LIN/COS280
Final Project
Daniel Douglas

Text Compression Based on Context Free Grammars

Introduction

For my final project, I wanted to design and text the feasibility of a text compression system that could potentially deliver a fantastic compression rate, at the cost of lossiness - where decompression returns a text that is similar, but not necessarily identical, to the one compressed. To test the process, I performed a series of trials on a sample text with different compression rates to determine the tradeoff between compression rate, resultant lossiness, and runtime. While the system, even on a toy example, does not deliver very impressive results in these tests, several future optimizations could end up improving its usefulness to the point where it might be the algorithm of choice in a few niche applications.

Compression Process

Let there be a hash function that maps each word or punctuation mark in the lexicon to an integer in the range $[0, 2^n - 1]$. This hash function should be constructed to make collisions (pairs of words that map to the same value) as rare as possible, especially collisions between words that are of the same part of speech or grammatical function. For the sake of computational efficiency in the decompression process below, one value should also be set aside only for periods or other terminal punctuation of sentences. By feeding sentences into this function one word at a time, one obtains a series of numbers, each expressible using only n bits of memory. Most of the examples in this paper use an n of either 5 or 6, so each word is stored in 5 or 6 bits. For comparison, each character in an English word is typically stored in ASCII encoding with 8 bits per *character*, or about 49 bits per word on average.[1]

Decompression Process

Recovery of the original text from this series of numbers is, unfortunately, more complicated. For each sentence (determined by splitting the file along instances of the hash value denoting terminal punctuation), determine all possible sequences of words that could generate the given series of hashed values. For each such sequence, use a context free grammar that is capable of generating every sentence in the language to see whether the given sequence of words is a valid sentence in the language. Most sequences will be eliminated by this process, usually returning only one or two potential interpretations during testing. The decompresser can therefore return, for each sentence in the original text, either that sentence, or a small set of sentences that contains the original sentence. At this point, the original message is straightforward to recover by human inspection, since human knowledge can apply linguistic properties that are not covered by context free grammars alone.

Toy Example

For testing whether the system is feasible, I used the text of *Green Eggs and Ham*, the children's book by Dr. Seuss. The most desirable property of the text is that it uses a lexicon of only 52 words, making tagging the words by part of speech for the CFG easy to do by hand. While the complexity of the sentences in the text was greater than expected, compared to the rest of English it is a much simpler syntax subset. In the attached tarball are the files used in the test. I obtained a copy of the text from online (greeneggs), did simple processing to eliminate capitalization and separate punctuation from words (greeneggs-modified), and created a context free grammar that covers the whole text (greeneggs.cfg). Obviously, the most difficult part was trying to determine the simplest grammar that covers all 140 sentences in the text, complicated by the fact that Seuss used sentence fragments and awkward sentences often in his poetry. Before compression, the modified *Green Eggs* was **3479 bytes**.

Testing

Rather than try to design an effective hash function from scratch to minimize collisions, I generated random hash functions in multiple tests, hoping one among many would turn out to be particularly suitable. All testing data below must therefore be taken with the caveat that a carefully constructed hash function could conceivably be much more effective. All testing data was taken on a Dell Inspiron 1520 laptop running Ubuntu 8.10, with a 2.4 GHz processor.

If a 6-bit hash function (mapping words to 0 - 63) is used, each word can have a value to itself, and there is no ambiguity in reconstructing the original message. Compression occurs in constant time, and decompression is nearly constant - it takes approximately 0.02 seconds per sentence to confirm that the possibility is in fact correct English and that a compression error did not occur. At 6 bits per word, *Green Eggs* is compressed to **745 bytes**, or a compression factor of 4.67.

A 4-bit function (mapping words to 0 - 15) could conceivably get a much better compression rate - 7.01, or **497 bytes** - but because 3 to 4 words map to every value, the number of possibilities per sentence before checking with the grammar is exponential to the sentence length. Many sentences in the sample text have 10 words or more, and 3^{10} to 4^{10} possibilities per sentence, with 0.02 seconds to check each possibility, demonstrate the total infeasibility of a 4-bit function, even on toy examples. I attempted to run a single 4-bit test, and after 3 hours it had made no noticeable progress. Because the test never completed, it is not clear how much ambiguity the decompression process' results would have, but it would likely be phenomenal, rendering the results useless anyway.

The best compromise seems to be a 5-bit function (mapping words to 0 - 31), where most values either have a unique word associated with them or share with only one other word. The compressed text took up **621 bytes**, a compression factor of 5.21. The average decompression time at this level was 9.26 seconds per sentence, or 21 minutes for the whole story, with the best of 20 tests taking under 6 minutes, and the worst taking over 49 minutes. This seems pretty awful, but if all the end user desires is a good compression rate, it is at least not computationally infeasible. There is also not an excessive amount of ambiguity, measured in the number of possibilities returned by the decompresser per sentence. The best hash function had an ambiguity of 1.66 per sentence, the worst had 13.3 per sentence (this highlights the importance of a well-balanced function!), and the average was 3.8 per sentence. If each sentence in the text is either certain or a choice between one of two sentences, reconstruction of the text by a human is straightforward.

For comparison with the above values, the Gzip algorithm compresses the text to **801 bytes**. This is partially an unfair assessment because Gzip is designed to work on all types of data, not just text, and does not have the advantage of being tailored to this sample text in particular. However, it gives a good idea of how grammar-based compression used in specialized systems could be superior to traditional compression algorithms.

Least ambiguous mapping:
```
{'and': 16, 'sam': 14, 'be': 29, 'house': 11,
'am': 21, 'say': 18, 'are': 15, 'in': 8, 'sam-i-am': 7, 'mouse': 19,
'boat': 27, 'if': 26, '!': 0, 'will': 1, 'ham': 26, 'would': 13,
'there': 29, 'fox': 12, ',': 8, '.': 0, 'so': 21, 'you': 7, 'goat':
7, '?': 0, 'box': 27, 'them': 25, 'good': 12, 'do': 26, 'that': 1,
'may': 16, 'eggs': 30, 'rain': 30, 'dark': 13, 'me': 9, 'train': 22,
'let': 2, 'here': 5, 'they': 4, 'not': 11, 'with': 6, 'eat': 14,
'thank': 11, 'a': 13, 'on': 20, 'like': 1, 'i': 28, 'car': 15,
'could': 27, 'tree': 25, 'see': 15, 'try': 7, 'anywhere': 24,
'green': 6, 'the': 13, 'or': 3}
```
Most ambiguous mapping:
```
{'and': 26, 'sam': 25, 'be': 24, 'house': 11,
'am': 17, 'say': 3, 'are': 28, 'in': 2, 'sam-i-am': 3, 'mouse': 17,
'boat': 13, 'if': 18, '!': 0, 'will': 14, 'ham': 11, 'would': 7,
'there': 2, 'fox': 6, ',': 17, '.': 0, 'so': 9, 'you': 13, 'goat':
13, '?': 0, 'box': 22, 'them': 3, 'good': 1, 'do': 14, 'that': 29,
'may': 31, 'eggs': 24, 'rain': 9, 'dark': 17, 'me': 5, 'train': 22,
'let': 12, 'here': 30, 'they': 30, 'not': 9, 'with': 28, 'eat': 5,
'thank': 27, 'a': 19, 'on': 12, 'like': 23, 'i': 26, 'car': 27,
'could': 18, 'tree': 19, 'see': 26, 'try': 13, 'anywhere': 23,
'green': 3, 'the': 19, 'or': 20}
```

Applications

I can see this system being useful in a situation where messages of an already known and agreed upon structure and vocabulary need to be sent between communicators in a simple and brief way, such as semaphore or morse code communication between military units, where complex orders taking advantage of the grammar need to be sent, while minimizing the danger of transmission error and reducing the legibility of the sent message as much as possible.

It is not clear how increasing the complexity of the grammar or the size of the lexicon will affect compression rate, decompression time, and ambiguity. A lexicon of 1000 words or more will almost certainly need a 9-bit function (range of 512) to run in comparable time to the examples above, but since the average word length in such a lexicon would be much greater than in Green Eggs and Ham, the compression rate would likely be about the same. Ambiguity is probably a function of the sentence length and the ratio of lexicon size to hash range, and so a larger example, if scaled correctly, might not fare any worse than the toy example. A more complex grammar, however, will certainly increase ambiguity, as more sentences will be accepted as valid, even if to a native speaker they seem "off" or nonsensical.

Future Improvements

Use of Markov models of the language or measuring the co-location of words in the corpus could aid in the automatic determination of one sentence possibility as more likely than another. A system that determines "meaning" behind sentences could find sentences who do not fit in with neighboring sentences. Carefully constructing the hash function rather than randomly generating it would almost certainly result in much less ambiguous results for any text.

Citations

[1] http://blogamundo.net/lab/wordlengths/