Modeling ENIGMA with Integer Linear Programming

Kevin Knight
Threeven Labs
kevin.crawford.knight@gmail.com

Abstract

We model the German ENIGMA cipher machine as a large integer linear program (ILP), allowing us to solve certain encipherment and decipherment problems with a generic ILP solver.

1 Introduction

ENIGMA is an electromechanical device that was used by the German military to encrypt secret communications before and during World War II. It was the subject of intense code-breaking efforts in Poland (Rejewski, 1984) and the United Kingdom (Welchman, 1982). Researchers since then have conducted a number of ENIGMA analyses using specially-designed software (e.g., Gillogly, 1995; Bagnall et al., 1997; Sullivan and Weierud, 2005; Ostwald and Weierud, 2017; Lasry et al., 2019; Sommervoll and Nilsen, 2021; Kopal and Esslinger, 2022).

In this paper, we investigate the use of a general-purpose problem solver to model ENIGMA. We first describe the workings of the machine as a set of variables and constraints in an integer linear program (ILP).

Disclaimer: The use of the name Adolf Hitler in this paper is solely to illustrate a technical question concerning Enigma settings in a historical cryptographic context. No endorsement, glorification, or political meaning is intended.

We then use a generic ILP solver to answer questions such as "Is there an initial ENIGMA setting that encrypts the string KEVINKNIGHT as ADOLFHITLER? If so, what setting uses the fewest number of plugboard wires?"

Advantages of this approach include:

- Questions are put to the solver in declarative form, as values for specific variables or constraints on those values, and no cryptanalytic software or special search algorithms need to be written. Instead, the ILP solver employs general-purpose search strategies, such as branch-and-bound and cutting planes.
- The ILP solver returns optimal answers.

The main disadvantage is slow speed, due to the generic problem solver's optimality and its lack of ENIGMA-specific strategies. In this paper, we investigate what can and cannot be done by modeling ENIGMA this way.

2 Integer Linear Programming

Any ILP consists of variables, constraints, and an objective function. For example:

Integer variables: x, y

Maximize: $2x + y$

Subject to:

$x + y \leq 6.9$
$y - x \leq 2.5$
$y > 1.1$

The goal of an ILP solver is to find integer assignments[1] to the variables that satisfy the constraints and maximize the objective function. Readers unfamiliar with ILPs may want to solve the above puzzle by hand. Dantzig (1949) popularized linear programming after World War 2 and introduced the first generic solver, the simplex algorithm.

3 ILP for Simple Substitution

Ravi and Knight (2009) show how to model a simple substitution cipher as an ILP. Since this cipher is much simpler than ENIGMA, we briefly review their method.

[1] In this paper, we further constrain each variable to take on the value 0 or 1.

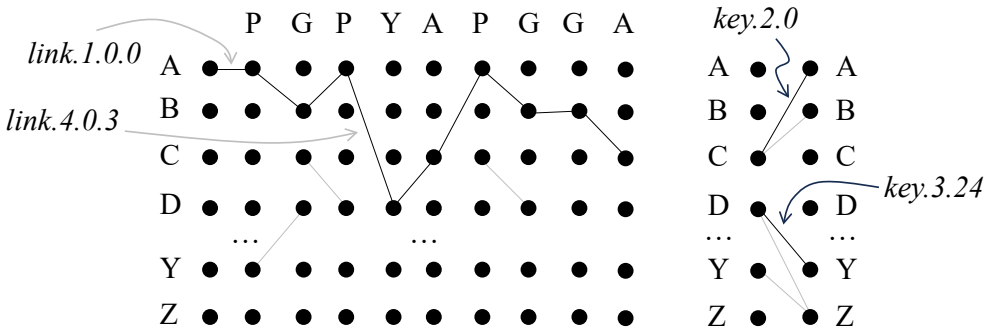

Figure 1: Decrypting a simple substitution cipher with integer linear programming (ILP). The ciphertext runs along the top, and there is a row for each plaintext letter A-Z. The ILP contains two types of binary variables, *link.k.i.j* and *key.i.j*. Their values are constrained so that *link* variables with value 1 make up a single, connected path that assigns a plaintext letter type to each ciphertext token. Here, the *link* variables assign the plaintext ABADCABBY to the ciphertext PGPYAPGGA.

Figure 1 visualizes decipherment using a rectangular lattice. Ciphertext letter tokens (n of them) run along the top, and plaintext letter types (26 of them) run down the left-hand side. Any particular decipherment is a linked sequence of nodes through the lattice, where each node assigns a plaintext type to a ciphertext token. Separately, there is a key connecting plaintext and ciphertext types.[2]

The ILP variables are binary, valued at 0 or 1.

- *link.k.i.j* = 1 iff the decipherment contains a link between plaintext types i and j underneath ciphertext token k. Here, k ranges from 1 to $n-1$, where n is the length of the ciphertext, while i and j range from 0 (A) to 25 (Z).

- *key.i.j* = 1 iff the key maps plaintext letter type i onto ciphertext letter type j.

For a ciphertext of length 50, the ILP will contain $49 \cdot 26 \cdot 26$ link variables and $26 \cdot 26$ key variables, for a total of 33,800 distinct variables.

To restrict the values these variables are allowed to take, we need a number of "Subject to" constraints. First, we ensure that link variables with value 1 form a connected path through the lattice (rather than a scattering of unconnected links), by requiring that the sum of "on" links entering a node equals the sum of "on" links exiting that same node.

$$\forall_{k=1}^{n-2} \; \forall_{i=0}^{25} \; : \; \sum_{j=0}^{25} link.k.j.i = \sum_{j=0}^{25} link.(k+1).i.j$$

Next, we require that the chosen links are consistent with the key:

$$\forall_{k=1}^{n-1} \; \forall_{i=0}^{25} \; : \; \sum_{j=0}^{25} link.k.i.j = key.i.c_k$$

where c_k is the type of the kth cipher token. Of course, this is only useful if the key is restricted to be one-to-one, so we also add:

$$\forall_{i=0}^{25} \; : \; \sum_{j=0}^{25} key.i.j = 1 \quad \Big| \quad \forall_{i=0}^{25} \; : \; \sum_{j=0}^{25} key.j.i = 1$$

By this time, we have also ruled out link settings that represent zero or multiple connected paths. The total number of constraints for a 50-letter cipher comes to 2728.

Lastly, we need a way to prefer one link sequence over another. A reasonable objective is to maximize the probability of the plaintext $p_1 \ldots p_n$, or equivalently, the negative log probability, which we approximate with letter bigram probabilities:

$$P(p_1 \ldots p_n) \sim \sum_{i=1}^{n-1} -logP(p_{i+1}|p_i)$$

$$P(p_{i+1}|p_i) = \frac{\text{count}(p_i p_{i+1})}{\text{count}(p_i)}$$

[2]Throughout, *token* refers to a letter found in running text, such as the second A in KOALABEAR, while *type* refers to a member of some fixed vocabulary, such as a letter in the range A-Z.

Counts are taken over a large plaintext corpus unrelated to the cipher at hand. Fortunately, we can write this objective as a weighted linear combination of the ILP variables:

$$\text{minimize:} \sum_{k=1}^{n-1} \sum_{i=0}^{25} \sum_{j=0}^{25} link.k.i.j \cdot logP(j|i)$$

Given a particular ciphertext, we can now write out the relevant constraints and invoke the solver, which sets the *link* and *key* variables to maximize the objective. Inspecting the variables with value 1 lets us read off the plaintext and key. After that, we can be sure there is no better-scoring plaintext than the one returned.

Finally, if we have additional information about the cipher, we can add it in the form of constraints before solving. For example, we may encode that no letter enciphers to itself:

$key.0.0 = 0$ where 0 stands for A
$key.1.1 = 0$ where 1 stands for B
...

or that space always enciphers as space:

$key.26.26 = 1$ where 26 stands for space

Ravi and Knight (2009) build ILPs for a large number of synthetically-generated ciphertexts and report decipherment accuracy rates under the ILP solver's exact search.

4 ENIGMA

ENIGMA is an electromechanical device that implements a complex substitution cipher. As illustrated in Figure 2, when a plaintext key is pressed on the keyboard, a ciphertext letter lights up in the display area. The wiring of ENIGMA dictates the mapping between plaintext and ciphertext letters. Electricity passes through a *plugboard* (or *steckerboard*), then through a series of three rotors, then through a *reflector*, and finally back through the rotors and plugboard to the cipher-letter display.

This would seem to implement a simple substitution cipher, but each time a keyboard key is pressed, rotors may turn, changing their wiring patterns.

Configuring ENIGMA with a secret key consists of these steps:

1. Select a reflector (inventory: B or C). Each of the two reflectors has a fixed wiring pattern.

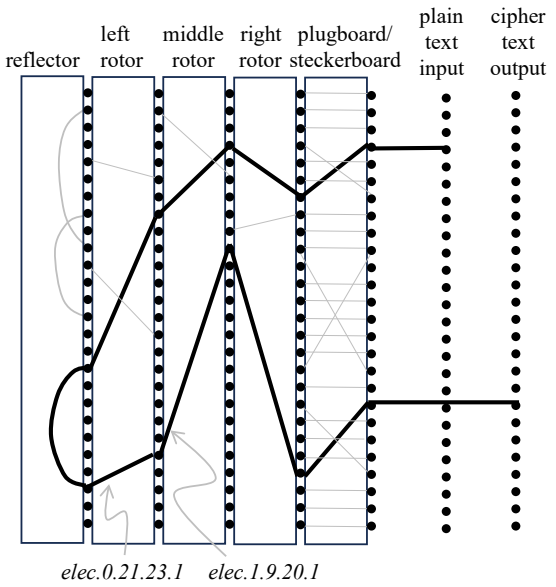

Figure 2: Electricity passing through wires inside ENIGMA. Electrical contact points are represented by black circles. Once a reflector is selected ("B-type" or "C-type"), the wiring inside the reflector is fixed. Likewise, once plugboard wires are inserted, that wiring is also fixed. Each time a plaintext letter is entered on the keyboard, one or more of the rotors will turn, changing the wiring pattern.

2. Select a rotor (inventory: I, II, or III) for the left-hand slot. Each of the three rotor-types has a fixed wiring pattern.
3. Select a ring setting for the rotor. This rotates the internal wiring relative to a physical notch on the rotor.
4. Place the rotor into the left-hand slot and turn it to an initial position, called the indicator setting.[3]
5. Repeat this process for the middle rotor and the right rotor. Each rotor should be of type I, II, or III, and each type should be used exactly once.

By manipulating the ring and indicator settings, we independently decide on the initial wiring pattern and the initial notch position for a rotor. As we type plaintext letters, the right rotor's notch advances. When it reaches the top, it forces the middle rotor to advance, changing *its* wiring pattern and advancing its own notch. When the middle rotor's notch reaches the top, it similarly advances

[3]We will use "position," "indicator," and "indicator setting" interchangeably.

the left rotor. The left rotor has a notch, but since it never engages another piece of ENIGMA, its position is irrelevant. Finally, due to a curious feature called *double-stepping*, the notch on the middle rotor never remains at the top. Once it gets there, it automatically advances again with the next keypress.

When two ENIGMAs are configured in the same way, they can send encrypted messages to each other. Sending and receiving are symmetric—to decrypt, we simply type the ciphertext and read the plaintext off the display.

5 ILP for ENIGMA

This section overviews our ILP for ENIGMA, with Figures 3-8 providing the complete specification. Readers uninterested in the details of the ILP model can skip this section and associated figures, and proceed to Section 6, where we describe tests.

5.1 Variables

Figure 3 shows the variables in the ILP model. All variables are binary, with values 0 or 1. In naming variables and writing constraints, these conventions apply:

- w ranges over the rotor inventory (0 ... 2)
- f ranges over the reflector inventory (0 ... 1)
- p ranges over rotor positions (0 ... 2)
- t ranges over time steps (0 ...)
- i, j range over letter types (0 ... 25).

Variables *elec.p.i.j.t* represent the flow of electricity through the rotors. Figure 2 previously illustrated this for two such variables. Similarly, *elec.u.i.j.t* and *elec.s.i.j.t* model electrical flow through the reflector and plugboard. Taken together, these variables are analogous to the *link* variables of Section 3.

To be precise, *elec.p.i.j.t = 1* if electricity passes through rotor p (0 = left, 1 = middle, 2 = right) between contact point i on the rotor's right side and contact point j on its left side, at time t. Note that i represents the rotor's right side regardless of the direction electricity is flowing.

Electricity between two positions at time t is only possible if there is a physical wire connecting those positions at time t. The presence (or absence) of a wire is modeled by *wire.p.i.j.t*.

The initial setting of rotors is modeled by *init.w.p.i.j* = 1 if a rotor of type w (0 = I, 1 = II, 2 = III) is slotted into position p (0 = left, 1 = middle, 2 = right) with indicator setting i and ring setting j. The reflector selection is captured by *reflector.0* (B-type) and *reflector.1* (C-type). The *init* and *reflector* are connected to *wire.p.i.j.0* and *wire.u.i.j* variables, which capture the state of the machine at time $t = 0$. The initial plugboard wiring is given directly through the values of *wire.s.i.j*.

If the plaintext letter at time t is i, then *plaintext.i.t* = 1, and likewise for *ciphertext.i.t*. The plaintext starts at time $t = 0$, while the ciphertext starts at time $t = 1$. This is because the first electrical circuit is closed only after the first plaintext letter is typed.

The rest of the variables track notch positions and rotor turning, the latter of which results in *wire.p.i.j.t* having a potentially different value from *wire.p.i.j.(t-1)*.

5.2 Constraints

Figure 4 shows constraints controlling electrical flow. First, *elec.p.i.j.t* $\leq$ *wire.p.i.j.t*, so that electricity can flow only if a wire is present. Second, for each contact point, the total electricity coming into the contact point equals the total electricity going out, the same idea as the constraints on *link* variables in Section 3.

Figure 5 gives constraints on the initial configuration of rotors. For example, each rotor from the inventory (I, II, and III) can be selected no more than once, and each rotor slot position (0 = left, 1 = middle, 2 = right) must be filled exactly once. Figures 6, 7, and 8 provide constraints that ensure consistency between the initial ENIGMA configuration, plaintext, ciphertext, and electrical flow between contact positions at each point in time.

In setting up an ILP for a cipher of length 50, we generate a total of 517,266 variables and 417,894 constraints (after de-duplication).

6 Fidelity of the ILP Model

We first test the model in "open mode" by setting the time steps to $n = 4$ and invoking the ILP solver with an empty objective function. The solver easily finds a value for each variable such that the constraints are all satisfied. Note that we have not supplied any plaintext or ciphertext, so finding values for *plaintext.i.t* and *ciphertext.i.t* is part of the solver's job.

Here is a subset of the variables assigned value 1:

- *elec.0.i.j.t* = 1 iff electricity flows from contact i (right side) of left rotor to contact j (left side).
- *elec.1.i.j.t* = 1 (same for middle rotor).
- *elec.2.i.j.t* = 1 (same for right rotor).
- *elec.u.i.j.t* = 1 iff electricity flows between contacts i and j on reflector.
- *elec.s.i.j.t* = 1 iff electricity flows from contact i to contact j on steckerboard.
- *init.w.p.i.j* = 1 iff rotor type w slots into position p with indicator setting i and ring setting j.
- *wire.s.i.j* = 1 iff the plugboard connects position i (on keyboard side) with position j
- *reflector.f* = 1 indicates choice of reflector type B ($f = 0$) or C ($f = 1$)
- *wire.u.i.j* = 1 iff a reflector wire connects contact position i with j
- *plaintext.i.t* = 1 iff tth plaintext letter (starting at $t = 0$) is ith letter of alphabet (a=0 ... z=25)
- *ciphertext.i.t* = 1 iff tth ciphertext letter (starting at $t = 1$) is ith letter of alphabet
- *notch.p.i* = 1 iff the notch on the pth rotor (0 ... 2) is initially at position i
- *notches.i.j.t* = 1 iff if the notch on the middle and right rotors are at position i and j at time t
- *wire.p.i.j.t* = 1 iff the pth rotor has a wire connecting contact i (right side) to j at time t
- *rotate.p.t* = 1 iff the pth rotor rotates at time t
- *wire.p.i.j.t.rotate* = 1 iff the wire connecting i and j is inside rotor p that rotates at time t
- *wire.p.i.j.t.norotate* = 1 iff the wire between i and j is inside non-rotating rotor p at time t

Figure 3: Variables in the ILP model for ENIGMA.

$$\forall_{t,p,i,j} \; : \; elec.p.i.j.t \leq wire.p.i.j.t$$

$$\forall_{t,p,i,j} \; : \; elec.s.i.j.t \leq wire.s.i.j \quad \Big| \quad \forall_{t,p,i,j} \; : \; elec.u.i.j.t \leq wire.u.i.j$$

$$\forall_{t,i} \; : \; plaintext.i.t + ciphertext.i.t = \sum_j elec.s.i.j.t$$

$$\forall_{t,i} \; : \; \sum_j elec.s.j.i.t = \sum_j elec.2.i.j.t \quad \Big| \quad \forall_{t,i} \; : \; \sum_j elec.2.j.i.t = \sum_j elec.1.i.j.t$$

$$\forall_{t,i} \; : \; \sum_j elec.1.j.i.t = \sum_j elec.0.i.j.t \quad \Big| \quad \forall_{t,i} \; : \; \sum_j elec.0.j.i.t = \sum_j elec.u.i.j.t$$

$$\forall_{t,i} \; : \; \sum_j elec.u.j.i.t = \sum_j elec.0.i.j.t \quad \Big| \quad \forall_{t,i} \; : \; \sum_j elec.0.i.j.t = \sum_j elec.1.j.i.t$$

$$\forall_{t,i} \; : \; \sum_j elec.1.i.j.t = \sum_j elec.2.j.i.t \quad \Big| \quad \forall_{t,i} \; : \; \sum_j elec.2.i.j.t = \sum_j elec.s.j.i.t$$

$$\forall_{t} \; : \; \sum_i plaintext.i.t = 1 \quad \Big| \quad \forall_{t} \; : \; \sum_i ciphertext.i.t = 1$$

Figure 4: Constraints on electrical flow, ensuring a single connected path such as that shown in Figure 2. Note that this formulation includes redundant constraints to emphasize the bidirectional electrical flow; redundancies are removed by the ILP solver in pre-processing.

$$\forall_w : \sum_{p,m,r} init.w.p.m.r \leq 1 \quad \Big| \quad \forall_p : \sum_{w,m,r} init.w.p.m.r = 1$$

Figure 5: Constraints on initial rotor selection and placement, ensuring no rotor type is selected more than once, and that no rotor position is left empty or filled with multiple rotors.

$$\sum_i wire.s.i.i \geq 26 - 2 \cdot \text{steckermax} \quad \Big| \quad \sum_i wire.s.i.i \leq 26 - 2 \cdot \text{steckermin}$$

$$\forall_i : \sum_j wire.s.i.j = 1 \quad \Big| \quad \forall_{i<j} : wire.s.i.j = wire.s.j.i$$

$$\sum_f reflector.f = 1 \quad \Big| \quad \forall_{f,i} : wire.u.i.R[f]_i \leq reflector.f \quad \Big| \quad \forall_i : \sum_j wire.u.i.j = 1$$

$$\forall_{p,i,j} : wire.p.i.j.0 = \sum_{w,m,r} init.w.p.m.r \quad \text{(add to sum only when } P[w]_{i+m-r} = j+m-r\text{)}$$

```
Note:
    P[0] = rotorIwiring     = EKMFLGDQVZNTOWYHXUSPAIBRCJ
    P[1] = rotorIIwiring    = AJDKSIRUXBLHWTMCQGZNPYFVOE
    P[2] = rotorIIIwiring   = BDFHJLCPRTXVZNYEIWGAKMUSQO
    R[0] = reflectorBwiring = YRUHQSLDPXNGOKMIEBFZCWVJAT
    R[1] = reflectorCwiring = FVPJIAOYEDRZXWGCTKUQSBNMHL
```

Figure 6: Constraints on initial wiring of plugboard, reflector, and rotors, bounding the amount of steckering, and ensuring wiring is consistent with the initial configuration of the rotors and reflector.

$$\forall_{p,i} : notch.p.i = \sum_{w,j} init.w.p.Q[w] - i \pmod{26}.j$$

$$\forall_i : notch.1.i = \sum_j notches.i.j.0 \quad \Big| \quad \forall_i : notch.2.i = \sum_j notches.j.i.0$$

$$\forall_{t,i,j} : notches.i.j.t+1 = \begin{cases} 0 & \text{if } i = 0 \text{ and } j < 25 \\ notches.i + 1.0.t & \text{else if } j = 25 \\ notches.25.j+1.t + notches.0.j+1.t & \text{else if } i = 25 \\ notches.i.j+1.t & \text{otherwise} \end{cases}$$

```
Q[w] is notch position of rotor w, namely:
    Q[0] = Q     Q[1] = E     Q[2] = V
```

Figure 7: Constraints on notch positions, ensuring that notches move when rotors turn. When a notch reaches top position, it engages the rotor to the left, forcing it to turn. In addition, the middle rotor "double steps," as its notch cannot remain in top position. The update is complicated by the fact that while each configuration of rotors has a deterministic successor, some have two possible predecessors.

$$\forall_t \; : \; rotate.2.t = 1$$

$$\forall_t \; : \; rotate.1.t = \sum_{i,j} notches.i.j.t \text{ such that } i = 0 \text{ or } j = 0$$

$$\forall_t \; : \; rotate.0.t = \sum_{i} notches.0.i.t$$

$$\forall_{t,p} \; : \; \sum_{i,j} wire.p.i.j.t.rotate = 26 \cdot rotate.p.t$$

$$\forall_{t,p,i,j} \; : \; wire.p.i.j.t = wire.p.i.j.t.rotate + wire.p.i.j.t.norotate$$

$$\forall_{t,p,i,j} \; : \; wire.p.i.j.t = wire.p.i+1.j+1.t-1.rotate + wire.p.i.j.t-1.norotate$$

Figure 8: Constraints on rotor turning, ensuring that physical locations of rotor wires are updated appropriately.

init.2.0.21.0	*init.0.1.24.17*	*init.1.2.4.23*
reflector.0	*wire.s.2.3*	*wire.s.3.2*
plaintext.24.0	*plaintext.12.1*	*elec.0.24.5.1*
ciphertext.12.1	*ciphertext.8.2*	...

We can interpret this as follows:

Left rotor III, indicator V, ring setting A
Middle rotor I, indicator Y, ring setting R
Right rotor II, indicator E, ring setting X
AB CD EF MO NP QR ST UV WX YZ
Reflector B
Plaintext: (A) Y M D W
Ciphertext: M I A V (A)

Or, more compactly:

III I II (Reflector B) VYE ARX
AB CD EF MO NP QR ST UV WX YZ
YMDW → MIAV

Using an independent ENIGMA simulator GUI, we confirm that plaintext YMDW is encrypted as MIAV under these settings.

We also want to test edge cases like *double stepping*. A quick test is to add these two constraints:

notch.2.0 = 1	right rotor
notch.1.1 = 1	middle rotor

The first constraint requires the right rotor to be in top notch position 0, ready to engage the middle rotor on the first keypress. The middle rotor starts in notch position 1. The solver now yields:

I III II (Reflector B) RUE ACT
AB CD EF MO NP QR ST UV WX YZ
WQYV → HLGG

An independent ENIGMA simulator confirms the double-stepping action of the machine as it encrypts WQYV as HLGG.

7 Experiments

In this section, we use our ILP to answer questions about ENIGMA.

7.1 A Constant Sequence Question

> *Question 1:* What is the largest n for which an unsteckered ENIGMA can be configured to encipher A^n as Z^n?

To prohibit plugboard wires, we add 26 constraints of the form *wire.s.i.i* = 1, one for each i from 0 to 25 (A-Z). We constrain *plaintext* and *ciphertext* variables to hold various amounts of As and Zs.

This configuration encrypts AAAA as ZZZZ:

II I III (Reflector C) VQS KVJ
AAAA → ZZZZ

However, there is no configuration that encrypts AAAAA as ZZZZZ. When we ask for a solution, the ILP solver simply returns Infeasible.[4]

If we allow steckering, we *can* find solutions for $n > 4$. For example:

[4] All runs use v13.0.1 Gurobi Optimization (2024) on an Intel Core i9-13950HX CPU laptop with 128 GB RAM, running Ubuntu 22.04.5 LTS. The open-source HiGHS 1.14.0 is slower, failing to solve AAAA → ZZZZ within 3 hours.

II I III (Reflector B) QQT KTY
AM CJ EP FV GQ HR LZ OT SW UY
AAAAA → ZZZZZ

II III I (Reflector B) RUQ OEC
BP CD FJ GX HY KS MQ NW OU RZ
AAAAAA → ZZZZZZ

7.2 An Eleven-Letter Question

Question 2: Is there an ENIGMA configuration that encrypts KEVINKNIGHT as ADOLFHITLER, using the standard German Army practice of 10 plugboard wires?

We cast this question as an equivalent maximization problem.

Question 3: Given plaintext KEVINKNIGHT, what ENIGMA setting generates a ciphertext that maximally resembles ADOLFHITLER?

We supply the plaintext KEVINKNIGHT and add this objective function:

ciphertext.0.1 +	(first letter = A)
ciphertext.3.2 +	(second letter = D)
ciphertext.14.3 +	(third letter = O)
...	
ciphertext.17.11	(eleventh letter = R)

As the ILP solver strives to maximize this objective, it emits a series of increasingly better solutions, ultimately answering *Question 2* in the affirmative:

III I II (Reflector B) URF ZKY
AB CD EF MO NP QR ST UV WX YZ
KEVINKNIGHT → **LDQBMADZNEA**
Objective value: 1

I II III (Reflector B) QDZ HXF
AS BW CE DR FL GV IX JT KO NZ
KEVINKNIGHT → **SAOLFXITVER**
Objective value: 7

III II I (Reflector B) AYR DEA
AS CL DO ER FG HZ IY KX NQ TU
KEVINKNIGHT → **ASOLFHHTLLR**
Objective value: 8

I II III (Reflector B) ZGL QAR
AV BC DG EY FL HM IX JT NZ OS
KEVINKNIGHT → **ADOLFHITQER**
Objective value: 10

III II I (Reflector B) ADP FXK
AU BL CI EV FR GW HS JN OT QZ
KEVINKNIGHT → **ADOLFHITLER**
Objective value: 11

Question 4: Is the solution unique? Are there other configurations that also encrypt KEVINKNIGHT as ADOLFHITLER with 10 plugboard wires?

There are several ways to answer this question. One is to add the following constraint and re-run the solver, preventing it from choosing the same rotor settings:

$$init.0.2.15.10 + init.1.1.3.23 + init.2.0.0.5 < 3$$

Another is to ask the solver for multiple optimal solutions, a strategy that tends to yield only minor changes in plugboard wiring:

III II I (Reflector B) ADP FXK
AU BL CI EV FR GW HS JN **MZ** OT
KEVINKNIGHT → **ADOLFHITLER**

Or, we can insist on using Reflector C instead of B (by adding constraint *reflector.1* = 1), in which case we get yet another solution:

I II III (Reflector C) QER TYQ
AG BT CV DO FX HQ IS KW LN RU
KEVINKNIGHT → ADOLFHITLER

Question 5: What ENIGMA configuration encrypts KEVINKNIGHT as ADOLFHITLER with the fewest number of plugboard wires?

Here, we simply maximize the sum of *wire.s.i.i* variables, i.e., one for each letter that the steckerboard maps to itself. Although this objective is easy to state, it is difficult for the ILP solver to optimize. We interrupt the solver after this 6-wire solution:

II I III (Reflector C) VRR AKO
AL DS HP JK TX VW
KEVINKNIGHT → ADOLFHITLER
38.4hrs

Because of the interruption, we cannot rule out a solution with fewer plugwires. However, a similar query for the shorter pair KEVIN → ADOLF terminates with one plugwire, so we at least conclude that neither name pair can be mapped by an unsteckered ENIGMA.

7.3 Lazy Plaintext

During World War II, British codebreaker Mavis Lever noticed a long Italian Navy ENIGMA ciphertext message without any L's. ENIGMA cannot encode a letter as itself, so she assumed the Italian operator had sent a test message by lazily choosing a plaintext of repeated L's (Woo, 2013).

> *Question 6:* How long must such a "lazy" ENIGMA intercept be to permit key recovery?

First, we randomly construct an unsteckered key and use it to encrypt L* = LLLLL ...

III I II (Reflector C) KFS ISC
L* → JUJOFGZQGVPAGPGGRGI
MDRYKRGOABAMAWTRETW
JZPBVUCTKKQYDCZGDNY
TNRWPQEZHZONANSJCUW
AJSIEZOWUOCJDGMKAIB
USYJYMDPZMNWRIDFDFP
QAOKEGDVUIKIAKVUOUH ...

Given an intercepted plaintext/ciphertext pair of length 3 *only* (namely, LLL → JUJ), the ILP solver guesses:

I II III (Reflector C) IDV PFU
LLL → JUJ M...

The proposed solution is consistent with LLL/JUJ, but it fails to match the original key and incorrectly encrypts a fourth L as M instead of O.

Below are keys recovered from intercepts of lengths 5, 8, and 14, respectively. The third key's encryption of the next 100 Ls matches the key exactly.

III I II (Reflector C) TQW SEG
LLLLL → JUJOF GZQ R...
5.2hrs

III I II (Reflector C) SBF QOP
LLLLLLLL → JUJOFGZQ GVPA R...

III I II (Reflector C) TQS SEC
LLLLLLLLLLLLL → JUJOFGZQGVPAGP
14.8hrs

Now we make a different key, this time using ten stecker wires:

III I II (Reflector C) KFS ISC
AF CN DY EB GH IR JQ KL OT PS
L* → HSTSTTQGBSXJZGAGVZST ...

With an intercept of length 10, the solver's solution achieves the goal, although it is totally unrelated to the original key:

I III II (Reflector B) IUD ANW
AE BM GY HO IQ JT LV NP RX SW
LLLLLLLLLL → HSTSTTQGBS P W R

As discussed by Ostwald and Weierud (2017), accurate key recovery for such a short plaintext/ciphertext pair is impossible in the presence of steckering.

7.4 Crib Attack Largely Unsuccessful

In a practical situation, we may intercept a long ciphertext with a guess as to how some small part of it decodes. For example, consider a 32-letter ciphertext generated by an unsteckered ENIGMA, plus a guessed decipherment (or *crib*) of the first seven letters:

```
CT: QYPUQNWNBBVHRMANZCWQSLBYSSSDQPZN
PT: FARBEIT????????????????????????
```

In this case, we want to recover a key that:

- is consistent with the crib
- yields sensible plaintext when applied to the entire cipher

We can enforce the former with ILP constraints (as before) and encourage the latter with an ILP objective. Perhaps the simplest approximation to "sensible plaintext" is a unigram objective function:

$$\text{minimize: } \sum_{t=1}^{n} \sum_{i=0}^{25} plaintext.i.t \cdot logP(i)$$

where P(0) is the probability of A, P(1) is the probability of B, etc. Because log probabilities are negative, we minimize instead of maximize.

Question 7: Which key setting yields the most probable plaintext for the 32-letter cipher above?

The ILP solver is unable to find a solution after four days of runtime. However, if we fix the middle rotor to match the key (rotor I, position F, ring setting S), the solver decides to optimize the "English-ness" of the plaintext by finding a very good ENIGMA setting:

```
CT: QYPUQNWNBBVHRMANZCWQSLBYSSSDQPZN
PT: FARBEITFROMMETOPAINTAROSYPICTURE
 III I II (Reflector C) AFT YSD
 Objective: 141.54
```

Allowing for the possibility of steckered configurations, however, confounds the crib attack. The ILP solver finds a setting that produces a plaintext completion with a better letter-unigram score, but which is gibberish:

```
CT: QYPUQNWNBBVHRMANZCWQSLBYSSSDQPZN
PT: FARBEITDUGNEISDHSEAMRWOHITDTNMAK
 II III I (Reflector B) UVK WOY
 AR BL CI DZ EP FN GM OT QU SX
 Objective: 114.13
```

Question 8: Can a higher-order n-gram model distinguish the correct ENIGMA key from others?

Unknown. The answer boils down to whether two different ENIGMA settings could produce two distinct yet equally-sensible plaintexts. In theory, the ILP solver could answer this question by returning the top N solutions under a strong English objective function, but it is not fast enough in practice. However, longer ciphers are known to succumb to simple letter n-gram scoring.

8 Ciphertext-Only Attack

In the previous section, the crib serves to narrow down the possible ENIGMA keys, while the objective function prefers keys that generate sensible plaintexts. Of course, nothing stops us from simply deleting the crib constraints.

Question 9: Can ILP solve a long ENIGMA cipher without a crib?

It seems not, at least for the current model and search algorithm; the runtimes for ILP ciphertext-only attacks with n-gram English models are prohibitive.

9 Conclusion

By modeling ENIGMA as an integer linear program (ILP), we are able to ask questions about the machine very simply, by adding constraints on the plaintext, ciphertext, and/or settings of the machine. These questions can then be answered by a generic ILP solver without any special-purpose programming. The flexibility to constrain any part of the machine's operation makes it easy to explore.

At the same time, we find that for more difficult questions, the ILP solving speed decreases and its genericity/optimality can become liabilities rather than strengths. We would like to experiment in future work on whether significant speedups can be obtained by changes to the ILP model or the search parameters of a generic solver. Just for example, an ILP designer can suggest which variables might be given higher priority in early branching.

All software used in this paper appears at `github.com/kevincrawfordknight/enigma-ilp`.

10 Acknowledgments

We thank the anonymous reviewers for their valuable comments.

References

Anthony J. Bagnall, Geoff P. McKeown, and Victor J. Rayward-Smith. 1997. The cryptanalysis of a three rotor machine using a genetic algorithm. In *Proceedings of the Seventh International Conference on Genetic Algorithms*, pages 712–718. Morgan Kaufmann.

George B. Dantzig. 1949. Programming of interdependent activities: II mathematical model. *Econometrica*, 17(3):200–211.

James J. Gillogly. 1995. Ciphertext-only cryptanalysis of Enigma. *Cryptologia*, 19(4):405–413.

Gurobi Optimization. 2024. Gurobi Optimizer Reference Manual.

Nils Kopal and Bernhard Esslinger. 2022. New Ciphers and Cryptanalysis Components in CrypTool 2. In *Proceedings of the 5th International Conference on Historical Cryptology, HistoCrypt 2022*, pages 127–136. Linköping University Electronic Press.

George Lasry, Nils Kopal, and Arno Wacker. 2019. Cryptanalysis of Enigma double indicators with hill climbing. *Cryptologia*, 43(4):267–292.

Olaf Ostwald and Frode Weierud. 2017. Modern breaking of Enigma ciphertexts. *Cryptologia*, 41(5):395–421.

Sujith Ravi and Kevin Knight. 2009. Attacking decipherment problems optimally with low-order n-gram models. *Cryptologia*, 33(4).

Marian Rejewski. 1984. How Polish mathematicians deciphered the Enigma. In Władysław Kozaczuk, editor, *Enigma: How the Poles Broke the Nazi Code*, pages 1–33. University Publications of America, Frederick, MD.

Åvald Åslagson Sommervoll and Leif Nilsen. 2021. A genetic algorithm attack on Enigma's plugboard. *Cryptologia*, 45(3):194–226.

Geoff Sullivan and Frode Weierud. 2005. Breaking German Army Ciphers. *Cryptologia*, 29(3):193–232.

Gordon Welchman. 1982. *The Hut Six Story: Breaking the Enigma Codes*. McGraw–Hill, New York.

Elaine Woo. 2013. Mavis Batey dies at 92; renowned code-breaker for Britain in WWII. *Los Angeles Times* obituary, November 23, 2013.