Tiny Data Compression with td512

L. S. Leonard, November 11, 2021

# Introduction

Tiny data compression is not usually supported by compression programs. Now with td512 you can compress data from 6 to 512 bytes. Td512 is available under the GPL-3.0 License at <https://github.com/lsleonard/tiny-data-compression>. Although algorithms in Zstandard and Snappy get better compression at 512 bytes than td512, Zstandard is very slow for tiny datasets and both programs steadily decline in compression ratio as the number of bytes decreases to 128. At 64 bytes, neither program produces compression. td512 combines the compressed output of td64 for each block of 64 bytes in the input, meaning that the compression achieved at 512 bytes is the same as that for 64 bytes. The algorithm emphasizes speed, and based on data in this paper, gets 21% average compression at 215 Mbytes per second on the Squash benchmark test data (see [https://quixdb.github.io/squash-benchmark/#](https://quixdb.github.io/squash-benchmark/)). Although Huffman coding, with its optimal compression using frequency analysis of values, has been used effectively for many applications, for tiny datasets the compression modes used in td512 approach or exceed the results of using the Huffman algorithm. And with a focus on speed of execution, Huffman and arithmetic coding are not practical algorithms for applications of tiny data. Two areas where high-speed compression using td512 might be applied are small message text and programmatic objects.

For compression and speed comparison with td512, this paper presents compression data using QuickLZ, where its streaming mode supports compression of as few as 200 to 300 characters (see [quicklz.com](http://quicklz.com/)). Also presented are compression values for the arithmetic encoding program fpaq0 (<http://mattmahoney.net/dc/fpaq0.cpp>) to give a picture of the possible compression when runtime is not a concern. The next section presents this comparison data.

The final section of the paper discusses the fundamentals of td64, the compression interface that the td512 algorithm uses. Keep in mind that td512 will maintain the same compression performance for 64 bytes as for 512 bytes because the algorithm combines the output from td64 and its 64-byte compression modes.

# Comparison of Compression Performance: td512, QuickLZ, and fpaq0

The td512 algorithm is packaged with a testbed that runs the algorithm iteratively over an input file using 512-byte blocks until a final block, possibly smaller, concludes the run (see <https://github.com/lsleonard/tiny-data-compression>). The QuickLZ public distribution of version 1.5.1 Beta 7 was modified to iterate over a given block size as was the fpaq0 test bed. The td512 test bed, QuickLZ with 256- and 512-byte blocks, and fpaq0 with 64-byte block size were run on the Squash benchmark test data on a MacBook with a 1 GHz Dual-Core Intel Core M processor. The test beds for td512 and QuickLZ tests read file data into memory and compute loop count between 10 and 1000 based on file size to pick the best (fastest) compression and decompression speed. The test bed for fpaq0 was run only to get compression percent.

Compression as a percent reduction of the original file size is shown in Table 1. Across all the benchmark files, compression percent average, assuming a block of data from one file is as likely as from another, for td512 is 21%, for QuickLZ 512 is 22%, and for QuickLZ 256 is 14%. QuickLZ gets better compression for XML data, while td512 gets better compression for text data and data in file x-ray that contains frequently repeated values in small blocks, for example 16 or more 0s in a 64-byte block, a td64 compression mode called single value.

The compression percent for fpaq0 with 64-byte blocks gives a value to be expected when compression and decompression runtime are not a concern. The compression reached by td512 is 25% below the form of arithmetic encoding used by fpaq0 and 5% below QuickLZ 512.

Table . Compression Percent: td512 and QuickLZ 512-Byte Blocks, QuickLZ 256-Byte Blocks, and fpaq0 64-Byte Blocks

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Filename  Squash Benchmark | Compression  Percent  td512 | Compression  Percent  QuickLZ 512 | Compression  Percent  QuickLZ 256 | Compression Percent  fpaq0 64 Bytes |
| alice29.txt | 24.77 | 8.43 | -0.43 | 27.98 |
| asyoulik.txt | 21.47 | 6.93 | -1.10 | 24.84 |
| cp.html | 13.89 | 19.25 | 2.09 | 21.00 |
| dickens | 25.48 | 4.67 | -2.40 | 28.13 |
| fields.c | 17.78 | 33.98 | 20.13 | 25.45 |
| fireworks.jpeg | -0.35 | -1.56 | -3.30 | -7.08 |
| geo.protodata | 3.11 | 13.82 | 8.28 | 6.99 |
| grammar.lsp | 19.27 | 40.15 | 30.26 | 28.00 |
| kennedy.xls | 30.22 | 59.87 | 53.33 | 50.23 |
| lcet10.txt | 26.45 | 8.31 | -0.02 | 28.94 |
| mozilla | 17.42 | 30.86 | 22.91 | 22.36 |
| mr | 47.59 | 30.15 | 26.73 | 53.50 |
| nci | 38.71 | 62.05 | 53.08 | 57.03 |
| ooffice | 7.60 | 12.18 | 5.84 | 12.82 |
| osdb | 1.62 | -1.74 | -3.51 | 8.50 |
| paper-100k.pdf | 5.79 | 9.41 | 6.75 | 3.02 |
| plrabn12.txt | 25.57 | 3.52 | -2.71 | 27.27 |
| ptt5 | 70.06 | 74.94 | 68.82 | 75.17 |
| reymont | 18.39 | 25.51 | 13.87 | 27.27 |
| samba | 22.29 | 33.56 | 22.10 | 26.00 |
| sao | -0.39 | -1.75 | -3.51 | -3.81 |
| sum | 28.86 | 27.72 | 20.10 | 35.86 |
| urls.10K | 15.79 | 23.46 | 13.29 | 23.84 |
| webster | 16.37 | 17.60 | 3.23 | 23.68 |
| x-ray | 23.34 | -0.98 | -3.07 | 28.59 |
| xargs.1 | 22.69 | 18.31 | 7.26 | 25.31 |
| xml | 23.22 | 35.74 | 23.90 | 24.74 |
| Average Compression % | td512  21.00 | QuickLZ 512  22.01 | QuickLZ 256  14.14 | fpaq0 64  26.13 |

Compression and decompression speed as shown in Table 2 and Table 3 are affected by the compressibility of the data and can easily vary by 10% with other activities on the test system. The average numbers provide a good benchmark for comparison. Uncompressible data such as fireworks.jpeg compresses and decompresses quickly because it contains mostly random data. Different types of data compress and decompress at different rates. Only the file reymont has a compression speed that is much slower for td512 than QuickLZ. This type of data tries string mode, the slowest of the modes, for nearly every block, and ends up using 7-bit mode (removal of high-order bit) for 26% of blocks, resulting in longer compression time than for other data types. On average, td512 compresses nearly twice as fast as QuickLZ 512, and with average compression percent very similar between them, this is a benefit for td512. With 2.5 times the average compression speed and nearly 50 percent better average compression, td512 performs far better than QuickLZ 256.

Table . Compression Speed: td512 Versus QuickLZ 512 and 256

|  |  |  |  |
| --- | --- | --- | --- |
| Filename  Squash Benchmark | td512  Compression  MB/sec | QuickLZ 512 Compression  MB/sec | QuickLZ 256 Compression  MB/sec |
| alice29.txt | 138262736 | 111912440 | 72217000 |
| asyoulik.txt | 134168272 | 111567728 | 71694728 |
| cp.html | 89791976 | 111831816 | 73005936 |
| dickens | 143130208 | 111248168 | 85049032 |
| fields.c | 111500000 | 157042256 | 96120688 |
| fireworks.jpeg | 905095552 | 188215600 | 107410992 |
| geo.protodata | 208049136 | 117763656 | 86120552 |
| grammar.lsp | 218882352 | 148840000 | 106314280 |
| kennedy.xls | 291960288 | 175484656 | 103899104 |
| lcet10.txt | 147716848 | 112362824 | 86160704 |
| mozilla | 191677648 | 122775536 | 84784008 |
| mr | 238290816 | 123954952 | 78551056 |
| nci | 189799104 | 145842688 | 90311288 |
| ooffice | 143024336 | 89003544 | 77686056 |
| osdb | 218484544 | 105099712 | 80688696 |
| paper-100k.pdf | 380669120 | 143618512 | 104918032 |
| plrabn12.txt | 126872296 | 92116424 | 88512304 |
| ptt5 | 194916816 | 136675360 | 103117544 |
| reymont | 67358512 | 111843960 | 80472128 |
| samba | 113269584 | 124723064 | 87278136 |
| sao | 759127360 | 118042552 | 89546752 |
| sum | 102245992 | 111162792 | 79171840 |
| urls.10K | 83661464 | 108046624 | 81269472 |
| webster | 79533880 | 107375920 | 84175832 |
| x-ray | 227838912 | 128494920 | 87856928 |
| xargs.1 | 234833328 | 132093752 | 84540000 |
| xml | 72294088 | 104340904 | 70672976 |
| Average Compression Speed | td5612  215276117 | QuickLZ 512  124128902 | QuickLZ 256  86723928 |

Average decompression speed in Table 3 shows some interesting numbers. Both QuickLZ 512 and 256 have much better average decompression speeds, 25 percent or greater, than td512. The values for two files, fireworks.jpeg and osdb, push the values much higher for QuickLZ. These are both files that are mostly uncompressible and QuickLZ handles the decompression much quicker than td512. Handling uncompressible data quickly is a benefit for QuickLZ.

Table . Decompression Speed: td512 Versus QuickLZ 512 and 256

|  |  |  |  |
| --- | --- | --- | --- |
| Filename  Squash Benchmark | td512 Decompression MB/Sec | QuickLZ 512 Decompression MB/Sec | QuickLZ 256 Decompression MB/Sec |
| alice29.txt | 272073344 | 299338560 | 652635200 |
| asyoulik.txt | 302364736 | 317076128 | 621532352 |
| cp.html | 410050016 | 270065920 | 491520000 |
| dickens | 277988448 | 343108608 | 944526336 |
| fields.c | 247777776 | 282947392 | 262095248 |
| fireworks.jpeg | 4923720192 | 8192000000 | 6826666496 |
| geo.protodata | 898393920 | 363913856 | 461198432 |
| grammar.lsp | 338272736 | 238933344 | 256000000 |
| kennedy.xls | 865331136 | 477565888 | 447082912 |
| lcet10.txt | 293907712 | 326566592 | 651529792 |
| mozilla | 624899136 | 376361024 | 420019040 |
| mr | 915486592 | 368447008 | 469042304 |
| nci | 666271680 | 462856672 | 425087200 |
| ooffice | 446328512 | 260300064 | 374562656 |
| osdb | 2106450176 | 4058501376 | 4839554560 |
| Paper-100k.pdf | 1765517184 | 1600000000 | 1796491136 |
| plrabn12.txt | 237956048 | 299808352 | 1271218944 |
| ptt5 | 688880512 | 485358560 | 521896224 |
| reymont | 135152480 | 237724768 | 237589072 |
| samba | 280089696 | 346339648 | 414280768 |
| sao | 3243266560 | 4789601280 | 5025441280 |
| sum | 267412592 | 264951056 | 276405792 |
| urls.10K | 264838560 | 339928352 | 544570944 |
| webster | 239482352 | 257283664 | 444125664 |
| x-ray | 642377216 | 1089777792 | 1261216256 |
| xargs.1 | 422700000 | 215578960 | 292571424 |
| xml | 132024608 | 249627792 | 248144480 |
| Average Speed Decompression | td512  811444960 | QuickLZ 512  993109728 | QuickLZ 256  1128777945 |

# Fundamentals of td512 and td64 Interfaces

With the td512 interface, you can call the td512 and td512d functions to compress and decompress 1 to 512 bytes. The td512 interface performs compression of 6 to 512 bytes, but accepts 1 to 5 bytes and stores them without compression. td512 acts as a wrapper that uses the td64 interface to compress blocks of 64 bytes until the final block of 64 or fewer bytes is compressed. Along with the number of bytes processed, a pass/fail bit is stored for each 64-byte (or smaller) block compressed, and the compressed or uncompressed data is output.

With td64, you can call the td5 and td5d functions to compress and decompress 1 to 5 values. This interface is not used by td512 because the number of bytes generated is often more than the number of values to compress. Compression of these miniscule datasets requires bit handling not supported by td512. Or you can call td64 and td64d functions to compress and decompress 6 to 64 values. The td64 interface returns pass (number of compressed bits) or fail (0) and outputs only compressed values. Decompression requires input of the number of original values and data that successfully compressed.

## Encoding Used by the td5 Interface

The td5 interface uses three modes to encode 1 to 5 values:

* For 4 or 5 input values, decode 1 or 2 unique values that are the only ones that occur in the data.
* For 1 to 5 input values, encode frequently occurring text characters (see text mode below).
* For 2 or 3 input values, encode 2 unique 4-bit nibbles.

## Encoding Used by the td64 Interface

The td64 interface integrates the following encoding modes: fixed bit coding, text mode, single value mode, string mode, and 7-bit mode. Initially, the program searches 1/3 of the input values for unique values and accumulates a count of each one, accumulates a high-bit value, and counts the number of frequently occurring text characters. If more than the accepted limit of unique values is encountered, then if all characters have the high bit clear, 7-bit mode is used; otherwise, the program fails, assuming random data for this block. If the values match to indicate text data, text mode is used if it gets at least 12% compression. Processing continues by looking for any single value that occurs 25 percent of the time in addition to looking for new unique values. If fewer than the limit of unique values is encountered, fixed bit coding is used. Otherwise, if a single value is found, single value mode is used. Next, string mode is used to encode repeating strings of two or more characters. If string mode fails and all characters have the high bit clear, 7-bit mode is used. If all modes fail, the program fails to compress the input data. Each of the td64 compression modes is described next.

Fixed bit coding is well known as using only the number of bits required to encode the number of unique characters in a dataset. The td64 algorithm predetermines the maximum number of unique values in the data that will allow compression to occur. With these limits known ahead of time (see uniqueLimits25 in td64.c), decisions about compressibility can be made quickly. This mode gets at least 25% compression for 64 input values.

Text mode uses the most frequent characters as defined by Morse code to identify text data that can be compressed (see Table 4). When 75 percent of the data values are one of these characters, text mode gets 25% compression for 64 input values.

Table : Most Frequently Occurring Text Chars Plus Space Character

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
| space | e | t | a | i | n | o | s | h | r | d | l | u | c | m | f |

Single value mode encodes a frequently occurring value, at least 25% of the number of values, as a 1 bit and all other values as a 0 bit plus their 8-bit value. This results in a minimum of 9% compression for 64 input values.

String mode encodes repeating strings of at least two characters in length. Single repeating characters are also encoded and save at least one bit. This algorithm is the most time-intensive of the modes and is used after all but the 7-bit mode. String mode can fail to compress, and is only used if it gets at least 6% compression, or 12% when 7-bit mode can be used, for 64 input values.

7-bit mode encodes the 7 lower bits of each value, leaving off the high zero bit. This mode gets 11% compression for 64 input values and is used last because other modes can get higher compression.

# Summary

This paper has shown that td512 compresses tiny datasets nearly as well as QuickLZ using 512-byte blocks that results in 21% average compression over the Squash benchmark data. Although QuickLZ 512 has slightly better average compression and decompression speed, td512 has better compression speed and for datasets of 256 bytes, is much better performing than QuickLZ 256. The huge benefit of td512 is that its performance will remain the same for datasets down to 64 bytes, a number of values that very few compression programs can support, fpaq0 arithmetic compressor being an exception. On average, fpaq0 with 64-byte blocks gets 25% better compression than td512.

The implementation of td512 uses fixed bit coding, text, single value, string, and 7-bit modes to compress data. This variety of encoding modes reflects the fact that data compression is data dependent. The td64 interface can be easily modified to support additional compression modes for special data types. Any changes to the td64 interface will not affect the td512 interface.