LZ77 is a lossless dictionary-based compression algorithm first published in 1977 (Ziv & Lempel, 1977). Modifications of it form the basis of many modern compression algorithms (REF). Proceeding iteratively through the data, the compression algorithm maintains a search buffer of W symbols as a dictionary for encoding the current string which directly precedes the current encoding position. Compression occurs when strings of data at the current encoding position are replaced with a reference to a repeat copy of that data existing in the buffer; A space efficient compression comes from finding the longest such repeating string that exists in the search buffer at each encoding position. This match is encoded by a triplet consisting of how many symbols in preceding the current encoding position that the match occurred (D); the length of the match (L); and the next symbol in the unencoded data following the match string (s). The coding position is then advanced to directly succeeding this character and the algorithm continues until all text is encoded.

Decoding works via parsing the sequence of triplets iteratively. For each triplet [D, L, s] the data is reconstructed by copying L symbols from the position D symbols ago, followed by the symbol S. Because this is done sequentially and with reference to previous data, decompression can only be started from the beginning.

Implementation Details

The implementation used for testing in this report parameterises the buffer size W, which is the number of preceding symbols to the current encoding position; the maximum lookahead L, which is an absolute maximum length of string searched for in the buffer before moving on to the next encoding position; and the size of what is considered a symbol measured in byte length, S. A symbol would typically be considered to be one byte of data long. Varying these parameters will demonstrate in the following tests the trade-off between worst case running time of the encoder and the compression ratio achieved.

Compressed files contain a header encoding the number of bits used for each element of the reference triplets.

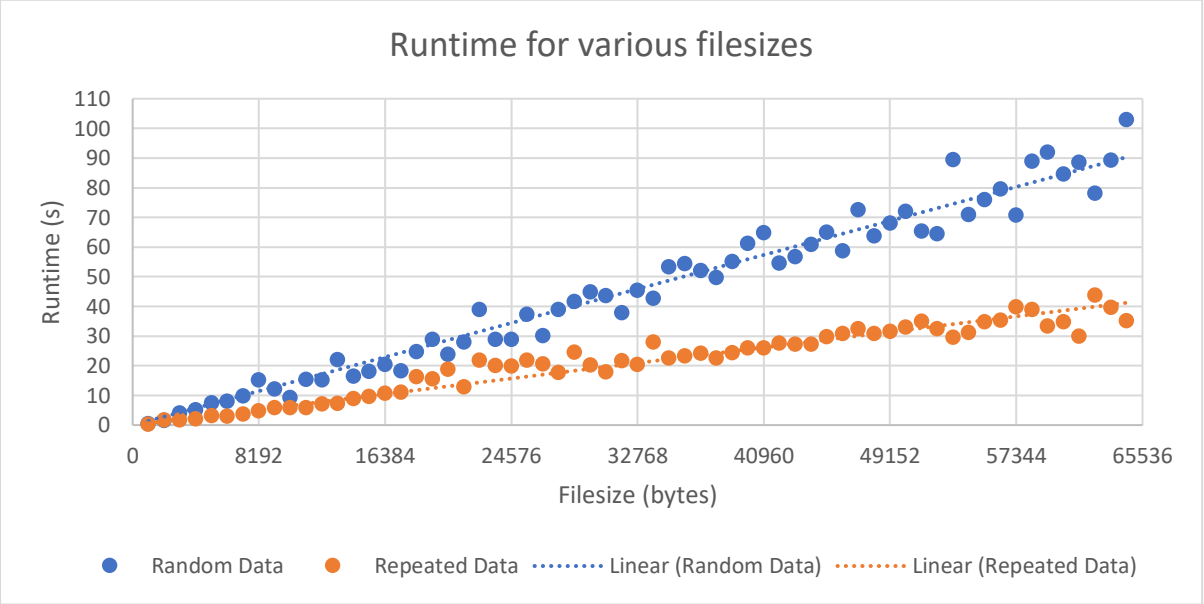
Test Data

Different of formats of data with different redundancy characteristics will demonstrate the properties of LZ77 best. For instance, highly repetitive data like bitmap images with few distinct colours will achieve a high compression ratio because the values for individual pixels are repeated. Both a solid black and rgb bitmap are used for testing. Some Image file formats already use a compression scheme, and therefore will not contain as much repetitive data and may even increase in size, an issue which is discussed in section 3. The PNG format (Portable Network Graphics) uses the DEFLATE algorithm, which is a derivation of both LZ77 and Huffman coding. The JPEG format uses lossy compression. Text files containing English natural language are a good candidate for compression because its plaintext encoding will have some redundancy. An even better example of this is a text encoded DNA string. DNA contains only 4 distinct characters, and these repeat in sequence so there will be a lot of redundancy. Less repetitive data like a stochastic binary file will compress badly because the values are not likely to repeat.

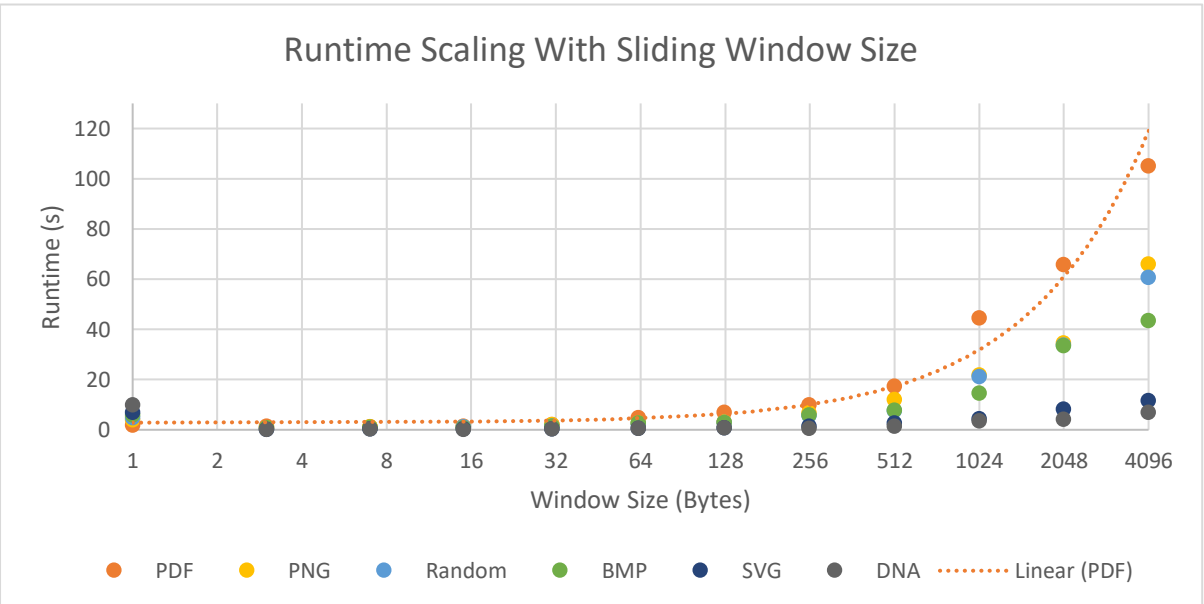
All files for testing have been included in the submission. Most tests were done on the Durham MIRA Server, unless otherwise stated; Some have been done on my own Laptop. The algorithm is not easily parallelisable and so the implementation is single threaded.

1 - Running Time of the Encoder

For each iteration of the algorithm, each encoding position is considered. The window is searched once for each additional matching character which takes a constant amount of time for a fixed window size. Each additional matching character moves the encoding position along by a single character.



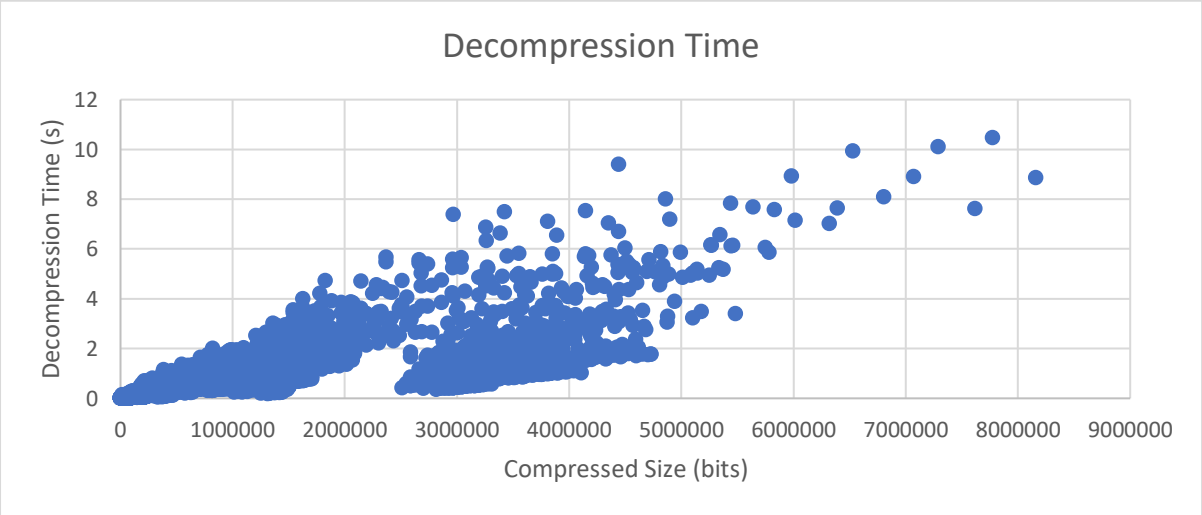
This graph shows the runtime for various truncated file sizes of two different datasets. One is random data, it will not contain much repeating data and therefore the entire window needs to be searched for each encoding position. The other is a string of identical bytes. This is a best case because the encoding position is advanced in greater steps due to the repetition, meaning there are less iterations of the algorithm. As can be deduced from the graph, the average runtime of the algorithm is O(n). In terms of memory complexity, the algoritm only needs to hold in memory the data in the buffer window and the data in the lookahead, so is O(W + L)



This graph shows that runtime is roughly linear with the size of the sliding window. Any variations are likely due to other tasks that the machine was running at the time affecting throughput of the algorithm.

2 - Running Time of the Decoder

The decoder takes a binary sequence of triplets and assembles the original data. It knows, either through hardcoding or through a file header, the number of bits needed for the distance-length pair and the number of bits for a symbol. Proceeding iteratively through each of the triplets (W, L, S), it constructs the original data string by appending the substring W symbols ago of length L, followed by S.



As can be deduced from the chart the average running time of the decoding algorithm is O(n).

3 - Compression Ratio

File	File Size (bits)	Symbol Size (bits)	Window Size (Symbols)	Search Size (Symbols)	Compressed Size (bits)	Compression Ratio
All Black BMP	691648	32	1023	1023	2208	313.2463768
BMP	368628	4	8191	31	142606	2.584940325
DNA Text	67716	4	4095	15	30680	2.207170795
English Text	1216720	8	8191	15	646674	1.881504437
SVG	80076	4	8191	15	51894	1.543068563
JPEG	12224	64	1	1	12234	0.999182606
PDF	316480	64	1	1	325734	0.971590316
WAV	2436672	64	1	1	2512842	0.969687708
PNG	182400	64	1	1	188124	0.96957326
Random Text	131136	64	1	1	135258	0.969524908

The above shows the best-case compression ratio for all the test files from several tests. It shows that the JPEG, PDF, WAV and PNG files were not compressible. All these files already use a compression algorithm when they are encoded, so it makes sense that they increase in size when put through LZ77. The algorithm encodes all data as a triple, even if there is no match. In practice, this results in using more bits to store a match of length zero as opposed to encoding the literal characters, this is because several all zero bits are used for storing the distance-length pair. This

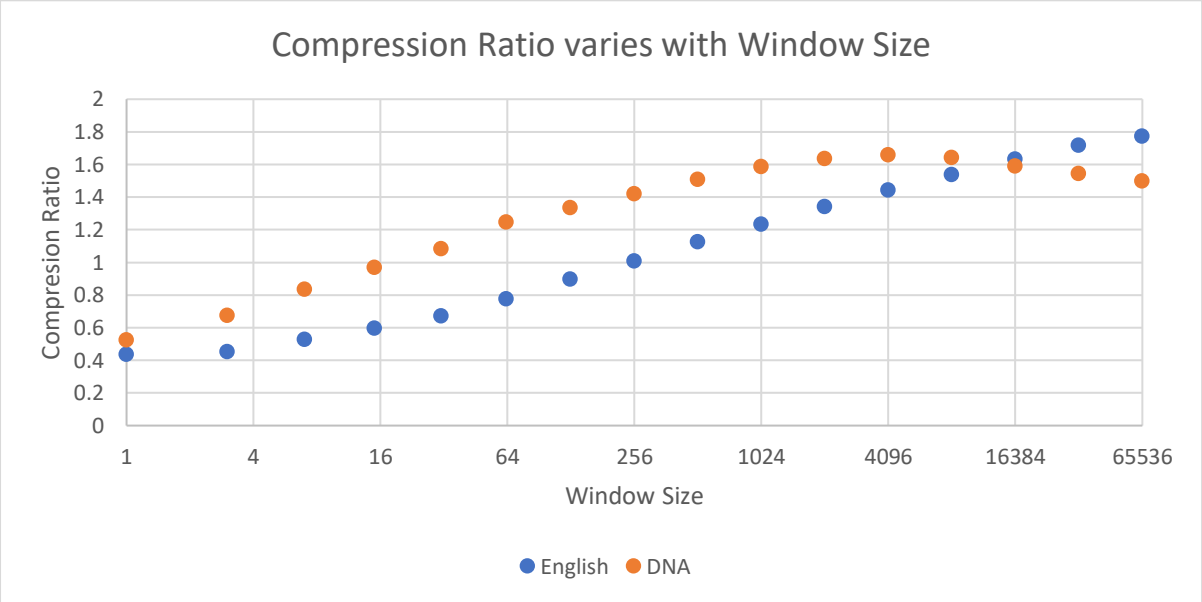
results in an increase in the file size. As can be seen, the minimal increase in file size for these files occurred when symbols were encoded at longer values, and the distance-length pair only take up 1 bit each.

The random text file was also uncompressible because it is highly unlikely that there will be any repeating strings in the file.

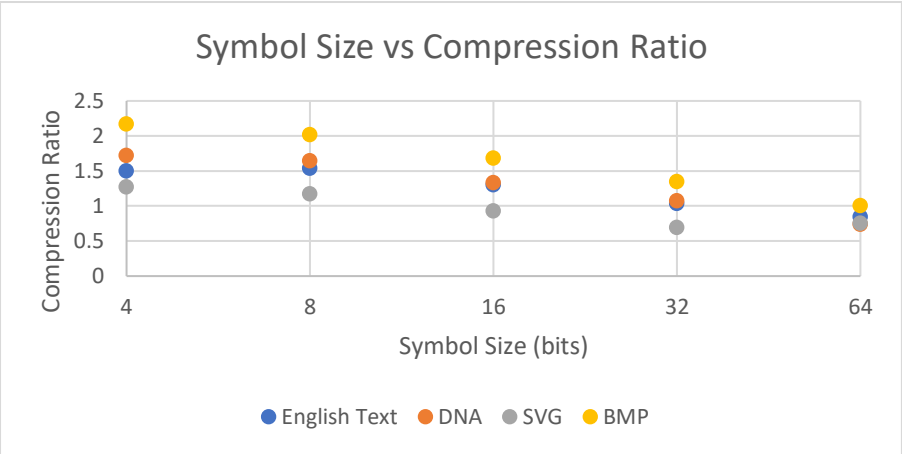
The best-case ratio for the files that were compressible suggests there is no global optimal parameters for compression, and that the optimal parameters depend on the source of data. It makes sense that the SVG, which is encoded in XML text, and English text files which are both languages designed to be human readable, have some redundancy in them.

The highest ratio was a bitmap containing only black pixels. This file is essentially a string of repeating symbols, so it makes sense that it can be compressed by a factor of 313.

One of the main assumptions of LZ77 is that repeating data is grouped locally. Data just prior to the search buffer may contain redundant information that is not utilized. While the search window can be made longer so it includes this data, this will negatively affect runtime and will, in some cases increase the encoded file size because more bits are needed to store the value of the distance in each triplet. For a window size W, a maximum lookahead L, and a symbol size S, the number of bits needed to store each reference triplet is $\lceil \log_2 W \rceil + \lceil \log_2 L \rceil + S$ where $\lceil x \rceil$ is the smallest integer greater than or equal to x.



As can be seen, the extra matches afforded by an increase in the window size starts to become offset by the increased number of bits needed to store the data. LZSS improves on this. The same would be true of the maximum length.



This graph shows the effect of symbol size on compression performance. ASCII data is encoded in 8 bit segments, however a Unicode symbol can be up to 32 bits long. Smaller symbols perform better because, for example, there are less distinct 4 bit symbols than 8 bit symbols, so they are more likely to repeat. This also depends on the original encoding of the input. For instance, ASCII text is encoded in 8 bit segments, while RGBA is encoded in 64 bit symbols.

4 - Other Compression Techniques - LZSS

The original LZ77 algorithm encodes all repeated data, even if the match length is only of length 1. In practice, this results in using more bits to store a match of length 1 as opposed to encoding the literal characters, this is because several all zero bits are used for storing the distance-length pair. LZSS uses a single bit flag in each reference which when set tells the decoder that a distance-length pair follows; if not set, a literal symbol follows. A breakeven point will be declared, where a match shorter than the breakeven point will be declared as literals.

File	File Size (bits)	Symbol Size (bits)	Window Size (Symbols)	Search Size (Symbols)	Compressed Size (bits)	Compression Ratio
All Black BMP	691648	32	1023	1023	2208	313.2463768
BMP	368628	4	8191	31	148835	2.476756139
DNA Text	67716	4	4095	15	32149	2.106317459
English Text	1216720	8	8191	15	671435	1.81211882
SVG	80076	4	8191	15	54228	1.476654127
JPEG	12224	64	1	1	12061	1.013514634
PDF	316480	64	1	1	31592	0.98448784
WAV	2436672	64	1	1	2474769	0.984605836
PNG	182400	64	1	1	185274	0.98448784
Random Text	131136	64	1	1	133209	0.984437988

The above table shows the compression ratio of running LZSS on the test files under the optimal parameters shown in section 3.

References

Ziv, J. & Lempel, A., 1977. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory,* 23(3).