CRYPTOLOGIA
https://doi.org/10.1080/01611194.2020.1721617

OPEN ACCESS

Genetic algorithm attack on Enigma's plugboard

Åvald Åslaugson Sommervoll and Leif Nilsen

ABSTRACT
The history, operating principles, strengths, and weaknesses, of the German cipher machine Enigma, have been widely studied for almost 50 years. Even though Bletchley Park regularly broke Enigma encrypted traffic during World War II, new pieces of information and fresh analysis are still aggregated to the remarkable "puzzle" called Enigma. This paper shows that Enigma's plugboard is vulnerable to Genetic Algorithm (GA) attacks, which solves Enigma's plugboard faster than earlier published ciphertext-only techniques. The Genetic Algorithm does this using the counting measure *Index of Coincidence (IC)*. Independently of the GA, but related to the analysis, we introduce a new measure *Progress Index of Coincidence (PIC)*. PIC is a measure of the relative progress in decryption between the ciphertext and plaintext measured by IC.

KEYWORDS
Enigma; Enigma plugboard; genetic algorithm; index of coincidence

1. Introduction

The Enigma Machine represents a milestone in the history of cryptography. The machine combines the rotor system, invented by two Dutch navy officers in 1915 (de Leeuw 2003), with a plugboard; resulting in a cipher so advanced that it was thought to be unbreakable (Copeland 2004). Enigma's strength, mobility, and user-friendliness allowed its widespread use by the German military during the Second World War. Its importance in the war and cryptanalysis made the Enigma perhaps the most famous cryptographic machine in history. Its fame is also reflected in modern textbooks, for example, in Paar- and Pelzl's "Understanding Cryptography," where the Enigma is used to illustrate a classical encryption machine (Paar and Pelzl 2009). The machine has even had books and movies centered around it and its cryptanalysis, with perhaps the most recent release of "The Imitation Game" on the 25th of December 2014 (IMDb 2014).

The Enigma represents a special form of a *polyalphabetic substitution cipher* and cannot, by any means, be considered to provide secure

CONTACT Åvald Åslaugson Sommervoll aavalds@ifi.uio.no Department of Informatics, Faculty of Mathematics and Natural Sciences, University of Oslo, Sem Saelands vei 24, Oslo, 0371 Norway.

encryption by modern standards.[1] Building on the pre-WWII analysis of Polish mathematicians, a huge effort by English and American scientists developed cryptanalytical tools and methods that could break German Enigma traffic daily (Singh 2000). Significant members of this activity included the classical scholar Dennis Knox, mathematicians from Oxford and Cambridge like Peter Twinn, Alan Turing, and Gordon Welchman, as well as the international chess masters Hugh Alexander and Stuart Milner-Barry.

However, even if the Enigma represents an outdated crypto technology, it still inspires researchers to fill gaps in the Enigma history and to improve on Enigma cryptanalysis. The purpose of such research is twofold, to develop modern cryptanalysis or to attack unread authentic traffic from WWII. One recent example is the paper by Ostwald and Weierud (Ostwald and Weierud 2017), who, in 2017, released "Modern breaking of Enigma ciphertexts" in Cryptologia. It is to be anticipated that new analysis for breaking the Enigma could apply to other ciphers that build their security on the same principles. For this reason, decryption techniques that prove effective on Enigma encryption may also prove effective on other encryption techniques as well. If not by themselves, they may provide useful building blocks for future crypto-attacks. This paper aims to provide one such building block in the form of a ciphertext-only attack based on genetic algorithms (GA). The proposed GA attack is faster than earlier ciphertext-only attacks. We also build upon the existing measure Index of Coincidence, creating a more human-readable representation of the measure which we call Progress Index of Coincidence.

The remaining paper is organized as follows. Section 2 provides background information on the construction and operation principles of Enigma. Then, the Genetic Algorithm is described. The measure, Index of Coincidence, is defined and explained. The box plot variant, notch plot, is also described and defined. The section finishes with a brief review of related research. In Section 3, the different settings of the Enigma are analyzed, and the vulnerability in the plugboard is outlined. The first attempts to use genetic algorithms for this task required unrealistic long pieces of ciphertext, but it is shown that the technique can also succeed for much shorter messages.

2. Background

2.1. Properties of the enigma

The Enigma is a portable encryption machine that was mainly used for battlefield communications and to protect tactical links. Physically the

[1]**Polyalphabethic substitution ciphers** are substitution ciphers that utilize multiple letter mappings, in that the substitution depends on a changing state.

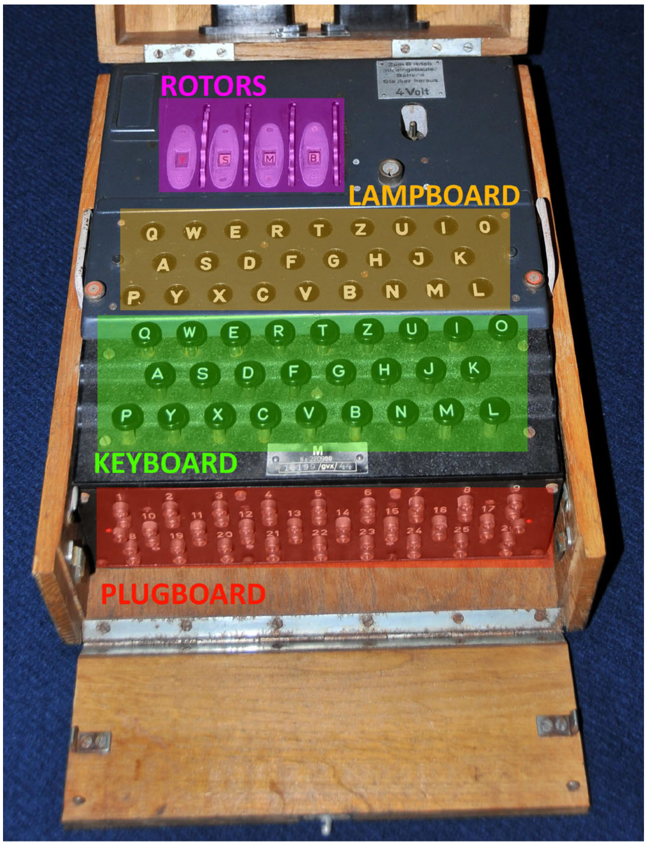


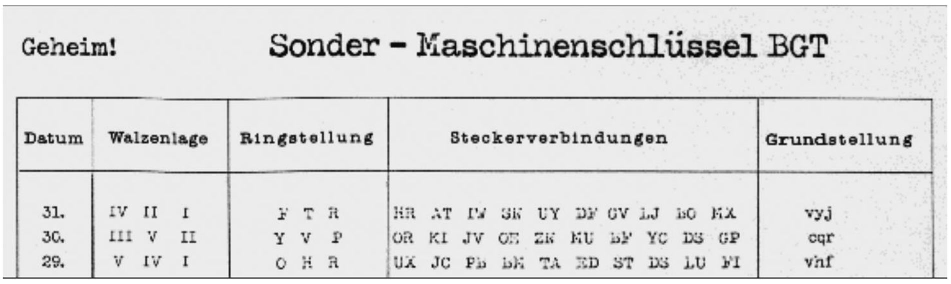
Figure 1. The four main components of the Enigma: Four rotors, a lampboard, a keyboard and a plugboard.
(Photo taken by Leif Nilsen edit by Åvald Sommervoll)

Enigma Machine was embedded in a wooden or metallic box, consisting of four main components (highlighted in Figure 1):

1. *Three rotors* (four after 1941 in the German navy). (The display values are visible in the windows next to the outer disks.)[2]
2. A *lampboard* with 26 lamps, one for each letter in the Latin alphabet.
3. A *keyboard* with 26 buttons, one for each letter in the Latin alphabet.
4. A *plugboard* also called a steckerboard with 26 connector points.

The rotors and the plugboard shown in Figure 1 define the *state* of the Enigma. This state consists of four parts: 1. rotor selection and order, 2. ring settings, 3. display values and 4. plugboard settings. We say that *rotor settings* are defined by the first three, and the plugboard's settings are

[2]Traditionally this was given by three letters. However, some Enigmas used numbers instead of letters, and as is shown in fig. 1, four-rotor Enigmas used four letters (YSMB).



Geheim! Sonder - Maschinenschlüssel BGT

Datum	Walzenlage	Ringstellung	Steckerverbindungen	Grundstellung
31.	IV II I	F T R	HR AT IW SK UY DF GV LJ BO MX	vyj
30.	III V II	Y V P	OR KI JV OE ZN MU BF YC DS GP	cqr
29.	V IV I	O H R	UX JC PB BM TA ED ST DS LU FI	vhf

Figure 2. Enigma key book.
Photo from authentic German codebook. (From before September 1938 as it has a "Grundstellung")
"Datum": Date, "Walzenlage": Rotor selection and order, "Ringstellung": Ring settings, "Steckerbindungen": Plugboard settings, "Grundstellung": Basic setting (Daily initial display value).
Image from The Late Tony Sale's Codes and Ciphers Website (Tony Sale 2001).

defined by the last. The union of these settings is referred to as the *key*. It defines the starting point for the encryption of a message. Due to the reciprocal characteristic of the Enigma, the same starting point is used for decryption and encryption. After the state of the Enigma is set, the keyboard is used for input, and the corresponding output is read off from the lampboard. Of the four parts that make up the state, all but one remains constant during encryption and decryption, the display value. Since the display value changes for every letter pressed, we introduce two additional terms when it comes to talking about the display values: *basic setting* and *message setting*.[3] The *basic setting* gives the daily initial display value, and the *message setting* gives the display value used at the start of the message.

In practice, there was typically one operator and one assistant that handled encryption and decryption. If they wanted to encrypt a message, the operator would type the message into the keyboard letter by letter (Copeland 2004). For each letter pressed, one of the 26 lamps would light up on the lampboard. The resulting sequence of lit letters was noted by the assistant. The noted sequence would then be the ciphertext. For every letter pressed, the display value would change, changing Enigma's state. Therefore if the operator presses the same letter twice, it will most likely be encrypted as two different letters. This is to make the Enigma robust against some of the most common cryptanalytic attacks, such as *frequency analysis* which was described as early as the 9th century (Singh 2000). The above applies to decryption also as Enigma is a reciprocal symmetric-key[4] encryption technique.

[3]Also called *message key* by Gillogly (1995) and *text setting* by Welchman (1982).

[4]Symmetric- key encryption means that encryption and decryption use the same key. Reciprocal means that encryption and decryption is the same mathematical operation.

Before decryption or encryption, however, the operator must set the Enigma machine's state. This state must be agreed upon between the two or more communicating parties before the communication can take place. During the Second World War, this was generally done by the distribution of pre-shared codebooks which provided a different setting for each day. Figure 2 shows a scanning of a page in such a codebook. Here "Datum" gives the actual date for the use of this key. "Walzenlage" gives the selection and order of the three rotors out of a total of five rotors (8 rotors for the naval Enigma). Before the selected rotors were placed in the machine, the ring setting of each rotor was set, given by "Ringstellung" in Figure 2. The next entry in the codebook is the plugboard setting, "Steckerbindungen" which is set by adding plug-connections between two different characters in the Latin alphabet in a one-to-one connection. Typically 10 plugs were used, leaving 6 characters without any connection in the plugboard (A plugboard with 0 plugs connected means that each letter is connected by default to itself, which is shown in Figure 1). The final setting listed is the "Grundstellung" which roughly translates to *basic setting*, and gives the daily initial *display value*. It is initial since the Germans always broadcasted some specified changes to the daily Enigma settings at the beginning of the message. Perhaps most famously is the double indicator operational procedure used by the Germans up to September 1938 (Lasry, Kopal, and Wacker 2019).[5] The first 6 letters of the message would contain the *message setting*, by encrypting the new display value twice. For example, if the *message setting* was to be "RCM," then "RCMRCM" would be encrypted from the *basic setting* given by the codebook. Then the operator would change the display value to "RCM" and encrypt the rest of the message. This procedure was done to ensure that different messages were encrypted from different starting points and thus protecting against well-known attacks on polyalphabetic substitution cipher. Note that the codebook lists the keys in "opposite" order, with the latest date at the top and earlier dates at the bottom. As a result, it was easy to remove and securely destroy keys from past dates.

2.2. The inner workings of the enigma

2.2.1. Electrical coupling

The Enigma uses an electrical current, traveling through a circuit to light up the correct lamp on the lampboard. Figure 3 shows a simplified version of the inner workings of the Enigma, with a plugboard, rotors, keyboard, and lampboard. Note that before the "A" (item 2) is pressed on the

[5]There are, of course, other double indicator operational procedures used by the Germans. During the war, the procedures would often not only vary over time but also across different groups.

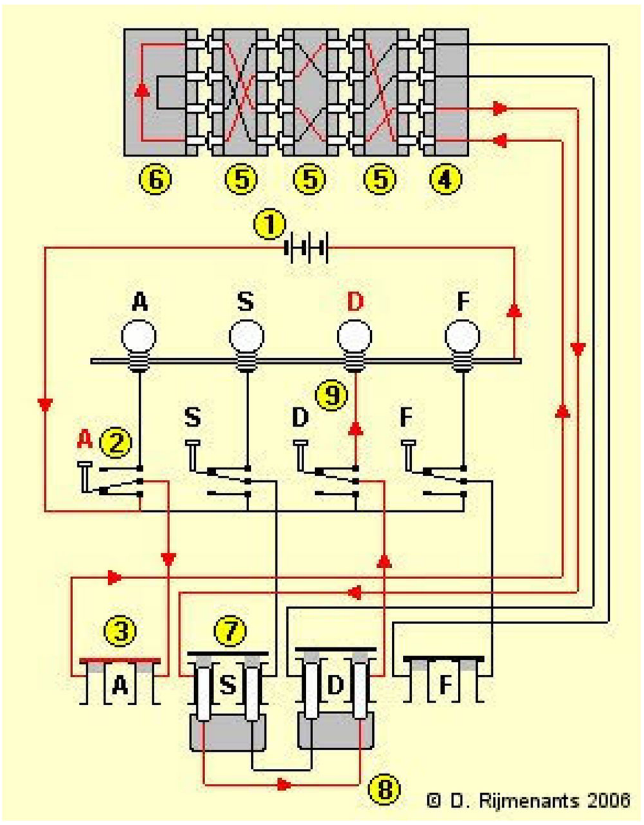


Figure 3. Enigma example wiring.
1. We have a battery; it provides electricity for the lamps.
2. Shows the letter pressed. In this example, "A" is pressed, which lets the current from the battery in 1 enter the circuit as shown by the red lines.
3. Since A is not steckered to any other letter the signal/current continues to the rotors.
4. The current enters through the A position in the entry ring.
5. The current is scrambled in the rotors.
6. Then the signal is reflected in a reflector sending the current back through the rotors.
7. The current arrives at S, but because the circuit going out to S is broken by a stecker the current continues to the steckered letter D.
8. From D the current continues up to the lampboard
9. Arriving at the lampboard the letter D lights up, encrypting A to D.

Image credits to Dirk Rijmenants.

keyboard, the electrical circuit is disconnected, and no lamp would light up. Then when "A" is pressed the circuit is complete and the current can travel from the battery to the plugboard (3), the entry ring (4), the rightmost rotor (5), the middle rotor (5), the leftmost (5), the reflector (6), the leftmost rotor again (5), the middle rotor again (5), the rightmost rotor again (5), the plugboard again (7 and 8), until finally reaching the lampboard (9). From this, it is clear that the encryption goes through the plugboard and each rotor twice, once on the way in and once again on the way out. Because of this, a small change in the plugboard or the rotors may

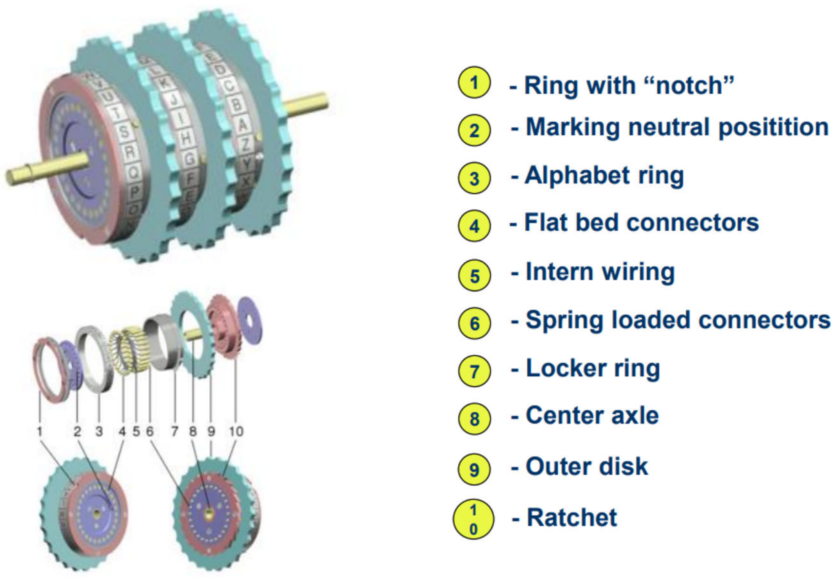


Figure 4. Enigma rotor diagram. Created by Wapcaplet in Blender. [CC BY-SA 3.0 (http://creativecommons.org/licenses/by-sa/3.0/)]

result in a large change since almost no matter where the change is, it will be applied twice and go through further changes in the other encryption components. Also note that if "D" was pressed instead of "A," the circuit would be the same, however "A" would light up instead of "D." This is an important characteristic of the Enigma encryption machine and explains why the encryption and the decryption settings are the same.

2.2.2. Rotors in detail

The rotor setting in the army Enigma is a selection of three rotors among five *I, II, III, IV*, and *V*, and is typically written in order. For example: *IV II I*, means rotor *I, II* and *IV* were selected and *IV, II* and *I* are the leftmost, middle and rightmost rotors respectively. Each of the individual rotors contains a 26 to 26 rewiring of 26 potential inputs, one for each letter in the English language, as shown in Figure 4 as item 5, internal wiring.

The wiring is constant; however, its position in relation to the alphabet ring and notch (item 1 and 3) is not constant but is defined by the *ring setting*, which is set with the locker ring, item 7 in Figure 4, locking the wiring in the specified position. The ring setting is set prior to the insertion of the rotor into the Enigma. The *display value* on the other hand can be changed after inserting the rotor into the Enigma, and is set with the *outer disk* (item 9). The current *display value* is shown in a small window next to its respective rotor and is given by a single letter on the alphabet ring (item 3). For every keypress, the rightmost rotor takes a single step,

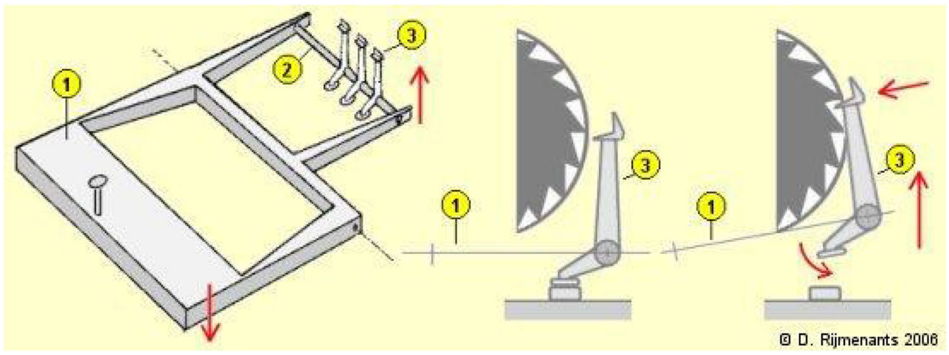


Figure 5. Mechanical setup of the Enigma Machine.
From the figure we observe that when a key is pressed on the keyboard (marked with 1), it acts as a jack pushing up on the ratchet teeth of the rightmost rotor, moving it one step.
Not shown in the final two pictures is how the middle and leftmost rotor is moved. They are only moved at a specific index determined by the ring with "notch" market with 1 in Figure 4.
Image credit to Dirk Rijmenants.

changing the display value as mentioned in Section 2.1. This is because the ratchet teeth, item 10 in Figure 4, are engaged for every keypress. Figure 5 shows this more in-depth, how the pressed key is used to nudge the rotor one step further. Only the rightmost rotor is engaged for every keypress. The middle rotor and the leftmost rotor are only engaged when the corresponding pawl aligns with the notch of the rotor to the right, item 1, the ring with "notch" in Figure 4. The display value for the rotor determines the position of this "notch." Different rotors have different locations for the notch. For example, rotors I and IV would step their neighbor rotor at display value "Q" and "J" respectively. If the rightmost rotor is IV, the middle rotor will take a step whenever the display value passed "J," like an odometer. Furthermore, since there is no rotor to the left of the leftmost rotor, the completion of one full cycle by this rotor has no effect. This means that for encryption of just one message, the ring setting has only $26 \cdot 26 = 676$ effective settings. In other words, the attacker only needs to recover only the physical orientation of the internal wiring in the leftmost rotor rather than actual the ring setting.[6] The ring setting defines the relation between the internal wiring and the rest of the rotor, while the display value determines the orientation of the rotor in the Enigma machine. Therefore they together define the initial pattern of the rotors' scrambling, and it is enough to find the message setting. Short texts give minimal rotor stepping reducing the impact of the middle rotor's ring setting greatly.

[6]However, this leftmost ring setting is still relevant if we study how this message was setup. The recovery of a full 3-letter ring setting is needed in order to decrypt additional messages sent within the same network on the same day.

In addition to the three rotors, there are two extra elements mentioned with regards to the Enigma machine which has some impact on the encryption:

- An entry ring, in which the current enters and exits the right-most rotor.
- A reflector, where the incoming current is reflected back through the rotors a second time, before exiting through the entry ring.

The reflector is itself a self-reciprocal transformation, and its inclusion makes Enigma encryption and decryption the same operation as the current in rotors flows in the same circuit, albeit in the opposite direction.

2.2.3. Plugboard

Enigma's plugboard is located at the front of the Enigma (typically). It defines a pairwise substitution between the 20 (typically in WW2 traffic) of the letters with the use of 10 plugs. A plugged connection between two letters is often referred to as a *stecker*. Each stecker defines a self-reciprocal substitution between two letters. The plugboards stecker substitutions are applied twice, both before and after entering the rotors. We have three cases: zero plugboard substitutions, one plugboard substitution, and two plugboard substitutions. The plugboard substitution is often listed as letter pairs separated with space, as shown in Figure 2. Letters that are not part of a stecker pair are often referred to as *self-steckered letters*. These self-steckered letters are essential to many attacks on Enigma encryption (Gillogly 1995; Williams 2000; Ostwald and Weierud 2017). The introduction of the plugboard (around 1928-1930) was a big improvement over the early commercial Enigmas and protected against well-known cryptoanalytical attacks.

2.3. The complexity of the enigma

The version of the Enigma described above is quite complex. Some simple calculation shows that there are $5 \cdot 4 \cdot 3 = 60$ different rotor selections, $26^3 = 17576$ different ring settings, $26^3 = 17576$ different message settings and $\frac{26!}{6! \cdot 10! \cdot 2^{10}} = 1.5073827 \cdot 10^{14}$ plugboard settings. In total this gives:

$$5 \cdot 4 \cdot 3 \cdot 26^6 \cdot \frac{26!}{6! \cdot 10! \cdot 2^{10}} = 60 \cdot 26^6 \cdot \frac{26!}{6! \cdot 10! \cdot 2^{10}} \approx 2.7939259 \cdot 10^{24} \approx 2^{82},$$

different settings. However, the complexity of these settings doesn't perfectly represent the complexity of Enigma's encryption. It is possible to simplify and remove some redundancy; for example, as mentioned in Section 2.2, the leftmost ring setting can be perfectly represented by the leftmost display value; therefore, in practice, it is common to refer to only the $26^2 = 676$ impactful ring settings.[7] In addition to this, some papers (Matthews 1993; Williams 2000) reduce this number further from 26^2 to 26. This reduction is because, in practice, the messages are very short, 250 letters or shorter (Ostwald and Weierud 2017), this means that the leftmost rotor almost never steps. After the first step, the middle rotor only steps every 26 characters, and after the first step, the leftmost rotor only steps every $26^2 = 676$ characters. This means that as long as the messages are under 250 letters long, the leftmost will most likely not step, and at most step once.[8] For this reason, the stepping of the leftmost rotor is often abstracted away, since while ignoring this one may still decrypt at least 50% of the message. If we abstract away from this, the fraction will instead be:

$$5 \cdot 4 \cdot 3 \cdot 26^4 \cdot \frac{26!}{6! \cdot 10! \cdot 2^{10}} = 60 \cdot 26^4 \cdot \frac{26!}{6! \cdot 10! \cdot 2^{10}} \approx 4.1330264 \cdot 10^{21} \approx 2^{72}$$

However, even with such a reduction the complexity is considerable. Even by modern computing power, an exhaustive search over the complete space of states will be a demanding task.

2.4. Genetic algorithms

Genetic algorithms (GA) draw their inspiration from evolution. They start by creating multiple candidate solutions to the problem. Each candidate solution is packaged within an object, referred to as an *individual*. The collection of individuals makes up the genetic algorithm's *population*. The parameters that vary across individuals are called *genes* (Mitchell 1998). The collection of these genes are referred to as the *genome* or *genotype* of the individual. Random draws are usually used when creating the first individuals, to assure some initial *genetic diversity*.[9] The individuals' *fitness* can be determined by a *fitness function*. Individuals with high fitness relative to the other individuals survive and reproduce, similarly to evolution in the real world. The evolution is naturally divided into *generations*, where each generation requires:

1. Evaluating the individuals.
2. Finding the fittest individuals (for reproduction).
3. Replacing the least fit individuals with the offspring of the fittest.

[7]This is because the notch of the leftmost rotor as described in Section 2.2 is ignored.

[8]Similarly for the middle rotor it will step at most $\frac{250}{26} + 2 < 12$ times. (We add 2 instead of 1 to account for the rare case of double stepping).

[9]In nature, genetic diversity refers to the diversity of the genes in a specific species.

This process is repeated until the models stop improving significantly. The best individual in a population is called the *alpha* individual.

2.4.1. Cross-over and mutation

Reproduction between two or more individuals, is called *cross-over*. The cross-over allows a new individual to inherit some of the elements from each parent. This cross-over can be done in many ways, but in nature, a new individual (typically) inherits roughly 50% of its genes from two parent individuals. During cross-over, some mutations may occur in the offspring's genome, and this is also used in genetic algorithms. This mutation introduces some (needed) variation in the population. It is common to have a smaller number of individuals than what is present in more extreme real-life examples, such as wildebeest populations. The population used in the genetic algorithm is more like a population of individuals which inhabit a small island that is roughly 10 to 500 individuals. A concern with small populations is that it is prone to loss of genetic diversity, while this may be an issue, there is a tradeoff. Smaller populations require fewer computations per generation since each individual in the simulation has to be assigned a fitness. Also, a smaller population allows for good gene variations to spread through the population quicker than it would have with a large population. Even in the real world, a smaller island population may have a more rapid evolution than the larger populations on the mainland (Gross 2006). This indicates that a smaller population can converge faster than a larger population, though at the expense of genetic diversity.[10] The main danger of a small population is that one may get stuck in a local optimum. In nature, genetic diversity also helps the population adapt to a changing environment. However, in this study, the Enigma plugboard is a stationary target for each simulation, so genetic diversity was not prioritized. Bletchley Park, on the other hand, was not attacking a stationary target, and benefited greatly from its "genetic diversity." They had to handle varying amounts of information, changing protocols, and working in a limited timeframe.

2.4.2. Index of coincidence

The Genetic algorithm needs a fitness measure, a way of comparing a partially decrypted ciphertext to other partially decrypted ciphertexts. To a human, it is typically obvious whether a given text is plaintext or ciphertext. However, quantifying how close the text is to plaintext, or if a given text is closer to plaintext than another, is more difficult. Luckily several different techniques can be used to measure the "closeness" to plaintext. A lot of them exploit the biased nature of natural languages; for example, letter frequencies

[10]This accelerated evolution may also be because it takes more time for a favorable genetic variation to spread through the population when the population is large.

can indicate how close one is to true German or true English. However, this measure is not ideal when working with an unsolved plugboard, as only roughly 5% of the text is left unaffected by the 10 plugs.[11] Furthermore, these frequencies are very vulnerable to noise, as the relative frequencies of letters can vary greatly, especially when working with very short texts. We need a measure that works even when the number of incorrect characters is large. The index of coincidence (IC) suggested by William Frederick Friedman (1922) is a candidate for such a measure. It is defined mathematically as:

$$IC = \frac{\sum_{i=1}^{26} f_i \cdot (f_i - 1)}{N \cdot (N-1)},$$

where IC is the index of coincidence, f_i is how frequent the letter i is in the text, and N is the number of letters in the text.

Informally the index of coincidence gives the probability that two letters randomly drawn from the text are equal. This measure is better as the self-steckered plugs result in a monoalphabetic substitution regardless of their exit plug. This is essential as it allows IC to pick up some statistical biases when using an empty plugboard as roughly, $100 \cdot \frac{6}{26}\% \approx 23\%$ of the key-presses result in monoalphabetic substitutions (ignoring rotors). It is this weakness that a series of previous work exploit when attacking the Enigma (Gillogly 1995; Williams 2000; Ostwald and Weierud 2017), keeping the plugboard empty while applying a partial brute-force of the rotors.

Under the assumption that all the characters are just as likely in an incorrect decryption, we have:

$$f_i(N) \approx \frac{N}{26},$$

which means that a random text should have an approximate IC of:

$$\begin{aligned} IC_{rand} &\approx \frac{\sum_{i=1}^{26} f_i(N) \cdot (f_{i-1}(N-1))}{N \cdot (N-1)} \\ &= \frac{\sum_{i=1}^{26} \frac{N}{26} \cdot \frac{N-1}{26}}{N \cdot (N-1)} \\ &= \frac{\sum_{i=1}^{26} 1}{26^2} \\ &= \frac{1}{26} \\ &\approx 0.0385, \end{aligned}$$

[11]The unaffected plugs are $\frac{6 \cdot 5}{26 \cdot 25} \approx 0.046$.

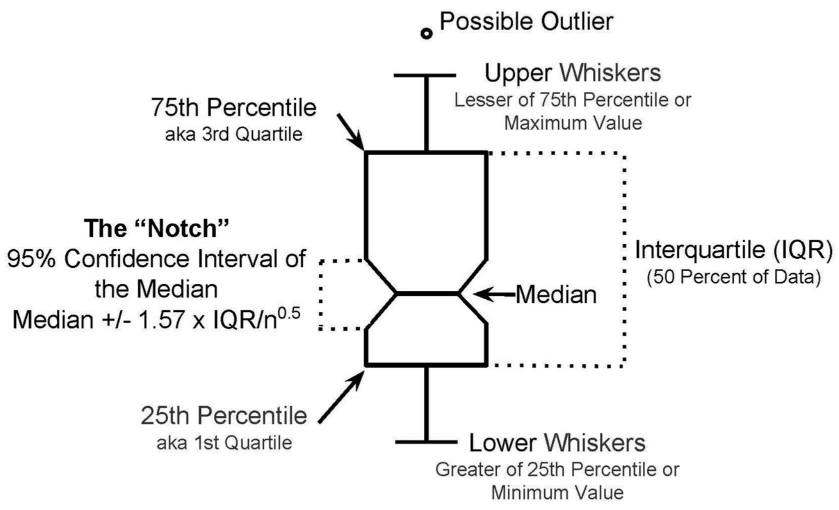


Figure 6. The key features of a notch plot.
Key features of a notch plot explained (from David's Statistics (2016))

while English is closer to 0.066, and according to Gillogly standard German is 0.07 (Gillogly 1995).

2.4.3. Notch plot

The genetic algorithm is not deterministic. This means that the time to find the correct plugboard settings will vary even for the same ciphertext. However, by conducting many runs and comparing the runtime between them, we can state something about the efficiency of the algorithm, and how fast we expect to find a solution. When visualizing such results, it is common to use a notch plot. A notch plot visualizes such a result by creating a box plot where the middle line represents the median of the data and letting the ends of the box define the 75th and 25th percentile of the supplied data. In other words, 50% of the data is inside the interval defined by the box. Around the median, there is funnel-like shape, a *notch* which constitutes the 95% confidence interval of the median. Outside of the box, there are two *whiskers* on each side which span the remainder of the observations. Then finally, there may be some dots outside the whiskers; these illustrate the *outliers*, which are extreme and atypical observations. An outlier can come from a human error like a typo or a strange event. In our runs, an outlier is typically due to a very lucky or unlucky attack. Figure 6 gives an overview of the features of a notch plot.

2.5. Previous work

Long after the war, in 1995, Gillogly used the index of coincidence in a ciphertext-only attack on Enigma encryption (Gillogly 1995). He does this with an initial brute-force[12] of the rotor order, rotor selection, and message setting. In this initial brute-force, he uses an empty plugboard and ring settings "AAA." Note here that this initial brute-force involves $60 \cdot 26^3 = 1054560 \approx$ 1 million different decryptions of the message. Then with the rotors and message setting that gave the highest index of coincidence, the ring settings are calculated. This is also done with the index of coincidence, however, here we start with the rightmost rotor, and testing the 26 different ring settings. The tests are conducted by moving the message setting in unison with the tested ring setting. For example, if the brute-force found the message setting D for the rightmost rotor, then he tests ring setting B with message setting E, ring setting C with message setting F et cetera. After the rightmost rotor ring setting is found, the middle rotors ring setting is found, and the leftmost is left as it is as it is perfectly represented by the message setting. For the plugboard, he no longer uses IC, but instead trigram frequencies, where the "true" distribution is found from the communist manifesto. He begins by searching the $26 \cdot 25$ possible swaps of just one stecker, then the $24 \cdot 23$ possible swaps of two steckers, and so on until he has found all six steckers. Gillogly tested his technique on 0 to 11 steckers and found very limited success on 10 plugs with a 5% success rate on 1463 letter messages, but more than 40% success on 4 plugs with 316 letter messages.

Williams (2000) builds on the work done by Gillogly (1995). In her work, she begins by locking the plugboard settings to be: "DR JX FW HS CL MU GY KV QZ BP." This may look random, but note that the most frequent letters in English plaintext remain unaffected by the plugboard (A, E, I, N, O, and T). This is particularly important because, like Gillogy, she finds the message setting and the rotor selection with a form of brute-force which relies on the letters not affected by the plugboard. With these settings, she achieves a 100% decryption accuracy on a 450 letter message encrypted with an Enigma with all 10 steckers. She improves on Gillogly's method by storing the best 3000 message settings and rotor selections from the initial brute-force, so her algorithm does not fail if the best one does not match. For this brute-force, she tests multiple measures, including IC, and found that the Sinkov statistic[13] applied to unigrams gave the best results. (for details see her paper (Williams 2000)).

Bagnall et al. attempted a genetic algorithm cryptanalysis of the three rotor system (Bagnall, McKeown, and Smith 1997). However, their success

[12]Brute-force means to try all the possible solutions. In this case, it refers to trying all the rotor orders and ring settings.

[13]The Sinkov statistic outcompeting trigrams makes perfect sense as the plugboard's influence on the trigram frequencies is very large. The probability of a trigram being unaffected by the plugboard is $\left(\frac{6 \cdot 5}{26 \cdot 25}\right)^3 \approx 9.83 \cdot 10^{-5}$.

was limited, only cracking the two rotor systems, and failing on systems using three or four rotors.

Ostwald and Weierud in 2017 published another paper on the Enigma machine in Cryptologia titled *Modern breaking of Enigma ciphertexts* (Ostwald and Weierud 2017). Their paper is a comprehensive work which attacks and manages to break many previously unbroken Enigma messages. They do this using a hill-climbing algorithm paired with the brute-force approach described by Gillogly (1995). Their success is in large part due to their in-depth analysis of Enigma's plugboard and their extensive knowledge of the protocols and techniques used to improve the security of Enigma encryption during WWII. Of particular interest to this study is their hill-climbing attack on the plugboard, which similarly to Gillogly and previous attempts start with an empty plugboard. Oswald et al. argue for this approach since it is guaranteed to have six correctly self-steckered letters—in contrast to a completely random steckering which may have no correct steckers. From this empty steckering, the authors describe the various techniques they use to find the first steckers of the plugboard since the hillclimber alone was not always successful. Described are approaches for a brute-force of the first stecker, a brute-force of the first and second stecker, a brute-force of the first, the second and third stecker, and finally a brute-force of the first, the second, the third and the fourth stecker. In other words, they may brute-force $1.6 * 10^8$ different steckerings after their initial brute-force of the rotors. Because this was often too slow, they implemented a targeted stecker search which prioritized more frequent letters, with great success.

A more recent study by Lasry et al. *Cryptanalysis of Enigma double indicators with hill climbing* (Lasry, Kopal, and Wacker 2019) in 2019 introduced new attacks on two of the double indicator operational procedures: the one used until September 1938 and the one used from September 1938 to May 1940. In doing so, they first covered Rejewski's attack, which he devised at the beginning of the 1930s. Rejewski's attack was on the double indicator which was in use by the Germans until 1938. This double indicator was the six first letters of each message, denoting the message setting by encrypting it twice.[14] Both Rejewski and Lasry et al. begin by trying to compute the cyclic structures of $A_4 \cdot A_1, A_5 \cdot A_2$ and $A_6 \cdot A_3$, where A_i is the state of Enigma's encryption when the i_{th} letter is typed. If they manage to compute their cyclic structure, then they can brute-force parts of the Enigma. Rejewski ignored the ring setting and brute-forced the rotor order[15] and message setting. However, Lasry et al. do not ignore the ring setting and accounts for the middle rotor movement using internal hill climbing. Their hillclimbing

[14](Covered at the end of Section 2.1)

[15]In the beginning, there were only three rotors to choose from so he only had to deduce the rotor order.

techniques also allow them to continue even though the initial computation of the cyclic structures fail. They continue by trying to reproduce the cycles given by the indicator states, by looking for all possible rotor orders, ring settings, and basic state options. They uncover the plugboard settings with hill-climbing, but in contrast to Ostwald and Weierud (2017) they start with a random plugboard instead of an empty one. This brute-force paired with hill-climbing enables them to solve the Enigma using only 6-8 double indicators, while Rejewski's attack required 70-90 double indicators. Additionally, they handle turnover by the middle rotor. They also handle the 1938-1940 protocol similarly with hill climbing except here they base their attack on the Zygalski method and improve upon its reliability.

3. GA-based enigma attack

We consider encryption and decryption of the first chapter of "Alice in Wonderland" (http://www.gutenberg.org/files/11/11-h/11-h.htm[16]). The reason for choosing this text in contrast to an authentic WWII message[17] is twofold. First, we can freely vary the actual message length, and we may also vary the Enigma settings. The latter is especially important as we are not interested in revealing a particular historic Enigma setting, but the ability to decrypt a random Enigma setting. Moreover, as we vary the message length, we can study the decryption attack sensitivity to message length.

3.1. Enigma decryption settings impact on IC

All successful decryptions[18] of the full Enigma relies on some kind of partial brute-force. To highlight this, we first find the IC of the plaintext, denoted IC_{pt}, of the first chapter of "Alice in Wonderland":

$$IC_{pt} = 0.06649$$

This index of coincidence is similar to the one we would expect from English. The Enigma is then used to encrypt the entire chapter with the Enigma settings shown in Table 1. The index of coincidence of the resulting ciphertext, IC_{ct} is:

$$IC_{ct} = 0.03854,$$

which is roughly equal to the IC of a random text, and considerably lower than the IC of the plaintext. This ciphertext will be the basis for the analysis below.

[16]All non-letter characters are removed from this first chapter for easy Enigma encryption.

[17]Several earlier contributions rely on authentic Enigma messages (Gillogly 1995; Ostwald and Weierud 2017).

[18]See Section 2.5 for details.

Table 1. Enigma settings.

Rotors	Ring settings	Plugboard settings	Message setting
IV II I	FTR (5, 19, 17)	AT BO DF GV HR IW JL KS MX UY	VYJ (21, 24, 9)

(This is the Enigma settings described as date 31 in the authentic codebook excerpt shown in Figure 2.)

To amplify the contrast, the differences between the cipher- and plaintexts IC, we introduce a progress measure which we will call *Progress Index of Coincidence (PIC)*. By design, we want $0\%(=0)$ progress if nothing is done, and we want $100\%(=1)$ progress if we have found the plaintext. We define the PIC of an attempted decoding PIC_{ad} by:

$$PIC_{ad} = 1 - \frac{IC_{pt} - IC_{ad}}{IC_{pt} - IC_{ct}},$$

where IC_{ad} is the index of coincidence of the attempted decoding of the ciphertext. From this we observe that PIC_{ad} is linearly dependent on IC_{ad} since both IC_{pt} and IC_{ct} are constant for a given ciphertext. Moreover, this relation is:

$$PIC_{ad} = \frac{1}{IC_{pt} - IC_{ct}} \cdot IC_{ad} - \frac{IC_{ct}}{IC_{pt} - IC_{ct}}.$$

In other words the two measures are equivalent as a fitness measures for the GA, however, PIC gives a clearer and more human-readable image of the progress. Also noteworthy is that PIC_{ad} uses IC_{pt}, which is typically assumed to be unknown for a ciphertext-only analysis. However, as PIC_{ad} and IC_{ad} are linearly dependent, this should not be an issue, especially since they are only used for fitness measures. To be sure we will only allow the GA to work with IC_{ad}, and not PIC_{ad} until the run terminates.

Table 2 shows that the measure works as intended. The correct Enigma settings leading to the correct decryption gives 100% PIC. Table 2 also shows that if just one of the rotors is wrong or the ordering is wrong the PIC drops from 100% to roughly 0% PIC. Even though the message setting, 3 ring settings, and 10 plugs in the plugboard are correct, the IC shows no indication of just how "close" we are to the correct key. This discreteness of the correct rotor selection poses a challenge for machine learning approaches, as most of them rely on some form of hill climbing. Furthermore, this property is not unique to rotor selection it also applies to the ring- and display- settings.[19] Despite this discouraging insight, Gillogly (1995) showed that only parts of the Enigma needs to be bruteforced, since the ring settings and the message setting preserve some of the IC properties

[19]For some examples please check the appendix Section 4 table 11 and table 12.

Table 2. Enigma decryption changing the rotors.

Rotors	IC	PIC
IV II I(No change)	0.06649	100%
V II I	0.03852	−0.1%
III II I	0.03844	−0.4%
IV III I	0.03846	−0.3%
IV II III	0.03846	−0.3%
IV I II	0.03852	−0.1%
I II IV	0.03846	−0.3%

Red is used to highlight which rotors are changed from the correct decryption settings to the attempted decryption, while blue is used to highlight which rotors are swapped before the attempted decryption.

Table 3. Enigma decryption changing ring settings and message setting with the same index.

Ring settings	Message setting	IC	PIC
F T R(No change)	VYJ(No change)	0.06649	100%
A T R	QYJ	0.06649	100%
F T S	VYK	0.06436	92.4%
F U R	VZJ	0.06404	91.2%
F U S	VZK	0.06214	84.4%
K V I	AAA	0.04805	34.0%
A A A	QFS	0.03860	0.2%

Red is used to highlight which settings are changed from the correct decryption settings to the attempted decryption.

of the decryption when changed in unison. This may not be too surprising; in Section 2.1, we observed that the two settings are highly related. Table 3 shows this in practice; a synchronized change in the message setting and the ring setting allows the IC to measure the quality of the partially decrypted ciphertext. We see that when only the leftmost ring- and message-setting is changed in unison, there is no decrease in PIC as they perfectly represent each other. We also observe that the minor change of incrementing the rightmost ring- and message-setting by one barely reduces the PIC. So, in this case, the encryption is the same except when the middle rotor (and possibly the leftmost rotor) steps prematurely. Since the rotors work almost like an odometer, this only happens once every 26 characters. However, the encryption before any stepping is the same given "synchronized" ring- and message-settings. Therefore it is natural for hill climber to pay more attention to the first characters of the ciphertext that may not be influenced by an asynchronous stepping. This connection between ring- and message-setting is of great importance as it allows for a partial brute-force attack by keeping either the message setting or the ring settings fixed. The message setting "AAA," for example, achieves a PIC of 34% with otherwise correct settings. Moreover, it achieves a PIC of 1.2% with zero plugs, and the correct rotor selection, this is something we could pick up on with a partial brute-force. However, it is possible to be "unlucky"; if we fix the ring setting to "A A A," as Gillogly did, and brute-

Table 4. Enigma decryption changing plugboard settings.

New plugboard settings	IC	PIC
AT BO DF GV HR IW JL KS MX UY (No change)	0.06649	100%
AH BO DF GV IW JL KS MX RT UY	0.05902	73.2%
AH BZ DF GV IW JL KS MX RT	0.05353	53.6%
AH BZ FO GV IW JL KS MX RT UY	0.05053	42.9%
AB CD EF GH IJ KL MN OP QR ST	0.03852	−0.1%
<No plugs>	0.03993	4.9%

Red is used to highlight which settings are changed from the correct decryption settings to the attempted decryption. The final row < No plugs > is used to symbolize the decryption with correct rotor settings, ring settings, and message setting, but the plugboard is left un-steckered.

force the rotor selection and message setting. The resulting PIC is 0.2% with the correct plugboard and −0.3 using zero plugs. This is astonishingly low and is unlikely to be picked up during a partial brute-force.[20] For this reason, maybe a variant of Williams, (2000) approach where we try 1-3 fixed values for the ring setting or the message setting during the partial brute-force will work well.

The plugboard on the other hand with its $1.50738 \cdot 10^{14}$ possible states, is typically not bruteforced. Table 4 shows that its change in IC is not as discrete as the earlier settings. This makes it vulnerable to machine learning approaches, given that the rest of the Enigma is solved. Also, note from the above table that the empty plugboard results in a positive PIC of about 5%. The empty plugboards small but positive PIC is an essential premise in the ciphertext-only analysis of the Enigma. As discussed earlier, this allows for a brute-force attack of the rotors and message setting with an empty plugboard. It is also the starting point for Ostwald et al.'s hill-climbing algorithm (Ostwald and Weierud 2017), guaranteeing six correctly self-steckered plugs. However, a genetic algorithm starting from 0 plugs will have the unnecessary complexity of dynamically decreasing and increasing the genome size, which will probably slow it down. Therefore, the genetic algorithm that this paper introduces has a genome of exactly ten plugs.

The above analysis shows the discreteness of Enigma's rotor selection, rotor order, ring setting, and message setting. From this, it is clear that some brute-force is needed. However, Gillogly's techniques allow us to narrow the search for the rotor settings to some candidates, given an initial brute-force attack. The remaining plugboard was shown to be vulnerable to hillclimbing and other machine learning approaches. In the next three sections, we will design and use a Genetic Algorithm attack which can solve the plugboard in a matter of minutes.

[20]This is likely part of the reason why Gillogly only saw a 5% success rate on 10 plugs.

Table 5. Population.

	top-third			middle-third			bottom-third			
fitness in $100 \cdot IC$	6.06	6.05	6.03	6.02	6.00	5.99	5.98	5.94	5.94	5.93
individual id	4	0	7	3	5	8	2	9	6	1

Table 6. Cross-over combinations.

cross-over individuals (ids)	4,3	0,5	7,8
replaced individuals (ids)	9	6	1

3.2. Genetic algorithm for determining the plugboard settings

In this section, we will consider a genetic algorithm attack on the plugboard.[21] The specification of a genetic algorithm involves:

1. A representation of the individuals' genome.
2. A fitness function.
3. A selection function for cross-over.
4. A cross-over function.
5. A mutation rate.
6. A population size.
7. A number of generations the function is run

We let each stecker pair constitute a gene. Furthermore, we let the individuals **genome** consist of 10 genes represented by a list of 20 indices with numbers from 0 to 25, where each pair defines a stecker. For example the genome:

[(0,1), (2,3), (4,5), (6,7), (8,9), (10,11), (12,13), (14,15), (16,17), (18,19)]

Defines the plugboard settings:

AB CD EF GH IJ KL MN OP QR ST

The initial individuals are chosen to be a random selection of 20 such indices. **The fitness function** is chosen to be the IC of the attempted decryption with the plugboard settings defined by the individuals' genome. This measure is then used to rank the individuals from most fit to least fit for **cross-over**. Before cross-over, the population is divided into three parts, the top-third, the middle-third and the bottom-third. The top-third cross-overs with the middle-third, in the respect, that the fittest in the top-third cross-overs with the fittest in the middle-third, the second fittest with the second fittest and so on. The offspring of this cross-over then replaces the bottom-third of the population. Tables 5 and 6 exemplify this with a mock population of 10 individuals. If the population size is not divisible by three,

[21]In Bletchley Park Gordon Welchman's invention, the" Diagonal Board" improved the British Bombes attack on the plugboard significantly.

there may be one or two individuals that does not partake in the cross-over. In the example shown in Table 5 individual number 2, between the bottom and middle-third, does not partake in the cross-over for this reason.

The *cross-over* between two individuals starts by randomly selecting one of the individuals to be *parent*1 and making the other *parent*2. We then draw five indices in the range of 0 to 10. These random indices then access and copy five steckers (genes) from *parent*1 to the offsprings genome. The indices that were not drawn in the previous step are then used to access and copy five steckers from *parent*2's genome to the offspring. However, to avoid duplicate plugs, we do not copy plugs that are already present in the offsprings genome. This may results in incomplete or missing steckers in the offsprings genome. At such incomplete or missing steckers, random vacant plugs are assigned until the offspring has a valid genome of 10 steckers. Below is an example cross-over where the pairs 1,3,4,5 and 7 are selected to be inherited from *parent*1:

$$\text{individual :} \quad \begin{bmatrix} GENOME \end{bmatrix}$$

$$\begin{array}{rl}
\text{parent1 :} & \begin{bmatrix} AB & CD & EF & GH & IJ & KL & MN & OP & QR & ST \end{bmatrix} \\
\text{parent2 :} & \begin{bmatrix} AT & BO & DF & HR & IW & JL & KS & MX & PQ & UY \end{bmatrix}
\end{array}$$

$$\text{offspring :} \quad \begin{bmatrix} AT & CD & WF & GH & IJ & KL & ZS & OP & EQ & UY \end{bmatrix}$$

Here we see that all the red pairs with indices 1,3,4,5 and 7 are completely inherited from *parent*1, however some of the steckers from *parent*2 are changed (marked in green). For example the stecker *DF* could not be entirely inherited because the *D* plug is used in the pair *CD*, which has already been inherited from *parent*1, so another vacant plug is picked at random instead, in this case stecker *W* was picked. The GA draws its foundation from evolution; each new offspring has a probability of getting a mutation in their genome. In this paper, we will refer to probability of at least one mutation occurring in an offsprings **genome** as the *mutation rate*, while we let the probability of a mutation occurring in a specific **gene** of the genome be the *mutation probability*. In other words, *mutation rate* refers to the probability of a mutation in the plugboard, while the *mutation probability* is the probability of a mutation in a specific stecker. The two terms have the following relation:

$$mutation\_rate = 1-(1-mutation\_prob)^{10}$$
$$mutation\_prob = 1-(1-mutation\_rate)^{\frac{1}{10}}$$

A gene selected for mutation removes the stecker pair associated with it. Then a new stecker pair is randomly selected from the now eight available plugs. An example of an offspring with two mutations is shown below to illustrate this. The stecker pairs selected for mutation are colored red and the available plugs are also red, and the chosen replacement steckers are blue.

$$\text{offspring}: \begin{bmatrix} AT & CD & WF & GH & IJ & KL & ZS & OP & EQ & UY \end{bmatrix}$$

available=V,N,X,R,M,B

$$\text{offspring}: \begin{bmatrix} AT & CD & \_\_ & GH & IJ & KL & ZS & OP & EQ & UY \end{bmatrix}$$

available=V,N,X,R,M,B,W,F (Chosen index 5 and 7)

$$\text{offspring}: \begin{bmatrix} AT & BF & CD & GH & IJ & KL & ZS & OP & EQ & UY \end{bmatrix}$$

available=V,N,X,R,M,W

$$\text{offspring}: \begin{bmatrix} AT & BF & CD & GH & IJ & KL & ZS & \_\_ & EQ & UY \end{bmatrix}$$

available=V,N,X,R,M,W,O,P(Index 2 and 5 chosen)

$$\text{offspring}: \begin{bmatrix} AT & BF & CD & GH & IJ & KL & ZS & WX & EQ & UY \end{bmatrix}$$

available=V,N,R,M,O,P

From this it is clear that mutations can dramatically change Enigma's encryption and decryption capabilities. Like in nature most mutations (but not all) will be useless or add unnecessary noise. Therefore evolution in GA is typically faster with a low mutation rate; however, with a low mutation rate, the probability of being stuck in a local optimum and not finding the correct decryption is increased. For this reason, two mutation rates will be tested, one with a mutation rate set to be roughly 50%; this corresponds to a mutation probability of 0.067 (6.7%).

$$\begin{aligned} \text{mutation prob} &= 1-(1-0.5)^{\frac{1}{10}} \\ &\approx 0.067 \end{aligned}$$

This fairly high mutation rate has a low probability of getting stuck, but will most likely be slower than a lower mutation probability of 0.001 (0.1%) corresponding to a mutation rate of about 1%.

$$\begin{aligned} \text{mutation rate} &= 1-(1-0.001)^{10} \\ &= 0.009955 \\ &\approx .01 \end{aligned}$$

Furthermore, just because a mutation occurs does not mean that the stecker is changed as there is a $\frac{1}{8*7} = \frac{1}{56}$, chance that the stecker will be unchanged by the mutation.

Table 7. Default GA settings.

Genome	list of 20 indices
Fitness function	Index of Coincidence
Cross-over	as described in Section 3.2
Mutation probability	0.067 or 0.001
Population size	100
Number of generations	100 (may vary)

3.3. Genetic algorithm runs and results

A summary of the design choices of this GA is shown in Table 7. These settings efficiently solve Enigma's plugboard. 100 separate genetic algorithm runs were conducted with default settings (Table 7 with mutation probability 0.067) to find the plugboard settings described in Table 1. All of which were successful in finding the correct plugboard settings. However, running the genetic algorithm for 100 generations is a little bit overkill, since all of them find the correct deciphering before then, as seen in Figure 7. Also, 100 individuals for 100 generations correspond to decrypting the ciphertext with an Enigma 3400 times, which is more than the best case of Gillogly's approach, which decrypted the ciphertext 3050 times (Gillogly 1995). In number of generations, 3050 encryptions correspond to between 90 and 91 generations[22] consisting of 3037 and 3070 decryptions.

From Table 8 we see that the median run finishes in 2344 decryptions which is much faster than the best case of 3050. In other words this approach gives a significant increase in the plugboard recovery speed over Gillogly given the Enigma settings described in Table 1. To ensure that this improvement is independent of the Enigma settings we draw 9 additional Enigma settings, for details check the appendix Section 4 Table 9. We then run 100 genetic algorithm runs on each of the different Enigma settings to show that its efficiency is independent of the encryption settings. Table 10 shows the minimum, median, and max runtime of the GA before finding the correct solution across the 10 different Enigma settings. This clearly shows that the GA attack on Enigma's plugboard works on many underlying Enigma settings. We observe that our worst median is at 71 generations (2410 decryptions), which is pretty good. Of course, we see even better results as the fastest attack only took 35 generations (1222 decryptions), more than twice as fast as Gillogly's best case of 3050 decryptions. We also observe that there seems to be an extremely "unlucky" run on Enigma nr 10, which does not find the solution until it has completed 166 generations. Fortunately, it is an extreme outlier as the second-longest run on Enigma 10 took 102 generations, which is also an outlier, but a more reasonable one.

[22]Number of decryptions $= 100 + 33\cdot$ (Number of generations - 1).

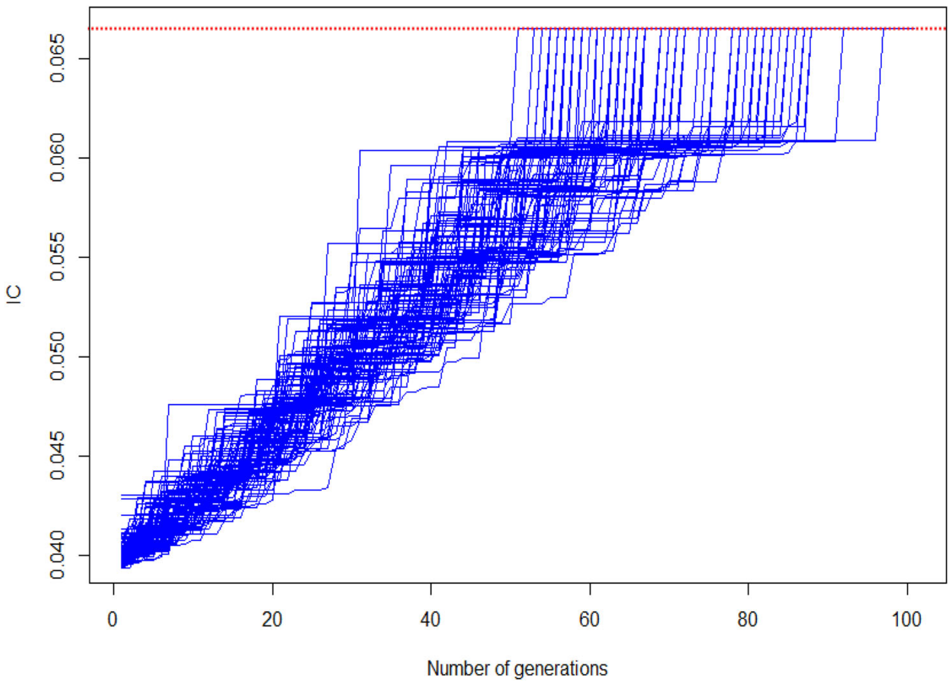


Figure 7. IC of a 100 GA runs with default settings finding the plugboard key from Table 1. Each blue line maps the IC of the best individual in each of the 100 GA runs for each generation.
The red dotted line gives the IC of the plaintext; in this case, the solution appears to be unique, as each decryption with this IC results in the correct plaintext.

Table 8. Finish times of the different GA runs on Table 1.

Measure	Min	Median	Mean	Max
Generations	51	69.0	70.2	97
Decryptions	1750	2344	2384	3268

Table 9. Drawn Enigmas.

name	Rotors			Ring settings	Plugboard settings	Message setting
1	IV	II	I	F T R	AT BO DF GV HR IW JL KS MX UY	VYJ
2	I	IV	III	W C I	BE CG DW FN HU JS MX OV PT QR	RHB
3	I	III	V	N E R	AB CS DM FP GT JL KU NR QY XZ	OAY
4	II	I	V	R R Y	AZ BS DL EI FG HU JV MW NX RT	FBU
5	III	I	IV	S E M	AP BQ CW DZ EL FM IT NU OR SX	OHT
6	II	V	I	J R T	AC BO ES FQ GX HZ IV JL MY PW	SDO
7	III	V	II	P X E	BY CR DN EH IS JT LV MW OP QZ	EYL
8	V	II	III	J A C	AP BH CY ES FG IQ JM KW LV NR	USJ
9	IV	III	II	W K V	AO BH DF EK GJ IS NR QV TY UZ	JOH
10	I	II	V	Y D S	AP BW CI DR FM GN HY JX KS LU	BKJ

Table 10. A 100 GA run finish time comparison across 10 different Enigma settings.

Name	1	2	3	4	5	6	7	8	9	10
Min	51	35	41	44	39	40	43	45	47	42
Median	69.0	71.0	70.5	67.5	67.0	65.0	69.0	70.0	66.5	69.0
Max	97	103	97	99	106	101	110	102	95	166

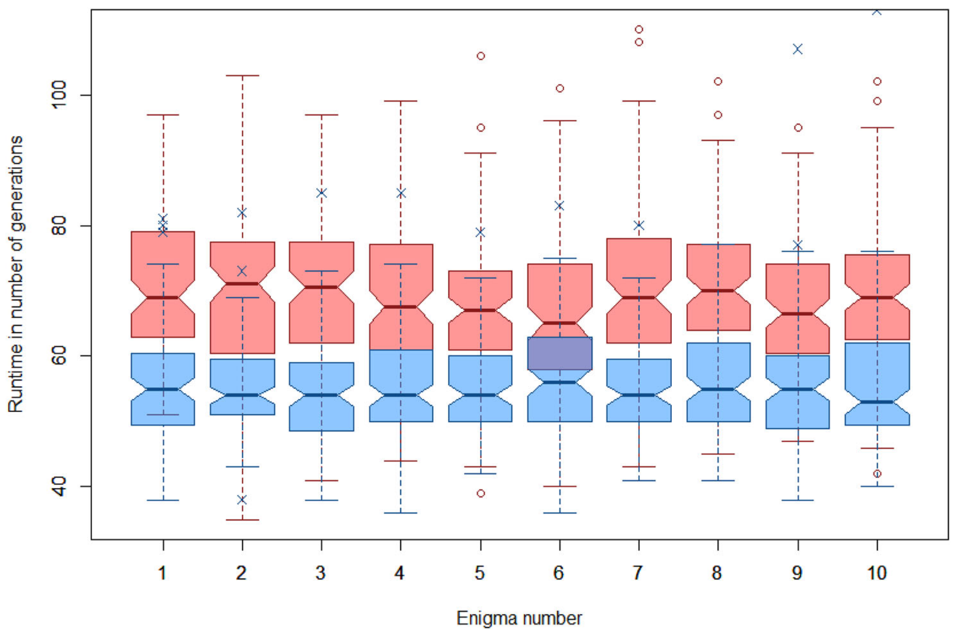


Figure 8. Notch plot comparison of 100 GA attacks with mutation rate 0.5 (red) and 0.01(blue) across 10 different Enigma settings.
In this notch plot the outliers of the low mutation rate (0.01) runs are shown as a darkblue × and the outliers of the high mutation rate(0.5) are shown as a darkred °.

However, this was with a high mutation rate of about 0.5. We also check for *mutation_probability* 0.001. To test this we run 100 GA's on the 10 Enigmas with a maximum number of generations set to 1000. With such a low mutation rate, some of these runs never finish or take almost the full 1000 generations. Of the 1000 runs, 29 of them were worse than Gillogly's best case of 3050 decryptions (90 to 91 generations), and these 29 are much worse, and some do not find the solution. This is because they have lost some essential genetic diversity, which their low mutation rate is unable to replace. However, as we can see, this only occurs in less than 3% of all the runs, most of the GAs also succeed with a low mutation rate. The runs that do succeed generally find the solution faster than their counterparts with a higher mutation rate, as the low mutation rate has a median runtime between 53 and 56 generations. Figure 8 shows a more indepth comparison between the two mutation rates across the 10 different Enigmas. We see here that for most of the Enigmas, the 75% fastest runs of the low mutation rate runs are faster than the 75% slowest runs of the GA with a high mutation rate. The only exceptions being Enigma numbers 4 and 6. In terms of speed, the low mutation rate is the obvious choice. However, as the runs lose most of their genetic diversity in the early generations, this approach is prone to getting stuck in a local optimum. A way to escape the local optimum and increase the genetic diversity is through

Table 11. 100 GA's run on smaller subsets of Alice in Wonderland.

		Runtime				
No. of characters	Correct	Median	Mean	Min	Max	Max PIC
100	0%	69	74.65	32	137	122.76%
150	0%	1000	822.13	69	1000	103.39%
200	47%	1000	611.22	83	1000	100%
250	97%	123	161.9	67	1000	100%
300	97%	118	168.17	74	1000	100%
350	100%	117	121.08	73	228	100%
400	100%	101.5	107.27	67	192	100%
450	100%	92.5	100.07	60	376	100%
500	100%	91.5	91.92	56	173	100%

The GA was run for a 1000 generations or until a text with an IC greater or equal to the IC of the solution was found.

mutations, which by construction is set to be low in this case. For consistent results, the high mutation rate (0.5) performed better, and even managed to get faster decryption than low mutation rate GA on Enigma 2 and Enigma 5. A possible best of both worlds is to run multiple GA attacks in parallel, compensating for its lower success rate with "strength in numbers."

3.4. Genetic algorithm on smaller texts

From Section 3.3 it is clear that the GA paired with IC solves Enigma's plugboard efficiently on the first chapter of "Alice in Wonderland," a text containing 8596 characters. However, it remains to be shown that it also works on shorter texts. To investigate this, we create subsets of the first chapter of Alice in Wonderland, selecting the first n characters of the text, letting $n = 100, 150, 200, 250, 300, 350, 400, 450, 500$. These subsets are encrypted with the default Enigma settings, stated in Table 1. For these new ciphertexts, a hundred GA runs were conducted with the settings defined in Table 7, a mutation probability of 0.067 and the stopping criteria of a 1000 generations or reaching an IC greater than the IC of the plaintext. Table 11 gives an overview of the results of these runs. Note here that short messages that use 100 and 150 characters gain a maximum IC greater than the IC of the plaintext as indicated by achieving a PIC greater than 100%. This is not unique to Alice in Wonderland, but a common occurrence, Ostwald and Weierud (2017) and Gillogly (1995) both used some extra tricks and extra measures to get around this. For the GA to work on such short texts, we would also have to implement alternative measures to IC. We have not done this, and as a result, it has 0% success for texts where it is possible to achieve a PIC greater than 100%. However, the algorithm does not just fail in the instances where a PIC = 100% does not offer an upper bound, as texts with 200, 250 and 300 characters are not successful on every run. Even though the global optimum may be 100% PIC, the

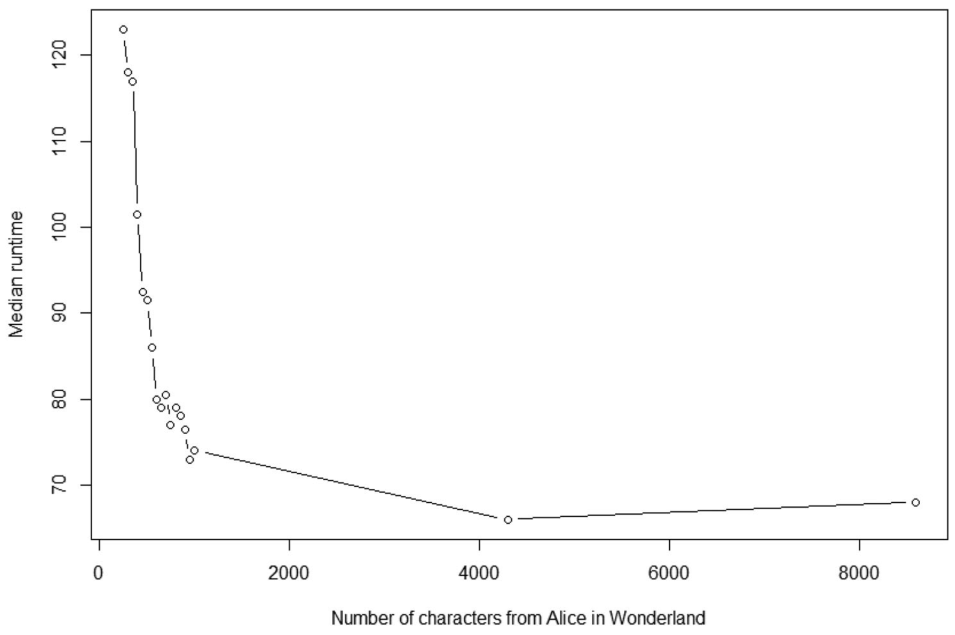


Figure 9. Median runtime vs Number of generations on subsets of Alice in Wonderland. GA runs that did not finish within 1000 generations had their runtime in number of generations set to 1000.

results show that we only have a 47% success rate on ciphertexts with 200 characters. This means that even with a high mutation_probability the GA can get stuck in a local optimum. This occurs because there are too many different plugboard settings that increase the IC on short texts, that a local optimum may be "too" far away from the global optimum in some cases. In extreme cases a local optimum may not have any steckers in common with the correct steckering.

Evident from Table 11 is that the median number of generations decrease as the number of characters increase from 250 characters and up. This is also a likely evidence that the global optimum becomes easier to distinguish with more characters. To further investigate this development we added subsets of the first 550, 600, 650, 700, 750, 800, 850, 900, 950, 1000 and 4298 characters, and ran 100 GA's on each of these subsets, Figure 9. As expected these runs show that the GA works better with larger texts, however, it seems to reach some saturation between 1000 and 4298 characters. The logarithmic shape stops at 4298 characters as the GA preforms slightly worse with all 9596 characters.

We can also observe that the success rate of the GA also increases with the number of characters in the ciphertext. To sketch this development we conduct 100 GA runs with character subsets of (150, 152, 154, 156, … , 348, 350) and plot the percentage of runs that found the correct solution against the number of characters in the "Alice in Wonderland" subset, shown in

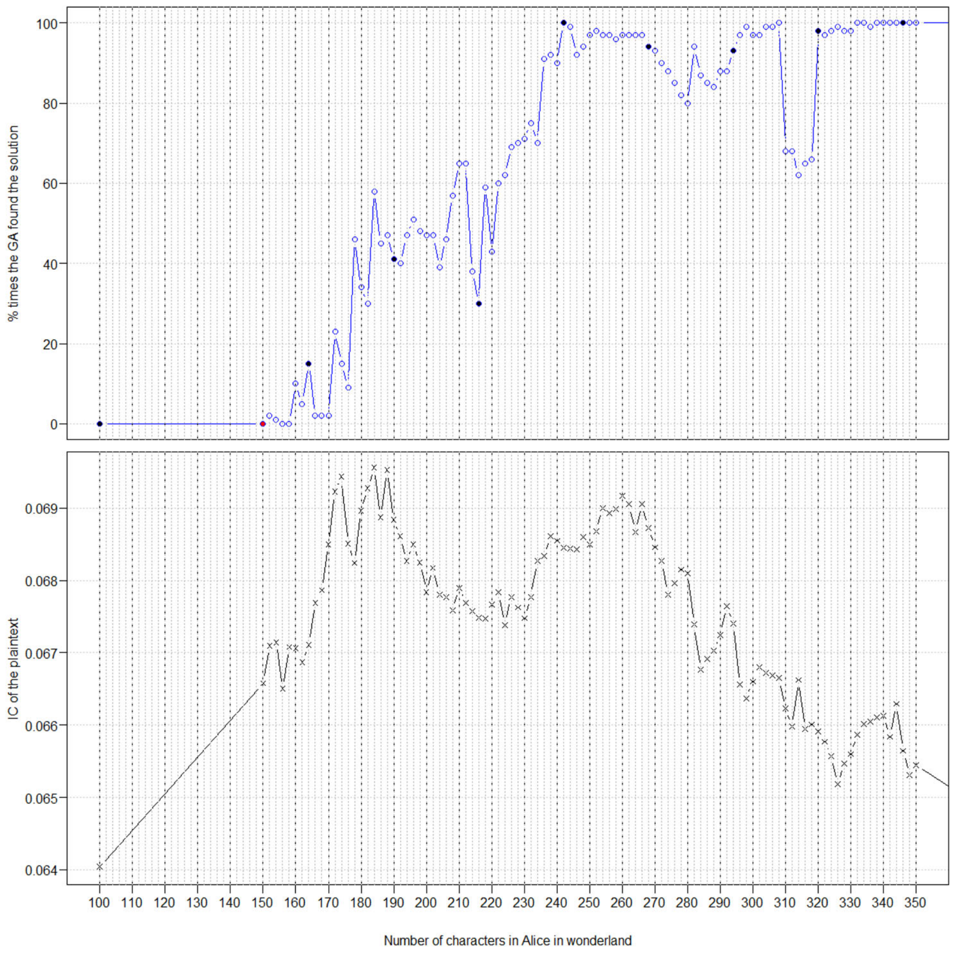


Figure 10. The number of characters in the plaintext plotted against the GA success-rate and the IC of the plaintext.
Marked in black are the first encryptions that includes the rotation of the middle rotor. Similarly marked in red is the first encryption that includes a leftmost rotor rotation.

the top plot of Figure 10. Like with the runtime a key indicator to the GA's success rate is the number of characters, however, this trend/development is more jagged. Also, notice that 152 characters has two successes. These successes are rare, which makes sense since the greatest PIC found is 102.87%, which means that in these cases the local optimum is the solution and the global optimum is something else. Despite that, two of the GA runs are lucky and find the solution. If we add two more characters, we only find the solution in one of the 100 runs. Then if we step up to 156- and 158-characters all 200 GA runs are unsuccessful. Initial speculation by the authors thought that this had something to do with the dive in IC from 0.0671 to 0.0665 between 154- and 156- characters, and that there may be a tiny correlation between the GA's "lucky" successes with small texts and the IC of the underlying plaintext. But in the absence of a deeper analysis it is more likely that the minor reduction in successes are due to chance rather than plaintext IC. After all the plaintext IC of 158 characters is very similar to the IC of 160 characters where the solution is found 10 times. However, this trend diminishes as we get more plaintext to work with. The majority of the jaggedness observed in the success rate of the GA is not due to the changing of the plaintexts IC as can be seen from the bottom plot of Figure 10 which shows the IC of the plaintext. The success rate of the GA is jagged, this may be because we only did 100 trials on each subset, but it has more structure than one would expect from random chance. For example, there is a drop in the success rate of almost all the runs on texts with 310 to 318 characters. We, therefore, think that this jaggedness, in particular, this drop is due to a peculiar interaction between; our current GA approach, the added letters, and how they influence Enigma decryption. A natural theory would be that this is because of rotor stepping. However, the decryption capabilities seem to be agnostic of this as is shown by the red and black dots representing stepping of the leftmost and middle rotor respectively.

4. Conclusion

The Enigma Machine as a whole is built to distort the letter frequencies of a plaintext message. This distortion, paired with the discreteness of the correct decryption settings, especially the rotor settings, makes a ciphertext-only attack difficult. To illustrate this, we introduced a new measure, Progress Index of Coincidence (PIC), which is a more human-readable version of the measure: Index of Coincidence (IC). Our analysis with PIC showed that Enigma's plugboard was vulnerable to a machine learning attack. To capitalize on this vulnerability, we introduced a genetic algorithm attack for solving Enigma's plugboard using a ciphertext only attack. This genetic algorithm attack proved to be very efficient. It found the plugboard settings faster than earlier attacks. Intriguingly the algorithm is the fastest with a low mutation rate but at the cost of its reliability. In other words, the algorithm has a higher success rate with a high mutation rate, but at the cost of its speed. This tradeoff may be of consequence for a broader range of genetic algorithm attacks beyond the Enigma. In particular, one can get the best of both worlds by considering attacks with a low mutation rate in parallel. This way, we can increase the solution probability through the "strength in numbers." We also observe that the decryption success rate is not a completely monotone function of the number of characters in the ciphertext. In particular, we observe a significant dip in success rate for texts with 310 to 318 characters. Intriguingly, this dip does not seem to be driven by plaintext IC nor the Enigma's rotor stepping. It may be due to some non-trivial property of the Enigma encryption, and its interplay with the IC. Future research may shed light on this surprising property of Enigma encryption.

Appendix

The ring- and message-settings impact on enigma decryption

In Section 3.1 of the paper we cover the difficulties of measuring closeness in the Enigma decryption key. Absent from the main paper was a table showing this in practice for ring settings (Table 12) and message setting (Table 13). The full Enigma setting used to encrypt the plaintext (the first chapter of "Alice in Wonderland") corresponds to Enigma nr 1. in Table 9.

Enigma settings used in this paper

Table 9 shows a table detailing the encryption settings of the 10 Enigma decryptions studied in this paper.

The GA development across the different enigma settings

In this paper, we conducted 100 GA runs, (with mutation rate 0.5 and 0.01), for each of the 10 different Enigma settings. For a closer inspection of their performance, we have split Figure 8 into two separate plots: Figure 11(0.5) and Figure 12(0.01). Also included is an uncropped notch plot with mutation rate of 0.01 (Figure 13), which clearly shows how extreme some of the outliers are.

Table 12. Enigma decryption changing ring settings.

New ring settings	IC	PIC
F T R(No change)	0.06649	100%
E T R	0.03846	−0.3%
G T R	0.03846	−0.3%
F S R	0.03842	−0.4%
F T E	0.03846	−0.3%

Red is used to highlight which settings are changed from the correct decryption settings to the attempted decryption.

Table 13. Enigma decryption changing message setting.

New message settings	IC	PIC
VYJ(No change)	0.06649	100%
AYJ	0.03851	−0.1%
VZJ	0.03842	−0.4%
VYK	0.03849	−0.2%

Red is used to highlight which settings are changed from the correct decryption settings to the attempted decryption.

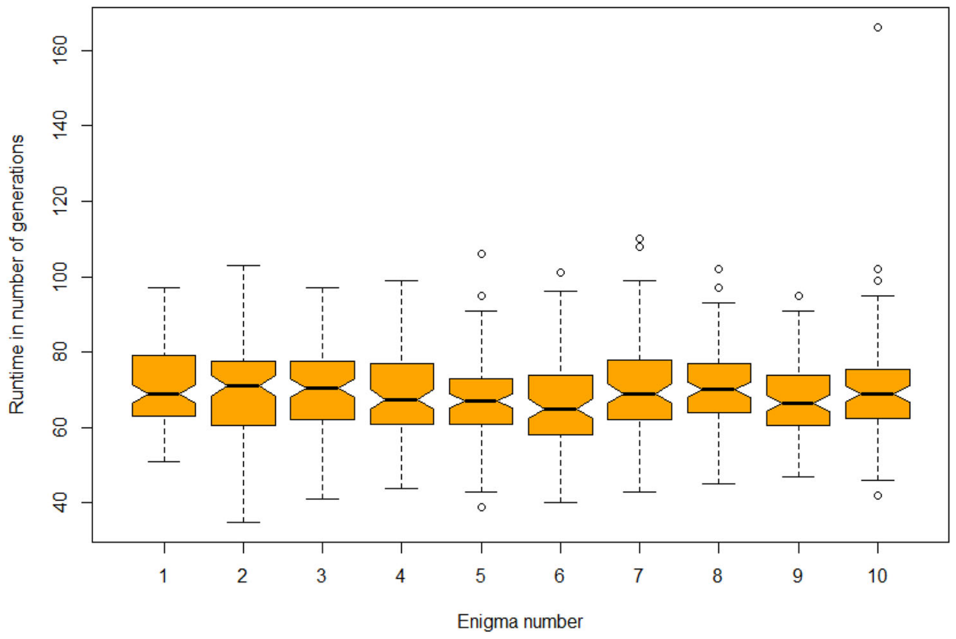


Figure 11. Notch plot of the number of generations used by 100 genetic algorithm runs with mutation rate 0.5 for the 10 different Enigmas.

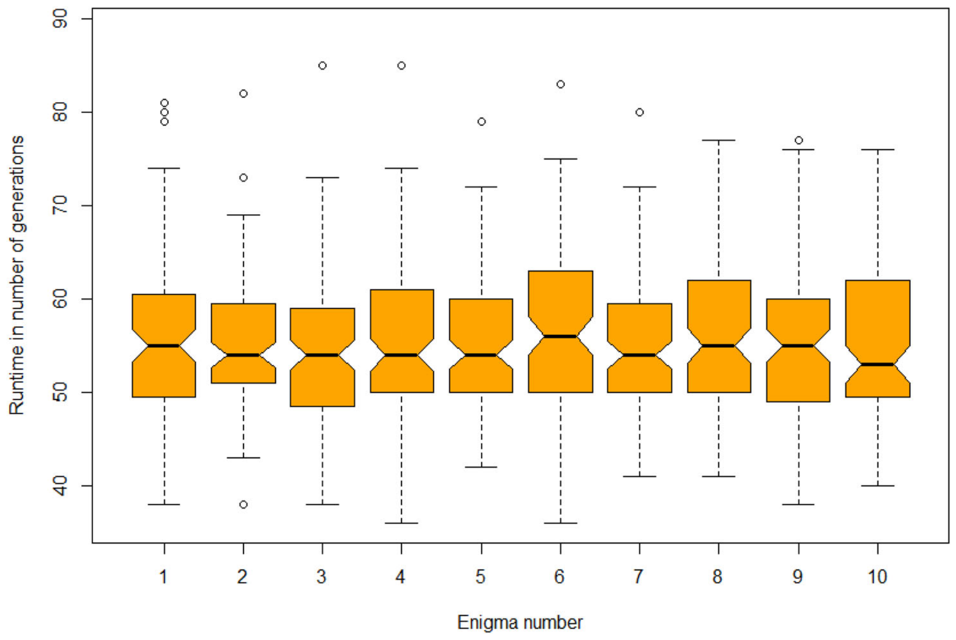


Figure 12. A cropped notch plot, ignoring extreme outliers, of the number of generations used by 100 genetic algorithm runs with mutation rate 0.01 for the 10 different Enigmas.
(The cutoff was at 90 generations.)

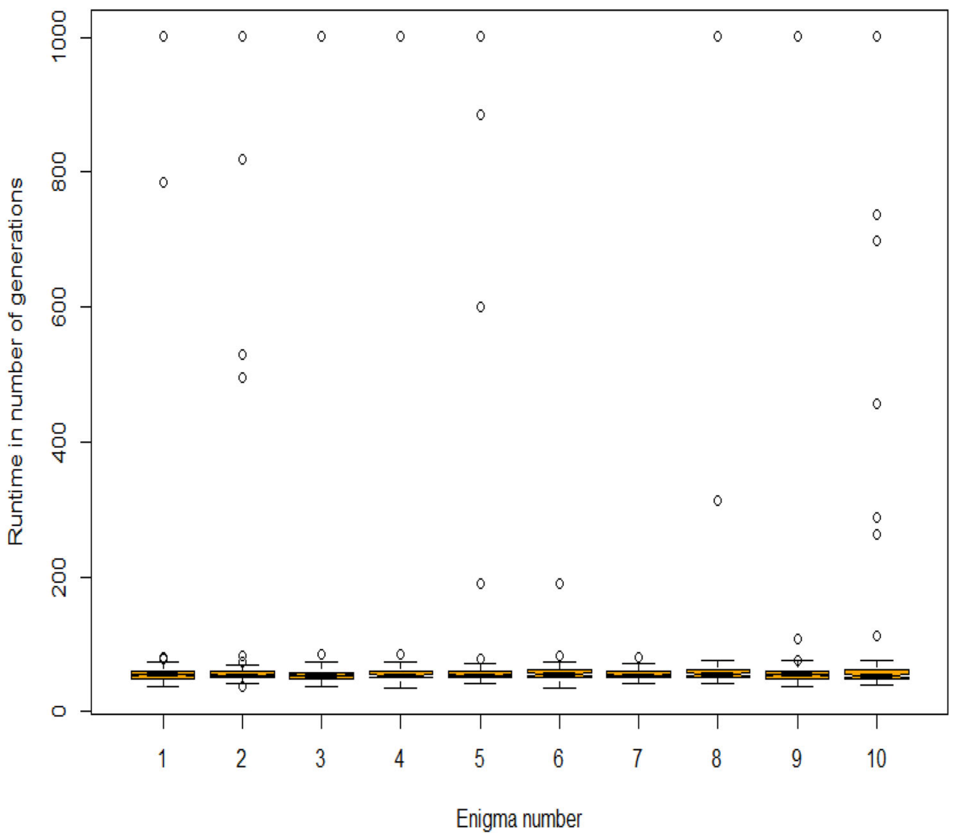


Figure 13. Notch plot of the number of generations used by 100 genetic algorithm runs with mutation rate 0.01 for the 10 different Enigmas.
GA runs that did not finish within 1001 generations had their runtime in number of generations set to 1001.

About the authors

Åvald Åslaugson Sommervoll is a 24-year-old PhD student at the University of Oslo. He has a bachelor degree in Mathematics and obtained a Masters degree in Informatics in 2018. His master thesis focused on machine learning and its applications for spatial aggregation. He now aims to bring machine learning techniques to the world of information security and cryptography.

Leif Nilsen is a Norwegian crypto specialist at Thales Norway AS and is also an Adjunct Professor at the Department of Technology Systems of the University of Oslo. He holds a Cand. Real. Degree in Pure Mathematics from the University of Oslo and has been teaching mathematics, physics, and informatics since 1977. Since 1988 he has worked within research and development of high assurance crypto products, mainly for the governmental market. He has been an active participant in International Standardization of Crypto and Information Security in ISO, NATO, and EDA, and is a permanent member of the ETSI Security Algorithms Group of Experts (SAGE). For more than 20 years he has lectured introductory and advanced courses in cryptology and supervised master and PhD students. His main areas of interest are crypto engineering, design, and analysis of algorithms, elliptic curve cryptography, the history and teaching of crypto.

Acknowledgments

The authors wish to give a special thanks to Audun Jøsang for valuable discussion and words of encouragement. The Authors would also like the anonymous reviewers as their generous comments helped clarify and improve the paper.

References

Bagnall, A. J., G. P. McKeown, and V. J. R. Smith. 1997. The cryptanalysis of a three rotor machine using a genetic algorithm. In *Proceedings of the 7th International Conference on Genetic Algorithms*, ed. Thomas Bäck, East Lansing, MI, USA, July 19–23, 712–718. Morgan Kaufmann.

Copeland, B. J. 2004. *The essential turing*. Oxford: Oxford University Press.

David's Statistics. 2016. Notched box plots. Accessed March 29, 2018. https://sites.google.com/site/davidsstatistics/home/notched-box-plots

de Leeuw. K. 2003. The Dutch invention of the rotor machine, 1915–1923. *Cryptologia* 27 (1):73–94. doi:10.1080/0161-110391891775.

Frederick Friedman, W. 1922. *The index of coincidence and its applications in cryptography*. Walnut Creek, CA: Aegean Park Press.

Gillogly, J. J. 1995. Ciphertext-only cryptanalysis of enigma. *Cryptologia* 19 (4):405–13. doi: 10.1080/0161-119591884060.

Gross, L. 2006. Islands spark accelerated evolution. *PLoS Biology* 4 (10):e334. doi:10.1371/journal.pbio.0040334.

IMDb. 2014. The imitation game imdb. Accessed November 12, 2018. https://www.imdb.com/title/tt2084970/

Lasry, G., N. Kopal, and A. Wacker. 2019. Cryptanalysis of enigma double indicators with hill climbing. *Cryptologia* 43 (4): 1–26. doi:10.1080/01611194.2018.1551253.

Matthews, R. A. 1993. The use of genetic algorithms in cryptanalysis. *Cryptologia* 17 (2): 187–201. doi:10.1080/0161-119391867863.

Mitchell, M. 1998. *An introduction to genetic algorithms*. Cambridge, MA: MIT.

Ostwald, O., and F. Weierud. 2017. Modern breaking of enigma ciphertexts. *Cryptologia* 41 (5):395–421. doi:10.1080/01611194.2016.1238423.

Paar, C., and J. Pelzl. 2009. *Understanding cryptography: a textbook for students and practitioners*. Springer Heidelberg Dordrecht, London, New York: Springer Science & Business Media.

Singh, S. 2000. *The code book: the science of secrecy from ancient Egypt to quantum cryptography*. London: Fourth Estate.

Tony Sale. 2001. Lecture on naval enigma. Accessed 15 April, 2019. https://web.archive.org/web/20180923005612/; http://www.codesandciphers.org.uk/lectures/naval1.htm

Welchman, G. 1982. *The hut six story: breaking the enigma codes*. New York: McGraw-Hill.

Williams, H. 2000. Applying statistical language recognition techniques in the ciphertext-only cryptanalysis of enigma. *Cryptologia* 24 (1):4–17. doi:10.1080/0161-110091888745.