LZ77 Report

Jun Eeo (njzj22)

Running Time of the Encoder and Decoder

Measurements were taken by compressing 4 files: an XML file taken from the XML Data Repository, an excerpt from the KJV Bible, a JPEG file, and a file with just zeroes. All file sizes were around 0.51MB. We found that changing the length of the lookahead buffer didn't increase the running time without an increase in the length of the window, thus we consider fixing $L = 255$ and increasing W.

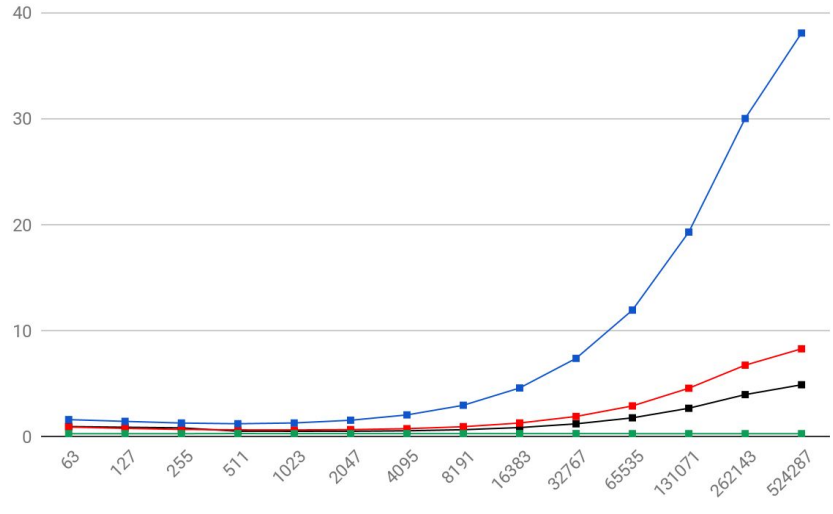

Figure 1: Running time of the encoder vs W. Key: KJV Bible, XML File, JPG Image, Zeroes File.

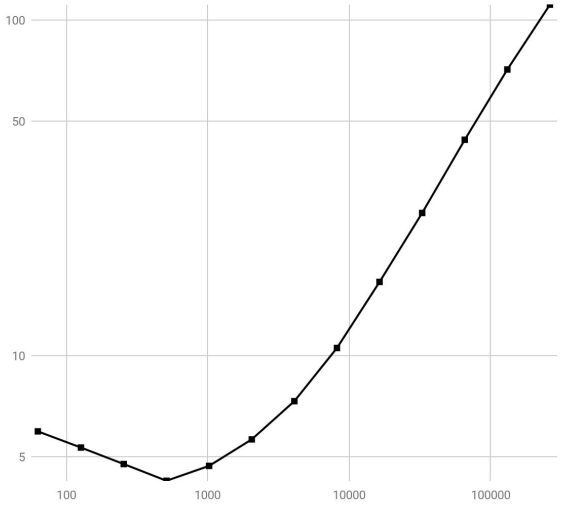

Figure 2: log-log plot of slowdown (time for JPEG image to compress / time for zeroes file) vs W.

Figure 1 shows that **the running time of the algorithm depends on how structured the input is (how much it repeats itself) and W**. Higher W leads to longer runtimes; this increase is much higher for inputs which are unstructured. This can be explained by having more mismatches before (potentially) finding a match in the window for unstructured input, hence finding substrings would take longer time. Figure 2 shows that (past some values for W) there is a power relationship between W and the slowdown in encoding time for highly unstructured vs structured data.

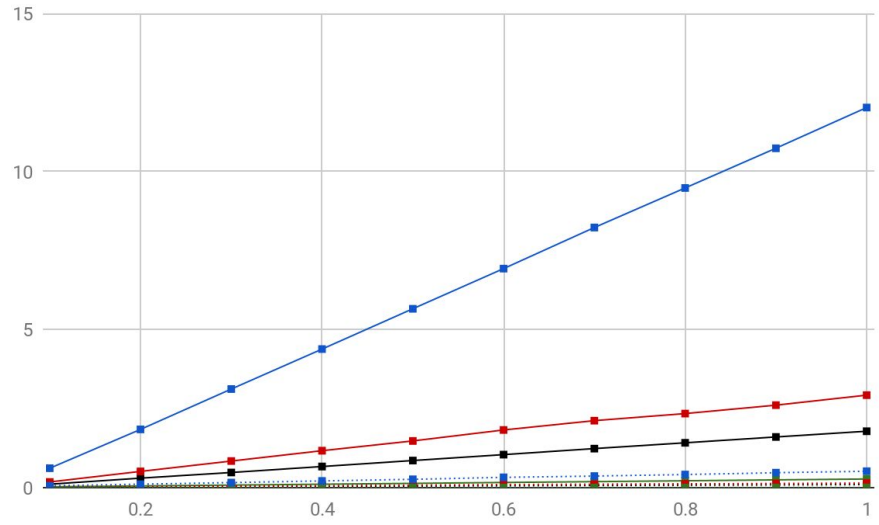

Figure 3: Encoding (solid lines) and decoding times (dotted lines) for increasing portions of the input, for fixed W=65535 and L=255.

From Figure 3 we see that both the decoder and encoder takes time linear in the size of the input, regardless of how (un)structured the input is.

The running time of both the decoder and encoder for fixed W and L is $O(kn)$ where k depends on the amount of structure in the input.

Key: KJV Bible, XML File, JPG Image, Zeroes File.

With decoding times we find that **less time is spent on compression means more time will be spent on decompression** (i.e. higher W => lower decompression time). However there is an ideal value W^* such that if $W > W^*$, the savings in decoding time no longer make up for the increased encoding time, as shown below.

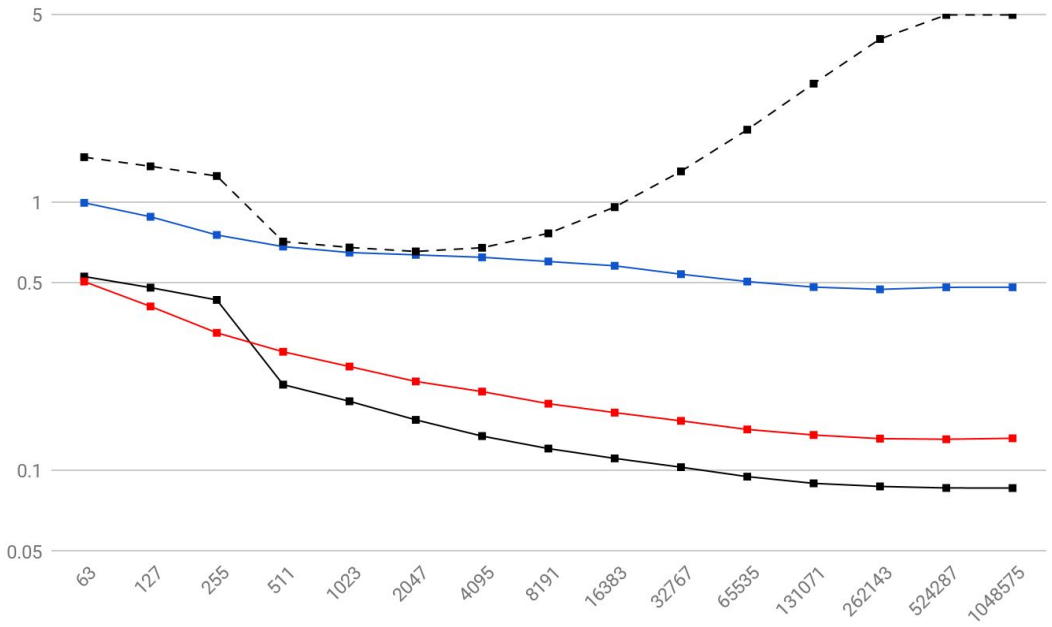

Figure 4: Logarithmic plot of decoding times vs W. Key: KJV Bible, XML File, JPG Image. Zeroes file was left out because all values < 0.001s and it made the graph less readable. Dashed black line = combined encoding and decoding times for XML File.

We also find that **decoding times for unstructured inputs are higher**. This is due to the decoder having to process fewer bits/triplets, because structured input yields comparatively fewer triplets. This is further highlighted by compressing sequences of 256 bytes; a 256-byte sequence has *n*-repetition if it can be divided in to *n* equal substrings. E.g. a sequence of 2 repetition is 0,1,2,3,...127,0,1,2,...,127, and a sequence with 1-repetition is 0,1,2,...255.

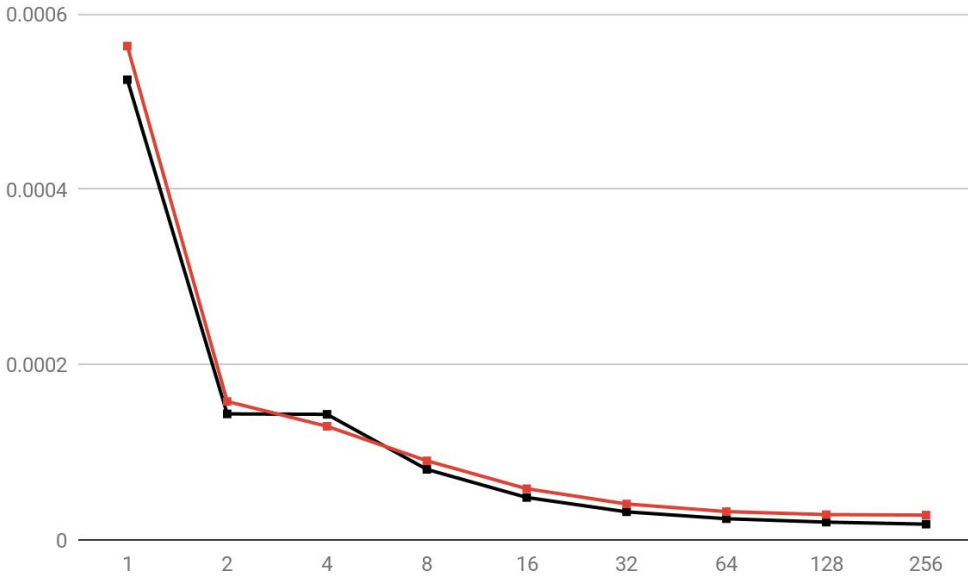

Figure 5: Decoding times vs n-repetition. Key: Compressed size / original size (scaled to fit), decoding time. For all inputs, L=W=255. Graph shows that better compression ratio => fewer bits => lower decoding time.

Compression Ratio

We show in Figure 6 that our observations (decoding time is proportional to 1/compression ratio) on small, constructed sequences also apply to “real life” data. A good property of LZ77 is that **data with more structure is compressed more compared to data with less structure** - e.g. the zeros file is compressed to below 5% of the original size. We use compressed / original size instead of the inverse because it is allows easier visualisation of the relationship with other results, namely the decoding time.

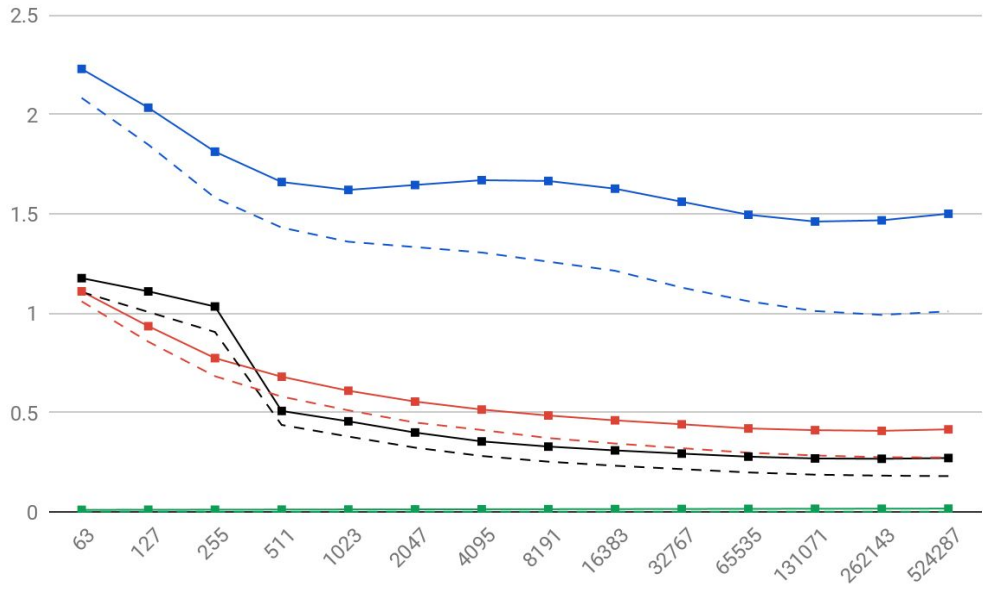

Figure 6: Compressed size/original size vs increasing values for W. Key: KJV Bible, XML File, JPG Image, Zeroes File. Dotted lines are decoding times, scaled as necessary.

Though Figure 6 suggests that increasing W yields better compression ratios (because the encoder can re-use more previously seen subsequences), this does not hold above some value of W. To show this, we compress a 33KB section of the Genesis chapter. Once W > 2**14 - 1, the cost of encoding triplets increases but no further compression of the input is possible, and the compression ratio worsens.

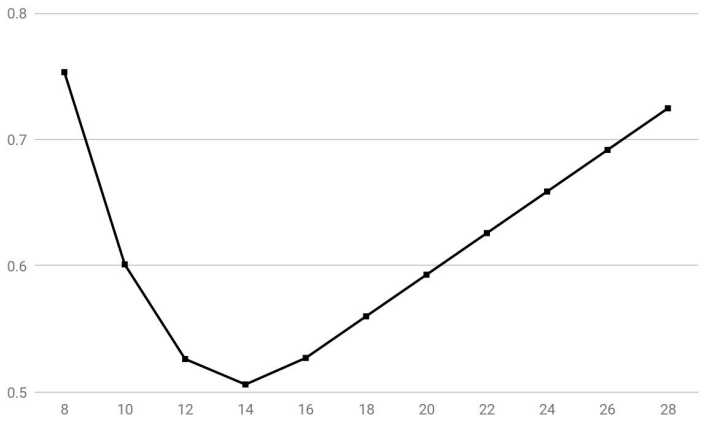

Figure 7: Compressed size/original size vs log2(W+1), i.e. number of bits used for encoding the distance. L = 255 for all cases.

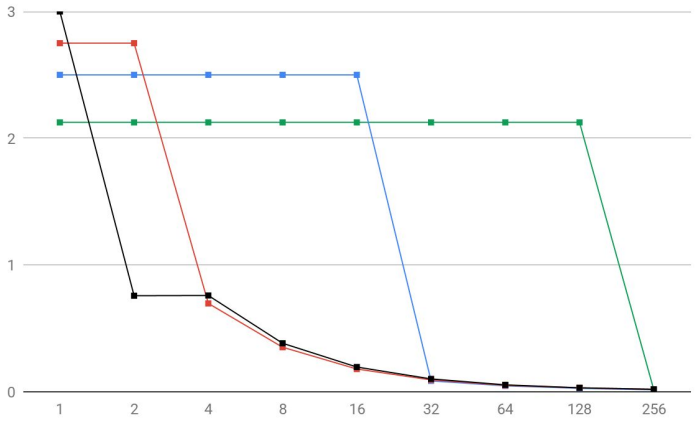

Figure 12: Compressed / Original size vs n-repetition. (L=255) Key: W=255, 63, 15, 1. We see that the compression ratios improve massively when W/2 >= 256/n, and is the best out of all values for W when W/2 ~= 256/n.

From Figure 12 we see that to improve compression ratios, picking a good value for W depends on the amount of structure in the data; in particular the “period” of the repetition should fit into the window.

Comparison Between LZSS and LZW

LZW uses a variable-bit length encoding. Measurements were made against the same set of files used to test the LZ77 algorithm. The zeroes file and XML file were left out. For LZ77 and LZSS, L is fixed to be 255 for all tests. Across all graphs we define n to be such that W=(2**n)-1 for LZ77/LZSS, or in the case of LZW, 2**n is the max size for the dictionary.

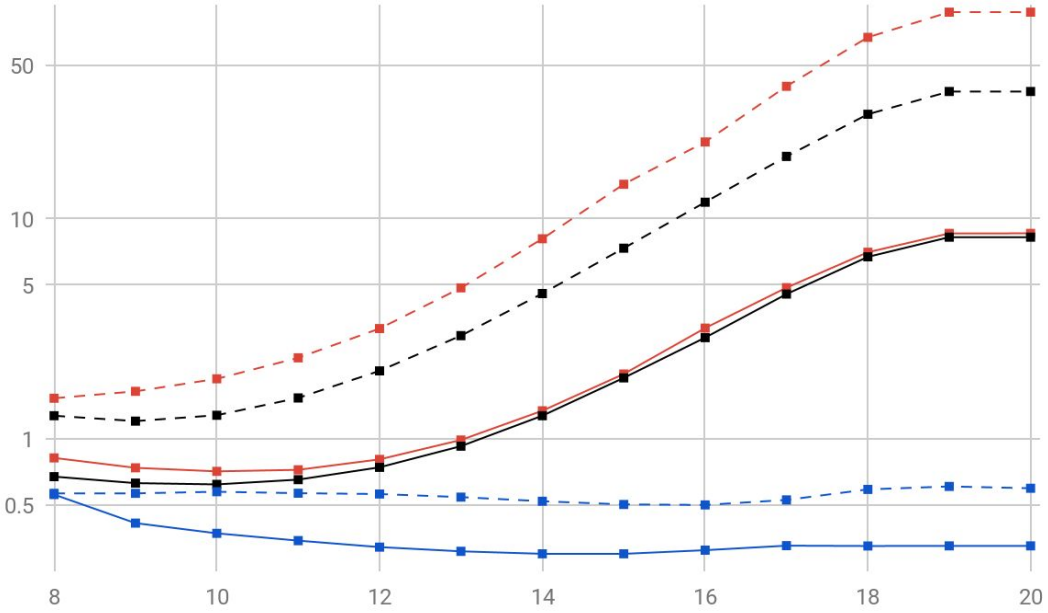

Figure 8: Logarithmic plot of encoding times using LZSS, LZW, and LZ77 vs n. Dotted line is for the JPG image, and solid line for the KJV Bible excerpt.

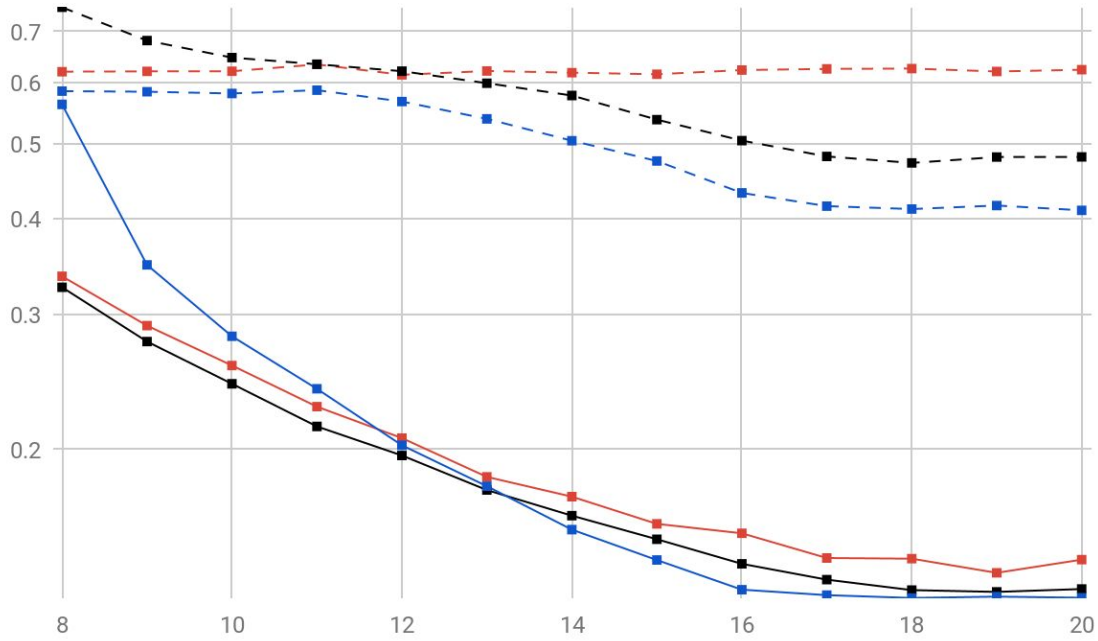

Figure 9: Logarithmic plot of decoding times using LZSS, LZW, and LZ77 vs n. Dotted line is for the JPG image, and solid line for the KJV Bible excerpt.

From Figures 8 and 9 we see that the **decoding and encoding times for LZSS is competitive with LZ77**, the constant amount of slowdown for most inputs may be explained by the additional complexity of the encoding and decoding process. We also see that the **encoding times for LZW is much faster than both LZSS and LZ77**, because LZW encoding is much simpler and no exhaustive search of a window is required.

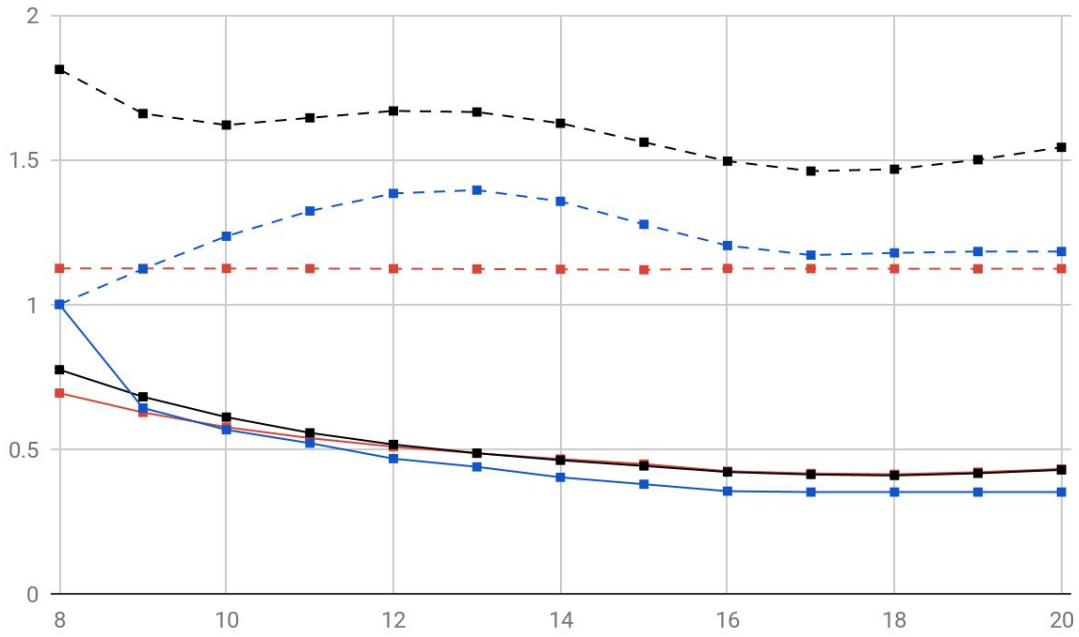

Figure 10: Compressed size/original size using LZSS, LZW, and LZ77 vs n. Dotted line is for the JPG image, and solid line for the KJV Bible excerpt.

From Figure 10 we see that the **compression ratio of LZSS is always better than or almost the same with LZ77 across all test inputs**, in particular the JPG image which LZ77 performed much better on. Also, **the compression ratio of LZW is very competitive against LZ77.** Figure 11 shows that LZSS also suffers from the same problem as LZ77 - compression ratio worsens when W increases beyond some value. However this does not apply for LZW with variable bit-length encoding.

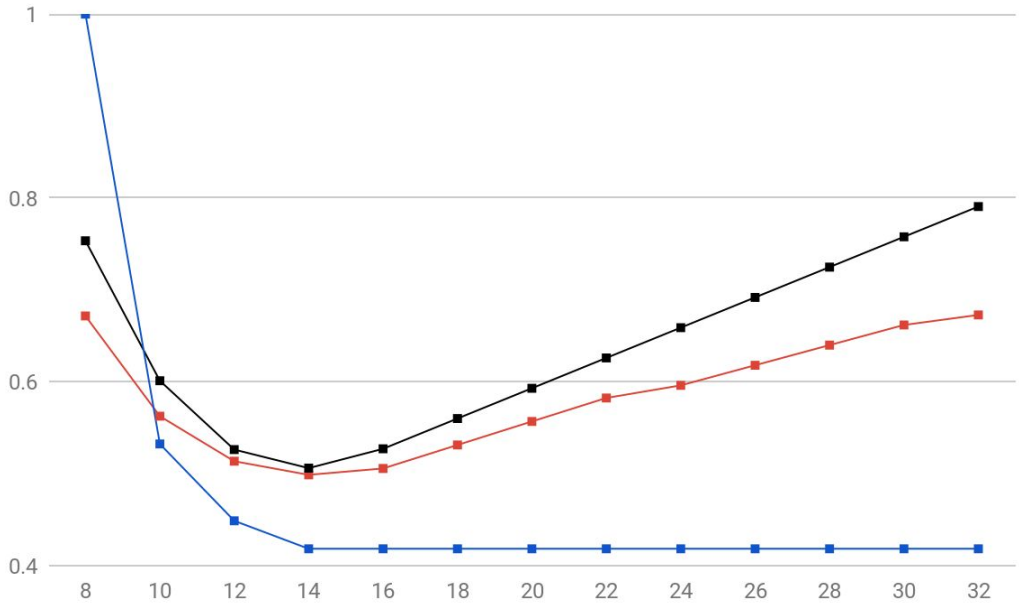

Figure 11: Compressed size/original size of the 31KB excerpt of Genesis vs n for LZ77, LZW and LZSS.

In conclusion **LZW is better than LZ77 in terms of total compression and decompression time and compression ratio.** It also has the added benefit that increasing W doesn't harm the compression ratio, whereas both LZSS and LZ77 are very sensitive to changes in W for non-trivial inputs. However **LZ77 is better than LZSS in terms of performance, but at the cost of compression ratio.**