Optimizing Lempel Ziv Welch for DNA Compression

A Thesis

Presented to

The Division of Mathematics and Natural Sciences

Reed College

In Partial Fulfillment

of the Requirements for the Degree

Bachelor of Arts

Caden Corontzos

May 2023

Approved for the Division
(Computer Science)

Eitan Frachtenberg

Acknowledgements

I want to thank a few people.

Preface

This is an example of a thesis setup to use the reed thesis document class (for LaTeX) and the R bookdown package, in general.

List of Abbreviations

LZW Lempel Ziv Welch

Table of Contents

List of Tables

List of Figures

Abstract

The Lempel Ziv Welch compression algorithm is a lossless data compression algorithm used for numerous applications, including the Unix file compression utility `compress` and the GIF image format. Storing, reading, and transferring enormous amounts of data is often an issue in the biological field, especially when concerning DNA. This thesis explores the application of Lempel Ziv Welch to the compression of DNA. A variety of different optimization of the original LZW algorithm are explore included palatalizing, multiple dictionaries, and some other cool thing here broh.

Dedication

You can have a dedication here if you wish.

Introduction

When dealing with DNA, it

Chapter 1

Backround and Motivations

This thesis deals with some high level topics and uses language specific to compression research. This chapter tries to give brief summaries and examples of the relevant topics to be discussed so readers of all experience levels can put our results into context.

1.1 What is information?

Suppose you had an idea that you wanted to share with another person. Humans have many ways to communicate information; you could send a text message, you could tell them with words, you could tell them with sign language. But regardless of the medium, you have some idea that you want to get across. Does it matter if the other person gets your message exactly? Or can it be part of the message? If someone asks you "Where library", despite the lack of prepositions you still understand what they mean. So did that person convey any less information than a person who asks "Where is the library?" Clearly, information is fundamental to how humans interact and how they understand the world, but defining it proves difficult. For our purposes, let us assume that information is something that can be interpreted to glean information that you didn't know before.

1.2 Compression: A history

1.3 Compression Metrics

1.3.1 Compression Ratio

Compression Ratio is the measure of size reduction achieved by a compression algorithm. It is typically expressd as a ratio of the size of the uncompressed data (OS) to the size of the compressed data ({CS}).

$$CR = \frac{OS}{CS}$$

So a higher compression ratio means a more effective compression algorithm, and means that we were able to store more data in less space, allowing for easier storage and transfer.

1.3.2 Runtime

The runtime is also an important part of evaluating the effectiveness of a compression algorithm. If you have the option of two compression algorithms, one with a compression ratio of 2.0, and another with a compression ratio of 2.15 but takes twice as long as the other, you may opt for a lower compression ratio to save time.

1.3.3 Memory Usage

Memory usage is closely tied with runtime when it comes to compression algorithms. Memory generally refers to information that programs track as they are running on a computer. So do reduce our runtime and make a more effective compression algorithm, we want to be saving only the most important data that our algoritm needs in order to reduce our memory usage.

1.4 Lossless vs. Lossy Compression

1.4.1 Lossy

Lossy compression is based on the idea that not all information is vital. For instance, when saving a picture on your computer, your computer may save it in the .jpeg format to save space. Jpegs lose some of the information in the original picture and

produce an overall lower quality picture, but the general information in the picture is preserved. Another example

1.4.2 Lossless

Lossless compression is the compression of data with the goal of preserving all the information in the data. As a result, lossless compression algorithms usually don't compress as well as their lossy counterparts. Examples of lossless compression algoritms are Huffman Encoding and Lempel Ziv Welch, which is the focus of this thesis.

1.5 Examples of Compression Algorithms

1.5.1 Run Length Encoding

Run Length Encoding (RLE) is on of the simplest and most intuitive forms of compression. We can take advantage of redundant runs of characters in a sequence by just giving the number of times each character appears. Suppose you want to send the following message

AAGCTTTTTTTTGGGGGCCCT

Even if this message did mean something, we can get the information across without repeating ourselves. When writing a grocery list, you don't write "egg egg egg egg", you say "4 eggs". RLE uses this same strategdy.

2A1G1C8T5G3C1T

We could compress this even further if we omit the 1 on characters that only appear once.

1.5.2 Huffman

Huffman Encoding is a strategdy that assigns variable length code to certain symbols in the data. The goal is to assign short codes to frequently appearing symbols and longer codes to less frequent symbols.

Put example here

1.5.3 Arithmetic

Arithmetic encoding is another lossless compression algorithm that uses probability to asssign codes to symbols in the message. Unlike Huffman, arithmetic enoding assins

a single code to the whole message, rather than seperate codes for each symbol.

Here is a simple example. Say we want to encode a string of characters "ACCGGGGTTT". The probability of each symbol in the message are

- P(A) = 1/10
- P(C) = 2/10
- P(G) = 4/10
- P(T) = 3/10

We want to represent the message as a fractional number between 0 and 1. We will divide the interval [0,1] into sub intervals using the probabilities of each character in the message. That way, each symbol is reprsented by the sub-interval that corresponds to its probablity.

Arithmetic encoding can have a better compression ratio that Huffman in some cases, but the computation time is often not worth the payoff.

1.5.4 Lempel Ziv Welch

Lempel Ziv Welch is another lossless compression algorithm. When compressing, LZW builds a dictionary of codewords, where codewords represent strings previously seen in the message. As it compresses the message, the dictionary grows. The compression algorithm leaves behind the codewords and some of the original characters, allowing the decompression algorithm to build up the same dictionary as it decompresses the message.

Here is a simple example. We may be sending messages with the characters {'A', 'C', 'T', 'G'}, so I will start with those in my dictionary. Say we want to send the message

"AAGGAATCC"

When we compress, we start at the beginning of the message and scan through.

Chapter 2

Optimizing LZW: Approach

To restate the goal of this thesis, we seek to optimize LZW for use in compression of DNA. I chose to write in C++.

2.1 Supporting Research

There has been several attempts to optimize LZW by computer science researchers.

There has also been attemps to generally improve performance of LZW

2.2 Corpora

Most compression papers make use of a Corpus, which is a collection of files to run a compression algorithm on in order to ascess performance and to compare different algorithms to one another.

In the world of DNA compression, there are several academic papers on the subject. One of the first and most popular of the papers was published in 1994, and the selection of DNA sequences used in the paper have become an informal corpus for the subject of DNA compression (Grumbach & Tahi, 1994).

```
Warning in read.table(file = file, header = header, sep = sep,
quote = quote, : incomplete final line found by readTableHeader on
'data/corpus_1_summary.csv'
```

Name	Size.bytes.
hehcmv	229354

Name	Size.bytes.
humdyst	38770
humghcs	66495
humhbb	73308
humhdab	58864
humprtb	56737
mpomtcg	186609
mtpacga	100314
vaccg	191737

Another, newer paper aimed to create a corpus specifically for compressing DNA (Pratas & Pinho, 2018). They put together a corpus of DNA sequences for this purpose, as summarized below. Since the papers publishing, it has been cited by several DNA compression papers.

Name	Size.bytes.
AeCa	1591049
AgPh	43970
BuEb	18940
DaRe	62565020
DrMe	32181429
EnIn	26403087
EsCo	4641652
GaGa	148532294
HaHi	3890005
HePy	1667825
HoSa	189752667
OrSa	43262523
PlFa	8986712
ScPo	10652155
YeMi	73689

This particular dataset is publicly available at this link.

2.3 A Starting Point

As a starting point, we thought it was best to get a working implementation of LZW in C++ on regular text files, optimize it as much as we could, and then try variations from there, opitmizing it for DNA.

2.3.1 Growing Codewords and Bit Output

When reading files on the computer, most characters are stored as bytes, which is made up of 8 bits. For instance `01000001` stands for the letter 'A' in ASCII encoding. Numbers are more simple to display, so `00000001` is 1, `00000010` is 2, and so on.

But if we are translating numbers to binary, we don't need all of the bits in a byte. In binary, `1` is the same as `01` is the same as `00000000000001`. So when we are outputting codewords for LZW, we don't necessarily need to ouput a whole byte. We can have growing codewords.

As the number of codewords grows, the number of bits needed to represent it also grows. So if we are on codeword 8, we need 4 bits since 8 is `1000`. As our dictionary grows, we can grow the number of bits needed to display a codeword and save a lot of space in our compressed document.

So we needed a method of ouputting bits one by one, and reading in bits one by one. This is not something that is supported in C++ on its own. We were able to create this functionality bydefining a class.

```
// BitInput: Read a single bit at a time from an input stream.
// Before reading any bits, ensure your input stream still has valid inputs.
class BitInput {
 public:
  // Construct with an input stream
  BitInput(const char* input);

  BitInput(const BitInput&) = default;
  BitInput(BitInput&&) = default;

  // bool eof();
  // Read a single bit (or trailing zero)
  // Allowed to crash or throw an exception if called past end-of-file.
  bool input_bit();
```

```
  int read_n_bits(int n);
}

// BitOutput: Write a single bit at a time to an output stream
// Make sure all bits are written out by the time the destructor is done.
class BitOutput {
 public:
  // Construct with an input stream
  BitOutput(std::ostream& os);

  // Flushes out any remaining output bits and trailing zeros, if any:
  ~BitOutput();

  BitOutput(const BitOutput&) = default;
  BitOutput(BitOutput&&) = default;

  // Output a single bit (buffered)
  void output_bit(bool bit);

  void output_n_bits(int bits, int n);
}
```

So when we are encoding and need to output a codeword, we can `output_n_bits`, where `n` is the number of bits needed to display our greatest codeword. When decoding, we can just `read_n_bits`.

2.3.2 Getting EOF to work

One of the very early issues with the implementation was how to denote the end of a file. The early implementation would work for some files, but for others the very last part of the file would be lost after encoding and then decodeing.

In theoretical implementations of LZW, computer scientists tend to denote the end of a message with a special character, one that isn't seen anywhere else in the file. In this initial implementation, that wasn't possible because we wanted to be able to compress any file with any characters.

The solution was to reserve a codeword to mark the end of the file. So we start

with a starting dictionary containint all ASCII characters.

```
std::unordered_map<std::string, int> dictionary;
for (int i = 0; i < 256; ++i){
    std::string str1(1, char(i));
    dictionary[str1] = i;
}
```

Then use the code 256 to denote the end of file. So the algorithm goes along reading a file. It builds up a current string character by character, adding the character to the string and checking if it has seen that sequence before. Once it find the end of file, we stop and output the EOF codeword.

The problem was, what about what is left over? Suppose we are reading a file, and the file ends with "ACCT". If "A" is in the dictionary, we see if "AC" is in the dictionary, and so on. This leaves us with three possible cases when we reached the end of the file

1. "ACC" was in the dictionary but "ACCT" was not. This means we can output the codeword for "ACC", follow it by the character "T", and we are done. This is the ideal scenario, because nothing is left over when we output the EOF codeword
2. "ACCT" was in the dictionary: This means we have one more codeword to output, but since we reached the end of the file, we never got to output it.
3. "AC" was in the dictionary, but "ACC" was not: in this case, we would output the codeword for "AC" output the character "C", and then start looping again starting at "T". But we reach the end of the file, so we output EOF before outputting T.

We solved this issue by adding 2 extra bits after the EOF codeword. These bits denote the case that occured

```
// after we've encoded, we either have
// no current block (case 0)
// we have a current block that is a single character (case 1)
// otherwise we have a current block > 1 byte (default)
switch (currentBlock.length()){
case 0:
    bit_output.output_bit(false);
```

```
        bit_output.output_bit(false);
        break;
    case 1:
        bit_output.output_bit(false);
        bit_output.output_bit(true);
        bit_output.output_n_bits((int) currentBlock[0], CHAR_BIT);
        break;
    default:
        bit_output.output_bit(true);
        bit_output.output_bit(true);

        int code = dictionary[currentBlock];
        bit_output.output_n_bits(code, codeword_size);
        break;
    }
```

So when the decoded is reading and encounters the EOF codeword, it can look at the next two bits to see if anything is left over.

2.3.3 Dictionary Accesses

Another way that we

2.3.4 Using Const Char *

2.4 Evaluating Performance

2.5 Trying Different Dictionaries

2.5.1 Direct Map

2.5.2 Multiple Indexed Dictionaries

Chapter 3

Graphics, References, and Labels

3.1 Figures

If your thesis has a lot of figures, *R Markdown* might behave better for you than that other word processor. One perk is that it will automatically number the figures accordingly in each chapter. You'll also be able to create a label for each figure, add a caption, and then reference the figure in a way similar to what we saw with tables earlier. If you label your figures, you can move the figures around and *R Markdown* will automatically adjust the numbering for you. No need for you to remember! So that you don't have to get too far into LaTeX to do this, a couple **R** functions have been created for you to assist. You'll see their use below.

In the **R** chunk below, we will load in a picture stored as `reed.jpg` in our main directory. We then give it the caption of "Reed logo", the label of "reedlogo", and specify that this is a figure. Make note of the different **R** chunk options that are given in the R Markdown file (not shown in the knitted document).

```
include_graphics(path = "figure/reed.jpg")
```

Figure 3.1: Reed logo

Here is a reference to the Reed logo: Figure 3.1. Note the use of the `fig:` code

here. By naming the **R** chunk that contains the figure, we can then reference that figure later as done in the first sentence here. We can also specify the caption for the figure via the R chunk option `fig.cap`.

Below we will investigate how to save the output of an **R** plot and label it in a way similar to that done above. Recall the `flights` dataset from Chapter **??**. (Note that we've shown a different way to reference a section or chapter here.) We will next explore a bar graph with the mean flight departure delays by airline from Portland for 2014.

```
mean_delay_by_carrier <- flights %>%
  group_by(carrier) %>%
  summarize(mean_dep_delay = mean(dep_delay))
ggplot(mean_delay_by_carrier, aes(x = carrier, y = mean_dep_delay)) +
  geom_bar(position = "identity", stat = "identity", fill = "red")
```

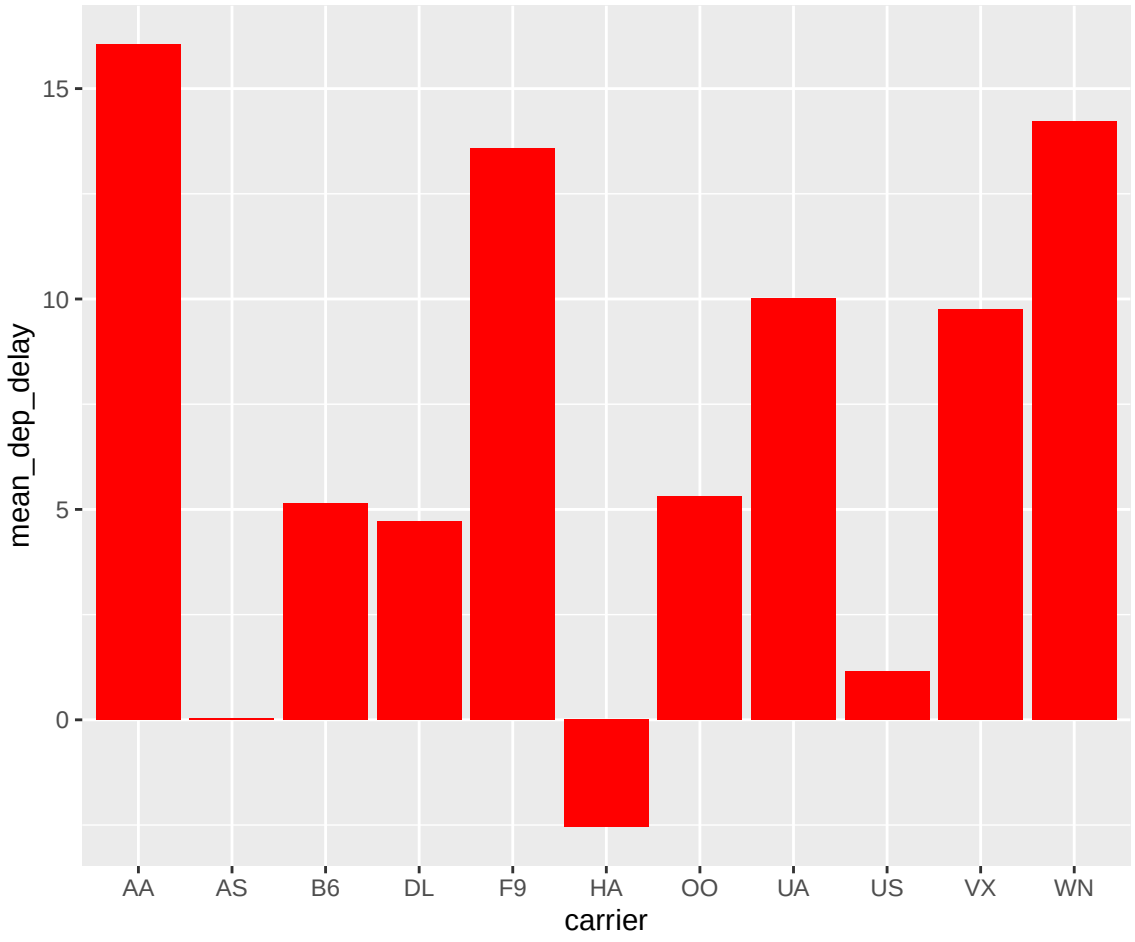

Figure 3.2: Mean Delays by Airline

Here is a reference to this image: Figure 3.2.

A table linking these carrier codes to airline names is available at `https://github.com/ismayc/pnwflights14/blob/master/data/airlines.csv`.

Next, we will explore the use of the `out.extra` chunk option, which can be used to shrink or expand an image loaded from a file by specifying `"scale= "`. Here we use the mathematical graph stored in the "subdivision.pdf" file.

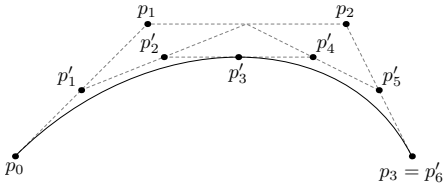

Figure 3.3: Subdiv. graph

Here is a reference to this image: Figure 3.3. Note that `echo=FALSE` is specified so that the **R** code is hidden in the document.

More Figure Stuff

Lastly, we will explore how to rotate and enlarge figures using the `out.extra` chunk option. (Currently this only works in the PDF version of the book.)

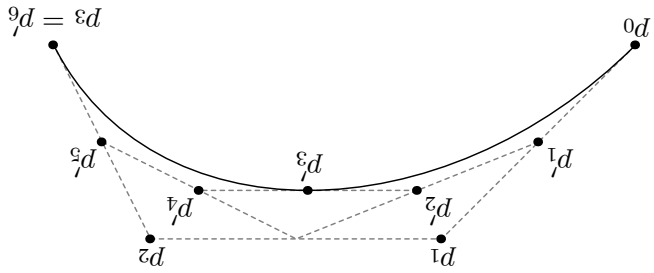

Figure 3.4: A Larger Figure, Flipped Upside Down

As another example, here is a reference: Figure 3.4.

3.2 Footnotes and Endnotes

You might want to footnote something. [1] The footnote will be in a smaller font and placed appropriately. Endnotes work in much the same way. More information can be found about both on the CUS site or feel free to reach out to `data@reed.edu`.

3.3 Bibliographies

Of course you will need to cite things, and you will probably accumulate an armful of sources. There are a variety of tools available for creating a bibliography

[1] footnote text

database (stored with the .bib extension). In addition to BibTeX suggested below, you may want to consider using the free and easy-to-use tool called Zotero. The Reed librarians have created Zotero documentation at `https://libguides.reed.edu/citation/zotero`. In addition, a tutorial is available from Middlebury College at `https://sites.middlebury.edu/zoteromiddlebury/`.

R Markdown uses *pandoc* (`https://pandoc.org/`) to build its bibliographies. One nice caveat of this is that you won't have to do a second compile to load in references as standard LaTeX requires. To cite references in your thesis (after creating your bibliography database), place the reference name inside square brackets and precede it by the "at" symbol. For example, here's a reference to a book about worrying: (Molina & Borkovec, 1994). This `Molina1994` entry appears in a file called `thesis.bib` in the `bib` folder. This bibliography database file was created by a program called BibTeX. You can call this file something else if you like (look at the YAML header in the main .Rmd file) and, by default, is to placed in the `bib` folder.

For more information about BibTeX and bibliographies, see our CUS site (`https://web.reed.edu/cis/help/latex/index.html`)[2]. There are three pages on this topic: *bibtex* (which talks about using BibTeX, at `https://web.reed.edu/cis/help/latex/bibtex.html`), *bibtexstyles* (about how to find and use the bibliography style that best suits your needs, at `https://web.reed.edu/cis/help/latex/bibtexstyles.html`) and *bibman* (which covers how to make and maintain a bibliography by hand, without BibTeX, at `https://web.reed.edu/cis/help/latex/bibman.html`). The last page will not be useful unless you have only a few sources.

If you look at the YAML header at the top of the main .Rmd file you can see that we can specify the style of the bibliography by referencing the appropriate csl file. You can download a variety of different style files at `https://www.zotero.org/styles`. Make sure to download the file into the csl folder.

Tips for Bibliographies

- Like with thesis formatting, the sooner you start compiling your bibliography for something as large as thesis, the better. Typing in source after source is mind-numbing enough; do you really want to do it for hours on end in late April? Think of it as procrastination.
- The cite key (a citation's label) needs to be unique from the other entries.

[2]Reed College (2007)

- When you have more than one author or editor, you need to separate each author's name by the word "and" e.g. `Author = {Noble, Sam and Youngberg, Jessica},`.
- Bibliographies made using BibTeX (whether manually or using a manager) accept LaTeX markup, so you can italicize and add symbols as necessary.
- To force capitalization in an article title or where all lowercase is generally used, bracket the capital letter in curly braces.
- You can add a Reed Thesis citation[3] option. The best way to do this is to use the phdthesis type of citation, and use the optional "type" field to enter "Reed thesis" or "Undergraduate thesis."

3.4 Anything else?

If you'd like to see examples of other things in this template, please contact the Data @ Reed team (email `data@reed.edu`) with your suggestions. We love to see people using *R Markdown* for their theses, and are happy to help.

[3]Noble (2002)

Conclusion

If we don't want Conclusion to have a chapter number next to it, we can add the `{-}` attribute.

More info

And here's some other random info: the first paragraph after a chapter title or section head *shouldn't be* indented, because indents are to tell the reader that you're starting a new paragraph. Since that's obvious after a chapter or section title, proper typesetting doesn't add an indent there.

Appendix A

The First Appendix

This first appendix includes all of the R chunks of code that were hidden throughout the document (using the `include = FALSE` chunk tag) to help with readibility and/or setup.

In the main Rmd file

```
# This chunk ensures that the thesisdown package is
# installed and loaded. This thesisdown package includes
# the template files for the thesis.
if (!require(remotes)) {
  if (params$`Install needed packages for {thesisdown}`) {
    install.packages("remotes", repos = "https://cran.rstudio.com")
  } else {
    stop(
      paste('You need to run install.packages("remotes")",
            "first in the Console.')
    )
  }
}
if (!require(thesisdown)) {
  if (params$`Install needed packages for {thesisdown}`) {
    remotes::install_github("ismayc/thesisdown")
  } else {
    stop(
      paste(
        "You need to run",
```

```
        'remotes::install_github("ismayc/thesisdown")',
        "first in the Console."
      )
    )
  }
}
library(thesisdown)
# Set how wide the R output will go
options(width = 70)
```

In Chapter 3:

```
# This chunk ensures that the thesisdown package is
# installed and loaded. This thesisdown package includes
# the template files for the thesis and also two functions
# used for labeling and referencing
if (!require(remotes)) {
  if (params$`Install needed packages for {thesisdown}`) {
    install.packages("remotes", repos = "https://cran.rstudio.com")
  } else {
    stop(
      paste(
        'You need to run install.packages("remotes")',
        "first in the Console."
      )
    )
  }
}
if (!require(dplyr)) {
  if (params$`Install needed packages for {thesisdown}`) {
    install.packages("dplyr", repos = "https://cran.rstudio.com")
  } else {
    stop(
      paste(
        'You need to run install.packages("dplyr")',
        "first in the Console."
```

```
      )
    )
  }
}
if (!require(ggplot2)) {
  if (params$`Install needed packages for {thesisdown}`) {
    install.packages("ggplot2", repos = "https://cran.rstudio.com")
  } else {
    stop(
      paste(
        'You need to run install.packages("ggplot2")',
        "first in the Console."
      )
    )
  }
}
if (!require(bookdown)) {
  if (params$`Install needed packages for {thesisdown}`) {
    install.packages("bookdown", repos = "https://cran.rstudio.com")
  } else {
    stop(
      paste(
        'You need to run install.packages("bookdown")',
        "first in the Console."
      )
    )
  }
}
if (!require(thesisdown)) {
  if (params$`Install needed packages for {thesisdown}`) {
    remotes::install_github("ismayc/thesisdown")
  } else {
    stop(
      paste(
        "You need to run",
        'remotes::install_github("ismayc/thesisdown")',
```

```
        "first in the Console."
      )
    )
  }
}
library(thesisdown)
library(dplyr)
library(ggplot2)
library(knitr)
flights <- read.csv("data/flights.csv", stringsAsFactors = FALSE)
```

Appendix B

The Second Appendix, for Fun

References

Huffman, D. A. (1952). A method for the construction of minimum-redundancy codes. Pr

Sayood, K. (2017). Introduction to data compression. Academic Press.

Witten, I. H., Neal, R. M., & Cleary, J. G. (1987). Arithmetic coding for data compr

Sayood, K. (2017). Introduction to data compression. Academic Press.

Pratas, Diogo & Pinho, Armando. (2018). A DNA Sequence Corpus for Compression Benchm

Angel, E. (2000). *Interactive computer graphics : A top-down approach with OpenGL*. Boston, MA: Addison Wesley Longman.

Angel, E. (2001a). *Batch-file computer graphics : A bottom-up approach with QuickTime*. Boston, MA: Wesley Addison Longman.

Angel, E. (2001b). *Test second book by angel*. Boston, MA: Wesley Addison Longman.

Grumbach, S., & Tahi, F. (1994). A New Challenge for Compression Algorithms: Genetic Sequences. *Information Processing and Management*, *30*. Retrieved from https://hal.inria.fr/inria-00180949

Molina, S. T., & Borkovec, T. D. (1994). The Penn State worry questionnaire: Psychometric properties and associated characteristics. In G. C. L. Davey & F. Tallis (Eds.), *Worrying: Perspectives on theory, assessment and treatment* (pp. 265–283). New York: Wiley.

Noble, S. G. (2002). *Turning images into simple line-art* (Undergraduate thesis). Reed College.

Pratas, D., & Pinho, A. (2018). A DNA sequence corpus for compression benchmark. In (pp. 208–215). http://doi.org/10.1007/978-3-319-98702-6_25

Reed College. (2007). LaTeX your document. Retrieved from https://web.reed.edu/cis/help/LaTeX/index.html