Tiny Data Compression with td512

L. S. Leonard, November 4, 2021

# Introduction

Tiny data compression is not supported by major compression software tools. Now with td512 you can compress data from 6 to 512 bytes. Although the algorithm emphasizes speed, the minimum compression supported within each 64-byte block is 10%, with a goal of 25% or greater when possible. Because of its optimal compression using frequency analysis of values, the Huffman algorithm has been used effectively for many applications. For small datasets, however, the compression modes used in td512 can approach or exceed 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/)). The next section presents this comparison data.

The final section of the paper discusses the fundamentals of td512 and 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 is built on td64 and its 64-byte compression modes.

# Comparison of Compression Performance: td512 versus QuickLZ

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. The td512 test bed and QuickLZ running 256- and 512-byte blocks were run against the Squash benchmark test data (see [https://quixdb.github.io/squash-benchmark/#](https://quixdb.github.io/squash-benchmark/)) on a MacBook with a 1 GHz Dual-Core Intel Core M processor. The test beds for all three 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.

Compression as a percentage 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 20%, for QuickLZ 512 22%, and for QuickLZ 256 14%. QuickLZ achieves better compression for XML data, while td512 achieves better compression for text data and data in file x-ray that contains frequently repeated values in small blocks, such as 16 or more 0s in a 64-byte block.

Table 1. Compression Percentage: td512 Versus QuickLZ 512 and 256

|  |  |  |  |
| --- | --- | --- | --- |
| Filename  Squash Benchmark | Compression  Percent  td512 | Compression  Percent  QuickLZ 512 | Compression  Percent  QuickLZ 256 |
| alice29.txt | 24.53 | 8.43 | -0.43 |
| asyoulik.txt | 21.14 | 6.93 | -1.10 |
| cp.html | 13.52 | 19.25 | 2.09 |
| dickens | 25.25 | 4.67 | -2.40 |
| fields.c | 16.79 | 33.98 | 20.13 |
| fireworks.jpeg | -0.35 | -1.56 | -3.30 |
| geo.protodata | 3.26 | 13.82 | 8.28 |
| grammar.lsp | 16.93 | 40.15 | 30.26 |
| kennedy.xls | 30.22 | 59.87 | 53.33 |
| lcet10.txt | 26.15 | 8.31 | -0.02 |
| mozilla | 16.88 | 30.86 | 22.91 |
| mr | 47.57 | 30.15 | 26.73 |
| nci | 38.66 | 62.05 | 53.08 |
| ooffice | 6.80 | 12.18 | 5.84 |
| osdb | 5.70 | -1.74 | -3.51 |
| paper-100k.pdf | 5.63 | 9.41 | 6.75 |
| plrabn12.txt | 25.18 | 3.52 | -2.71 |
| ptt5 | 69.87 | 74.94 | 68.82 |
| reymont | 14.54 | 25.51 | 13.87 |
| samba | 20.45 | 33.56 | 22.10 |
| sao | -0.39 | -1.75 | -3.51 |
| sum | 27.37 | 27.72 | 20.10 |
| urls.10K | 13.10 | 23.46 | 13.29 |
| webster | 15.37 | 17.60 | 3.23 |
| x-ray | 23.28 | -0.98 | -3.07 |
| xargs.1 | 21.36 | 18.31 | 7.26 |
| xml | 20.78 | 35.74 | 23.90 |
| Average Compression  Percent | 20.35 | 22.01 | 14.14 |

Compression and decompression speed as shown in Table 2 and Table 3 are affected by the compressibility of the data. 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. Using td64, this type of data requires string mode, with a backup of 7-bit mode (removal of high-order bit) when string mode fails, which results in a longer compression time than for other data types. On average, td512 compresses twice as fast as QuickLZ 512, and with average compression percentage 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 2. 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 | 144571296 | 111912440 | 72217000 |
| asyoulik.txt | 147965728 | 111567728 | 71694728 |
| cp.html | 109834816 | 111831816 | 73005936 |
| dickens | 131823296 | 111248168 | 85049032 |
| fields.c | 128160920 | 157042256 | 96120688 |
| fireworks.jpeg | 1000756032 | 188215600 | 107410992 |
| geo.protodata | 337857568 | 117763656 | 86120552 |
| grammar.lsp | 195842112 | 148840000 | 106314280 |
| kennedy.xls | 281197152 | 175484656 | 103899104 |
| lcet10.txt | 154285600 | 112362824 | 86160704 |
| mozilla | 276717888 | 122775536 | 84784008 |
| mr | 233710672 | 123954952 | 78551056 |
| nci | 188660416 | 145842688 | 90311288 |
| ooffice | 233231936 | 89003544 | 77686056 |
| osdb | 371356960 | 105099712 | 80688696 |
| paper-100k.pdf | 463348416 | 143618512 | 104918032 |
| plrabn12.txt | 130621040 | 92116424 | 88512304 |
| ptt5 | 212776128 | 136675360 | 103117544 |
| reymont | 71562648 | 111843960 | 80472128 |
| samba | 137623024 | 124723064 | 87278136 |
| sao | 980787648 | 118042552 | 89546752 |
| sum | 131408936 | 111162792 | 79171840 |
| urls.10K | 101988240 | 108046624 | 81269472 |
| webster | 105289000 | 107375920 | 84175832 |
| x-ray | 248445856 | 128494920 | 87856928 |
| xargs.1 | 211350000 | 132093752 | 84540000 |
| xml | 96878664 | 104340904 | 70672976 |
| Average Compression Speed | 252890815 | 124128902 | 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 3. 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 | 273050272 | 299338560 | 652635200 |
| asyoulik.txt | 313731808 | 317076128 | 621532352 |
| cp.html | 439339264 | 270065920 | 491520000 |
| dickens | 255117296 | 343108608 | 944526336 |
| fields.c | 398214272 | 282947392 | 262095248 |
| fireworks.jpeg | 4244585984 | 8192000000 | 6826666496 |
| geo.protodata | 1022310336 | 363913856 | 461198432 |
| grammar.lsp | 620166656 | 238933344 | 256000000 |
| kennedy.xls | 770766464 | 477565888 | 447082912 |
| lcet10.txt | 296769120 | 326566592 | 651529792 |
| mozilla | 748716992 | 376361024 | 420019040 |
| mr | 916580608 | 368447008 | 469042304 |
| nci | 617585920 | 462856672 | 425087200 |
| ooffice | 498799392 | 260300064 | 374562656 |
| osdb | 1099856384 | 4058501376 | 4839554560 |
| Paper-100k.pdf | 1347368448 | 1600000000 | 1796491136 |
| plrabn12.txt | 237486928 | 299808352 | 1271218944 |
| ptt5 | 771753408 | 485358560 | 521896224 |
| reymont | 248367936 | 237724768 | 237589072 |
| samba | 403956096 | 346339648 | 414280768 |
| sao | 3191876608 | 4789601280 | 5025441280 |
| sum | 484050656 | 264951056 | 276405792 |
| urls.10K | 403498272 | 339928352 | 544570944 |
| webster | 374456544 | 257283664 | 444125664 |
| x-ray | 651563904 | 1089777792 | 1261216256 |
| xargs.1 | 528375008 | 215578960 | 292571424 |
| xml | 217650544 | 249627792 | 248144480 |
| Average Decompression  Speed | 791703523 | 993109728 | 1128777945 |

# Fundamentals of td512 and td64

With td512, you can call the td512 and td512d functions to perform compress and decompress of 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. The primary function of td512 is to use 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 either the compressed or uncompressed data is output.

With td64, you can call the td5 and td5d functions to compress and decompress 2 to 5 values. 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.

## Encoding Used by the td5 Interface

The td5 interface uses three modes to encode 2 to 5 values:

* Encode 1 or 2 unique values that are the only ones that occur in the data. The encoding of 2 unique values is done only for 4 or 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, the program fails, assuming random data for this block. If the values match to indicate text data, text mode is used. 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.

Fixed bit coding is well known as using only the number of bits required to encode the number of unique characters in a data set. The key is to determine 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 as early as possible.

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, 25 percent compression is achieved.

Table 4: 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 percent of the number of values, as 1 bit and all other values as a 0 bit plus their 8-bit value. This results in a minimum of 9 percent compression.

String mode encodes repeating strings of at least two characters in length. Single repeating characters are also encoded to save one bit. This algorithm is the most time intensive of the modes and is used after all but the 7-bit mode.

7-bit mode encodes the 7 lower bits of each value, leaving off the high zero bit. This mode achieves 11 percent compression and because it usually produces less compression than the other modes is used last.

# Summary

This paper has shown that td512 is a good program for compressing tiny data sets. Although QuickLZ 512 has slightly better average compression and decompression speed, td512 has better compression speed and for data sets of 256 bytes, is much better performing than QuickLZ 256. The huge benefit of td512 is that its performance will remain the same for data sets down to 64 bytes, which very few compression programs can support, fpaq0 arithmetic compressor being an exception.

The implementation of td512 uses fixed bit coding, text mode, single value mode, string mode, and 7-bit mode 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.