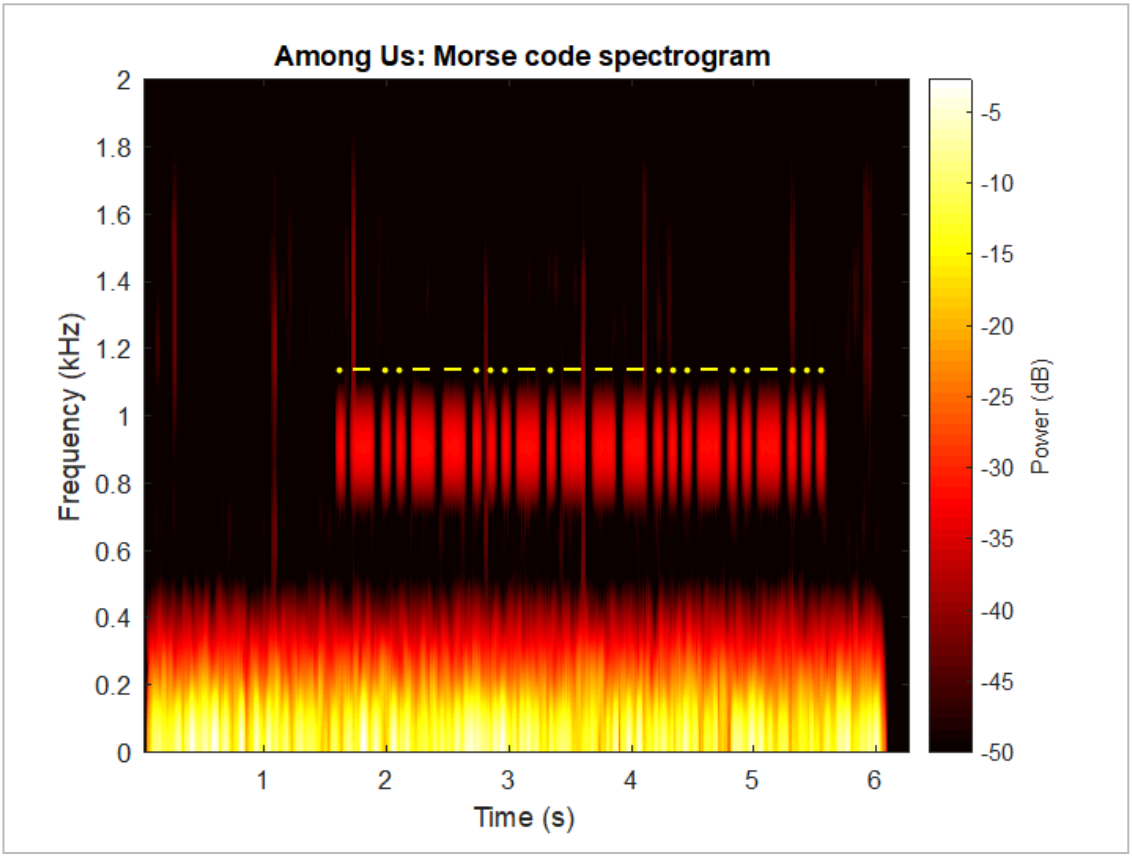
Possibly Wrong
On science, mathematics, and computing

Among Us: Morse code puzzle

Posted on **November 21, 2020**

In the online game *Among Us*, players who visit the Comms room hear a fuzzy audio recording of a series of high-pitched beeps that sound like Morse code. I first heard the recording here, but this more recent video also plays it at around 5:00, followed by a good explanation of the problem with trying to decipher the code.

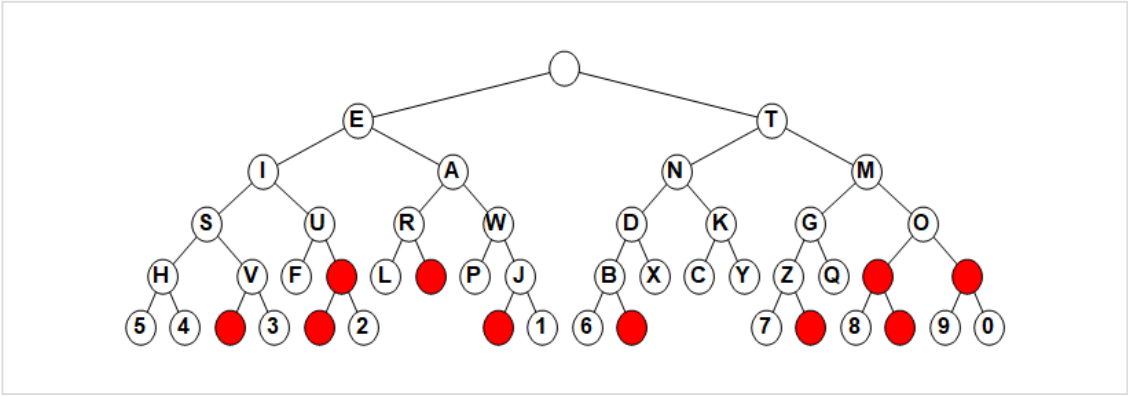
The following figure shows a spectrogram of the audio clip, with time on the *x*-axis, and each vertical slice showing the Fourier transform of a short (roughly 50 ms) sliding window of the signal centered at the corresponding time. We can clearly see the “dots” and “dashes” at around 1 kHz, with the corresponding translation overlaid in yellow.



— Spectrogram of the Comms room audio, with the translated Morse code also indicated in yellow.

Now that we have the Morse code extracted from the audio (which, for reference if you want to copy-paste and play with this problem, is "`.-.....-.....-...-..-...`"), we just need to decode it, right? The problem is that the dots and dashes are all uniformly spaced, without the required longer gaps between *letters*, let alone the still longer gaps that would be expected between *words*. Without knowing the intended locations of those gaps, the code is ambiguous: for example, the first dot could indicate the letter E, or the first dot *and dash together* could indicate an A, etc.

That turns out to be a big problem. The following figure shows the decoding trie for Morse code letters and digits; starting at the root, move to the left child vertex for each dot, or to the right child vertex for each dash. A red vertex indicates either an invalid code or other punctuation.



— Morse code represented as a decoding trie; each left and right edge indicates a dot or dash, respectively.

If we ignore the digits in the lowest level of the trie, we see that not only are Morse code letters ambiguous (i.e., not prefix-free), they are nearly "*maximally* ambiguous," in the sense that the trie of letters is nearly complete. That is, for almost any prefix of four dots and dashes we may encounter, the gap indicating the end of the first letter could be after *any* of those first four symbols.

This would make a nice programming exercise for students, to show that this particular sequence of 24 symbols may be decoded into a sequence of letters in exactly 3,457,592 possible ways. Granted, most of these decodings result in nonsense, like AEABKGEAEAEEE. But a more interesting and challenging problem is to efficiently search for *reasonable* decodings, that is, messages consisting of actual (English?) words, perhaps additionally constrained by grammatical connections *between* words.

Of course, it's also possible– probable?– that this audio clip is simply made up, a random sequence of dots and dashes meant to *sound* like "real" Morse code. And even if it's not, *we might not be able to tell the difference*. Which is the interesting question that motivated this post: if we generate a completely random, and thus *intentionally* unintelligible, sequence of 24 dots and dashes, what is the probability that it still yields a "reasonable" possible decoding, for sufficiently large values of "reasonable"?

This entry was posted in **Uncategorized** by **possiblywrong**. Bookmark the **permalink [https://possiblywrong.wordpress.com/2020/11/21/among-us-morse-code-puzzle/]** .

6 THOUGHTS ON “AMONG US: MORSE CODE PUZZLE”

Chris Wellons
on **November 21, 2020 at 2:13 pm** said:

efficiently search for reasonable *decodings*

In case you forgot, this was r/DailyProgrammer Challenge #380 back in August 2019. 🙂

There’s an old trick to use a general compressor (DEFLATE, etc.) to do language detection. Start with a corpus for each candidate language you want to detect. Append the unidentified text to each and compress the whole. The one that compresses the smallest is the likely language of the identified text since it’s the one most similar. For large enough Morse code sequences I wonder if this technique could be used to automatically identify reasonable decoding candidates.

possiblywrong
on **November 21, 2020 at 5:21 pm** said:

I remember seeing that challenge– this is essentially the same problem as the hard (Friday) version.

Some good and bad follow-up news: the bad news is that I learned that these beeps are typical of “channel markers,” or Morse code station identifiers commonly heard on police scanners… which is exactly the context of the surrounding audio as described by the guy in the originally linked video. See here for an example.

The (potentially) good news is that those channel markers seem to be “real” Morse code, with the required gaps between letters/digits. The above clip, for example, is unambiguously TAB269.

I say “good” news, because the code in the game, lacking the gaps needed to actually interpret it, seems unlikely to have been pulled from real/stock scanner audio. That is, my guess is that the developers “made it up”… and in the process messed it up by leaving out the gaps. That suggests– at least to my overly hopeful puzzle solving mindset :)– that they *intended* to put something of their own in that audio.

Chris Wellons
on **November 22, 2020 at 10:10 pm** said:

Your trie illustration and the comment that it's nearly maximally ambiguous (i.e. the trie has few "holes" when represented using an array) gave me an idea for a decoder automaton. The trie is encoded here as a byte-array:

https://gist.github.com/skeeto/f9e198b913b228f3fd773a0c4e266579

Plugging that into a little, recursive function to count all the decodings exactly matches your count (unsurprisingly).

>
>
> **possiblywrong**
> on **November 23, 2020 at 11:20 pm** said:
>
> Nice! We could take this further and look for actual words, with a larger trie constructed from a dictionary, with each path from the root corresponding to a word instead of just a letter (at least those paths that end with a terminal "this ends a complete valid word" vertex; there would be more internal "holes" in the trie). This would be much more efficient than my lazy Mathematica approach of recursive regex pattern matching. The results don't suggest anything earth-shattering: RATSTAMEFREE, stuff like that (although interpreting the code *backward* seems to yield more reasonable options, VIDEOLATER, VIDEOREWIN(d?), IRENEJRITTER (a developer easter egg?), etc.).

Filip Morawiec
on **January 31, 2021 at 5:29 am** said:

did you try to look only at the solutions, which consist of most commonly used english words in them? like 'the' for instance.

>
>
> **possiblywrong**
> on **January 31, 2021 at 1:05 pm** said:
>
> Sort of– I used a complete word list, but I sorted results by frequency of occurrence in the Google Ngrams dataset, so that more "reasonable" decodings would appear earlier in the list.

This site uses Akismet to reduce spam. Learn how your comment data is processed.