Wordle and Entropy (Preliminary)

Tom Davis
tomrdavis@earthlink.net
http://www.geometer.org/mathcircles
October 9, 2025

Abstract

In this article we will consider how to use information theory and entropy to find optimal strategies for playing the game Wordle that appears daily in the New York Times.

1 What is Wordle?

Wordle is a game where the goal is to discover a secret 5-letter English word by making a series of 5-letter word guesses. After each guess the user is informed which letters in the guess are exactly correct: the letter in the guess is the same and is in the same position as the secret word. Those are marked green. Then letters that are the same in the guess as in the secret word but are in the wrong position are marked yellow, and finally, letters in the guess that do not appear in the hidden word are marked black.

If the guess is completely correct (five greens) the game ends, but if not, the user makes a new guess. Up to 6 guesses can be made, but the object is to find the hidden word as quickly as possible.

Here is a sample game where the hidden word is BAGEL:

The first guess was TREND, none of whose letters match exactly with those in BAGEL, but it does have an E but the E is not in position 4, so it is marked in yellow. The other four letters, T, R, N, and D, do not appear in BAGEL, so they are marked in black.

The next guess, SLIME tries four new letters and includes the E, but in a different position. Again, no letters in the guess match exactly with those in BAGEL, but L is somewhere in BAGEL.

After the BELLY guess we know where B goes and since the E and L are still in the wrong positions, we can infer that the hidden word ends in EL, so the trial BOWEL makes sense. Finally BAGEL solves the puzzle.

In the actual New York Times version of the game the hidden words are not chosen at random, and once a word is used it is not used again as a hidden word, but in this article we will assume that there is a fixed set of words that are equally-likely to be the hidden word. We will also assume that any guess is legal, even if it it not a possible answer, given the results of previous test words. Given this problem statement, what is the best strategy to minimize the number of guesses needed to solve the problem, on average?

There were 2315 5-letter words in the original Wordle app before the New York Times purchased it, but some editing of the word list has occurred. Before a game starts, there are 2315 equally-likely possibilities for the hidden word. After each guess/response many of the possibilities are eliminated, so a reasonable strategy might be to try to reduce the number of possibilities as much as possible with each guess.

It will turn out that using the concept of entropy makes it easy (for a computer, at least) to come up with an optimal strategy. The next sections may seem unrelated to Wordle, but have patience, the following digression will make sense.

2 Introduction to Entropy

Imagine that Alice is sending a series of messages to Bob to convey some sort of information to him. The information might be the current temperatures, the current prices of a stock, the positions of the enemy, the results of flipping a coin or upon which number the ball stopped on a roulette wheel, et cetera. For now we will assume that all of the transmissions are made with a series of bits (0's and 1's)[1].

Let's begin with the assumption that Alice transmits a fixed set finite set of k possible messages: $M_1, M_2, M_3, \ldots, M_k$, and that the relative probability of sending message M_i is p_i. Since there are only k possible messages, the probabilities have to add to 1:

$$\sum_{i=1}^{k} p_i = p_1 + p_2 + \cdots + p_k = 1.$$

Let's start with a simple example where $k = 4$ and each of the four messages is equally likely; in other words, $p_1 = p_2 = p_3 = p_4 = 1/4$. We will also assume for now that messages are independent: just because a message was M_i gives no information about what the next message will be or what the previous message was.

A reasonable method for Alice to encode this information would be to assign to each of the four possibilities a two bit code, so a transmission of 00 indicates message M_1 and transmissions of 01, 10, and 11 would indicate, respectively, messages M_2, M_3, and M_4.

[1] If the transmissions consist of more than two characters, only a tiny change must be made in the formulas we will obtain for entropy and information

One way to measure the amount of information transferred is to count the number of yes/no questions that must be answered to find out which message was sent. In the case above, Bob needs to ask exactly two questions for every event; namely, "What was the first bit?" and "What was the second bit?"

Imagine that Alice has to pay one penny for every bit she sends. In this example, every message will cost two pennies, and so the average cost of sending a message is two cents, since every message has exactly two bits.

Now suppose that the messages are not equally likely. In fact, let's assume that $p_1 = 1/2$, $p_2 = 1/4$, and $p_3 = p_4 = 1/8$. Alice can still use the same 2-bit encoding for each message, and will still pay, on average, two cents per message, but she can do better!

Suppose Alice uses the following encodings for $M_1, \ldots, M_4$, respectively: 0, 10, 110, 111.

First note that there is no ambiguity in this. When Bob is looking for the next message, if he sees a 0, that's the whole message (M_1) and he starts immediately to listen for the next message. If the first bit is a 1, he waits for the second bit. If that one is 0, he is again done (the message is M_2) and he starts to wait for the next message. If he sees 11 as the first two bits, the next bit tells him which message it is.

Thus even though the messages have different lengths, you can tell which message has arrived, and there is no need to put "spaces" between the messages. (If there were markers between the messages, we'd have to transmit the markers (with zeroes and ones) and this would be even more costly.

But over a long time, how much will Alice have to pay for each message, on average, with this scheme? Half the time she pays one cent, a quarter of the time, two cents, and one quarter of the time, three cents (or two-eighths of the time, three cents). Thus Alice's average expenditure per message is:

$$\frac{1}{2} \cdot 1 + \frac{1}{4} \cdot 2 + \frac{1}{8} \cdot 3 + \frac{1}{8} \cdot 3 = 1.75.$$

In other words, she saves about a quarter of a cent, on average, for each message she transmits!

Can she do better? What about in the first case? These are the sorts of things that a calculation of entropy can tell us.

3 Information

We spoke of messages above, but in general it is equivalent to consider outcomes of experiments, et cetera, so when the term "message" is used below, it can have a far more general meaning.

A key idea is this: different messages convey different amounts of information. In fact, the less likely a message is, the more information it conveys. This isn't hard to see: if something is certain to happen, then learning that it happened tells you nothing since

you already knew it would happen. If something is quite likely to happen, then it is not a surprise when it does and you didn't learn much.

So in some sense, the amount of information Bob obtains when he gets a message from Alice is small when the message is likely and large when the message is rare, so a possible way to define the information content of message M_i might be something like $1/p_i$ which get larger as p_i gets smaller and vice-versa.

This is close, but not quite right. Here's why. Suppose Bob receives two messages with probabilities p and q (and that all the messages are independent in that none depend on the previous messages). Since the messages are independent events, the probability of getting the two messages in that sequence is pq (the product of the probabilities) but he got a certain amount of information from the first result and then another amount from the second. Intuitively the total amount of information received from the two messages should be the sum of the individual amounts of information, so if there is a function $I(p)$ that means, "the amount of information received when a message of probability p arrives, then $I(p)$ should satisfy the following equation:

$$I(pq) = I(p) + I(q).$$

This looks exactly like a logarithm function, and in fact, that's what it is:

$$I(p) = \log(1/p).$$

Notice that:

$$I(pq) = \log(1/(pq)) = \log(1/p \cdot 1/q) = \log(1/p) + \log(1/q) = I(p) + I(q).$$

It also has the nice property that the information associated with an event that is certain to occur (in other words, $p = 1$) is zero, since $\log(1/1) = \log(1) = 0$.

We did not specify the base of the logarithm above, and in fact, logarithms of any base would behave in exactly the same way. For reasons that will become clear later, we will usually use the logarithm base-2, so:

$$I(p) = \log_2(1/p) = -\log_2(p).$$

Here is another way to convince yourself that the information should be related to the logarithm of the number of different messages instead of to the number of different messages. Suppose you have a transmission line that sends one bit (0 or 1) every second. A certain amount of information will be transmitted. Now suppose we have four copies of the same transmission line running in parallel. It's reasonable to assume that you are now getting information four times as fast. But if we look at what arrives every second, we could receive up to $16 = 2^4$ different patterns of bits from the four lines. But we are only receiving $4 = \log_2 16$ times the amount of information.

Now let's ask a *very* interesting question. If an experiment or situation plays out, what is the average expected amount of information we will receive as a result? For each of the possible outcomes, we need to multiply the probability of that outcome by the amount of information that becomes available as a result, and that will be the

expected amount of information received. For our experiment with outcomes having probabilities $p_1, p_2, \ldots, p_k$, that value will be:

$$-(p_1 \log_2 p_1 + p_2 \log_2 p_2 + \cdots p_n \log_2 p_k) = -\sum_{i=1}^{k} p_i \log_2 p_i.$$

3.1 Uncertainty and Entropy

In the Alice/Bob transmission example we have a situation where there are a fixed number of possible messages (outcomes), and the probability of each is known. Entropy is a way to measure the uncertainty of the situation. If you are certain of the message (outcome), we'd like this measure to be zero, and the more uncertain the outcome, the larger we'd like the entropy to be.

What we would like to have is some sort of a mathematical expression that indicates, in a reasonable way, the uncertainty of a situation. You can think of uncertainly as the opposite of certainty. In other words, if you are almost certain how an experiment will come out, you will have low uncertainty, and vice-versa: a situation where it is very hard to predict an outcome will have high uncertainty. We'll be looking for a measure where if you are absolutely certain of what's going to happen, the uncertainty will be zero, and as you look at situations whose outcomes are harder and harder to predict, you should have uncertainties that are larger and larger.

The types of messages M_i don't really matter as long as they are independent; all that matters are the probabilities of the M_i which we will, from now on, call p_i. Our goal will be to find a function $H(p_1, p_2, \ldots, p_k)$ that measures the uncertainty of that situation. For situations where there are a different number of messages than k we will need different entropy functions with different numbers of parameters, but since you can just count the parameters to figure out which version of the entropy function to use, we will use the letter H to indicate all of them.

Exercise: Try to think of some properties that such an uncertainty function H should have.

3.1.1 Properties of H

Here are some obvious properties (and perhaps some not-so-obvious ones) that an uncertainty-measuring function should have:

1. If there is no uncertainty; namely, if there is only one possible outcome, then the value of H should be 0. The uncertainty of any event should be non-negative, and if situation A is less-predictable than situation B, the uncertainty of A should be larger than that of B.

2. The order of the parameters should make no difference. That is, it shouldn't matter which outcome we name as the first, second, third, et cetera. Mathematically, it means:
$$H(p_1, p_2) = H(p_2, p_1)$$

for two possible outcomes, and

$$\begin{aligned} & H(p_1, p_2, p_3) = H(p_1, p_3, p_2) = H(p_2, p_1, p_3) \\ = \quad & H(p_2, p_3, p_1) = H(p_3, p_1, p_2) = H(p_3, p_2, p_1) \end{aligned}$$

for three outcomes, et cetera. Mathematical functions where the order of the parameters is unimportant are called *symmetric functions*. Another way of stating this idea is to say that $H(p_1, p_2, \ldots, p_k)$ is symmetric. We can mathematically state this for the general case as follows: Let $\pi(i)$ be a permutation (a rearrangement) of the set $\{1, 2, 3, \ldots, k\}$. Then for any such π we have:

$$H(p_1, p_2, p_3, \ldots, p_k) = H(p_{\pi(1)}, p_{\pi(2)}, p_{\pi(3)}, \ldots, p_{\pi(k)}).$$

3. When there are k possible outcomes, the situation that is the most uncertain is when all the p_i are equal. If it's more likely that one outcome will occur than another, then there is less uncertainty. If two equally-able football teams are playing, you are quite uncertain about the outcome, but if the San Francisco 49ers are playing against your high-school football team, there is very little uncertainly about what will happen.

   We can state this mathematically as follows: For any set of $p_i \geq 0$ such that $p_1 + p_2 + \cdots + p_k = 1$ then:

   $$H(1/k, 1/k, 1/k, \ldots, 1/k) \geq H(p_1, p_2, p_3, \ldots, p_k).$$

4. Similarly, if you have more equally-likely outcomes, the uncertainty increases:

   $$H(1/k, 1/k, \ldots, 1/k) \leq H(1/(k+1), 1/(k+1), \ldots, 1/(k+1)),$$

   (where the first H has k parameters and the second, $k + 1$).

5. Adding an outcome to the list of possible outcomes which has zero probability of occurring will not change the value of H. In other words, it will have no effect on the uncertainty. Mathematically:

   $$H(p_1, p_2, \ldots, p_k) = H(p_1, p_2, \ldots, p_k, 0),$$

   (where the first H has k parameters and the second, $k + 1$).

6. If there are two completely independent situations, the first with outcomes having probabilities $p_1, p_2, \ldots, p_k$ and the other having probabilities $q_1, q_2, \ldots, q_m$ then when both situations have occurred, there are km possible outcomes, and since the situations are independent, the probabilities of these km outcomes will be:

   $$p_1q_1, p_1q_2, \ldots p_1q_m, p_2q_1, p_2q_2, \ldots, p_2q_m, \ldots, p_kq_1, \ldots p_kq_m.$$

   The total uncertainty should be the sum of the two uncertainties, so:

   $$H(p_1q_1, p_1q_2, \ldots, p_kq_m) = H(p_1, p_2, \ldots, p_k) + H(q_1, q_2, \ldots, q_m).$$

7. H should be continuous in its variables. In other words, if one set of probabilities is very close to another set, then the values of H for those two sets should also be close. (The standard calculus definition of continuity applies, if you know what that is.)

8. Here is a final condition that's a little more complicated. First we'll state the result mathematically and then try to describe it.

   Suppose $p_i \geq 0$, $q_i \geq 0$, $p = p_1 + \ldots + p_n$, $q = q_1 + \ldots q_m$, and $p + q = 1$. Then:

$$\begin{aligned} & H(p_1, \ldots, p_n, q_1, \ldots, q_m) \\ = \quad & H(p, q) + H(p_1/p, p_2/p, \ldots p_n/p) + H(q_1/q, q_2/q, \ldots, q_m/q). \end{aligned}$$

   In English, this basically says that if a situation with $n + m$ outcomes is divided into two classes, one having total probability p and the other, total probability q, then the total uncertainty is the sum of three things: the uncertainty of whether the first or the second class occurs, the uncertainly of the result if we were restricted to only the first class and the uncertainly of the result if we were restricted to only the second class. This is a little technical, so don't worry if it is not at first obvious.

3.1.2 A Formula for H

We will not prove this, but it turns out that there is basically only one type of function that satisfies all of the conditions above, and the total uncertainty, called the "entropy," has the following form:

$$H(p_1, p_2, \ldots, p_k) = -(p_1 \log p_1 + p_2 \log p_2 + p_n \log p_k) = -\sum_{i=1}^{k} p_i \log p_i.$$

The logarithms above can be taken to any base, but in information theory, it is customary (and we'll see why later) to use logarithms base-2. If we used other bases, the resulting values of H would simply be multiples of each other. The negative sign makes sense because all probabilities are at most one, so the logarithms are all zero or negative[2].

Notice that this formula for H is identical to the formula toward the end of Section 3! Thus another way to think of entropy is as the average amount of information received per outcome/message/event.

In the expression above, we will also assume that if some probability p_i is zero, then the term $p_i \log p_i$ is also zero in spite of the fact that the logarithm of 0 is undefined.

It is worth spending a bit of time checking so see that the conditions for an uncertainty measure are satisfied by that formula. Some of the checks, for example, the fact that the

[2]It turns out that if we use base-2 the entropy will tell us the optimal average number of bits required to send an average message. Had we used base-3, it would tell us the average number of symbols in a three-symbol encoding that would be required, et cetera

entropy is maximized when all the probabilities are equal, requires a bit of calculus, but you can check with some actual numbers to convince yourself, even without calculus, that it is the case. For example:

$$\begin{aligned} H(.5,.5) &= -(.5\log_2(.5) + .5\log_2(.5)) = 1.000000 \\ H(.51,.49) &= -(.51\log_2(.51) + .49\log_2(.49)) = 0.999711 \end{aligned}$$

3.2 Example: Twenty Questions

Consider the game of "Twenty Questions" where your opponent thinks of an animal and you can ask 20 yes/no questions to try to figure out what animal it is. From the discussion above, we know that with some perfect set of questions, there are $2^{20} = 1048576$ different sequences of 20 yes-no answers, so in principle, the guesser could distinguish among 1048576 different animals. If you want to win as often as possible, the best strategy with each question is to split the number of possibilities as close to in half as possible; in other words, you'd like to maximize the amount of information each answer gives you, no matter what it is.

To obtain the most information out of your first question, you would like to construct one whose answer is as close as possible to having equal probabilities of getting a "yes" or "no" answer. To make this obvious, the worst possible sort of initial question might be, "Is it a lion?" If your opponent knows 10000 different animals and selects one at random, then your probability of getting a "yes" is 0.0001 and is 0.9999 of getting a "no", yielding a very tiny gain in information, on average.

Or perhaps even more obvious: I have picked a number from 1 to 1048576 and you have 20 yes-no questions to find it. If each question splits the number of possibilities exactly in half, you'll be certain to get it. Your first question might be, "Is it smaller than or equal to 524288?" Each guess will split the remaining possible group of numbers exactly in half and at the end you'll be left with exactly one of them. If your first question is, "Is it 417?" and it isn't, even with the best possible play after that you'll only win about half the time.

Here's an even easier way to play 20 questions to find a number between (let's adjust the range by 1) 0 and 1048755: Every number in the range can be represented by a 20-bit binary number (perhaps with leading zeroes). The first question is: "Is the first binary digit equal to 1?" Then ask about the second, third, ..., and obviously, at the end, you know all 20 binary digits so you have the number defined exactly.

3.3 Example: Alice-Bob Messages

Let's return to the example in Section 2 where the probabilities of the four messages were: $1/2, 1/4, 1/8$, and $1/8$. The negative logarithms, base-2, of these probabilities are: $1, 2, 3$, and 3, respectively. Thus in this (ideal) situation, the entropy is exactly the same as the average amount of information in a message.

It is also interesting to note that if we look at a long stream of these four messages, encoded as 0, 10, 110 and 111, then (assuming the messages were sent at random in

accordance with the probabilities above) the stream of 0's and 1's produced would appear to be completely random. If the encodings had been the "obvious" 00, 01, 10, and 11 there would be far too many zeroes.

To see this, here is the result of sending the same 200 random messages with the above probabilities but using the two encoding schemes. This first is with the obvious inefficient encoding (400 total bits):

$$
\begin{aligned}
&1100000000000010000100010100001111000001110001010\\
&0100100011000000000000000100011100000001000100001\\
&0000010001000101001100000100000100000101000100010\\
&0\\
&1011000001001001010100000100000100000000100100100\\
&0\\
&0100110100100001100000110011000010110001010100000\\
&0\\
&0000001000100100011101110011011011000000100011111\\
&0\\
&0100111110110000001000000100101001011010000000001\\
&0\\
&1010111100000000001100000111010000110000100000110\\
&0
\end{aligned}
$$

and this is the transmission using the efficient encoding (346 total bits):

$$
\begin{aligned}
&1110000001100100101000111111001011101010010011001\\
&1\\
&0000000100101110001001000100010010010100111001000\\
&1\\
&0001010010010011011100100110101010001000100000110\\
&1\\
&0011001001111001100101100011101110011011101010100\\
&0\\
&0000011001101001011110111011110110111000110011111\\
&1\\
&1101001111111101110001100010011011010101100000 1\\
&0\\
&0110110111111000001110010111100011100110001110\\
&0
\end{aligned}
$$

Notice that the first stream has way more zeroes in it and also notice that it uses 400 bits (as it must: two bits per message) but the second stream uses 346 bits which is quite close to the expected number: $1.75 \cdot 200 = 350$.

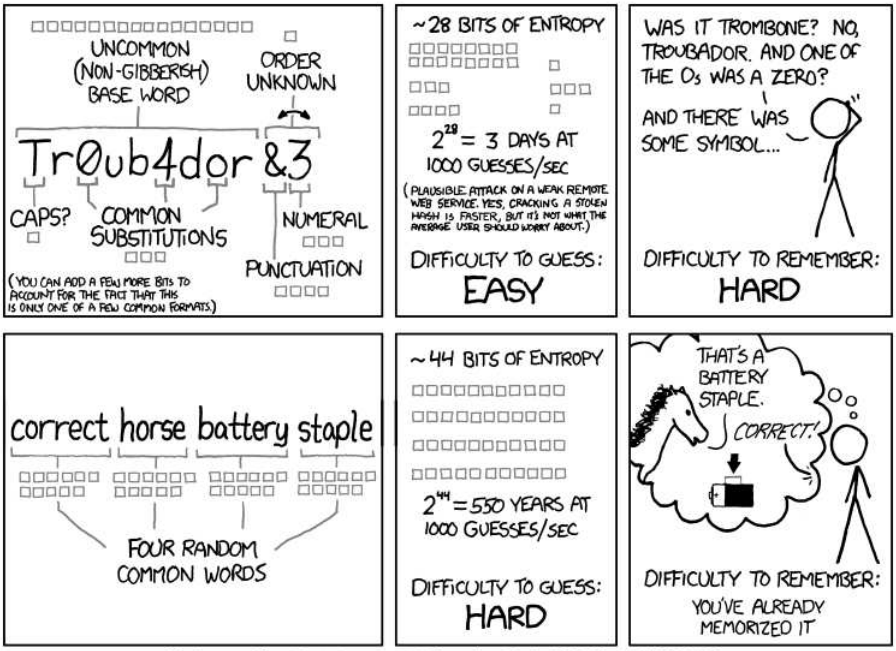
In the second stream there are 173 of the 0 bits which happens to be exactly half of the total number (346) of bits. In the first stream, there are 277 of the 0 bits and 123 of the 1 bits.

The outcomes above will be typical for very good encodings: a good encoding will generate a stream that seems to be completely random with equal probabilities of both bits and it will be shorter, on average, than steams generated by other encodings.

In fact the encoding for this set of probabilities of messages is "perfect" in the sense that it is impossible to do better in the long run with messages having the given probabilities of occurring.

3.4 Entropy As Bits/Symbol

From the web cartoon XKCD:



In the perfect example above, the entropy was exactly the same as the average number of bits per message. Let's look a a situation that's not so ideal. Again, assume that there are four outcomes, but that the probabilities are: 0.7, 0.1, 0.1, and 0.1. If we have to encode the messages individually, the best we can do (we'll see how to do this later) is the same as before: 0, 10, 110 and 111. It obviously doesn't matter how the last three are assigned to the three low-probability events.

The average number of bits per symbol will be:

$$0.7 \cdot 1 + 0.1 \cdot 2 + 0.1 \cdot 3 + 0.1 \cdot 3 = 1.5,$$

but the entropy is:

$$-(0.7 \log_2 0.7 + 0.1 \log_2 0.1 + 0.1 \log_2 0.1 + 0.1 \log_2 0.1 = 1.35678.$$

In other words, with our best simple encoding we cannot do as well as what the entropy tells us we should be able to do.

Here is a way to do better: pack the messages two at a time. If the messages were A, B, C, and D, then there are 16 possible pairs of messages: AA, AB, AC, AD, BA, $\ldots$, DC, DD. The probability of getting AA is $0.7 \cdot 0.7 = 0.49$, the probability of the 6 cases that include A and another message (in either order) is $0.07 \cdot 0.01 = 0.07$ and the probabilities of the other nine are all $0.01 \cdot 0.01 = 0.01$.

Without worrying for now about how the following encodings were produced (see Section 3.5 to learn how to do this), here they are:

$$\begin{aligned}
AA &= 0 \\
AB &= 1010 \\
AC &= 1011 \\
AD &= 1100 \\
BA &= 1101 \\
BB &= 1110 \\
`BC &= 1111 \\
BB &= 100000 \\
BC &= 100001 \\
BD &= 100010 \\
CB &= 100011 \\
CC &= 100100 \\
CD &= 100101 \\
DB &= 100111 \\
DC &= 1001100 \\
DD &= 1001101
\end{aligned}$$

With the encoding above, the average length of the bit pattern is (where each term is probability times number of items of that length of encoding times the length of the encoding):

$$0.49 \cdot 1 \cdot 1 + 0.07 \cdot 6 \cdot 4 + 0.01 \cdot 6 \cdot 7 + 0.01 \cdot 6 \cdot 8 = 2.73.$$

But remember that each of the bit patterns above stands for a pair of messages, so the per-message average length is $2.73/2 = 1.365$, which is extremely close to, but slightly larger than, the entropy (1.35678). You can see one more example of this in Section 3.7.

In fact the entropy is a limit to how well you can do. If we had an optimal bit assignment for triples or quadruples, et cetera, of messages, the average bit length would approach the entropy as closely as desired, but would never quite get there.

3.5 Huffman Encoding

How should we optimally assign bit patterns for a set of message probabilities? The answer is via something called a "Huffman Encoding."

It should be clear that whatever the best solution is, it ought to assign shorter codes to the more common messages and longer codes to the rarer messages. This is true since if there is a pair of messages M_1 and M_2 where M_1 is the most common but has a longer encoding than M_2, then you could just swap the encodings and have a more efficient final result.

We can think of the problem as a series of investments of bits. The longest strings of bits should go with the least common messages, so pick the two least-common messages and spend one bit on each (a 0 and a 1) to tell them apart. (If there are more than two of the least common messages, pick any particular pair.) This will be the final bit in the code for each, with some unknown number of bits preceding them which are the same. But now every time you add one bit to that preceding part, the cost, in terms of number of total bits spent, will be the sum of the costs for the individual messages, since that extra bit has to be added to both codes. This sentence is a little vague, but it provides an insight into how an optimal encoding is derived and to why it is the best.

As an aid to calculation, notice that it is not necessary to use the true probabilities when you are trying to optimize an encoding; any multiple of all the probabilities will yield the same minimum. In other words, suppose there are five messages ($A, B, C, D,$ and E) and you've done a count of the number of times each message occurs in some sample and you come up with the following counts that presumably are roughly in the same ratio of their true probabilities:

A	123
B	82
C	76
D	40
E	11

To find the probabilities you need to add all the counts: $123+82+76+40+11 = 332$ and then divide each of the numbers above by 332, so for example, the probability of obtaining an "A" is $123/332 = .3704\ldots$ — a bit more than 37% of the time. The same can be done for each of the numbers above, but notice that they all will be equal to 332 times the actual probability, so if you just use 123, 82, and so on, all your sums will be exactly the 332 times as large as what you'd get using the true probabilities.

So let's try to figure out logically an optimal encoding for the message set above. There are 5 messages, so 2 bits will not be sufficient; at least one of them will require 3 bits of data (and perhaps more). We can't allow four of them to have 2 bits and the other 3 bits, since the 3-bit encoding will have to start the same way as one of the 2-bit encodings and there will be no way to tell whether the 2-bit message ended, or whether we need to wait one more bit for the 3-bit message. So at least two of the messages will require at least 3 bits. As we stated above, the messages "D" and "E" should be the ones chosen to have the longest string of bits in their codes, so we can arbitrarily say that D has a code of $x\ldots x0$ and E a code of $x\ldots x1$ where the earlier bits indicated by the x's (we don't know how many yet) are the same.

But each time we add a bit to replace the x's above for the D and E characters, we add one bit to the representation of 51 characters (both the 40 D's and the 11 E's). In a sense, we can treat the combination of D and E as a single character with weight 51 together with the other three letters, and we'd have a chart that looks something like this:

A	123
B	82
C	76
(DE)	$40 + 11 = 51$

Now the least common "messages" are C and (DE), where (DE) is the combination of D and E. We should use a bit to distinguish between them, and we obtain the following chart:

$(C(DE))$	$76 + 51 = 127$
A	123
B	82

The C combined with (DE) now is the most-likely "message", so it moves to the top of the chart. The least likely of those left are A and B, which, when combined, yield:

(AB)	$123 + 82 = 205$
$(C(DE))$	$76 + 51 = 127$

Finally, one more bit needs to be added to distinguish between the two combinations above, and it could be indicated as follows, continuing the pattern above:

$((AB)(C(DE)))$	$205 + 127 = 332$

Each pair of parentheses contains two "subcodes" each of which is either a letter of a grouped pair of similar items. We can unwrap the parentheses as follows. Assign a leading 0 to one of the subcodes and a leading 1 to the other. Then continue recursively.

Thus the first bit tells us whether to go with (AB) or $(C(DE))$; arbitrarily use 0 for (AB) and 1 for $(C(DE))$. If we see a 0, we only need one bit to distinguish between A and B, so 00 means A and 01 means B. If we see a leading 1 then the next bit tells us whether we have a C or a (DE), et cetera, so the codes for the last three should be 10 for C, 110 for D and 111 for E:

Message	Frequency	Code
A	123	00
B	82	01
C	76	10
D	40	110
E	11	111

This optimal encoding requires only 3 bits for the least-common character, but there are situations with only 5 characters that 4 bits would be required for an optimal encoding. Here's an example:

Message	Frequency	Code
A	16	0
B	8	10
C	4	110
D	2	1110
E	1	1111

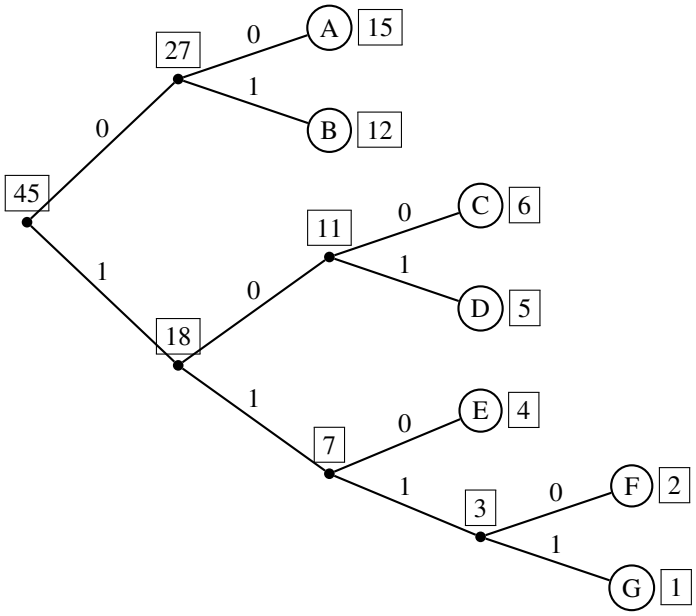
Follow the logic we used for the previous example to see that the structure for the encoding would be $(A(B(C(DE))))$.

3.6 Tree Formulation

Another way to visualize the Huffman encoding is as a tree. If we follow the same procedure as above for the following distribution of letters:

A	15
B	12
C	6
D	5
E	4
F	2
G	1

we obtain the encoding structure $((AB)((CD)(E(FG))))$. This can be displayed as the binary tree below, where the letters are the final nodes, the numbers in boxes are the weights, both of the final and internal nodes and the 0's and 1's above the lines indicate the encoding. To find the code for any particular letter, just find the unique path to the letter from the root (numbered 45) and read off the numbers on the connecting edges.



For example, to get to D from the root we follow three edges labelled 1, 0 and 1, so the Huffman encoding for D in this example is 101. Here is the complete encoding:

A	15	00
B	12	01
C	6	100
D	5	101
E	4	110
F	2	1110
G	1	1111

3.7 Entropy and Huffman Encoding

Suppose we have two possible messages, A and B, and the probability of message A is $3/4 = 0.75$ and of B, $1/4 = 0.25$. We can calculate the Entropy of this situation as follows:

$$H = -(.75 \log_2 .75 + .25 \log_2 .25) \approx 0.811278$$

and earlier we stated that this is measured in bits.

Now if we encode A as 0 and B as 1, then we "spend" one bit per message, so in a sense, we're wasting about $1 - 0.811278 = .188722$ bits for each message. Is there some clever way that we could get by with fewer bits? If we're willing to group the messages, we can do better using a Huffman encoding.

We can group as many messages as we want, but let's just group them three at a time, so that we send sequences of bits that correspond to sets of three messages. Here is a table of all the possible 3-message groups, together with their probabilities and an optimal Huffman encoding (there is more than one optimal encoding) for the groups:

Group	Probability	Encoding
AAA	$27/64 = 0.421875$	1
AAB	$9/64 = 0.140625$	001
ABA	$9/64 = 0.140625$	010
BAA	$9/64 = 0.140625$	011
ABB	$3/64 = 0.046875$	00010
BAB	$3/64 = 0.046875$	00011
BBA	$3/64 = 0.046875$	00001
BBB	$1/64 = 0.015625$	00000

Over the long run, about $27/64$ of the time we'll send a 1-bit message, $(9+9+9)/64 = 27/64$ of the time, a 3-bit message, and $(3 + 3 + 3 + 1)/64 = 10/64$ of the time, a 5-bit message. The expected (average) message length (in bits) will thus be:

$$\frac{27}{64} \cdot 1 + \frac{27}{64} \cdot 3 + \frac{10}{64} \cdot 5 = \frac{158}{64}.$$

Since each of these 1-, 3- or 5-bit grouped messages corresponds to a group of three of the original messages, the average number of bits sent per original (A or B) message will be: $158/(3 \cdot 64) = 158/192 \approx 0.822917$, which is very close to the 0.811278 bits predicted by the entropy calculation. Grouping more messages will yield even better approximations.

Exercise: Work out a Huffman encoding if we pack only two of the original messages above per group. Show that the bit efficiency is $27/32 = 0.84375$, which is already pretty close to the value predicted by the entropy.

4 Back to Wordle

How can entropy be used to try to solve Wordle puzzles efficiently? For the entropy discussions, we have been talking about messages between Alice and Bob and how are

these related to Wordle queries and responses?

For now let's consider a simple version of Wordle where all of the possible words are in a single list that is available to the questioner[3]. In the original version of Wordle this list contained about 2300 words. We also assume that the secret word is chosen at random from this list with a uniform distribution. In other words, humans do not select words, and just because a word has already been used, it is still equally-likely to be selected as an unused word.

If Alice is the human and Bob is the computer with the hidden word, then at each stage Alice selects a word from the list and the answer she receives is a series of five colors chosen from {black, green, yellow}. There are $3^5 = 243$ possible responses, but some are clearly impossible, like ones containing 4 greens and a yellow. (Are there any more impossible responses? Answer: no. Why?)

If Bob's response is "green-green-green-green-green" then Alice has guessed the correct word and the game is over. Otherwise Alice just makes another query which depends on her previous queries and Bob's responses to those queries.

If Alice has all the time in the world (or a computer) here's what she should do:

1. Alice's initial list of possible words consists of the entire list of possible legal guesses.

2. Alice has a data structure that keeps a list of words associated with each of the 238 possible responses that Bob can make.

3. Loop over the following actions until the puzzle is solved or Alice has made 6 total guesses without finding the hidden word.

   (a) For every word (say word1) in Alice's list.

   (b) First, clear the data structure so that all the lists are empty.

   (c) Then, for every word (say word2) in Alice's list calculate the response Bob would give if the guess were word1 and the solution were word2. Add word2 to the list corresponding to that pattern.

   (d) Finally, calculate the entropy associated with the distribution of words in the data structure and associate that with word1.

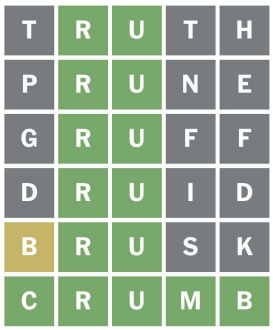
4. Choose any word1 with the largest entropy and make that the next guess, and at the same time, replace Alice's list with the smaller list of possible word2s corresponding to Bob's response.If there are N possible words at some stage, and there are n_1 words corresponding to green-black-yellow pattern 1, n_2 wors to pattern 2, et cetera, than the probabilities are $p_1 = n_1/N$, $p_2 = n_2/N$, et cetera.

[3]In the NYT version Wordle has two word lists. One is of more common words that are possible solutions, and the other contains words that can be used as guesses, but are not possible solutions. In fact, the possible Wordle solutions contain almost no plural words, so the set of possible guesses includes lots of four-letter wordss followed by an "s" to make them plural.

Each guess word1 divides all the possible solutions into lists that correspond to possible responses from Bob. Some are empty, and some might contain a lot of possibilities. For example from the initial list if Alice had guesses FUZZY, then the list corresponding to "black,black,black,black,green" would contain every word ending in Y but having none of U, Z, or F[4].

The basic idea is that you want to make the lists as equal in size as possible, and the list with the largest entropy corresponds to this.

5 Problems



6 Intuitive Justification for the Entropy Formula

Let's model any situation with a finite model. Suppose you have N balls of k different colors. Suppose there are n_1 of the first color, n_2 of the second, and so on, up to n_k of the k^{th} color. Of course we have:

$$N = n_1 + n_2 + \cdots + n_k,$$

and $n_i/N = p_i$ is the probability of getting ball number i. By choosing N large enough we can choose a set of n_i to approximate any real probability distribution as closely as we'd like.

If we draw out a single ball, that ball will have probability n_i/N of having color i, for every i. The multinomial coefficient:

$$\left(\frac{N!}{n_1!n_2!\cdots n_k!}\right)$$

represents the number of different arrangements there are of the N balls. The larger the number of arrangements, the less we know about the situation, so in a sense, the larger the multinomial coefficient above, the larger the entropy of the set of arrangements.

[4]In fact,, in some sense, FUZZY is the worst possible guess in the sense that it generates the smallest entropy for an input word.

Also, intuitively the multinomial coefficient will be larger if all the n_i are close to equal, and if the multinomial coefficient is large, so will be its logarithm, or even its logarithm divided by N. Thus one possible entropy-like measure of the complexity might be:

$$\frac{1}{N}\log\Big(\frac{N!}{n_1!n_2!\cdots n_k!}\Big)$$

We can use Stirling's approximation to the factorial function to estimate this value. Stirling's formula is this:

$$\ln m! \approx m\ln m - m,$$

where "ln" is the natural logarithm (the logarithm base-$e \approx 2.71828$).

Thus the expression for the multinomial coefficient above is approximately:

$$\frac{1}{N}\Big(N\ln N - N - (n_1\ln n_1 - n_1 + n_2\ln n_2 - n_2 + \cdots + n_k\ln n_k - n_k)\Big).$$

The N and all the n_i disappear since N is the sum of the n_i and this becomes:

$$\begin{aligned}
& \frac{1}{N}\Big(N\ln N - n_1\ln n_1 - n_2\ln n_2 + \cdots n_k\ln n_k\Big)\\
= \;& \frac{1}{N}\Big((n_1+n_2+\cdots n_k)\ln N - n_1\ln n_1 - n_2\ln n_2 + \cdots n_k\ln n_k\Big)\\
= \;& \frac{1}{N}\Big(n_1(\ln N - \ln n_1) + n_2(\ln N - \ln n_2) + \cdots + n_k(\ln N - \ln n_k)\Big)\\
= \;& \frac{1}{N}\Big(n_1(\ln(N/n_1)) + n_2(\ln(N/n_2)) + \cdots + n_k(\ln(N/n_k))\Big)\\
= \;& \frac{n_1}{N}\ln\frac{N}{n_1} + \frac{n_2}{N}\ln\frac{N}{n_2} + \cdots \frac{n_k}{N}\ln\frac{N}{n_k}\\
= \;& p_1\ln(1/p_1) + p_2\ln(1/p_2) + \cdots + p_k\ln(1/p_k)\\
= \;& -(p_1\ln p_1 + p_2\ln p_2 + \cdots + p_k\ln p_k).
\end{aligned}$$

Since $\log_2$ differs from $\ln$ by a constant factor, this is just a positive multiple of the entropy formula we have used in the previous sections.