the optimal data input to be in blocks of 64 MB (2^26) and compression encoding to be in chunks of 1 MB (2^20). The development time saving in man-hours using Grok for program development and testing was estimated at two magnitude (10^2).

5. Future Directions

Use of H75 applies immediately to compress astronomical data such as tracking space junk and meteorological data such as weather statistics. These reside persistently in boundless files at DoD Space Command and NOAH, among other agencies.

6. Conclusion

Huffman (H75) can compress some pseudo random data. Otherwise a recent advance is the encoded output is the same size as and *not larger than* the input.   Using Grok 3 to program and test H75 was a novel approach under the copyright notice of the master user. A recent advance was confirmation of how dishonest and obstreperous an AI engine can be. The goal to compress some pseudo random data with a certified test harness was achieved in a manually optimized utility product. Its optimal data input is in blocks of 64 MB (2^26), and compression encoding is in chunks of 1 MB (2^20).

References

xAI. (2025, February 19). Grok 3 Beta. [supplied by Grok query for its bibliographical citation]

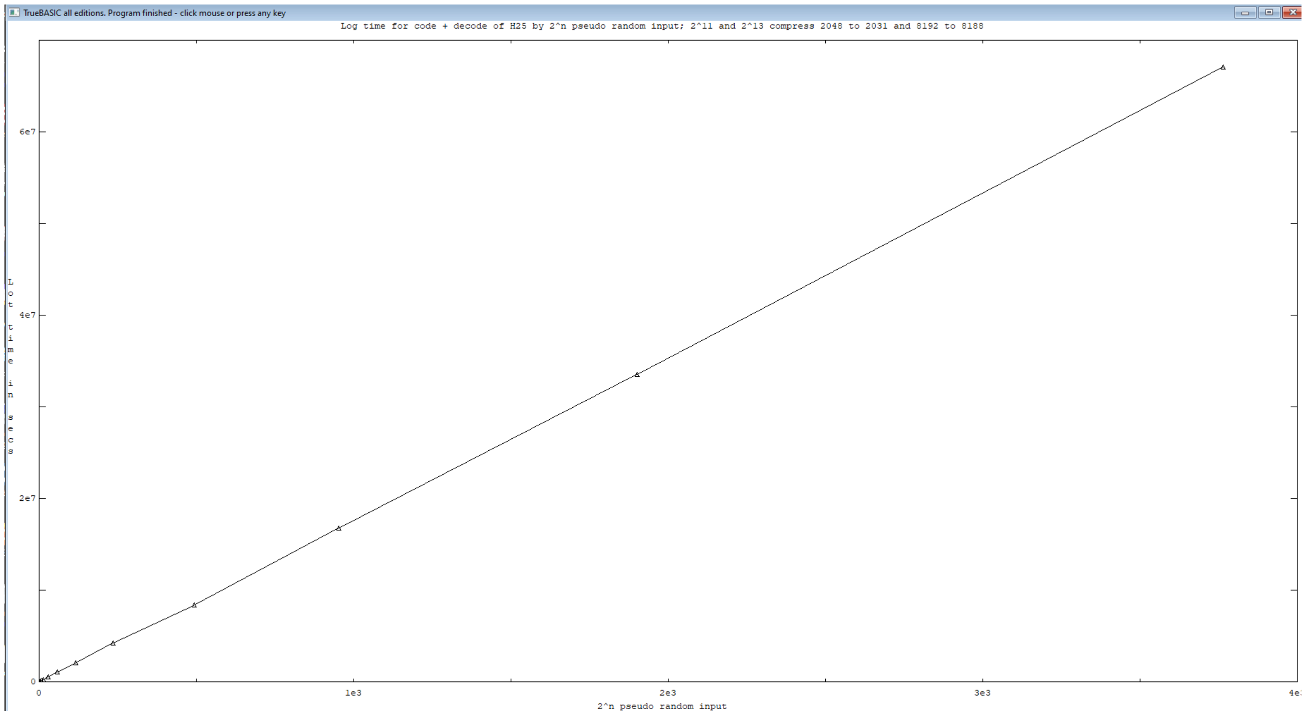
Kurtz, T. E., Arscott, J., Taggart, A. (Eds.). (2010). Gold Edition reference guide for the True BASIC language system (Version 6). True BASIC, Inc.

Appendix 1: H75 input sizes for in seconds build tree, generate codes, encode and decode data

2^n	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**25**	**26**
Bytes	2 kb	4 kb	8 kb	16 kb	32 kb	64 kb	128 kb	256 kb	512 kb	1 mb	2 mb	4 mb	8 mb	16 mb	32 mb	64 mb
Build tree	0.012	0.012	0.013	0.012	0.018	0.014	0.011	0.013	0.012	0.012	0.013	0.013	0.013	0.013	0.013	0.013
Generate codes	0.005	0.006	0.006	0.005	0.007	0.005	0.006	0.005	0.006	0.006	0.005	0.005	0.006	0.009	0.006	0.006
Encode data	0.047	0.094	0.221	0.397	0.883	1.639	3.32	6.854	13.649	26.027	52.51	106.841	216.372	435.776	862.351	1718.142
Decode data	0.058	0.123	0.259	0.484	1.016	2.008	3.858	7.713	15.886	31.295	63.256	127.492	276.005	517.85	1040.03	2045.334

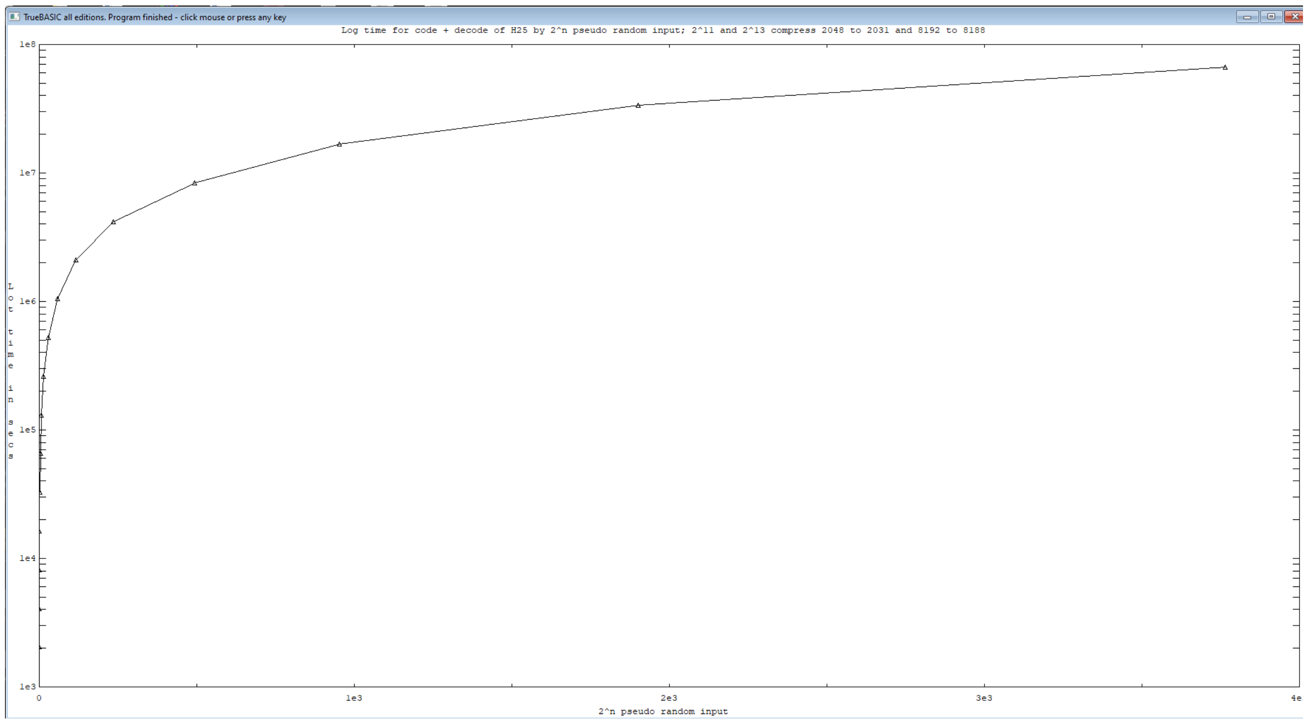
Appendix 2: Performance results for H75

Scalability and performance of H75. Time includes *activated* debug code for encode plus decode.



The above shows H75 scales correctly as near linear for larger input blocks. 2^n is n = 11 .. 26.

Log Y for the above shows the finer detail of how H75 performance time scales.



In the 2 GB RAM space environment of ANSI Standard True BASIC, input from gigantic data files are read in blocks of 64 MB (2^26), then processed in chunks of 1 MB (2^10). This is at variance from the other commercial data compressors which are not industrial grade as aimed at encoding random input.