**Cryptography**

1. **Code Architecture**

The code implemented consists of a slightly modified version of the Markov chain Monte Carlo (McMC) method used to solve a substitution cipher. It is executed as follows:

First, a 3-adjacent character combination distribution for the English language is found by parsing a Wikipedia page1 for 3-adjacent character combinations. Each possible 3-character combination (there are 27^3) in the Wikipedia page is added as a key to a dictionary with a value corresponding to its frequency of occurrence in the analyzed text. All the values of the dictionary are then smoothed to eliminate math range errors arising from frequencies of zero.

Once the English character combination distribution dictionary (Eccdd) has been populated, the code is fed a randomly encrypted version of the phrase “Jack and Jill went up the hill to fetch a pail of water” (spaces are included before and after the string to improve word recognition).

The scrambled phrase is then scored by adding the natural logarithms of the frequency in the Eccdd corresponding to every 3-adjacent character combination in the phrase. The scores are stored as the addition of natural logarithms of frequencies but will be converted to a probability score later by computing Euler’s constant to the power of the natural score, which equates to multiplication of the dictionary frequencies.

The natural score is stored as the best score. The code then begins to randomly swap pairs of letters in the randomly generated cipher. If the resulting score is better, the program always accepts it as a better solution and saves the generated cipher as the new best cipher and the score as the new best score. If the score is worse, the program accepts the generated cipher as the new solution with a probability inversely proportional to the distance between the two scores. After converting scores to probabilities, the probability of a bad swap becomes:

If the program accepts a bad swap, the cipher that generated the better phrase is stored to a list of future ciphers to investigate before the current best cipher is switched to the new cipher.

Once the program completes an initial seeding run, it moves on to apply this same method (without the storing of future ciphers and possibly with a modified bad swap acceptance rate), on all the stored ciphers from the simulation just executed. Once all these ciphers have stabilized, they are applied to the original scrambled phrase and scored for accuracy against the Eccdd. The cipher with the highest score is saved as the final cipher.

1. **Motivation**

Imagine you are following a path through the woods and all you have is a compass that points towards the exit. At every fork in the road, you decide to pick one of the new paths at random and compare it to the one you are on, and then take the path that lies in the direction closest to where your compass points; however, if the two paths are extremely close, you will just randomly choose between the two. Now, while the compass method will probably help you get as close to the exit as possible without an actual map of the trail, it is not *guaranteed* to get you there, and it probably never will. There is no telling what type of twists and turns a path may take after you’ve chosen, and a path that looked like it was leading you toward the exit might find you at a dead end, or worse, deeper into the forest. In order to account for this possibility, it would be wise to carry a handful of seeds with you on your journey and place one at the entrance of all the paths you encounter that also look promising. That way, if you reach the end of the path and find that you are still nowhere near out of the woods, you can just retrace your steps instead of taking a helicopter back to the entrance of the forest and starting all over.

1. **Results**

An algorithm with this many steps provides many possible avenues for adjustment. Of all possible adjustments, the ones tested thoroughly were:

1. Varying the timeless annealing rate for both a seedless and a seeded McMC.
2. Varying the time dependent annealing rate for both a seedless and a seeded McMC.
3. Accepting only positive swaps

Here, the timeless annealing rate refers to a scale factor applied to the random probability of switching based on the distance between two scores. A timeless annealing rate less than one increases the probability that a bad swap will be taken at a given score separation. The time dependent annealing rate scales the probability of a bad swap based solely on simulation run time. Both techniques produce more bad swaps in the beginning and fewer toward the end, since finding a worse swap that is still close in score to the current cipher is unlikely the closer you get to the maximum score. However, if there do happen to be two phrases with similar high scores, the timeless annealing method is liable to switch between them forever, never settling at an equilibrium phrase but jumping around an equilibrium score. This endless random jumping gives our method the chance to propel itself away from a local maximum even after it has settled into it for a long time. Another important difference between the two methods is that if a bad swap is taken using the timeless annealing method, we know the substituted score is likely to be close to our replacement score.

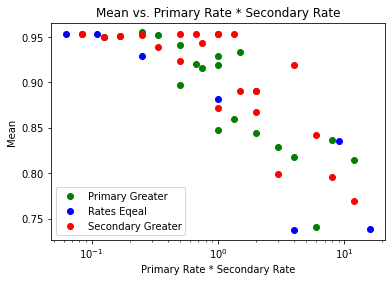
Chart, treemap chart

Description automatically generated

Figure 1 - Timeless Annealing Method Diagnostics

Figure 1 depicts the average deciphered phrase accuracy for 7 squared combinations of primary and secondary annealing rates. The figure shows that the system is very sensitive to both rates and performs best at low annealing rates. A surprising feature of this graph is that the variance for low annealing rate pairs is zero. Variances are based on differences in scores, not on the difference between actual phrases, Therefore, it is possible that a variance of zero could be obtained from different phrases if phrase scores are degenerate, meaning multiple phrases correspond to the same score. However, checking the actual output of each simulation verifies that each phrase is indeed the same, and there are likely no degenerate scores in our distribution. The invariant output phrase is not a perfect reconstruction of the input phrase “jack and jill went up the hill to fetch a pail of water,” but instead reads “jack and jill went us the hill to fetch a sail of water.” This result may appear discouraging at first. However, it is important to note that, with the scoring dictionary used, this version actually gets a better score than the correct version. Thus, by looking for the phrase with the highest score, we are dooming ourselves from the start.

The high accuracy and low variance of this model at low annealing rates is only made possible because the code involves cherry picking the best phrase while the seeded simulation is still running. The algorithm does not save the output phrase from each seed simulation, but rather the phrase encountered along our path that beats all other phrases on the path. Hence, it is not accurate to say that low annealing rates settle on the true root of the function every time, they just happen to stop by these roots on their journey. If we instead use the actual output phrase from each simulation, the final accuracy even at low annealing rates is around 73% with a variance of 3 points. This is likely because, while a low annealing rate gets you as close to the global maximum as possible, it is still prone to jump around similar scores toward the end, so we have the chance of ending our simulation after it has made a bad jump and scoring the wrong phrase.

Chart, scatter chart

Description automatically generated

Figure 2 - Mean vs. Combined Rates

With the simulations being divided into a primary and secondary process, each governed by its own independent rate, it is appropriate to question if the order of the rates plays an important role in determining final accuracy. As in, would it be better to plant the seeds more randomly than we grow the seeds, or vice versa? Figure 2 demonstrates the accuracy of each simulation in response to combined primary and secondary rates, with the data points being divided based on which is greater or whether they are equal. While it appears that having a lower secondary rate allows us to maintain maximum accuracy for a little longer, there is no clear distinction between the three groups.

Chart, scatter chart

Description automatically generated

Figure 3 - Initial success compared to seed success

Another question that must be considered regarding our distribution is whether the best accuracy achieved by the seeding run has any correlation with the output obtained from the seeds planted. Figure n demonstrates seeding run accuracy and corresponding best seed accuracy for several simulations. There appears to be no correlation between the success of a seeding run and the success of the seeds. Nevertheless, the fact that the plot contains points at an accuracy of 100% reveals that the algorithm *can* settle on our desired phrase. However, as the simulation progresses the algorithm tends to deviate towards the orange line, which corresponds to the accuracy of 95% that we invariantly achieve at low annealing rates.

Chart

Description automatically generated

Figure 4 - Time dependent annealing rate and accuracy

The accuracy of the output phrase is far less dependent on the secondary rate when the time dependent annealing rate is applied.

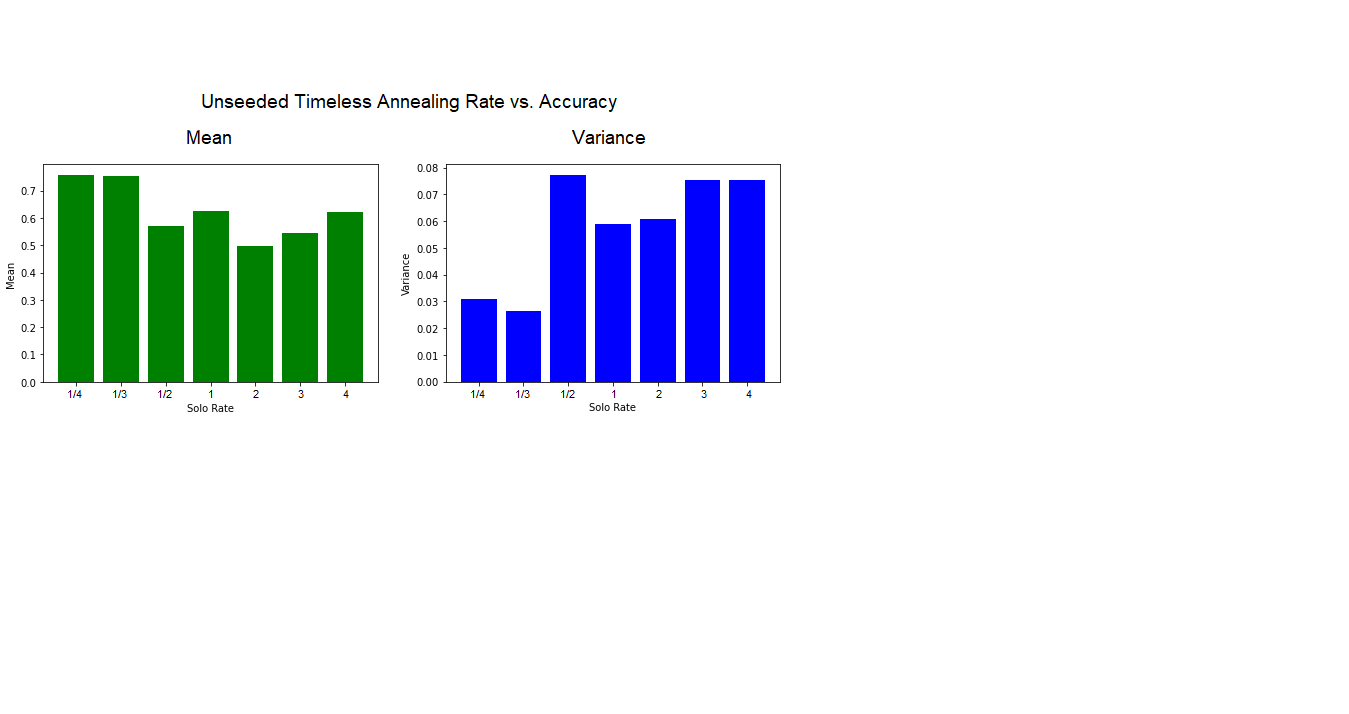


Figure 5 - Single run vs. Many runs

Unseeded simulations, which rely only on the output of the initial simulation, demonstrate a similar dependence on the timeless annealing rate. However, as we can see from Figure 2, the average accuracy of this method never surpasses 75%. Furthermore, even the least random simulations, corresponding to lower annealing rates, demonstrate far greater variance than the seeded method.

This single step version of the unseeded method, however, is not a suitable control for the performance of the seeded method. For every one simulation of the seeded method, we are computing multiple ciphers and picking the best out of all of them before adding it to our average. The fact that we are getting much more accurate results could just be a manifestation of the law of large numbers. The real comparison should be made against an application of the seeded method where the seeds are instead swapped out for randomly generated ciphers.

Chart

Description automatically generated

Figure 6 - Control annealing rate accuracies

Figure 6 shows the mean variance and accuracy for control simulations. The results are similar to the results obtained with the seeded method. Therefore, it does not appear that starting our seeds from ciphers gathered along an initial simulation provides any benefit over choosing a cipher at random, other than possibly a slight decrease in variance for larger primary rates.

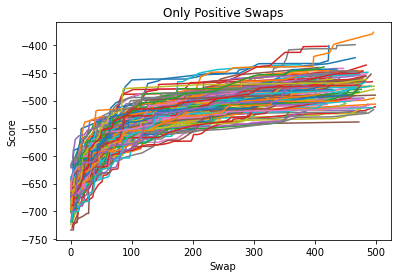


Figure 7 - Only positive swap trajectories

If only positive swaps are accepted, the seeded method cannot be applied in any meaningful way because it relies on randomness to generate the location for a seed. The average accuracy when only accepting positive swaps sits at around 20% with a 2-point variance. It is important to note that this average is not an appropriate comparison for the seeded method since this method only computes one phrase per simulation, unlike the seeded method and random replaced seeded method, both of which produce several phrases per simulation and choose the one with the best score.

1. **Corollary**

Another motivation for the seed approach was the apparent fractal qualities of deterministic root finding algorithms. Fractals like the one derived by applying the Newton-Raphson method to the complex plane arise because in proximity to a boundary between points that lead to different roots, there must lie points that lead to all other roots.

Diagram

Description automatically generated

Figure 8 - Sample Space Distribution

Figure 8 is a visualization of a sample space. Each point represents a value on the complex plane with a color corresponding to the root it settles on after repeated application of the Newton-Raphson method (For our problem the phase space is the space of all possible permutations of the input phrase generated by all possible permutations of the cipher. However, this is difficult to visualize). If we were to begin our application of the Newton-Raphson method on a random point drawn from ‘region a’, we would tend toward root e^(2\*π\*i/3) most of the time. However, if we start, instead, from a random guess within ‘region b’, we are much more likely to find all roots after application of the Newton-Raphson method to random points selected from within the region.

In our case the ‘roots’ of our cipher can be thought of as the output phrase that gets us the best score. There is no guarantee that there is just one maximum score for all possible permutations of a phrase. In fact, I could hand you a phrase that is apparently English and claim it is an encryption of some other English phrase, in which case there are at least two unique English solutions for this permutation of characters. The number of possible ‘roots’ for our cipher decreases as the length of the given encrypted phrase and the number of unique characters within it increase. Roots act as black holes for our guesses, and there are some regions within the space of possible permutations of our cipher that are inescapable on our recursive journey. These ‘black holes’ might be quite large, and, as we have seen in the Newton-Raphson example, they are not guaranteed to be uniformly distributed throughout the phase space.

Originally, I had hoped that placing seeds as you draw ever closer to a root would put you in proximity to one of these junctures leading to all possible roots. I made an error here. The only true ‘black hole’ in this permutation of characters is the phrase ‘jack and jill went us the hill to fetch a sail of water.’ This phrase generates the true minimum score for our function. I also made an error when I likened the seeded method and the randomly replaced seeded method to a deterministic root finding method. The only true deterministic method in this scenario would be a simulation with only positive swaps, and even then, we could have two of the exact same inputs lead to different outputs just from the random swaps generated. It would be better to say the positive swap method is the only method that guarantees you will get trapped in a local minimum. Every variation of the positive swap McMC method that involves randomness automatically includes small jumps around the permutation phase space that will eventually lead us to the true ‘root’ of the function at the right annealing rate and simulation length. Therefore, our phase space actually consists of one black hole surrounded by mere stars that tend to pull our guesses into orbit but, given enough randomness, can always be counted on to release them into the void. In order to decrypt this phrase using guaranteed minimum finding methods, we would have to catch our phrase while it happens to be orbiting the solution we are hoping for, which is unlikely. Maybe the solution to this problem boils down to making our training data say ‘pail’ a lot more or adding a filter that stops the simulation exactly when it outputs the score corresponding to the correct phrase. But both these steps require more intimate knowledge of what the encrypted phrase is talking about, which defeats the point.

1. **Our Phase Space**

I mentioned the difficulty of visualizing the phase space of this problem. That is, the space of all possible permutations of the input phrase. This space is so difficult to imagine because the dimension of the problem is not related to a spatial dimension. We can, however, contrive a way to make this problem fit into 3-dimensional space. The function acting on our phase space involves breaking the phrase up into all 3-adjacent letter combinations. If we assign each letter in our cipher to a number and the position of the letters within the 3-letter chunks to spatial dimensions, we can construct a 3-dimensional spatial distribution with a 4th dimension of weighted occurrence at each of these spatial coordinates. This is basically what we do when constructing the original scoring dictionary.

A picture containing graphical user interface

Description automatically generated

Figure 9 - Cipher Sample Space

Figure 4 depicts a snapshot of our phase space if the 4th dimension corresponding to character frequency was manifested as heat. The whole phase space for the problem consists of every snapshot that could be taken of this distribution based on permutations of the cipher.

1. **Orbit Tracking**

One step we *can* take with the knowledge that our guesses tend to orbit certain phrases is keep track of how long they remain in orbit around these phrases. That way, when we conclude our simulation, we do not just see where our guess happened to be by the final iteration, but instead where it spent most of its time. This process was implemented by running an initial McMC simulation to get our guess as close to the local minimum as possible, then feeding this guess into a second McMC simulation with the same annealing rate that documents each time our guess returns to a given phrase.

Console Output

phrase: jack and jill went us the hill to fetch a sail of water

frequency: 0.8918918918918919

score: -348.7226368776733

phrase: jack and jill went up the hill to fetch a pail of water

frequency: 0.03783783783783784

score: -348.8412243032847

phrase: jack and jill went us the hill to fetch a sail of watem

frequency: 0.010810810810810811

score: -355.0727391919332

The result of an orbit tracking simulation is shown in the example console output. As you can see, our guess spends most of its time around the phrase that is incorrect but still the true global maximum (or at least a local maximum). Our guess spends the second greatest amount of time around the correct solution, which only amounts to 3% of the time. This makes sense considering that the computer is guaranteed to accept a swap to a better score but only *likely* to accept a swap to a worse but similar score. The inspiration for implementing this process came from the results demonstrated in ‘[visual.mp4](visual0001-2000.mp4).’ This video shows in real time how our guess changes throughout the simulation. While the phrase never stops changing due to the low annealing rate necessary to push it to the global maximum, it is still possible to read the words ‘jack and jill went up the hill to fetch a pail of water’ flashing on the screen, which led me to believe that on average our guess remains at the correct solution.

Chart, scatter chart

Description automatically generated

Figure 10 - annealing rate vs. orbiting scores

Figure 10 shows all the scores our guesses were switching between by by the end of a single simulation for a range of annealing rates. The alpha value of each point corresponds to the proportion of time spent at that guess out of 1000 swaps. A negative annealing rate means we are running the simulation in reverse. As you can see from the plot, low annealing rates cause more random jumping between guesses by the end of the simulation, which creates more dots at corresponding annealing rates. It makes sense that our plot would follow a logarithmic distribution since we are jumping between scores based on a scale factor taken as Euler’s constant to the power of something. The initial increase in score along a single path until a rate of about two tenths represents a ‘one step forward two steps back type’ situation where our guesses cannot exceed a maximum score threshold because we are taking too many bad swaps. Beyond this rate, two distinct plateaus seem to emerge which represent prominent maxima our simulation is capable of being stuck at by the end of the simulation. Originally, I expected this plot to result in a kind of reverse bifurcation diagram, since at low annealing rates there would be more orbited points. However, it appears the decrease in orbited roots at higher annealing rates is balanced by an increase in possible roots to settle on, which makes the plot fuzzy even toward the right end.

1. **Future Modifications**

Another possible modification that could be made to our algorithm involves the exploitation of loops within the generating cipher. An interesting ‘paradox’ in probability theory reads as follows:

Say you are given 26 CD cases with their contents scrambled and are asked to return each CD to its proper case. You are only allowed to open 13 CD cases when looking for a given CD, and you have short term memory problems, so you forget what CD was in a case after you close it. At first, it would appear our probability of success is one half to the power of 26, since there are 26 CD cases to fill but we only get to check half of them every time. That is one out of every 67,108,864. However, we can increase our chances of success to around one third if we always check the case that corresponds to the CD we are currently looking for first, then, if the correct CD isn’t contained within that case, we check the case belonging to the CD that *is* in that case, and repeat this process until we find each CD. This remarkable increase in probability is thanks to the fact that, given a set of 26 CDs randomly scrambled, there is only a 69% chance that the scrambling produced a loop of cases to CDs greater than 13.

Chart

Description automatically generated

Figure 11 - Demonstration of cipher loops and probabilities

Admittedly, I am unsure exactly how we would go about applying this knowledge to our ciphers. Nevertheless, the problems appear similar. Finding the letter each letter is mapped to in the substitution cipher can be thought of as returning all the CDs to their cases. The original cipher contains a fixed number of letter loops (for the loop calculations ‘a’ mapped to ‘g’ , ‘g’ mapped to ‘p’, then ‘p’ mapped back to ‘a’ is considered a loop of length three) and for a randomly generated cipher there is a fixed probability that any loop within this cipher will exceed a given length. When we propose a new cipher, we are proposing to either join two loops into one or divide one into two. It might be beneficial to consider this in the swapping strategy. At the very least, we may be able to reject or accept some ciphers on the basis that they contain an unlikely number of loops and could be man-made. For our purposes, the ciphers are randomly generated, so they should fit this distribution.

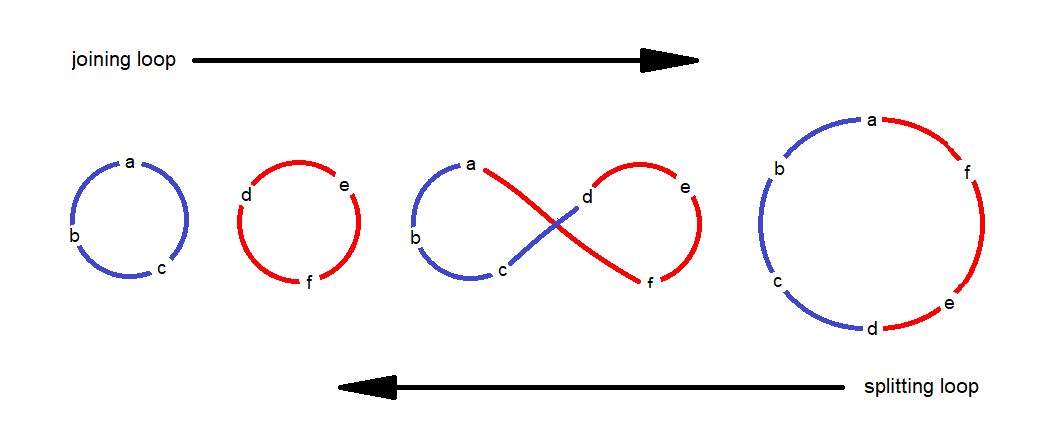


Figure 12 - Swaps generate a new loop configuration

Surprisingly, our original McMC method seems capable of indicating the number of loops contained within the original cipher even if it cannot decrypt the cipher. A randomly generated cipher contains five loops more often than any other number, as seen by the peak in figure 12. However, for ciphers generated by applying the McMC method to an encrypted phrase generated by a cipher with a statistically unlikely number of loops, this distribution shifts to the left or right, depending on if the number of loops is less or greater than, respectively, the expected number of loops for a randomly generated cipher. We may be able to use the difference between these two distributions to estimate how many loops are contained within the encryption cipher and only search ciphers that fall in this range.

Chart, histogram

Description automatically generated

Figure 13 – number of loops in 1000 random ciphers

Chart, line chart

Description automatically generated

Figure 14 - Encryption ciphers shift our distribution

Figure 14 shows the distribution of the number of loops in the output ciphers from 1000 McMC simulations for an initial encryption cipher containing specific loop counts, denoted by color. For our purposes, a cipher consisting of a single loop is impossible since ‘space’ is always mapped to itself. Note that for encryption ciphers containing a statistically unlikely number of loops, most of our guessed decryption ciphers still fall within the statistically allowed range of loops, so encryption ciphers containing an unlikely number of loops are almost completely immune to our decryption method if only the McMC method is used.

1. **The Wrong Thing to Say**

The most incorrect thing you could say in English is ‘kboi bxh ksuu zmxq vc qpm psuu qg lmqop b cbsu gl zbqmf.’ This is the result obtained by running the McMC method in reverse on this permutation of characters.

Chart, histogram

Description automatically generatedChart, histogram

Description automatically generated

Figure 15 - Worst scores at various swaps for several simulations

Unlike for the maximum score, there are many degenerate minimum scores. Or at least, there are many solutions extremely close to the minimum. Figure 15 shows the score of several simulations at every swap. While it appears all these phrases settle on one value, plus or minus a few tenths, they read very different things, which makes determining the true global minimum difficult.

**Works Cited**

Wikimedia Foundation. (2022, October 7). Jack and Jill. Wikipedia. Retrieved November 15, 2022, from <https://en.wikipedia.org/wiki/Jack_and_Jill>

Agarwal, R. (n.d.). Applications of MCMC for cryptography and Optimization. Retrieved November 16, 2022, from <https://towardsdatascience.com/applications-of-mcmc-for-cryptography-and-optimization-1f99222b7132>

1veritasium. (2022, June 30). The riddle that seems impossible even if you know the answer. YouTube. Retrieved November 15, 2022, from <https://www.youtube.com/watch?v=iSNsgj1OCLA>

YouTube. (2021, October 12). Newton's method, and the fractal it creates that Newton knew nothing about. YouTube. Retrieved November 15, 2022, from <https://www.youtube.com/watch?v=-RdOwhmqP5s>