# Digital Communication - Summative Assessment

## Lempel-Ziv Compression

The efficiency of the Lempel-Ziv (LZ77) algorithm is heavily based on the values chosen for the sizes of the sliding window and look-ahead buffer. If either values are too small then there is a risk the search space may be too small and matching text will be missed. On the other hand, if either are too large then the program’s runtime could increase dramatically, which would cause issue when compressing larger files (there is also the potential for wasted bytes).

For my implementation of LZ77 the sizes of the sliding window and look-ahead buffer are represented by a maximum number of bits allocated, as this improved compression rate of files. This meant that the maximum size for either value had to be a multiple of 2, and so if the look-ahead buffer was assigned 6 bits, the furthest it could look ahead would be 64 places.

Once I had implemented my algorithm I first decided to determine if either two values could had an optimum for all files being tested. If this were to be the case then it would massively help with later analyse of the algorithm. Using a selection of values for both the sliding window and look-ahead buffer, I compressed a variety of randomly generated txt and csv files.

**Compressing an individual file** (1.1)

Above are two graphs demonstrating the runtime and compression size of a 10kb text file. The x-axis represents the sliding window size, with the coloured series representing the look-ahead buffer’s size.

In analysing the two graphs it is clear that the increasing the look-ahead buffers value by a single bit causes the runtime to nearly double each time, and in terms of compression leads to a larger compressed file than using a smaller look-ahead value. We notice however that increasing the sliding windows value causes the size of the compressed file to decrease, and this is only at the expense of a slight increase in time (we can see increasing the value by two bits costs significantly less time than increasing the look-ahead value). The size of the compressed file does appear to increase slightly between 12 and 14 bit values for the sliding window, and so from trial and error you can determine an optimum sliding window value (for which I found it to be 13).

The discussed graphs above are just an example to show that increasing the look-ahead value past 8 bits only causes harm to the compression of a file, and there is a sliding window value for which compression is maximum. When repeated on other txt and csv files I found this to consistently be the case.

**Compressing different files** (1.2)

The graphs above show the runtime of the program when compressing various files at different sizes. The sliding window and look-ahead buffer values were kept constant for each file as to produce a fair representation of the runtime variation between different file sizes. The graphs demonstrate a fairly linear growth of runtime for the text files, with CSV files following the same trend.

Notice that for the CSV file with 278kb of data the runtime is significantly reduced. At first glance this appeared to be an error, but after looking into it I discovered that the file was simply very repetitive in terms of its data content. CSV files are a simple file format that store tabular data (often from a spreadsheet), and often tabular data will repeat (e.g. when storing job roles). LZ77 is much more effective when the data is repeated, and so the 278kb file benefited heavily from being compressed by LZ77.

Above displays two graphs that correlate to the same compression as done previously. These graphs show how well compressed the different files were when encoded by the LZ77 algorithm. The text files show a fairly similar compression size at around 60%, with the exception for the smaller 10kb text file. Since the algorithm relies on patterns in text, the exception with the 10kb file most likely stems from the file having few similarities in the text.

As for the CSV files, we can see that they typically compress at a much better rate than the text files. As mentioned above, CSV files store tabular data and tabular data tends to repeat. This means that more data can be stored in fewer bytes in the compressed file, and hence the compressed file takes up less room. When it comes to the 278kb file, we can see that is has been decreased to around 8% of the original file size. This just shows how effective LZ77 can be when working with a large amount of repeating data. We also notice that for the 2kb csv file that the size has increased after encoding\*\*\*

**Maximum, average & minimum runtimes** (1.3)

To demonstrate maximum, average and minimum runtimes of compression we will use different text files, all 68 bytes in length. Maximum runtime will occur when the file contains no patterns - the file is simply all different characters (i.e. abcdef…). The best case (or minimum) runtime will occur when all characters are the same, as the algorithm will recognise this pattern very easily. To calculate average runtime I simply produced 5 random text files of 68 bytes and averaged the results from them.

Since we’ve seen compression we should also look at maximum, average and minimum runtimes of decompression. Again we will use the same text document as before and simply decompress the files and record the time taken in a graph.

We notice that decompression appears to take longer than compression. This is primarily due to the fact that the file we are dealing with is so small, and more time is spent setting up the decompression than actually decompressing. As discussed later in 2.1, decompressing is normally much more efficient.

**Decoding various files** (2.1)

The two graphs above show the time taken to decode txt and csv files. The same previously compressed files in section 1.2 have been used here in decompression. Visually the graphs resemble the compression graphs of the same data extremely closely, only with a slight variation in the y axis, where we can see the time taken to decode the files is much shorter than that to encode them.

Interestingly though, in contract to encoding when decoding the data we can see the CSV files are decoded at much the same speed as the equivalent sized text file. This is because although the time taken to encode files depends heavily on the size of the original file, its structure and the values of the look-ahead buffer and sliding window, the time taken to decode files mainly depends on its size. The similarities between the two graphs is faithful to this idea, so therefore the time taken to compress a file has no effect on the time taken to decompress it.

**Decoding an individual file** (2.2)

To gain further insight into how the window sizes affect the runtime of the decode aspect of LZ77 the same tests have been run on the 10kb text file (as from 1.1), with the same values used for both the sliding window and look-ahead buffer. As in 1.1, the x-axis represents the sliding window size, with the coloured series representing the look-ahead buffer’s size. The graph only strengths the idea presented in 2.1 that values such as the sliding window and look-ahead buffer have no effect on the decode time.

**More on compression ratios** (3.1) \*\*\*

To further evaluate compression rates, the compression rate of a csv file (11kb) was taken for different combination of sliding window and look-ahead buffer values. We notice it appears the same as the text file analysed in 1.1, and we can deduce the file type has no effect on the ratios.

**Comparison to other compression techniques** (4.1)

Below is a table of data comparing the LZ77 algorithm described in this report with an ‘off-the-shelf’ compression technique called ‘Aricom’ (<https://sourceforge.net/projects/aricom/>). So far all the tests have been ran on a university machine, so for drawing comparisons between the two programs there will be two sets of data produced: one from a university machine, the other from a personal (much faster) computer.

The sliding window and look-ahead buffer sizes have been optimised through a trial and error technique (demonstrated in 1.1) for the LZ77 algorithm in the following tables.

**Ran on university machine**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | LZ77  COMPRESSED SIZE/ ORIGINAL SIZE (%) | Aricom  COMPRESSED SIZE/ ORIGINAL SIZE (%) | LZ77  COMPRESSION TIME (s) | Aricom  Compression TIME (s) |
| 10KB (TXT) | 0.65 | 0.60 | 2.466 | 0.208 |
| 20KB (TXT) | 0.63 | 0.55 | 9.682 | 0.355 |
| 50KB (TXT) | 0.60 | 0.55 | 50.414 | 0.801 |
| 100KB (TXT) | 0.58 | 0.55 | 108.56 | 1.535 |
| 200KB (TXT) | 0.59 | 0.55 | 233.497 | 3.143 |
| 2KB (CSV) | 1.148 | 1.757 | 0.117 | 0.090 |
| 11KB (CSV) | 0.601 | 0.75 | 3.222 | 0.247 |
| 69KB (CSV) | 0.44 | 0.62 | 44.340 | 1.175 |
| 119KB (CSV) | 0.42 | 0.69 | 71.566 | 2.130 |
| 278KB (CSV) | 0.07 | 0.62 | 24.420 | 4.455 |

**Ran on personal machine**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | LZ77  COMPRESSED SIZE/ ORIGINAL SIZE (%) | Aricom  COMPRESSED SIZE/ ORIGINAL SIZE (%) | LZ77  COMPRESSION TIME | Aricom  Compression TIME |
| 10KB (TXT) |  |  |  |  |
| 20KB (TXT) |  |  |  |  |
| 50KB (TXT) |  |  |  |  |
| 100KB (TXT) |  |  |  |  |
| 200KB (TXT) |  |  |  |  |
| 2KB (CSV) |  |  |  |  |
| 11KB (CSV) |  |  |  |  |
| 69KB (CSV) |  |  |  |  |
| 119KB (CSV) |  |  |  |  |
| 278KB (CSV) |  |  |  |  |

**Complete table above**

**Fix error with 2kb csv file for rest of tables**

**Write about above tables**

**Fix 3.1**