Nicholas Charchut
May 11, 2018

Efficient Substitution Cipher Cracking Using MCMC Sampling

When compared to the vanilla Metropolis-Hastings (MH) implementation, there was much room for improvement. Convergence occurred often, but far from all runs were 100% accurate. The primary goal of this "upgrade" was to dramatically increase efficiency while ensuring high accuracy. This was done primarily through a smarter initialization, log-likelihood (LL) calculation optimization, "break-early" strategy, and what I affectionately call "Smart Swap Technology", henceforth known as SST. The implementation of these suggested changes is described below, as well as their resulting effects on the end accuracy.

TEST SETUP

In order to properly evaluate performance of the upgraded MH algorithm, I first wrote a testing protocol that worked as follows. For each provided plaintext file, a new random encoding cipher was generated, and the plaintext was converted to ciphertext before being evaluated by the proposed MH algorithm. Each of the four provided plaintext files was tested 25 times in order to adequately capture the behavior of the algorithm. In addition, length of ciphertext was specified for testing, whether full text, 10000 characters or 1000 characters. Average accuracy, time per test (seconds, including initialization), and last iteration of maximum LL update were recorded for metrics. Initial results are included below.

Text	Plaintext	Paradise Lost	War and Peace	Feynman
Average Accuracy	0.81259	0.85030	0.87629	0.84821
Time / Test (s)	2.27952	1.90084	2.85112	2.12833
Last Iteration	2395.48	2934.56	2315.96	2329.20

Table 1. Metrics for full text with only LL optimization.

METHODOLOGY & REFINEMENTS

In terns of improving likelihood of perfect translations (100% accurate), there were several notable actions taken. First, a naïve but substantial improvement resulted from running the algorithm multiple times and taking the best result. This naturally led to a slowdown of the algorithm; running it multiple times meant efficiency was becoming increasingly key.

Thus I designed a simple but effective tool for efficiently visiting viable states in the Markov Chain, which I call SST. If a candidate cipher produces an invalid LL, i.e. the difference between the current LL and the candidate LL is greater than -15, then I make certain not to sample that same swap on the next iteration. I argue that this doesn't substantially effect the proposal distribution because the likelihood of acceptance is so small ($e^{-15} \approx 3 * 10^{-7}$), and so I need not change how

the acceptance rate is calculated. Using SST then implies that I sample from more viable swaps with a greater likelihood, which dramatically expedites convergence (note the decrease in "Last Iteration" between Table 1 and 2), without sacrificing accuracy in decoding. I note that the set from which I sample is reset when the proposed state is accepted.

Text	Plaintext	Paradise Lost	War and Peace	Feynman
Average Accuracy	1.00000	1.00000	1.00000	1.0000
Time / Test (s)	2.30430	2.23808	3.54828	2.49050
Last Iteration	1841.98	2025.38	1768.80	1943.67

Table 2. Metrics for full text with LL, multiple runs, and SST.

Efficiency is improved further when implementing an "break-early" strategy, which is two-fold. One, if the maximum LL has not been updated in 1000 iterations or I reach the end of 10000 total iterations, break. Because of SST, I know I do not need to iterate for long when hovering at a local maximum because I am constantly retrying potentially viable swaps. Two, if the majority of repetitions (i.e. multiple runs) break early, break from the algorithm early. For example, if I execute 10 runs for 1000 characters of *War and Peace,* and five of them terminate early, then it is likely that one of those five is the best cipher.

Lastly, initialization is made more effective via the following strategy. Identify the decoding values for the period and space character, and then align the remaining undefined alphabet by frequency in accordance with English language unigram frequency. Because it is given as an invariant that a space always follows a period, it is relatively simple to identify such a transition in a body of text: the transition beginning with character *n* that occurs the most often such that there exists only one transition that terminates on a character not equal to *n*. After identifying period and space, the frequency of the remaining letters in the ciphertext is calculated, sorted and aligned with sorted unigram frequencies. For example, if the character "z" is the most common character in the ciphertext and it is neither the space nor the period, then it would be assigned to "e". This initialization likely gives several correct mappings from the start and so we begin at a place closer to the global maximum. We can note the effectiveness of this strategy by again observing the dramatic decrease in time per test and last iteration updated, while not sacrificing accuracy.

Text	Plaintext	Paradise Lost	War and Peace	Feynman
Average Accuracy	1.00000	1.00000	1.00000	1.00000
Time / Test (s)	**0.92499**	**0.87741**	**1.25954**	**0.88373**
Last Iteration	**819.40**	**1071.42**	**817.46**	**804.52**

Table 3. Metrics for full text with LL, multiple runs, SST, smart initialization, and break-early.

I include below the same tests for length 1000 ciphertext, and extend the same analysis to them. The only difference, I note, is the time per test, which would naturally take longer for shorter

text, since maxima are much more shallow given the small number of characters. Thus, for shorter texts, the new algorithm takes at most twice as long, but guarantees near perfect accuracy.

Text	Plaintext	Paradise Lost	War and Peace	Feynman
Average Accuracy	0.49300	0.73356	0.71788	0.78392
Time / Test (s)	**1.29497**	**1.54635**	**1.49471**	**1.39262**
Last Iteration	9740.64	9815.76	8895.16	4027.84

Table 4. Metrics for 1000 characters with only LL optimization.

Text	Plaintext	Paradise Lost	War and Peace	Feynman
Average Accuracy	**1.00000**	**1.0000**	**0.99999**	**1.00000**
Time / Test (s)	2.92666	2.78086	3.10039	2.04501
Last Iteration	**2255.58**	**2168.30**	**2204.19**	**1318.713**

Table 5. Metrics for 1000 characters with all optimizations.

Overall, the proposed enhancements to MH gives incredible results at a fraction of the cost in terms of time and computation power. With more time, I would aim to explore non-symmetric proposal distributions that swap certain characters based on corresponding letter frequency, in addition to factoring in tri-gram probabilities to the initialization.