Project 3 - Group 8

Jeffrey Tao, Martin Ristovski, Vikram Rajan

November 3, 2022

Contents

1 The Problem

¡TODO¿: COPY AND EXTEND PROBLEM DESCRIPTION FROM COURSE WEBSITE.

2 Initial Implementation

The first idea that came to mind was to take our message of characters and encode it into a bitstring. To do this, we got a frequency distribution of the letters of the alphabet in English, and used that to generate a Huffman coding. Now that we had a mapping from characters to bits, we needed a mapping from bits to cards. This was done by ¡TODO¿: EXPLAIN BITS-TO-CARDS ENCODING.

¡TODO¿: EXPLAIN DECODING.

To make sure that our solution had few false positives, we implemented a checksum generator, which took as input the bitstring of the compressed message, and then we appended the 8-bit output of that checksum to our bitstring.

This allowed us to encode messages that consist of lowercase characters in the English alphabet and reliably decode them for high numbers of shuffles. A plot of our performance in this message domain is given below.

¡TODO¿: INSERT PLOT.

3 Improvements

While our soltuion performed well within this limited message domain, to do well on the tournament there were a number of extensions that we'd need to make.

3.1 Checksum

One of the easier adjustments was made in order to reduce the rate of checksum collision to further avoid false positives, To accomplish this, we increased the length of our checksum from 8 to 10 bits, which gave us a 4x reduction in the false positive rate.

3.2 Multiple Domains

Then, we added support for multiple message domains. Implementing multiple domains allowed us to use a more efficient encoding scheme for each domain, which in turn decreased the length of our message and made it more resistant to shuffles. However, we also had to dedicate 3 bits of our bitstring to let the decoder know which scheme was used to encode the message. In our testing, this was a worthwhile tradeoff. We added a number of character-based encoding schemes, which also used Huffman coding but with a different set of charcters and thus a different frequency distribution. In particular, we added a domain containing only numbers, a domain containing charcters typically found in English writing (uppercase and lowercase letters, numbers, space, and a few punctuation marks), and a domain containing all printable ASCII characters, which we used as a fallback of sorts, since it is able to encode any message at the cost of efficiency.

3.3 Word-based Encoding Schemes

We further extended our multiple domain approach by implementing word-based encoding schemes. These would map words to integers that referrenced each word's position in a dictionary, allowing the decoder to simply look up each word. This meant that if all words in a message are present in our dictionary, we could encode much longer messages than if we were to use a character-based encoding scheme. The dictionaries we initially included were one of a large number of words in the English language, one of names of people, and one of names of places.

¡TODO¿: INSERT PLOTS.

4 Eight Tournament Domains

On November 1, 2022, the class collectively decided to limit the types of messages to ones belonging to 8 domains, one proposed by each group, and each defined by a generator function. Our initial multi-domain apporach allowed us to easily adapt to this part of the tournament specifications, as we were able to build specific encoding schemes for each domain.

The domain we proposed is that of messages of type ¡verb¿ ¡name¿ ¡place¿, as we thought it would be a reasonable proposition that could realistically be used by a spy agency.

5 Tournament Performance

6 Limitations