Grady Robbins
AST4930
71536892

AST4930 Project 1

Code Process and Structure:

I started this cryptography project by importing the needed modules and loading the full text of *War and Peace*. I created user-defined functions to calculate the accuracy and probability of future phrases which will be used later. I created an empty 26 by 26 array to contain both letters from a-z as well as numbers within them (to assign point values based on letter probabilities in the future). I then converted the given text into a list of individual characters, then removed all special characters that are not letters. The first probability array will be ordered by row→column, so letters in the order 'ac' will go in row a, column c. I then calculated my probability matrix using a for loop and a dictionary that relates each letter to a number, allowing an index for the probability array. After converting that to a percentage array by dividing by the entire length of characters, a random solution key was created for comparisons of effectiveness. I tested the code by swapping characters in a random alphabet multiple times, and only keeping the new configuration if the probability score (given by the probability matrix) of the unscrambled phrase was higher than before.

Running the code with the scrambled phrase of "jack and jill went up a hill to fetch a pail of water" (no spaces) with 50 loops gives an ending accuracy of 2 to 17% with no annealing (not very good), and rare outcomes of 20-30% accuracy. An example of a run in my code is shown by figure 1, where the graph is of accuracy and probability over time. Because I cannot use accuracy as a guide for my iterations, it is only based on probability of letter ordering (the accuracy variable is there only to compare). This method is ineffective as seen below, so a different probability matrix may need to be developed in the future.

Grady Robbins
AST4930
71536892

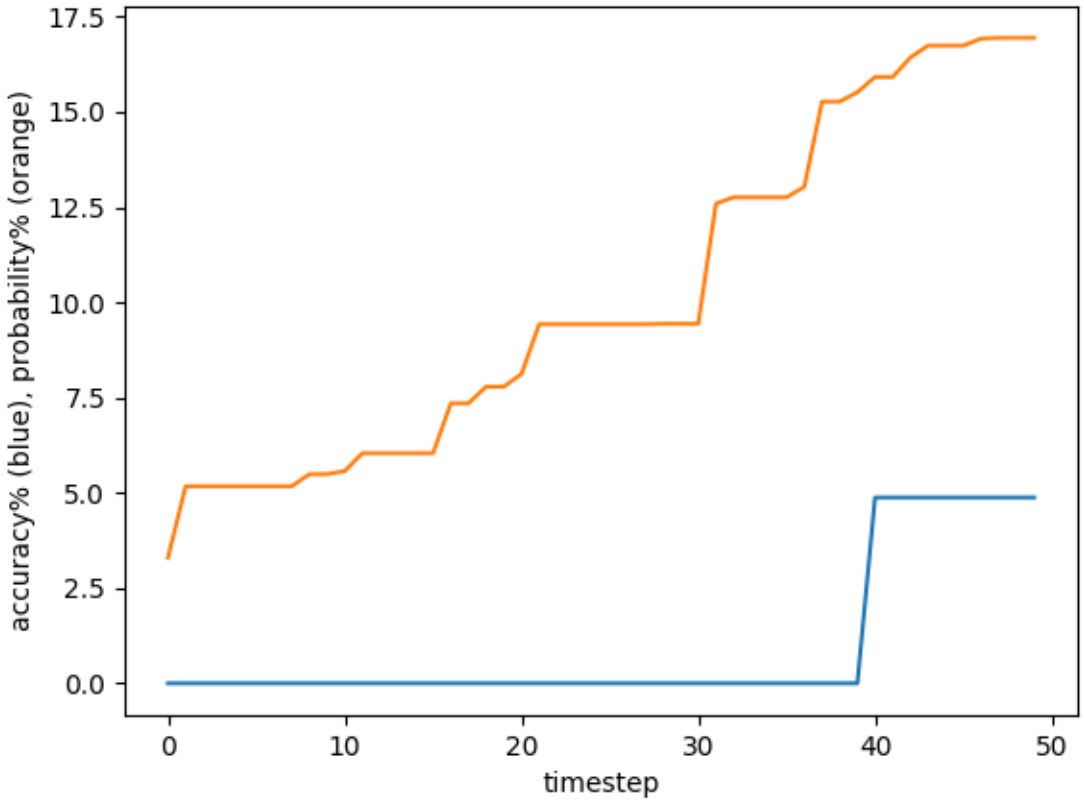

Figure 1: Graph of accuracy and probability over 50 timesteps with no annealing (positive only). Probability is scored by direct letter ordering.

Using an annealing factor of e ^ (1/k) where k is the iteration of the for loop, my final answer's probability score is 7% to 30% (14% average) and usually better than using no annealing. Because of this, using annealing seems to work better using my method. Running the same code, annealing and no annealing, with iterations above 50 are generally much less accurate, so I will be keeping the 50 iteration method. In most tests.

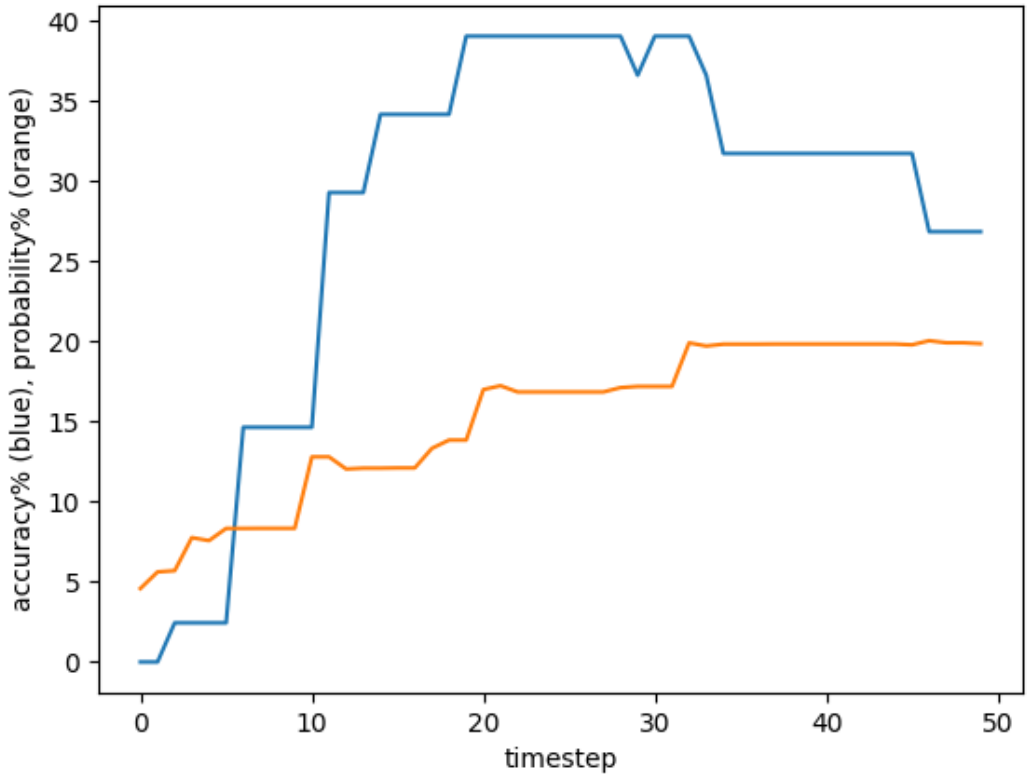

Figure 2: Graph of accuracy and probability over 50 timesteps with annealing of e^(1/timestep). Probability is scored by direct letter ordering.

This method of solving is difficult because the desired phrase can likely not be reached by probability score of letter order alone. To increase accuracy, I may create another probability array using letter ordering that skips a character (ie. “cat” follows a matrix of t one letter after c). Taking spaces into account could help as well, but these are just possibilities. At this point, I believe the probability calculation by direct letter ordering is good enough, now all that’s left is to change how the probability relates to the actual phrase.

The given phrase of “jack and jill went up a hill to fetch a pail of water” has a probability score of around 20 from my direct lettering matrix, which is actually less than the probability scores of my iterations when the timestep is above 50 (probability, not accuracy). To guarantee better results, I could create more probability tables that take words, word sequences, letter spacing, and surrounding words into account. If I were to calculate the final phrase and compare

it to a dictionary with real word sequences, this would allow me to only run the loop until actual words are formed. That being said, this would extend the time needed to run my code by a great amount, and I likely won't have enough time to develop.

I created a new probability matrix based on letter ordering but spaced it by one using the same method as the previous matrix. This means that the word "**o**n**e**" would add a point to the matrix indices row '**o**', column '**e**'. Using this matrix as well as the original and averaging them for probability calculation, my new average accuracy is around 17% as seen in figure 3.

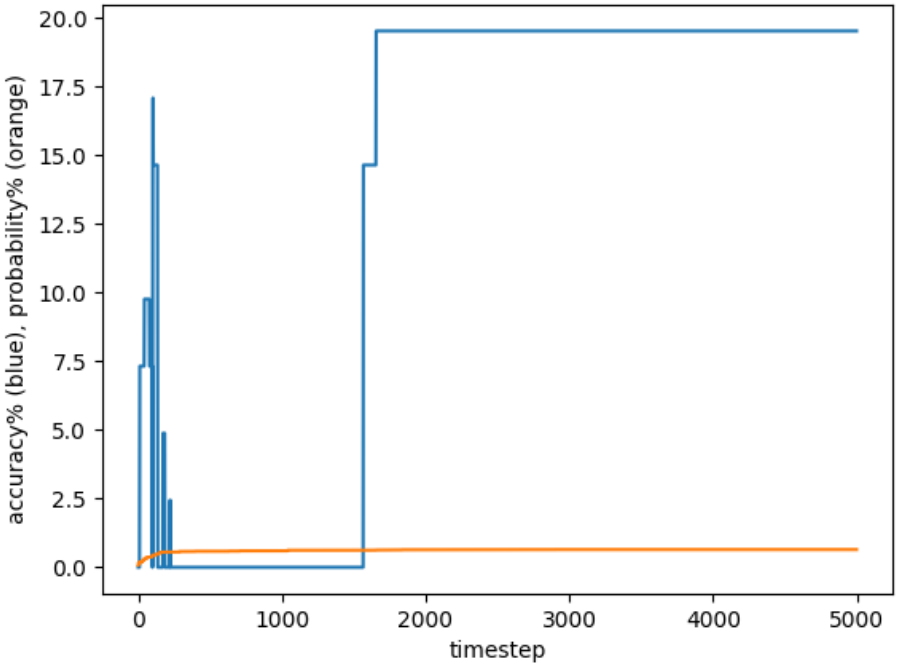

Figure 3:
Probability and accuracy over time with an average probability of letter ordering directly and spaced by one (abc→a,b and a,c)

If I simply sum the two probabilities instead of averaging them, I get a slightly better average accuracy (19%) with a tendency for better runs seen in figure 4. Although my accuracy to the true phrase is still low, the fact that the accuracy is continuously increasing with the changes made is a good sign that I'm on the right track. As a final test for the letter ordering method, I may create a third probability matrix to account for letters spaced apart by two (i.e. "**w**or**d**"). This will likely be my stopping point for letter ordering, as many words do not exceed 3 characters making further matrices negatively impact my probability.

Grady Robbins
AST4930
71536892

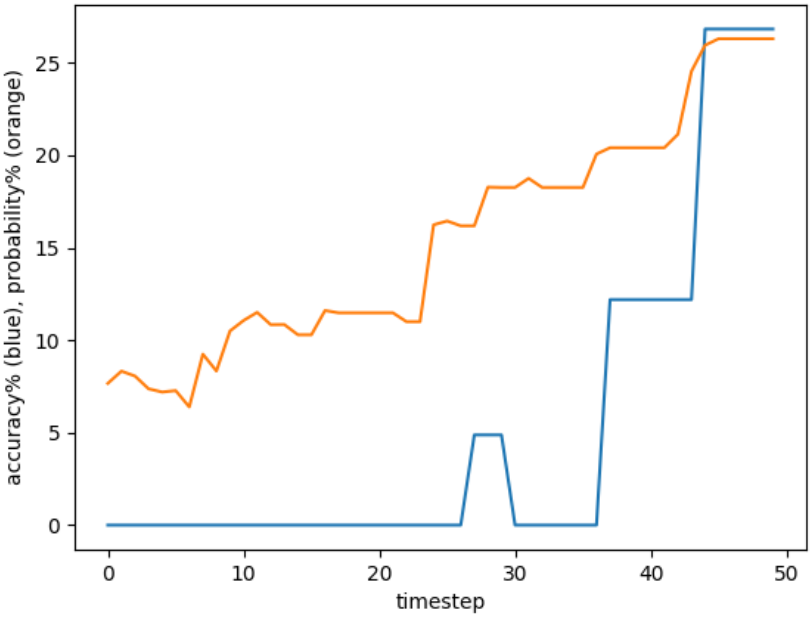

Figure 4:
Probability and accuracy over time with an addition probability of letter ordering directly + ordering spaced by one (abc→a,b and a,c)

Adding the third probability matrix of letter spacing 2 appears to have negatively affected my accuracy (see figure 5). Most of the time the accuracy ends on 0 with some 4%'s and 9%'s. This is understandable because the phrase and text are not separated, allowing longer spacing methods to bleed into other words or sentences.

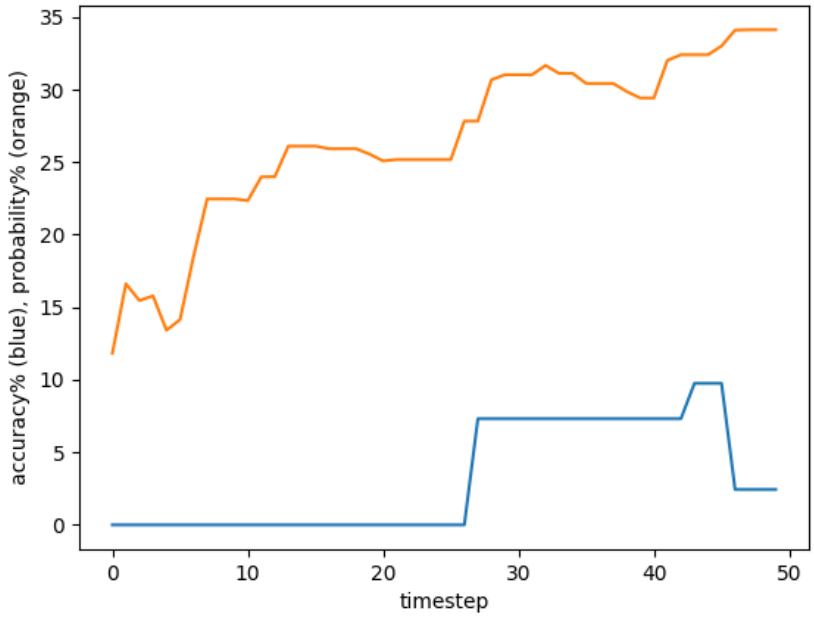

Figure 5:
Probability and accuracy over time with an addition probability of three letter ordering methods (abcd→a,b ; a,c ; a,d)

Grady Robbins
AST4930
71536892

To summarize my coding process and results, I ran multiple annealing experiments using differing probability arrays, and only kept the arrays and annealing variables that positively changed my average outcome. I tested summation, averaging, and standalone versions of probability calculations, and from that I reached a final average accuracy of ~ 19%, with some rare outcomes of up to 50% accuracy. To reach a better outcome, many more tests could be performed that change probability calculations, annealing factors, or tests that simply use a different example text (perhaps one that relates closer to the material of the true phrase).