***Abstract***

An attempt merge the work done by Gillogy[1] and Weierud[2] and to improve completeness of ciphertext-only attacks on enigma.

**1 Introduction**

Following writing my own enigma machine to crack down a rather amusing story of my university tutors using known-plaintext-attacks, and a significant amount of help with the enigma settings, I wanted to look into how ciphertext-only attacks had been devised to crack enigma codes. We are now in an age where we have sufficient computing power to launch such attacks on an enigma ciphertext, and as high-level languages allow less hassle in details such as registries, it has become a lot more open to even undergraduate students such as myself to write my own attacks.

In doing so, I initially tried to reproduce some research conducted by Gillogy before the turn of the millennium. But before I could complete this research I spotted something about the research that lead me to wonder whether I could improve the completeness of such techniques. Primarily, the fitness of the rotor settings. More specifically, how the fitness values are used to proceed with the algorithm.

**2 Gillogy's Rotor and Position Algorithm**

James Gillogy was one of the first (Gillogy, 1995) to theorise that modern computational power had the capabilities to crack enigma codes with knowledge only of the ciphertext and the complete list of settings in an encryption machine (in this case, enigma). It was based mainly on the weakness of the plugboard, that even with the plugs, successfully cracking the rotor settings would result in some of the plaintext being revealed.

He discovered that the natural-language fitness function, Index of Coincidence<sup>[3]</sup> was able to give a very good estimation to how well a particular setting had decrypted the ciphertext. The higher the index of coincidence, the better chance that setting has of being the one that was used in the original encryption.

Using this fitness function, we can use the raw computational power of modern computers to brute force each and every combination of rotors, along with their starting positions, and give a fitness value for each one.

However, this is **very** time-complex. Too time-complex, even with a standard laptop, to be done in a "reasonable" amount of time.

It took my laptop 1263.75 seconds to run through each permutation of rotors and settings.

This is because the algorithm is assessing 60 (permutations of 3 rotors) x 263 (each position permutation) iterations, and calculating the fitness of each and every one of them.

At the end of the brute-force approach, we are able to assess the "fittest" settings and go from there.

**3 Gillogy's Ring Setting Algorithm**

Having found the fittest rotors and their starting positions, we still have to tackle the problem of the ring settings. Gillogy noticed that the farthest left rotor (known as the "slowest rotor") doesn't have much any effect on the encryption with a different ring setting for even up to 500 characters.

Gillogy also found that calculating the fitness of each of the other two rotor's combinations wasn't necessary. This is very helpful at reducing the number of iterations down from 262 (combinations of rotor settings) to 2 (number of rotors) x 26 (rotor settings for each rotor).

We can use this knowledge to "optimise" each of the right-most rotors to their fittest setting, assessed by passing through a fitness function. Gillogy continued to use the Index of Coincidence, however perhaps a better function would be one suggested by Weierud.

**4 Weierud's Bigram Fitness Function**

It was suggested that a better fitness function for assessing the ring settings (and by extension, the later discussed plugboard search) would be to use the Bigram Fitness Function.

Given a comprehensive list of bigrams, along with their English fitness score, we are able to devise a fitness function that searches through each encryption and give it a total fitness score. Common bigrams such as "th", "ch", "it", "an", "he" can help us determine which encryptions are closest to what we're searching for.

This method is especially powerful at this stage of the decryption, because the only subsequent stage is the plugboard settings which swap out characters. For example, there is a high chance that the letters "I" and "T" aren't involved in the plugboard and so would show up as they are in the plaintext at this stage, allowing our fitness function to gauge with a high level of accuracy that this text is indeed the closest to the plaintext and hence the settings that were used originally.

**5 Finding the Plugboard Settings**

Finally, having found the fittest settings, we need to decipher the plugboard settings. Gillogy said that, if we have the correct rotor settings, some of the plaintext would be decrypted just fine (as not all letters have a plug attached to them).

With this knowledge, we know that we can simply use trial and error against a fitness function, and as they're independent we can apply a simple hill-climbing algorithm to slowly move towards the correct result.

The issue with this technique is that it is always unknown exactly how many plugs were used, and where to stop adding and removing, or swapping plugs from the board. Hence, our fitness function is probably our best bet.

It is written by Weierud that, similar to how bigrams should be used in rotor setting analysis, we should this time use trigram analysis to find common letters. If we've got a correct plug, we're more likely to have revealed a correct trigram than have revealed a correct bigram.

Putting all these methods together, we are able to make a very good guess at what the rotors may be. In fact, when passing in a ciphertext from Turing's Imitation Game paper<sup>[4]</sup> (Turing et al. 1950) we get the following plaintext out:

JBROPOSETOCONSIDERTHEQUESTPHNCANMACHINESTHINK...

This, for all intents and purposes of cracking a code, is sufficient for someone to recognise the contents of the message. However, this leaves a lot to be desired. It isn't accurate, and certainly lacks completeness.

**6 Rotor and Positions Analysis**

We use the Index of Coincidence to assess the fitness of rotor settings and starting positions, and this leads to the majority of inaccuracies observed by Gillogy. When putting this algorithm through my own algorithm, the following was observed:

Actual Encryption: II V III, J H W, 00 00 14, ["AR", "DM"]

Rotors Index of Coincidence Starting Positions

['IV II V', 0.026243182184691613, 'M D Q'],

['II IV III', 0.026219725557461403, 'B A E'],

['I II III', 0.02621386140065385, 'N E P'],

['V III IV', 0.02621386140065385, 'C S X'],

['II V III', 0.02620213308703874, 'M V L'] ]

As observed, we can see that the highest index of coincidence was for the rotor setting `IV II V` which is incorrect. However, down in 5th position, we can see the correct permutation.

In fact, let's observe the number of times each rotor appears in the top 5 fittest permutations:

I: 1

II: 4

III: 4

IV: 3

V: 3

Observe that the correct permutation appears the most in the top 5 fittest permutations.

I propose that, instead of observing the highest performing settings, we observe all permutations of rotors that appear the most in the top X fittest rotors.

I suggest that, as X increases, the completeness of our encryption will also increase, at the cost of time.

**7 Permutation Analysis**

We can write code that produces every permutation of the 3 rotors that appear the most in the top 5 fittest rotor settings. In this case:

II: 4

III: 4

IV / V: 3

From the 60 permutations that arise, we can rerun our original method for assessing every position and assessing the fitness function. We can observe the following results:

Actual Encryption: II V III, J H W, 00 00 14, ["AR", "DM"]

Rotors Index of coincidence Starting Positions

['II V III', 0.026219725557461403, 'M V L'],

['II III V', 0.026190404773423638, 'H R D'],

['II III V', 0.02613176320534811, 'L S I'],

['III II V', 0.026114170734925446, 'K R N'],

['II V III', 0.02610830657811789, 'J H W'] ]

Notice here how the correct rotor settings have bubbled to the top of the fitness scores, improving the accuracy from Gillogy's method.

**8 Completing Wiereud's Method**

Passing these new values through the existing bigram and trigram method devised by Wiereud, we can observe the following results:

Ring Setting Analysis with Bigram Fitness Function

Rotors Starting Positions Ring Settings Bigram Fitness

['II V III', 'J H W', [0, 14], -1613.469756054],

['II III V', 'H R D', [16, 12], -1609.3527085879996],

['II V III', 'M V L', [13, 7], -1588.9218158320002],

['III II V', 'K R N', [16, 16], -1584.0361097100006],

['II III V', 'L S I', [23, 20], -1541.8206898830003] ]

Plugboard Analysis with Trigram Fitness Function

We know that the encryption used only 2 plugs.

Rotors Starting Positions Ring Settings Plugboard

['II V III', 'J H W', '00 00 14', ['AR', 'DM']],

['II III V', 'H R D', '00 16 12', ['RN', 'AP']],

['II V III', 'M V L', '00 13 07', ['AL', 'AC']],

['III II V', 'K R N', '00 16 16', ['NR', 'GY']],

['II III V', 'L S I', '00 23 20', ['UO', 'AN']] ]

Final Analysis of Enigma Settings

Rotors Starting Positions Ring Settings Plugboard

II V III J H W 0 0 14 AR DM

Decryption:

IPROPOSETOCONSIDERTHEQUESTIONCANMACHINESTHINKTHISSHOULDBEGINWITHDEFINITIONSOFTHEMEANINGOFTHETERMSMACHINEANDTHINKTHEDEFINITIONSMIGHTBEFRAMEDSOASTOREFLECTSOFARASPOSSIBLETHENORMALUSEOFTHEWORDSBUTTHISATTITUDEISDANGEROUSIFTHEMEANINGOFTHEWORDSMACHINEANDTHINKARETOBEFOUNDBYEXAMININGHOWTHEYARECOMMONLYUSEDITISDIFFICULTTOESCAPETHECONCLUSIONTHATTHEMEANINGANDTHEANSWERTOTHEQUESTIONCANMACHINESTHINKISTOBESOUGHTINASTATISTICALSURVEYSUCHASAGALLUPPOLLBUTTHISISABSURDINSTEADOFATTEMPTINGSUCHADEFINITIONISHALLREPLACETHEQUESTIONBYANOTHERWHICHISCLOSELYRELATEDTOITANDISEXPRESSEDINRELATIVELYUNAMBIGUOUSWORDS

**9 Conclusion**

We have improved upon the method laid out by Gillogy and improved upon by Wiereud by assessing not just the highest performing settings as observed through a brute-force method, but by analysing which rotors performed the highest in general.

If we take the top x performing settings, we can observe the rotors that appeared the most in the top performing ones. This allows us to take the top 3 rotors (provided we know the encryption was done with 3 rotors) and assess again all permutations of those rotors.

We have observed that the highest performing setting under these new restrictions has a higher chance of being the correct setting, and hence we can move through the method with a higher level of completeness.

Going through just 3 rotors however, means we must assess 60 permutations of rotors and all their 263 possible starting positions. So this method comes at a time cost.

I hope that this finding allows us to think more broadly about the success of a particular setting, and not just jump at assessing the fittest setting defined by a particular fitness function. Instead, to observe the best performing settings in general, and work from there.

***Notes***

All written code can be observed in `advanced\_work.py` file and `advanced\_helpers` directory.

Data for the bigram and trigram fitness analysis can be observed in `data` directory.

To run this method yourself, open this repository's directory and run

python advanced\_work.py

***Acknowledgements***

Mike Pound, Department of Computer Science, University of Nottingham.

For providing me with a test encryption of the introduction of Turing's paper to launch an independent ciphertext-only attack that differed from the ones used in reference papers to ensure accuracy of my findings.

Also for providing me with bigram and trigram data and their fitness values.

***References***

1. James J. Gillogly (1995) CIPHERTEXT-ONLY CRYPTANALYSIS OF ENIGMA, CRYPTOLOGIA, 19:4, 405-413, DOI: 10.1080/0161-119591884060

2. Olaf Ostwald & Frode Weierud (2017) Modern breaking of Enigma ciphertexts, Cryptologia, 41:5, 395-421, DOI: 10.1080/01611194.2016.1238423

3. Kontou, Eleni. "Index Of Coincidence." Leicester Undergraduate Mathematical Journal 2 (2020).

4. Turing, A.M. (1950) Computing machinery and intelligence. Mind, 236, 433-460.