Tiny Data Compression with td512

L. S. Leonard, December 7, 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 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 td512 algorithm emphasizes speed, and based on data in this paper, gets 24% average compression at 266 Mbytes per second on the Squash benchmark test data (see [https://quixdb.github.io/squash-benchmark/#](https://quixdb.github.io/squash-benchmark/)) running on a 2 GHz processor.

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 running td64 8 times.

# 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>). In addition to running td512 with a 512-byte block size, the QuickLZ public distribution of version 1.5.1 Beta 7 was modified to iterate over a 256-byte and 512-byte block size, , and fpaq0 with a 64-byte block size. All were run on the Squash benchmark test data on a MacBook with a 2 GHz Quad-Core Intel Core i5 processor. The test beds for td512 and QuickLZ tests read file data into memory and compute loop count between 20 and 2000 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 24%, 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 best compression value for independent bytes with a uniform distribution throughout the file. The average compression reached by td512 is 10% above QuickLZ 512, 70% above QuickLZ 256, and 6.5% below fpaq0.

Table 1. 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 | 34.19 | 8.43 | -0.43 | 27.98 |
| asyoulik.txt | 29.85 | 6.93 | -1.10 | 24.84 |
| cp.html | 19.08 | 19.25 | 2.09 | 21.00 |
| dickens | 35.47 | 4.67 | -2.40 | 28.13 |
| fields.c | 25.57 | 33.98 | 20.13 | 25.45 |
| fireworks.jpeg | -0.35 | -1.56 | -3.30 | -7.08 |
| geo.protodata | 4.64 | 13.82 | 8.28 | 6.99 |
| grammar.lsp | 23.38 | 40.15 | 30.26 | 28.00 |
| kennedy.xls | 30.22 | 59.87 | 53.33 | 50.23 |
| lcet10.txt | 35.09 | 8.31 | -0.02 | 28.94 |
| mozilla | 18.63 | 30.86 | 22.91 | 22.36 |
| mr | 47.48 | 30.15 | 26.73 | 53.50 |
| nci | 38.14 | 62.05 | 53.08 | 57.03 |
| ooffice | 8.17 | 12.18 | 5.84 | 12.82 |
| osdb | 1.62 | -1.74 | -3.51 | 8.50 |
| paper-100k.pdf | 6.42 | 9.41 | 6.75 | 3.02 |
| plrabn12.txt | 35.50 | 3.52 | -2.71 | 27.27 |
| ptt5 | 70.07 | 74.94 | 68.82 | 75.17 |
| reymont | 21.58 | 25.51 | 13.87 | 27.27 |
| samba | 26.54 | 33.56 | 22.10 | 26.00 |
| sao | -0.39 | -1.75 | -3.51 | -3.81 |
| sum | 30.24 | 27.72 | 20.10 | 35.86 |
| urls.10K | 17.59 | 23.46 | 13.29 | 23.84 |
| webster | 22.84 | 17.60 | 3.23 | 23.68 |
| x-ray | 22.98 | -0.98 | -3.07 | 28.59 |
| xargs.1 | 29.22 | 18.31 | 7.26 | 25.31 |
| xml | 26.70 | 35.74 | 23.90 | 24.74 |
| Average Compression  Percent | td512  24.46 | QuickLZ 512  22.01 | QuickLZ 256  14.14 | fpaq0 64  26.13 |

Compression and decompression speeds, as shown in Table 2 and Table 3, are affected by the compressibility of the data and can vary by 10% or more with other activities on the test system. The average numbers provide a good basis for comparison. td512 compresses and decompresses uncompressible random data such as in fireworks.jpeg and paper-100k.pdf quickly because the program recognizes random data early, while QuickLZ attempts to compress all data. Different types of data compress and decompress at different rates. The files fields.c, reymont and xml have compression speeds that are much slower for td512 than QuickLZ. These files have repetitive strings that require longer to process than other data types such as text. This results in longer compression time than for other data types. On average, td512 compresses 7% slower than QuickLZ 512, but with better average compression, this is a benefit for td512. With 20% higher average compression speed and 70% 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 | 172632240 | 211823120 | 202785328 |
| asyoulik.txt | 166461424 | 225954864 | 213616048 |
| cp.html | 190720944 | 267423904 | 215815792 |
| dickens | 176394816 | 222975792 | 206563152 |
| fields.c | 285897440 | 446000000 | 253409088 |
| fireworks.jpeg | 885561152 | 506555552 | 317250016 |
| geo.protodata | 233901376 | 244008224 | 199979776 |
| grammar.lsp | 338272736 | 413444416 | 265785712 |
| kennedy.xls | 400055936 | 429776288 | 285564064 |
| lcet10.txt | 180674848 | 223197696 | 208071184 |
| mozilla | 227797440 | 257630144 | 202775488 |
| mr | 330337056 | 258794192 | 211236288 |
| nci | 273655456 | 322672704 | 234856256 |
| ooffice | 182092944 | 211757536 | 151364064 |
| osdb | 249571504 | 263553984 | 188376624 |
| paper-100k.pdf | 519796960 | 419672128 | 274530848 |
| plrabn12.txt | 180337200 | 211342560 | 209050320 |
| ptt5 | 315825216 | 362696832 | 256351632 |
| reymont | 101336464 | 227176816 | 191344080 |
| samba | 152993824 | 263454144 | 212519184 |
| sao | 586774336 | 314932224 | 233001680 |
| sum | 149960784 | 240503136 | 181232224 |
| urls.10K | 110652008 | 218378528 | 192299920 |
| webster | 133330448 | 214127392 | 196146512 |
| x-ray | 352037216 | 280306976 | 202577936 |
| xargs.1 | 325153856 | 352249984 | 264187504 |
| xml | 128116584 | 252421616 | 198827552 |
| Average Compression  Speed | td512  272234897 | QuickLZ 512  291215954 | QuickLZ 256  221093269 |

Average decompression speed in Table 3 shows that both QuickLZ 512 and 256 have better average decompression speeds than td512. The values for fireworks.jpeg, osdb, and paper-100k.pdf push the values significantly higher for QuickLZ. These are files that are mostly uncompressible and QuickLZ handles the decompression quicker than td512 because of handling one block of 512 bytes versus 8 blocks of 64 bytes. Decompressing data quickly is a benefit of 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 | 159422432 | 539234048 | 1551673472 |
| asyoulik.txt | 161313152 | 512000000 | 1301333248 |
| cp.html | 219669632 | 534260864 | 1068521728 |
| dickens | 153820384 | 629237184 | 1727229952 |
| fields.c | 247777776 | 467478240 | 407703712 |
| fireworks.jpeg | 13676999680 | 24576000000 | 15360000000 |
| geo.protodata | 1162627456 | 679724160 | 790186624 |
| grammar.lsp | 248066672 | 512000000 | 512000000 |
| kennedy.xls | 1300181888 | 718013952 | 665136960 |
| lcet10.txt | 158999264 | 585042496 | 1394614400 |
| mozilla | 673935936 | 640072000 | 740383616 |
| mr | 1553531264 | 644609536 | 909296128 |
| nci | 906188544 | 744126528 | 594265280 |
| ooffice | 537215552 | 554951488 | 706823488 |
| osdb | 4328619520 | 10397294592 | 8634958848 |
| Paper-100k.pdf | 1932075520 | 3657142784 | 3011764736 |
| plrabn12.txt | 156651824 | 667301888 | 2272603904 |
| ptt5 | 1199102848 | 838274496 | 791703744 |
| reymont | 164741024 | 381971072 | 400596768 |
| samba | 214736928 | 610505536 | 755151680 |
| sao | 9297363968 | 11455696896 | 10555621376 |
| sum | 374901952 | 451047616 | 482835456 |
| urls.10K | 206678544 | 638138176 | 985887616 |
| webster | 158841952 | 456402464 | 825541376 |
| x-ray | 1041445248 | 2053832192 | 2829419776 |
| xargs.1 | 281800000 | 315076928 | 512000000 |
| xml | 170623088 | 471947712 | 488733664 |
| Average Decompression Speed | td512  1506938224 | QuickLZ 512  2397458624 | QuickLZ 256  2232443983 |

# 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. 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, encode 1 or 2 unique values.
* 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 1/2 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 no more than 32 unique values occurred. 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 plus space, carriage return and comma to identify text data that can be compressed (see Table 4). When 90 percent of the data values are one of these characters as in most standard text, text mode gets 35% compression for 64 input values. From 3 to 7 bits are output for each character, based on frequency of occurrence. The adaptive text mode implemented in td64 supports replacement of the 8 lowest frequency characters with characters that are likely to occur in non-standard text. At present, XML (or HTML) and C code characters are handled.

Table 4: Most Frequently Occurring Text Chars Plus Space Character, Carriage Return, and Comma

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| # | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |
| Bits | 3 | 3 | 4 | 4 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 7 | 7 | 7 | 7 |
| Char | sp | e | t | a | i | n | o | s | h | r | d | l | u | c | m | f | cr | f | , | y | w | p | b |

Single value mode encodes a value that occurs at least 25% of the time in the data set as a 1 bit and all other values as a 0 bit plus an 8-bit value. This results in 11% compression for 64 input values, but is often much higher as checking for a single value is stopped when the number of occurrences reaches the minimum.

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, in groups of 7 bytes. This mode gets 11% compression for 64 input values and is used last because other modes can get higher compression. 7-bit mode requires at least 16 input values.

# Summary

This paper has shown that over the Squash benchmark data, td512 compresses 512-byte datasets at 24%, on average, while QuickLZ gets 22% using 512-byte blocks. Although QuickLZ has better average compression and decompression speed, td512 has better compression, and is much better performing that QuickLZ with 256-byte blocks. The huge benefit of td512 is that its performance remains 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. Over the Squash benchmark data, td512 gets 6.5% lower average compression than fpaq0.

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. New modes in the td64 interface will not affect the td512 interface.