Derek Klein
Computational Astrophysics
Cryptography Project

Code Architecture of Final Design

Source text, goal, and encrypted phrase
Import wp.npz
Convert source text to a string
Clean up text such as "- " from line breaks or "xxx" from chapter numbers
Input goal and encrypted goal as string variables

Probability matrix
Create alphabet index to convert from letters to numbers for indexing
Create probability matrix
Search through text for pairs of letters and set matrix values to counts
Calculate a running total of counts
Convert probability matrix to log10 space

Calculating Probability
Translate the coded message with the initial random key
Find the translated code's probability product
Initialize annealing

Main Loop
Copy a new key
Swap values
Test probability against initial test
If it's better, keep it and update key
If it's worse, compare the annealing value to a random value and occasionally keep it

Statistics
After finding the local maximum, decode using the final key found and save statistics on the 50 trial runs.

Derek Klein
Computational Astrophysics
Cryptography Project

Testing Phase

Baseline Code

For baseline, the following conditions were used:

- The phrase used was the first line of the Gettysburg Address
- No wrong answers accepted (No annealing)
- Summation probability
- Two letter combination testing
- Raw source text
- No penalties
- 10,000 iterations

Then each condition was altered until the best result was obtained. The next condition was then altered with the previous conditions left untouched. Each condition was tested with 50 trials, and the following statistics were recorded:

- Number of zero correct runs
- Average letters correct
- Average accepted swaps
- Histogram of percent correct
- Graph of accepted probability over time

Accepting Wrong Answers Through Annealing

- Linear Annealing
  A percentage of the wrong answers were kept regardless of the iteration number. Tested 50%, 25%.
  Code snippet:
  ```
  elif rn.random() < annealing_percent:
        # Keep wrong answer
  ```
- Scaled Annealing
  Fewer wrong answers are kept over time based on loop iteration. The speed was adjusted with a multiplier, M, of values 1, 2.
  Code snippet:
  ```
  elif rn.random() > (M*iteration/total_iterations)
        # Keep wrong answer
  ```
- Exponential Annealing

A decaying rate of acceptance of wrong answers. The speed was adjusted with a multiplier, M, of values 1, 2.

Code snippet:
```
tau, T = 1000., 1000.
```

```
while T < 10000:  # Main loop
     eng = np.exp(-M*iterations/tau)
     elif rn.random() < eng
          # Keep wrong answer
```

Cleaning Up the Source Text

Removed strings from word breaks and from chapter numbers from the source text. Code:

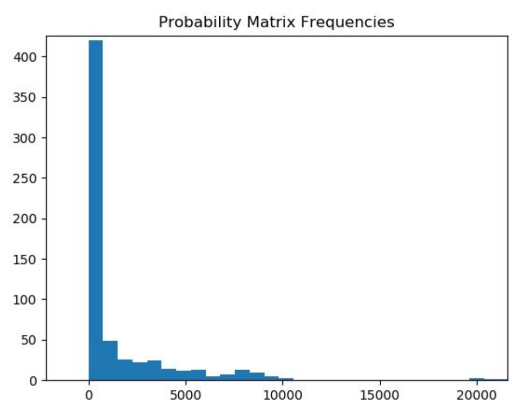
```
stext = stext.replace('- ', '').replace('xxx', '').replace('xx',
'').replace('iii', '').replace('ii', '').replace('xv', '')
```

Bonuses for Very Common Pairings

Added a bonus to very common letter pairs after analyzing the probability matrix frequencies using plt.hist(probmat.flatten(), bins=200). Zooming in on the 0-20000 counts range, the values greater than 10000 were chosen to be boosted to double their original values, weighing them more favorably.

Code:

```
if probmat[first,second] > 10000:
probmat[first,second] *= 2
```



Penalties for Zero Probability

Added a penalty to letter pairs with zero possibility of being next to each other in the source text, due to the high amount of zero counts found. This should prevent a key from being equally weighted when a letter is swapped to a non-likely pairing.

Code:

```
if probmat[first,second] == 0:
probmat[first,second] = -5000
```

Calculating a Product of Probabilities

From the http://probability.ca resource, they used a probability of product to calculate the decryption values to compare. I implemented a similar system after removing boosting and penalties, and adding 1 to zero count pairs, and another 1 after calculating the log10 of the count. The matrix was tested with multipliers of 1 and 2. Code snippet:

```
probability_matrix = (M*np.log10(probability_matrix)+1)+1
# Main loop
    probability *= probmat[aldex[key[i]],aldex[key[i+1]]]
```

Diagnostic Plots

Derek Klein
Computational Astrophysics
Cryptography Project

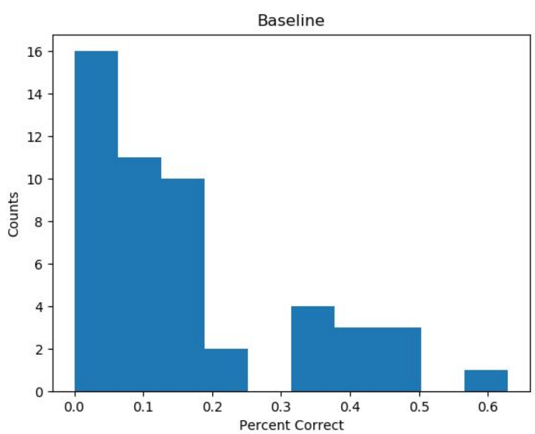
Results: Annealing

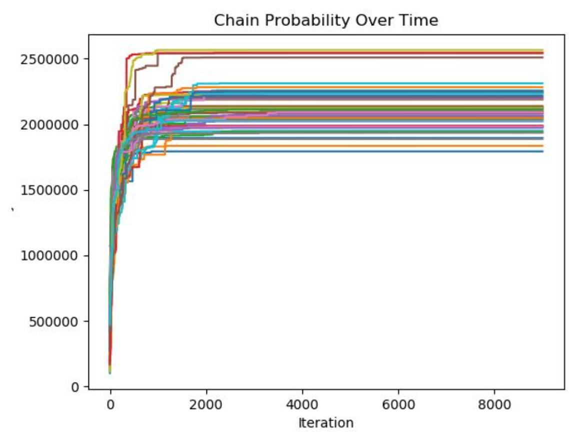
Baseline

Stats:

Number of zero correct runs: 5
Average letters correct: 0.15818
Average accepted swaps: 65.2

Graphs:



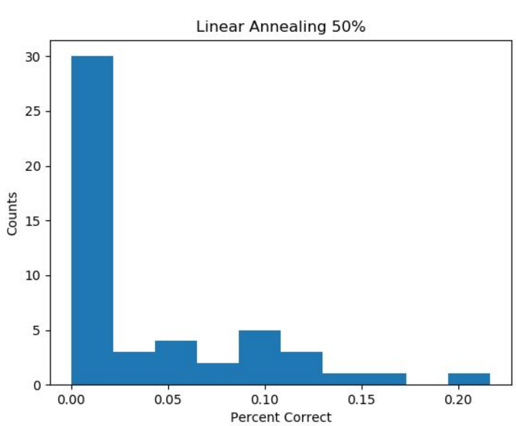


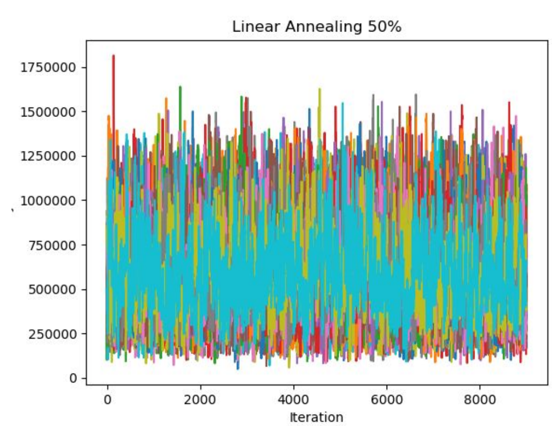
Linear Annealing 50% Kept

Stats:

Number of zero correct runs: 16
Average letters correct: 0.05455
Average accepted swaps: 6324.0

Graphs:





Linear Annealing 25% Kept

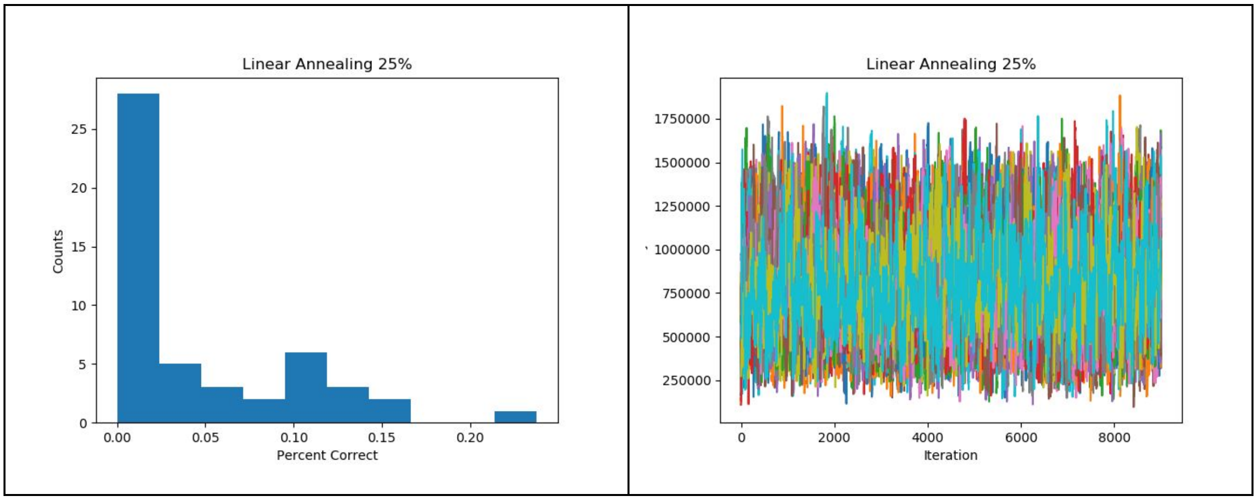
Stats:

Number of zero correct runs: 22

Average letters correct: 0.04280

Average accepted swaps: 4409.3

Graphs:



Scaled Annealing M=1 (M*iterations/total_iterations)

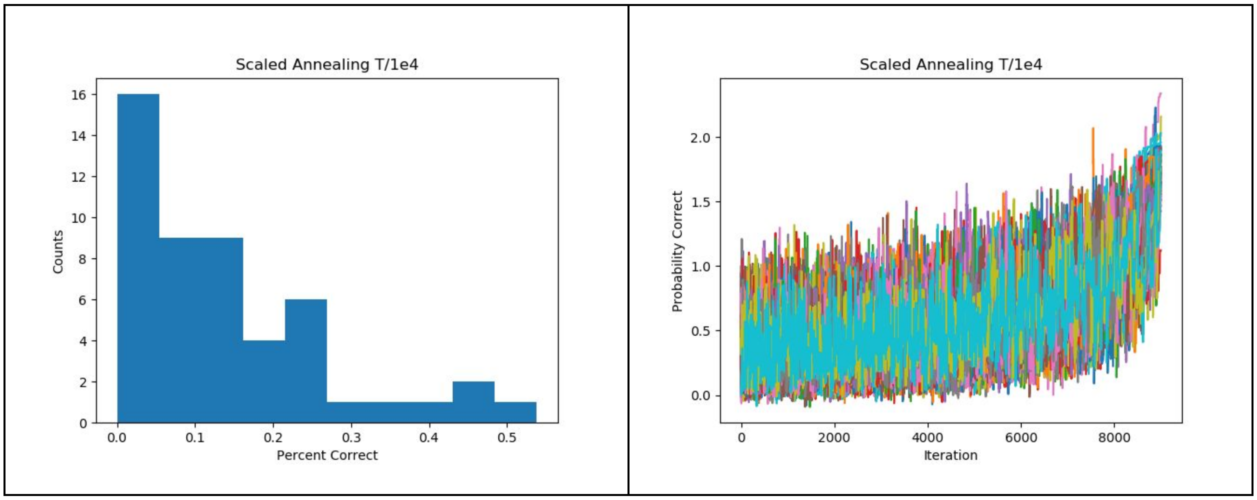
Stats:

Number of zero correct runs: 8

Average letters correct: 0.14208

Average accepted swaps: 5640.54

Graphs:

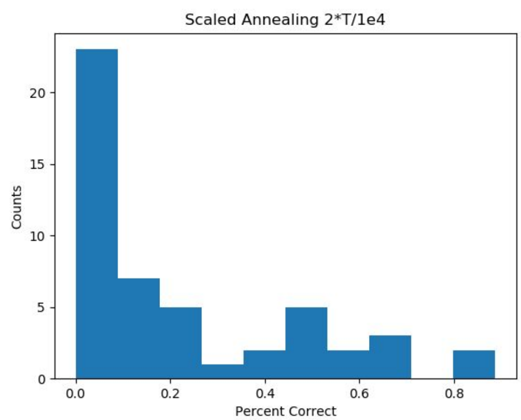


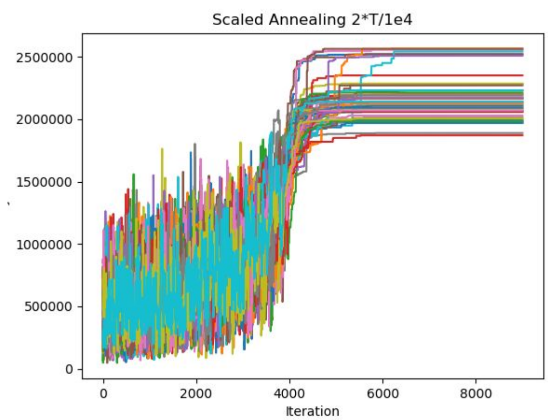
Scaled Annealing M=2

Stats:

Number of zero correct runs: 4
Average letters correct: 0.22223
Average accepted swaps: 2398.18

Graphs:



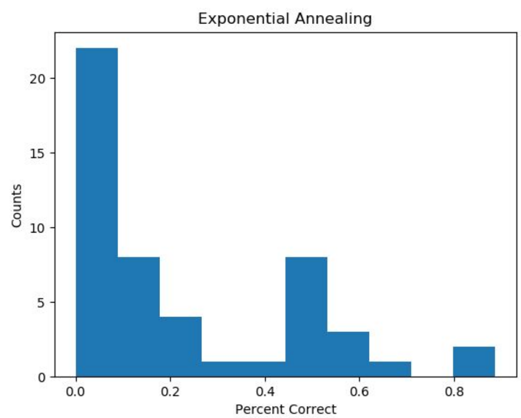


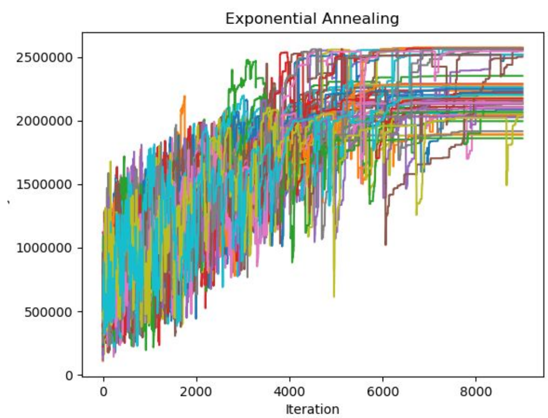
Exponential Annealing M=1; exp(-M*iterations/1000)

Stats:

Number of zero correct runs: 10
Average letters correct: 0.22210
Average accepted swaps: 1069.14

Graphs:





Derek Klein
Computational Astrophysics
Cryptography Project

Exponential Annealing M=2

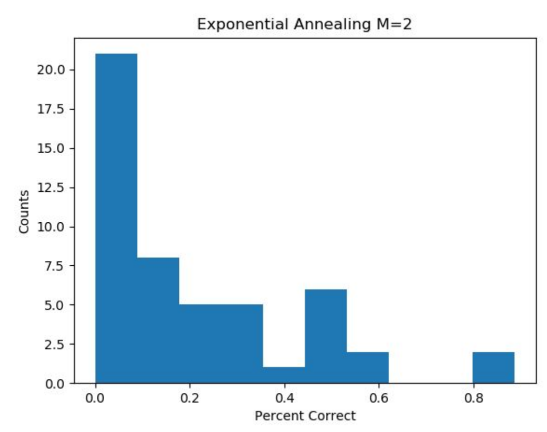
Stats:

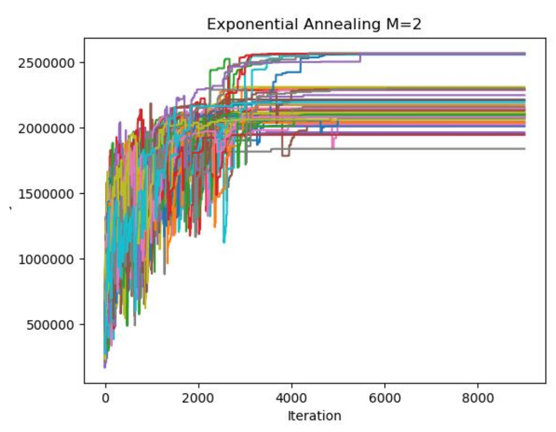
Number of zero correct runs: 8

Average letters correct: 0.20839

Average accepted swaps: 323.88

Graphs:





Annealing Synopsis

Linear annealing is garbage, the chain has to have some type of scaled or exponentially decaying annealing involved. I opted to keep the M=2 exponential annealing for the next set of tests, since the M=1 annealing appeared to not be fully finished settling down by the time the loop ended. I also lowered the iterations to 7500 which is well into the flat regime, but allows 75% quicker runs.

Derek Klein
Computational Astrophysics
Cryptography Project

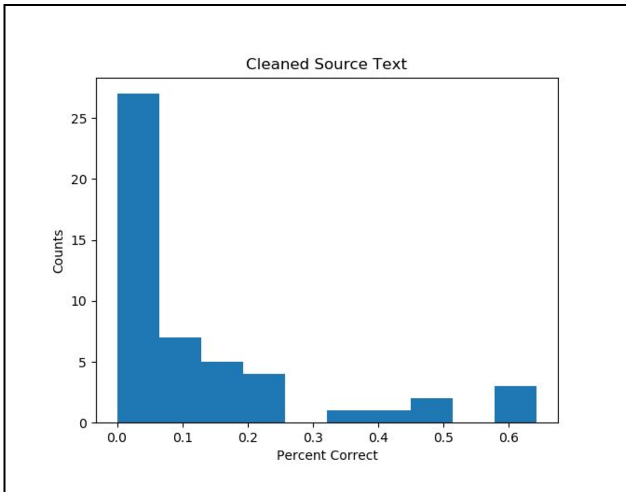
Results: Adjusting Source Text and Probabilities

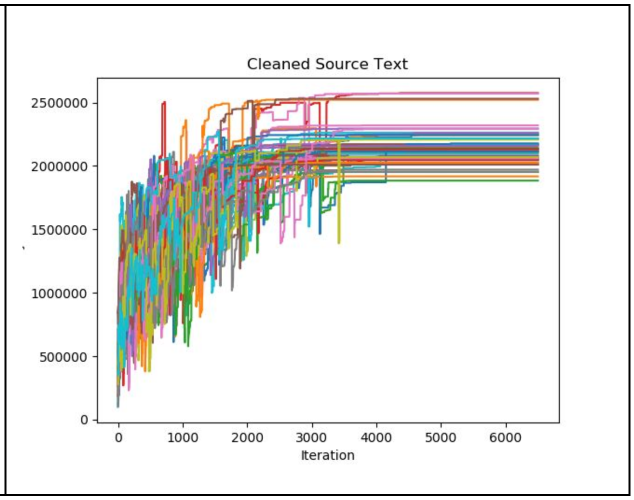
Cleaning up the Source Text

Stats:

    Number of zero correct runs: 4
    Average letters correct: 0.13524
    Average accepted swaps: 324.88

Graphs:



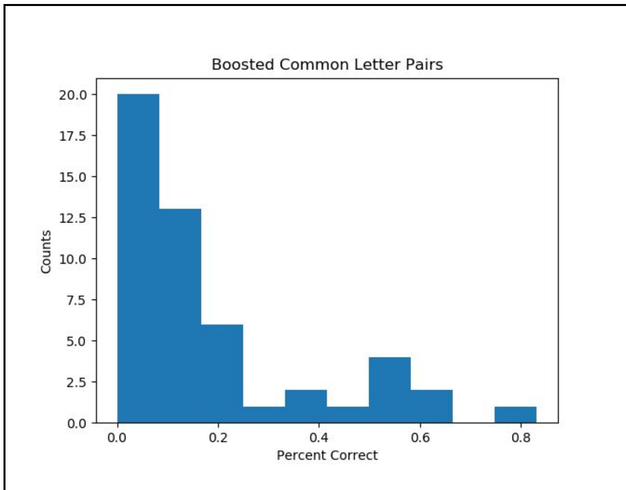


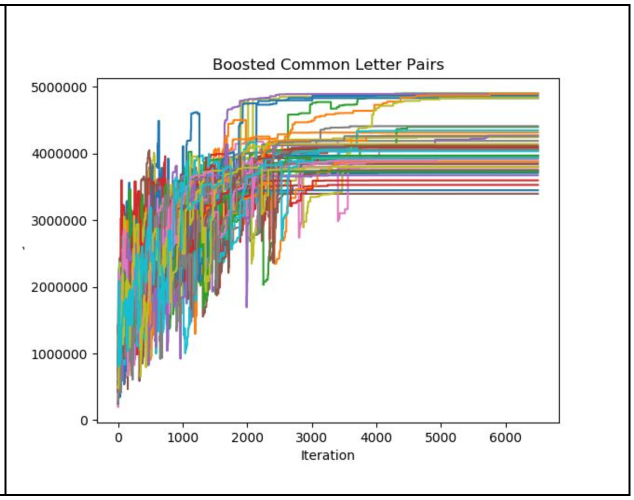
Boosting Common Letter Pairs 2x

Stats:

    Number of zero correct runs: 10
    Average letters correct: 0.17972
    Average accepted swaps: 323.98

Graphs:





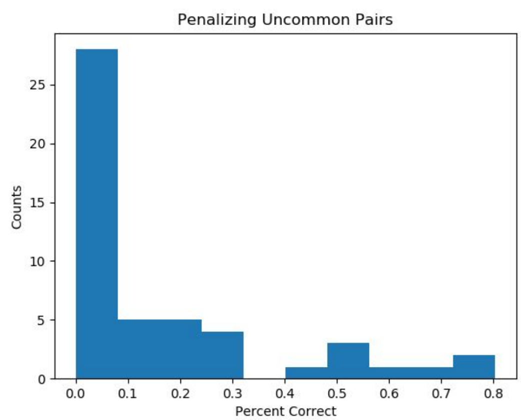
Derek Klein
Computational Astrophysics
Cryptography Project

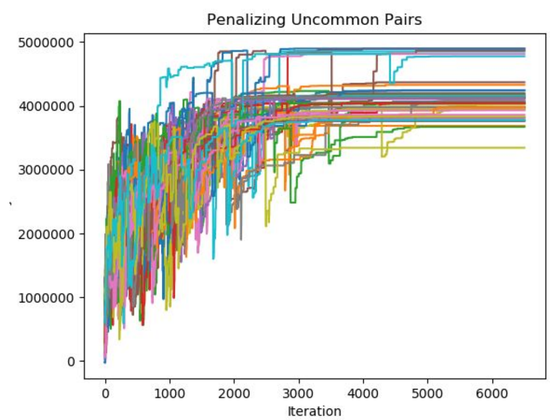
Penalizing Uncommon Letter Pairs

Stats:

  Number of zero correct runs: 7
  Average letters correct: 0.16699
  Average accepted swaps: 325.42

Graphs:



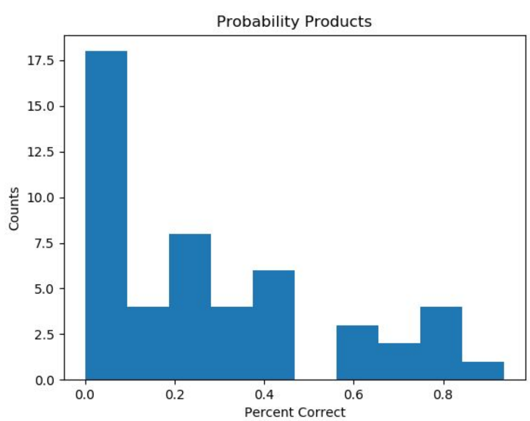


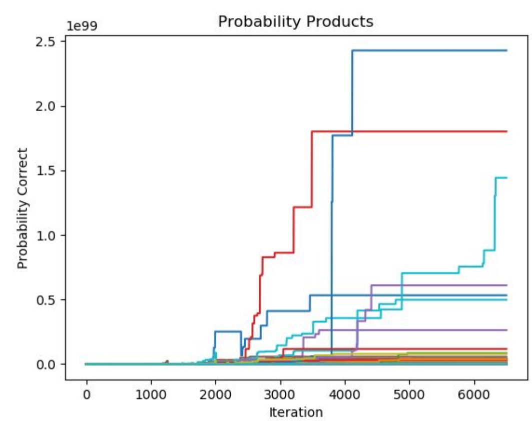
Probability of Products np.log10(1*(counts+1))+1

Stats:

  Number of zero correct runs: 4
  Average letters correct: 0.28839
  Average accepted swaps: 307.98

Graphs:





Probability of Products np.log10(2*(counts+1))+1

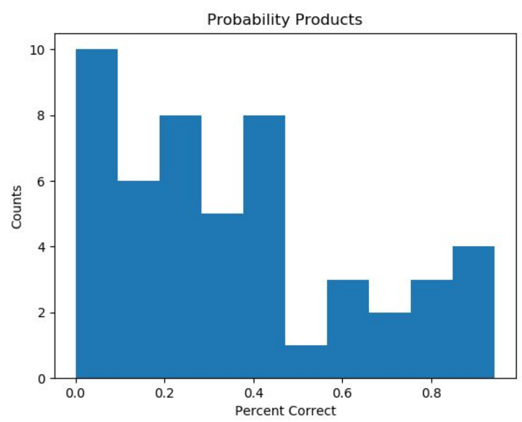
Stats:

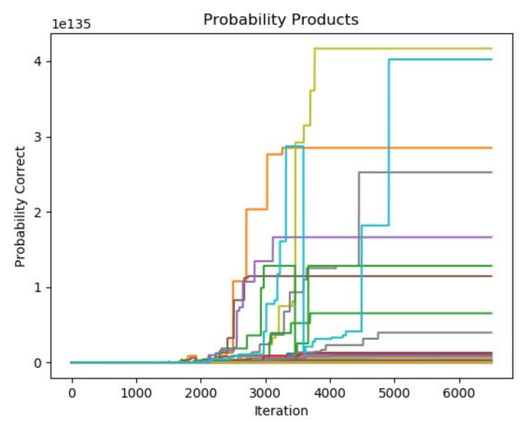
Number of zero correct runs: 1

Average letters correct: 0.36125

Average accepted swaps: 303.28

Graphs:





Probability and Letter Frequency Adjustments Synopsis

Removing, boosting, or penalizing certain letter combinations had very little effect on the chain's overall efficiency. On the other hand, switching to a product of probabilities had a drastic effect, with keys having higher percent corrects more often.

Conclusion

There are about a million more things I want to adjust and will most likely continue playing with. In essence, the best chains will calculate probabilities through products of common letter pairs and use exponential annealing to slowly cool down from accepting incorrect probabilities. The source text simply has to be long enough for the most common pairs to appear with enough frequency for the chain to pick up on, it doesn't necessarily have to be adjusted in any way. Ideally, start with a product probability and adjust annealing from that point on for the best results.

The best run from the final Markov Chain Monte Carlo method:
mourscoreandsevenyearsagoourmathersproughtmorthonthiscontinentanewnationconceivedinlipertyanddedicatedtothefrofositionthatallkenarecreatedebual