Playfair crib validation as a constraint satisfaction problem

Magnus Ekhall
Private researcher
magnus.ekhall@gmail.com

Abstract

This paper shows how a crib[1] for a Playfair enciphered message can be seen as a constraint satisfaction problem. This problem can then be solved with standard constraint programming tools. The solution will tell whether the crib is possible, and if so, can list all possible Playfair keys that would result in the given crib.

1 Introduction

The Playfair cipher, invented by the physicist and inventor Sir Charles Whetstone in the 19th century, is a bigram substitution cipher. As a key, 25 out of the 26 letters of the alphabet are written into a matrix of size 5×5. Usually the letter J is the omitted letter, with I used as a substitute (Kahn, 1996).

Playfair is a bigram cipher which means that it always enciphers or deciphers pairs of letters. Also, a letter pair cannot consist of the same letter. If needed, a padding letter (usually X) can be used to work around these situations. Example: If the plaintext that shall be enciphered is `MOONCHEESE` the padded plaintext letter pairs would be `MO ON CH EX ES EX`.

1.1 The three Playfair rules

The actual encipherment is carried out using three rules which will be referred to as the Playfair rules. The letter pair to be enciphered are looked up in the key matrix. If the letters happen to be on the same row in the key matrix, rule 1 applies. If the letters instead are in the same column in the key matrix, rule 2 applies. If the letters are neither on the same row nor in the same column, rule 3 applies.

[1] A crib is a known or guessed plaintext for (part of) an enciphered message.

K	T	A	F	D
M	X	O	Y	Q
C	L	Z	E	U
S	G	V	P	I
W	R	N	B	H

Table 1: A Playfair key

Rule 1: For each of the two plaintext letters, take the letter to the right in the key matrix. This is done in a cyclic manner.

Rule 2: For each of the two plaintext letters, take the letter directly under in the key matrix.

Rule 3: For each of the two plaintext letters, take the letter on the same line of the key matrix that also is on the same column as the other plaintext letter.

Using the key in table 1 the example plaintext above would be enciphered to:

`MO` $\rightarrow$ `XY` (Rule 1)

`ON` $\rightarrow$ `ZA` (Rule 2)

`CH` $\rightarrow$ `UW` (Rule 3)

`EX` $\rightarrow$ `LY` (Rule 3)

`ES` $\rightarrow$ `CP` (Rule 3)

`EX` $\rightarrow$ `LY` (Rule 3)

Deciphering is done in a similar fashion. When deciphering, rule 1 selects the letter to the left, and rule 2 the letter above. Rule 3 works just as when enciphering.

The three Playfair rules together with the nature of the key matrix imposes some constraints on what a plaintext letter can be enciphered into. For example, a plaintext letter can only be enciphered to five different cipher text letters: the four others on the same row as the plaintext letter and the letter directly under it. The constraints will be in focus in this paper.

Proceedings of the 8th International Conference on Historical Cryptology, pages 52-59
Poznań, Poland, 16-18 June, 2025

-	-	A	-	-
M	X	O	Y	-
-	-	Z	-	-
-	-	-	-	-
-	-	N	-	-

Table 2: A partial Playfair key

-	Z	-	-	-
-	-	-	-	-
-	N	-	-	-
-	A	-	-	-
X	O	Y	-	M

Table 3: An equivalent Playfair key

1.2 Related work

Modern cryptanalysis of Playfair enciphered messages involve computer algorithms such as hill climbing. This works most reliably with long messages and can be used as a ciphertext-only attack. For shorter messages other methods can be tried for example by testing whether certain words would be possible as the plaintext in different manners (Dunin et al., 2022).

When performing cryptanalysis with pen-and-paper methods, both cribs and taking advantage of the constraints imposed by the Playfair rules have been commonplace (US Government, 1990; Friedman, 1938).

1.3 Problem statement

This paper aims to show how the common pen-and-paper method for breaking Playfair messages (US Government, 1990) can be viewed as a constraint satisfaction problem and, as such, can be computerized using standard constraint programming tools. More specifically this applies to the tedious work related to calculating possible Playfair keys given a certain crib for an enciphered message.

2 Playfair crib validation

Whether a crib is possible for a Playfair enciphered text is determined by if there is at least one Playfair key that would encrypt the crib to the ciphertext. Such a key can be "partial" in the sense that only those letters used in the crib and its corresponding ciphertext need to be present in the key matrix.

For example, if the crib `MOON` is to be tested at the beginning of the ciphertext `XYZAUWLYCPLY`, then the partial key shown in table 2 would show that the crib is possible.

When a partial key derived from a crib is used to decrypt a message it might not be possible to decrypt the whole message. Only those bigrams where the plaintext letters and the corresponding ciphertext letters are present in the key matrix will be possible to decrypt. The crib itself, however, will always be possible to decrypt.

How many different keys exist for a given crib? In order to answer that question certain properties of the Playfair key should be considered.

A Playfair key matrix shifted horizontally or vertically results in a new matrix that encrypts or deciphers exactly the same as the original. All such keys are therefore equivalent with respect to decryption or encryption. As an example, table 3 shows a key which is equivalent to that shown in table 2, shifted to the left once and up twice.

The pen-and-paper method for deriving partial Playfair keys from a crib described in (US Government, 1990) is an iterative process.

It can be explained as follows:

Algorithm 1: Keys from Playfair crib

An empty key, k_0, is created as the only member of a set, K, of current keys.
The crib is processed bigram by bigram. For each bigram, all keys in the set K are examined, one by one. For each of the keys it is tested whether one or more of the three Playfair rules can be used to produce the current bigram from the crib. This may involve adding new letters to the key and can result in up to three new keys: one for each of the three Playfair rules that could successfully be used. The conclusion could also be that none of the three Playfair rules can be used to produce the bigram. In that case that particular key is removed. Otherwise, any new keys are kept as the new members of the key set K to be used in the next iteration.
In the first iteration, only k_0 is present but the number of possible keys can grow exponentially.
The algorithm ends when either the key set K becomes empty or all bigrams of the crib

have been successfully used to construct prospective keys.
If the key set ends up empty, this indicates that it is impossible to construct a key that decrypts the ciphertext into the crib, proving the crib incorrect.
Conversely, if there are keys in the key set at the end of the last iteration that shows that it is possible to decrypt the ciphertext into the crib at the given position, but there is no guarantee that the crib corresponds to the original plaintext.

A difficulty with algorithm 1 is that it is challenging to say exactly where in the key matrix the attempted use of the different Playfair rules shall be placed. In (US Government, 1990) this decision is left to the reader to handle. This problem can be handled by introducing the constraint satisfaction problem that stems from the Playfair rules.

3 Constraints from Playfair rules

The three Playfair rules imposes three different sets of constraints on the letters they operate on. All constraints are cyclic or modular with respect to the five-by-five size of the Playfair key matrix.

3.1 Rule 1 constraints

Rule 1 applies when both letters of plaintext are on the same line in the key matrix. When enciphering, the ciphertext letters are the letters immediately to the right of the plaintext letters.

For this to be possible the following must be true:

- The two ciphertext letters must be on the same row in the matrix.
- The first ciphertext letter must be directly to the right of the first plaintext letter.
- The second ciphertext letter must be directly to the right of the second plaintext letter.

3.2 Rule 2 constraints

Rule 2 is similar to rule 1, but here the two plaintext letters are in the same column in the matrix.

For this to be possible the following must be true:

- The two ciphertext letters must be in the same column in the matrix.
- The first ciphertext letter must be directly below the first plaintext letter.
- The second ciphertext letter must be directly below the second plaintext letter.

3.3 Rule 3 constraints

If the two plaintext letters are not on the same line nor in the same column, this produces the following constraints:

- The ciphertext letters must be on different rows in the matrix.
- The ciphertext letters must be in different columns in the matrix.
- The first plaintext letter must be on the same row as the first ciphertext letter.
- The second plaintext letter must be on the same row as the second ciphertext letter.
- The first plaintext letter must be in the same column as the second ciphertext letter.
- The second plaintext letter must be in the same column as the first ciphertext letter.

3.4 Stragglers

Sometimes a crib does not produce complete bigrams. The beginning or end of the crib may end up so that only one plaintext letter is present in a bigram. Take for example the crib `FOR` with the enciphered message from the example above. The two bigrams involved are then:

```
MR BY ...
FO R
```

These “stragglers” produce what can be seen as half bigrams. In this case, since only one plaintext letter is available, the constraints listed above do not fully apply to stragglers. Instead a reduced set of constraints can be used in this case: the constraints that involves the missing plaintext letter are simply ignored.

4 Constraint satisfaction problem

When implementing algorithm 1 on a computer, the constraints described in section 3 can be used to handle the fact that the precise positioning of the letters of the bigrams in the prospective Playfair key are not known. Even if the position itself is not known, the constraints imposed by the Playfair rules still apply and must hold true.

0	1	2	3	4
5	6	7	8	9
10	11	12	13	14
15	16	17	18	19
20	21	22	23	24

Table 4: Integers representing the positions in the key matrix.

This is a constraint satisfaction problem, which is a well-studied field. There are several computer implementations of tools to solve this kind of problem which go under the name "constraint programming".

4.1 Constraint satisfaction problem example

In order to illustrate what a constraint satisfaction problem can look like here follows a short (and unrelated) example:

Consider the task of assigning integers to three variables under specific constraints:

- **Variables:** X, Y, Z
- **Domains:** Each variable takes a value from $\{1,2,3,4,5\}$.
- **Constraints:**
  - $X + Y = Z$
  - $X < Y$
  - $X \neq Y, Y \neq Z, X \neq Z$

The goal is to find integer assignments to X, Y, and Z that satisfy all the given constraints simultaneously.

This problem has several solutions, for example $X = 2, Y = 3, Z = 5$.

4.2 Playfair crib constraints

To efficiently be able to describe the constraints in section 3 with a constraint programming tool it is helpful to use integers as references to specific places in the key matrix.

With the representation as in table 4 the various constraint types used in section 3 can be described mathematically in a way that many constraint programming tools use:

- x is on the same row as y: $\lfloor x/5 \rfloor = \lfloor y/5 \rfloor$
- x is in the same column as y: $x \bmod 5 = y \bmod 5$
- x is directly to the right of y: $\lfloor x/5 \rfloor = \lfloor y/5 \rfloor \wedge x \bmod 5 = ((y \bmod 5) + 1) \bmod 5$
- x is directly under y: $x \bmod 5 = y \bmod 5 \wedge \lfloor x/5 \rfloor = (\lfloor y/5 \rfloor + 1) \bmod 5$

5 Implementation

An implementation was made in the Python programming language. The Python library CPMpy (Guns, 2019) was used as an interface to a OR-Tools, a free and open source software suite developed by Google. OR-Tools includes an award winning SAT-solver for solving constraint programming problems (Perron and Furnon, 2024).

In CPMpy, a "model" consists of a set of decision variables and a set of constraints. With the representation of table 4 all decision variables used are integers $v \in [0,24]$, where each v is unique. A model can then be solved by the SAT-solver, which means that the SAT-solver will try to find specific values for the decision variables that fulfills all the constraints.

As input, the program takes a Playfair enciphered message and a crib for a part of the message. The program implements algorithm 1 using the mathematical constraints as described in section 4.

The program start with an empty set of models. For the first bigram of the crib three new models are created: one for each of the three Playfair rules. The newly created models are given decision variables corresponding to the letters involved in the plaintext and ciphertext bigram, as well as the constraints stemming from the Playfair rules.

For each of the remaining bigrams all models are tested with new decision variables and constraints from each of the three Playfair rules. If the SAT-solver can find at least one solution, the model is kept in the set of models. If not, the model is thrown away.

If this intermediate testing was not done, the size of the model set would grow exponentially which would make it difficult to handle cribs with more than a few bigrams.

When the last bigram of the crib has added, the program instructs the SAT-solver to retrieve all possible solutions for every model in the model set.

This results in the output of the program: a list of all possible partial keys for which the ciphertext deciphers to the crib. The program will also decrypt as much as possible of the ciphertext with

M	Z	N	X	B
C	V	L	A	K
S	H	D	G	F
P	Q	O	W	I
E	U	R	Y	T

Table 5: Key used in the example

E	U	R	Y	T
M	.	N	X	B
.	.	D	G	.
C	.	L	A	.
.	.	.	.	.

Table 6: One of the partial keys output from the example

every partial key. From this output it is sometimes possible to draw conclusions on plaintext letters *outside* the crib as will be shown in an example below.

As an alternative, the program can be instructed to stop as soon as one solution has been found. In that case the program will only answer the question whether it is possible for the crib to exist and present one example of a partial key that satisfies the crib.

5.1 Optimizations

The implementation also takes advantage of the fact that a Playfair key matrix can be shifted vertically and horizontally without affecting the result, as showed in section 2. This is done by fixing one of the letter variables, l, to a fixed position in the key matrix: $l = 0$. It does not matter which letter that is selected, or what position in the letter matrix it is fixed to. The fixing of one letter reduces the key space with a factor of 25.

One further optimization can be done. The order in which the various bigrams from the crib are tested matters. To reach a conclusion as quickly as possible it is advantageous to try to get a contradiction as soon as possible. Therefore the program first looks at all the available bigrams and starts with those that have many letters in common. This increases the probability to get contradicting constraints which in turn will reduce the number of models that the SAT-solver will need to investigate.

6 Examples

Below are two examples, the first with a plausible (and in fact correct) crib which results in a number of partial keys. The second example is with a crib that turns out to be impossible.

Both example use the plaintext message "Next telegram must be enciphered":

`NEXTTELEGRAMMUSTBEENCIPHERED`.

Enciphered with the key in table 5 the enciphered message becomes:

`MRBYEUCRDYCXZEFEMTRMKPQSUYRS`.

6.1 Plausible crib

If the (correct) crib `NE XT TE LE GR AM` is tested, the following sequence of events occur:

1. The first bigram chosen from the crib is `TE` which is matched against ciphertext bigram `EU`. After this step, the number of models in the model set is 2. That means that there are two different sets of constraints that both independently can produce the crib `TE` from the ciphertext `EU`.
2. Second bigram `NE` $\rightarrow$ `MR`. Model set size: 4.
3. `LE` $\rightarrow$ `CR`. Model set size: 4.
4. `XT` $\rightarrow$ `BY`. Model set size: 7.
5. `GR` $\rightarrow$ `DY`. Model set size: 1.
6. `AM` $\rightarrow$ `CX`. Model set size: 1.

As can be seen, the number of plausible models first increases, but later decreases when more bigrams and therefore more constraints are added.

The execution time on a normal PC with this example is about 0.5 seconds, and the output consists of 48 partial keys. The first of the partial keys is shown in table 6.

The partial decrypt with the key from table 6 is `NEXTTELEGRAM....BEEN....ER...` As can be seen three bigrams outside the crib have been deciphered with this particular key. The "extra" bigrams can vary with the various partial keys. If all 48 partial keys are considered it can be noted that the following plaintext is common for all keys: `NEXTTELEGRAM....BEEN....E....` That means that if the crib `NEXTTELEGRAM` is correct, then it is certain that the other letters are also correct. Given the "extra" letters, new words can be guessed and the crib thus expanded until the whole message is deciphered. If unlikely letter combinations appears as "extra" letters it can be suspected that

even if the crib was possible, it may be the wrong solution.

6.2 Impossible crib

For this example the crib "next message": `NE XT ME SX SA GE` is used with the same ciphertext as in section 6.1.

The sequence of events becomes as follows:

1. The first bigram chosen from the crib is `ME` which is matched against ciphertext bigram `EU`. After this step, the number of models in the model set is 2.
2. Second bigram `GE` → `CX`. Model set size: 4.
3. `NE` → `MR`. Model set size: 2.
4. `SX` → `CR`. **Model set size: 0.**

After four of the six bigrams have been added there are no models left that satisfy the constraints. The crib can be proven to be impossible and the program is terminated. Execution time about 0.35 seconds.

7 Results

To test the efficiency of the method with different cribs, a list of 1000 common English words with a length of at least two letters was created (Fletcher, 2003 2010). The words were then used as a crib at the beginning of the example message from section 6, one word at a time. The time it took for the program to come to a conclusion (either positive or negative) was recorded and is shown in figure 1. It can be seen that the median time to reach a conclusion is below half a second. The worst case is 2.45 seconds and the word in this case, `TALKING`, has no repeated letters, nor does it have any letters in common with the corresponding ciphertext. Of the 1000 words, 678 were found to be possible cribs with at least one partial key. The word length varied from two to thirteen letters, with the median being five.

The results from figure 1 can be partitioned into two parts: the possible cribs that were tested and the impossible cribs. This is shown in figures 2 and 3 respectively. It can be noted that the cribs with the longest processing time were the possible ones. The reason is that in the impossible case the algorithm will stop as soon as a contradiction is encountered, while in the possible case all bigrams need to be processed.

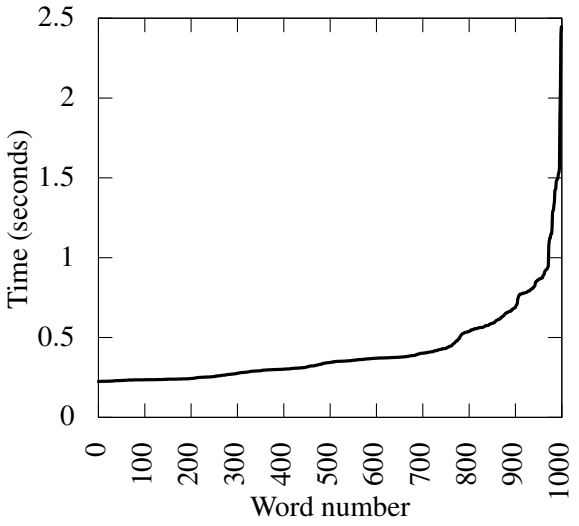

Figure 1: Runtime for 1000 common words, sorted by runtime.

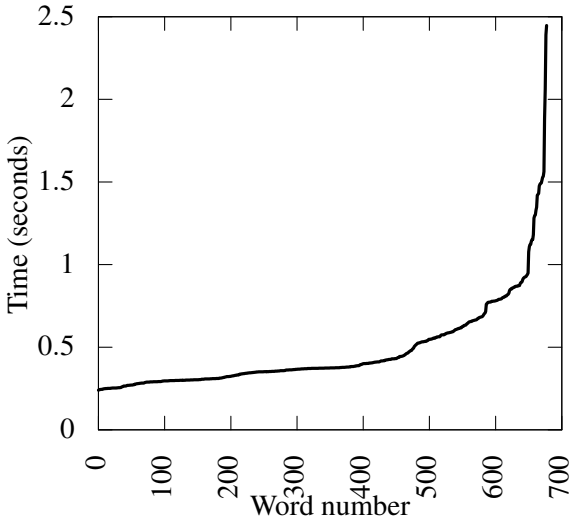

Figure 2: Runtime for the 678 of the 1000 words that were concluded to be possible cribs. Sorted by runtime.

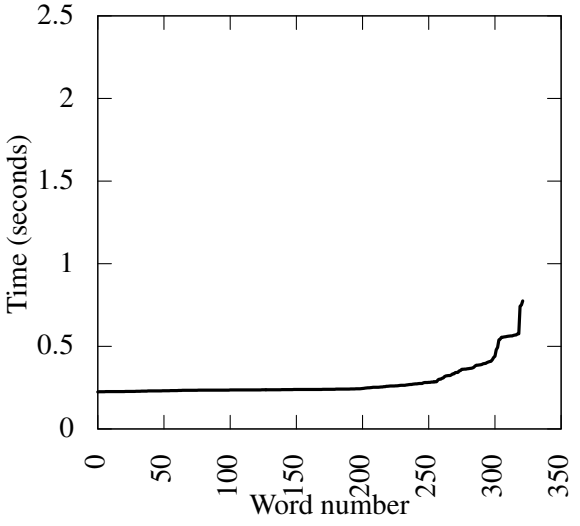

Figure 3: Runtime for the 322 of the 1000 words that were concluded to be impossible cribs. Sorted by runtime.

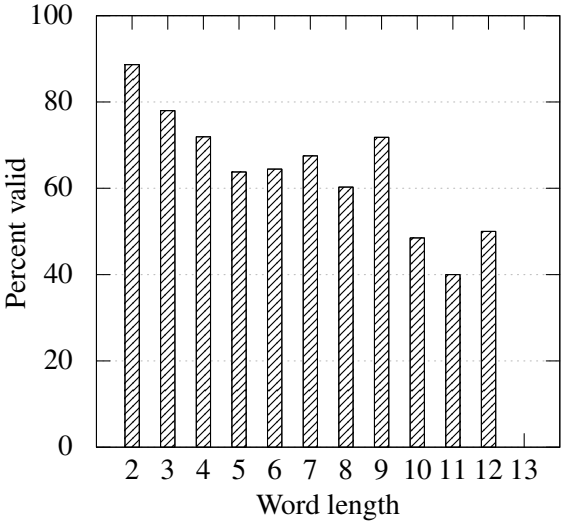

Figure 4: Histogram showing the percentage of possible cribs for all different word lengths in the 1000 common word line.

It might also be interesting to see the relationship between crib length and the chances of the crib being deemed possible. This is visualized in a histogram in figure 4 using data from the run of the 1000 common words. As can be seen, there is a clear tendency for shorter words to be possible cribs.

The longer the crib, the higher is the chance to quickly find a contradiction since the program generally has more letters to pick from. This can be seen in figure 5 where a list of 1000 long English words were used as cribs, one word at a time. The shortest word was fourteen letters and the longest twenty-two. The median word length was 15 letters. The word that took the longest time to test was `PHANTASMAGORIA`, which took 1.36 seconds to process. This word was found to be impossible as a crib, but only after processing all the bigrams of the word. Of the 1000 long words processed, only eight were found to be possible cribs.

If the whole, correct plaintext is used as a crib the program runs for 0.57 seconds before presenting a correct partial key with only three empty places in it.

In general it can be concluded that the Playfair cipher is very well suited to be used with constraint logic programming due to the nature of the key and the consequences of the three Playfair rules. The method to gradually build a set of prospective Playfair keys has been known since

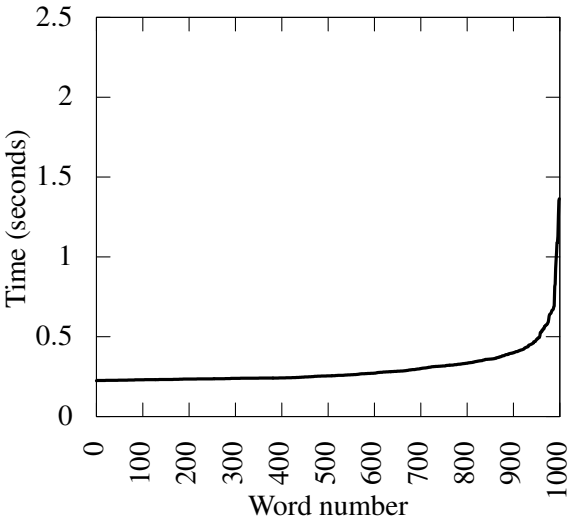

Figure 5: Runtime for 1000 long words, sorted by runtime.

before the advent of the computer. For example, this method is partially described in the novel "Have His Carcase" by Dorothy L. Sayers, published in 1932. By using state-of-the-art constraint programming software tools together with the old pen-and-paper method, things can be significantly accelerated.

Acknowledgments

The author would like to thank Henning Gideskog for his valuable comments on an earlier draft of this paper. Thanks are also due to Lisa Hallingström for drawing attention to the parallels between the method used in this study and the approach to the Playfair cipher in the novel *Have His Carcase*.

References

Elonka Dunin, Magnus Ekhall, Konstantin Hamidullin, Nils Kopal, George Lasry, and Klaus Schmeh. 2022. How we set new world records in breaking playfair ciphertexts. *Cryptologia*, 46(4):302–322.

William H. Fletcher. 2003-2010. Phrases in English. http://phrasesinenglish.org. Accessed 2025-01-04.

William F. Friedman. 1938. *Military Cryptanalysis, Part I*. War Department, Washington, Office of the Chief Signal Officer.

Tias Guns. 2019. Increasing modeling language convenience with a universal n-dimensional array, cppy as python-embedded example. In *Proceedings of the 18th workshop on Constraint Modelling and Reformulation at CP (Modref 2019)*, volume 19.

David Kahn. 1996. *The Codebreakers: The Comprehensive History of Secret Communication from Ancient Times to the Internet*. Scribner.

Laurent Perron and Vincent Furnon. 2024. OR-Tools. https://developers.google.com/optimization/. v9.10, Google.

US Government. 1990. Basic cryptanalysis, field manual no. 34-40-2. http://www.contestcen.com/ArmyFieldManual.pdf. Headquarters, Department of the Army, Washington, DC. Accessed 2024-12-29.