Automated cryptanalysis of substitution cipher using Hill climbing with well designed heuristic function

Article *in* Mathematica Montisnigri · January 2019

DOI: 10.20948/mathmon-2019-44-11

CITATIONS
2

READS
484

5 authors, including:

Luka Bulatovic
University of Donja Gorica
1 PUBLICATION **2** CITATIONS
SEE PROFILE

Andjela Mijanović
University of Montenegro
4 PUBLICATIONS **7** CITATIONS
SEE PROFILE

Vladimir Božović
University of Montenegro
16 PUBLICATIONS **147** CITATIONS
SEE PROFILE

MATHEMATICA MONTISNIGRI
Vol XLIV (2019)

AUTOMATED CRYPTANALYSIS OF SUBSTITUTION CIPHER USING HILL CLIMBING WITH WELL DESIGNED HEURISTIC FUNCTION

LUKA BULATOVIĆ*, ANĐELA MIJANOVIĆ*, BALŠA ASANOVIĆ*, NIKOLA TRAJKOVIĆ* AND VLADIMIR BOŽOVIĆ*

* Faculty of Natural Sciences and Mathematics (PMF)
University of Montenegro
Bulevar Džordža Vašingtona, 81000 Podgorica, Montenegro
e-mail: lukab@ac.me, web page: http://www.ucg.ac.me/pmf

DOI: 10.20948/mathmon-2019-44-11

Summary. In this paper, we propose new method for automated cryptanalysis of substitution cryptosystem using Hill climbing algorithm. New heuristic function is proposed in order to drastically improve overall fitness of the standard Hill climbing algorithm. Several fitness functions have been tested and we try to detremine which of them perform better in specific situations using our suggested time-dependent measure goodness. We conclude which of those functions should be used in case we are dealing with short or long texts.

1 INTRODUCTION

In classical cryptography, a substitution cipher is a very elementary method of encoding by which units of plaintext are replaced with ciphertext, according to a fixed system. The "units" may be single letters (the most common), pairs of letters, triplets of letters, mixtures of the above, and so forth. The receiver deciphers the text by performing the inverse substitution. In this article we use single letters as "units".

There are a number of different types of substitution cipher. If the cipher operates on single letters, it is known as a simple substitution cipher. A cipher that operates on larger groups of letters is called polygraphic.[2]

A	B	C	D	E	F	G	H	I	J	K	L	M
F	A	D	G	J	K	L	V	W	X	M	N	S

N	O	P	Q	R	S	T	U	V	W	X	Y	Z
P	R	O	Z	Y	I	B	C	Q	U	E	T	H

Figure 1: Example of substitution cryptosystem

Although this cryptographic method has very limited application, mainly for educational purposes, cryptographers are still interested in the problem of efficient automatic cryptoanalysis, which is mainly the subject of this paper.

The classical approach for cryptanalysis of substitution cipher is based on language properties, where frequency analysis is the main tool used for this purpose.[3,4] It is well known

2010 Mathematics Subject Classification: 94A60, 68P25, 68T20.
Key words and Phrases: substitution cipher, cryptanalysis, hill climbing, heuristics.

that, in each language, some letters tend to occur more often than the others and we use this property to analyze ciphertext.

The key of the substitution cipher is a bijection $E: L_E \rightarrow L_E$ where L_E is set of letters in alphabet (English alphabet in our case). Let $x \in L_E$ be a fixed letter from English alphabet. After encryption, it will be replaced with $E(x)$ in the ciphertext. Considering this, the inverse of this bijection would reverse the ciphertext to the original text. So obtaining this inverse could be the attackers goal.

There are various cryptanalysis techniques for automated breaking of substitution cipher and we list some of them:

- stochastic local search techniques based on n-gram Frequency Analysis or Markov models[6]
- automatic decryption include the ones based on the AI approach by Carroll[7,8]
- genetic algorithm technique
- simulated annealing attack and tabu search technique[11]

In [9] the idea is based on single letter probabilities which are iteratively improved. Quite possibly the most interesting for us is the work of Jakobsen[11], where he proposes a fast algorithm for cryptanalysis of simple substitution ciphers based on the premise where the initial key for the cipher text is guessed and then iteratively improved. In each iteration the resulting text would be "graded" to give us an idea how close to the correct key we are. This approach has a few things in common with our work when it comes to the base idea.

However, we offer a completely different algorithmic method to reach our goal. By using a deceptively simple Hill Climbing, we try to introduce a refreshingly straightforward, yet possibly very powerful and fast tool for automated cryptanalysis. We are looking for brilliant elegance in the manner of simplicity.

You can find the implementation and test the performance of our algorithm at the following URL: http://www.vladimirbozovic.net/univerzitet/subst-crypto/crypto-subst.html.

2 HILL CLIMBING

Hill climbing is a standard search technique[5]. The algorithm is initialized with a random key. In each step of the algorithm, the current key is evaluated by calculating fitness function of the ciphertext decrypted using that key. That tells us how well this key decrypts our ciphertext according to language properties – higher fitness means “better” key. We then try to improve the current key by swapping the images of two randomly chosen letters (each with the same probability) in a key. The obtained key is also evaluated by decrypting ciphertext and calculating fitness function. If the fitness is bigger than the currently best fitness, we use that key in the following steps of the algorithm. Otherwise, we discard that key and continue executing the steps of the algorithm. If we fail to improve the fitness for T consecutive steps, the algorithm stops and the best obtained key represents the best found solution of the algorithm. T denotes the threshold and is a parameter which we define at the beginning of the algorithm.

Since this algorithm often gets stuck in local maxima, we repeat the whole process for a predefined number of iterations, and we pick the best key in all of them as a solution to our problem. The pseudocode is given here:

```
for a pre-selected number of iterations do
key = getRandomPermutation(0,25)
```

```
        bestIterationKey = key #currently best it. key
        fitnessOfkey = getFitness(decrypt(cipherText, key))
        while true do
                newKey = generateNewKey()
                fitOfNewKey = getFitness(decrypt(cipherText,newKey))
                if fitOfNewKey > fitnessOfKey then
                        #new key and its fitness are memorized
                        #as the best ones for this iteration
                        fitnessOfKey = fitOfNewKey
                        bestIterationKey = newKey
                end if
                #break if better key hasn't been found in T steps
        end while
        #Update best ever key if bestIterationKey is better
end for
```

We use the following fitness function:

$$Fitness[red] = (1 - \sum_{i=1}^{26}\{|SF[i] - DF[i]| + \sum_{j=26}^{26}|SDF[i,j] - DDF[i,j]|\}/4)^8 \quad (1)$$

(taken from [1]) based on n-gram Frequency Analysis, where $SF[i]$ is a standard frequency of letter i in English language, and $DF[i]$ is a frequency of letter i in a decrypted text. SDF and DDF are frequencies of bigrams in English language and decrypted text, respectively. We see that decrypted text, which has frequencies close to those of English language has fitness close to 1, while texts that have larger differences in frequencies have lower fitness. The fitness value is between 0 and 1.

In Figure 2, we observe how mean fitness in first x iterations behaves, for all $x \in [1,100]$, using two different threshold values ($T = 350, T = 600$). We can see that increasing threshold gives us better fitness, since the algorithm has more attempts to fix the key in each iteration.

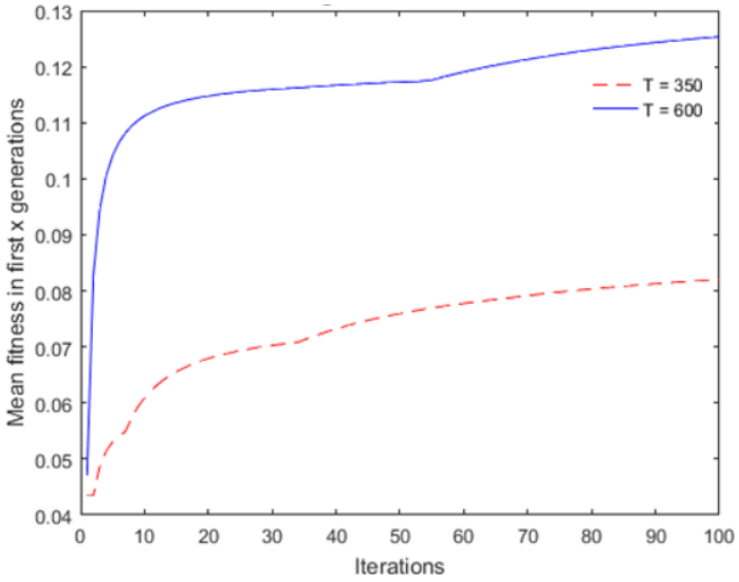

Figure 2: Hill climbing with different thresholds

3 HILL CLIMBING WITH HEURISTIC FUNCTION

After further analyzing the previously mentioned fitness function, we hypothesized that in certain conditions results could be far more precise. We try to achieve this by introducing a certain modification to the already used fitness function for a specific set of circumstances. These circumstances are actually some text characteristics which could have a great impact on decryption process. Text length was the main characteristic observed for these modification.

We now modify the hill climbing search by adding a heuristic function. Let $\mathbf{L_E} = \{\mathbf{a_1}, \mathbf{a_2}, \dots, \mathbf{a_M}\}$ be an alphabet of a language (English language in our case), such that $\boldsymbol{SF}[\mathbf{a_i}] < \boldsymbol{SF}[\mathbf{a_j}], \boldsymbol{i} < \boldsymbol{j}$. We define function $\mathbf{F_E}: \mathbf{L_E} \rightarrow \mathbf{N}$ such that:

$$\mathbf{F_E}(\mathbf{a_i}) = \mathbf{i}, \mathbf{i} \in [\mathbf{1}, \mathbf{M}] \tag{2}$$

Similarly, let $\mathbf{L_C} = \{\mathbf{b_1}, \mathbf{b_2}, \dots, \mathbf{b_K}\}, \mathbf{L_C} \subseteq \mathbf{L_E}$ be a set of letters occuring in the ciphertext, such that: $\boldsymbol{DF}[\mathbf{b_i}] < \boldsymbol{DF}[\mathbf{b_j}], \boldsymbol{i} < \boldsymbol{j}$. We define function $\mathbf{F_C}: \mathbf{L_C} \rightarrow \mathbf{N}$ such that:

$$\mathbf{F_C}(\mathbf{b_i}) = \mathbf{i}, \mathbf{i} \in [\mathbf{1}, \mathbf{K}] \tag{3}$$

Let $\boldsymbol{b} \in \mathbf{L_C}$. We define heuristic function $\mathbf{H}: \mathbf{L_C} \rightarrow \mathbf{N}$ such that:

$$\mathbf{H}(\mathbf{b}) = \mathbf{max}(\mathbf{1}, |\mathbf{F_C}(\mathbf{b}) - \mathbf{F_E}(\mathbf{E^{-1}}(\mathbf{b}))|) \tag{4}$$

Let:

$$\mathbf{P}(\mathbf{b}) = \frac{\mathbf{H}(\mathbf{b})}{\sum_{\mathbf{k} \in \mathbf{L_C}} \mathbf{H}(\mathbf{k})}, \forall \mathbf{b} \in \mathbf{L_C} \tag{5}$$

Function P represents probability distribution that we use to choose the element which will be swaped in a key.

Namely, we randomly choose $\boldsymbol{b} \in \mathbf{L_C}$ according to *P*, $\boldsymbol{a} = \mathbf{E^{-1}}(\mathbf{b})$. Next, we choose $\boldsymbol{d} \in \mathbf{L_C}$ from the uniform distribution, $\boldsymbol{g} = \mathbf{E^{-1}}(\mathbf{d})$. We try to improve fitness with newly generated key $\boldsymbol{E_1}$, such that:

$$\begin{aligned} \mathbf{E_1}(\mathbf{x}) &= \mathbf{E}(\mathbf{x}), \forall \mathbf{x} \in \mathbf{L_E} \backslash \{\mathbf{a}, \mathbf{g}\} \\ \mathbf{E_1}(\mathbf{a}) &= \mathbf{d} \\ \mathbf{E_1}(\mathbf{g}) &= \mathbf{b} \end{aligned} \tag{6}$$

Now, let's consider $\boldsymbol{d} \in \mathbf{L_C}, \mathbf{b} = \mathbf{E}(\mathbf{a})$. Letter $\boldsymbol{b}$ will be chosen with a large probability if the difference between frequency $\boldsymbol{SF}[\mathbf{a}]$ and $\boldsymbol{DF}[\mathbf{b}]$ is large.

As a result, the algorithm will quickly minimize these differences. As the length of ciphertext increses, these differences are expected to be smaller.

According to our simulations, using heuristic function gave us better overall fitness. This shows us that Hill climbing, when used with this heuristic function, requires less iterations and smaller threshold in order to find almost perfect solution. In Figure 3, we compare the two techniques with two different thresholds, and observe how mean fitness in first x iterations behaves, for all $\mathbf{x} \in [\mathbf{1}, \mathbf{100}]$. We also see that the overall fitness is higher in the situation where the threshold is higher (for both algorithms).

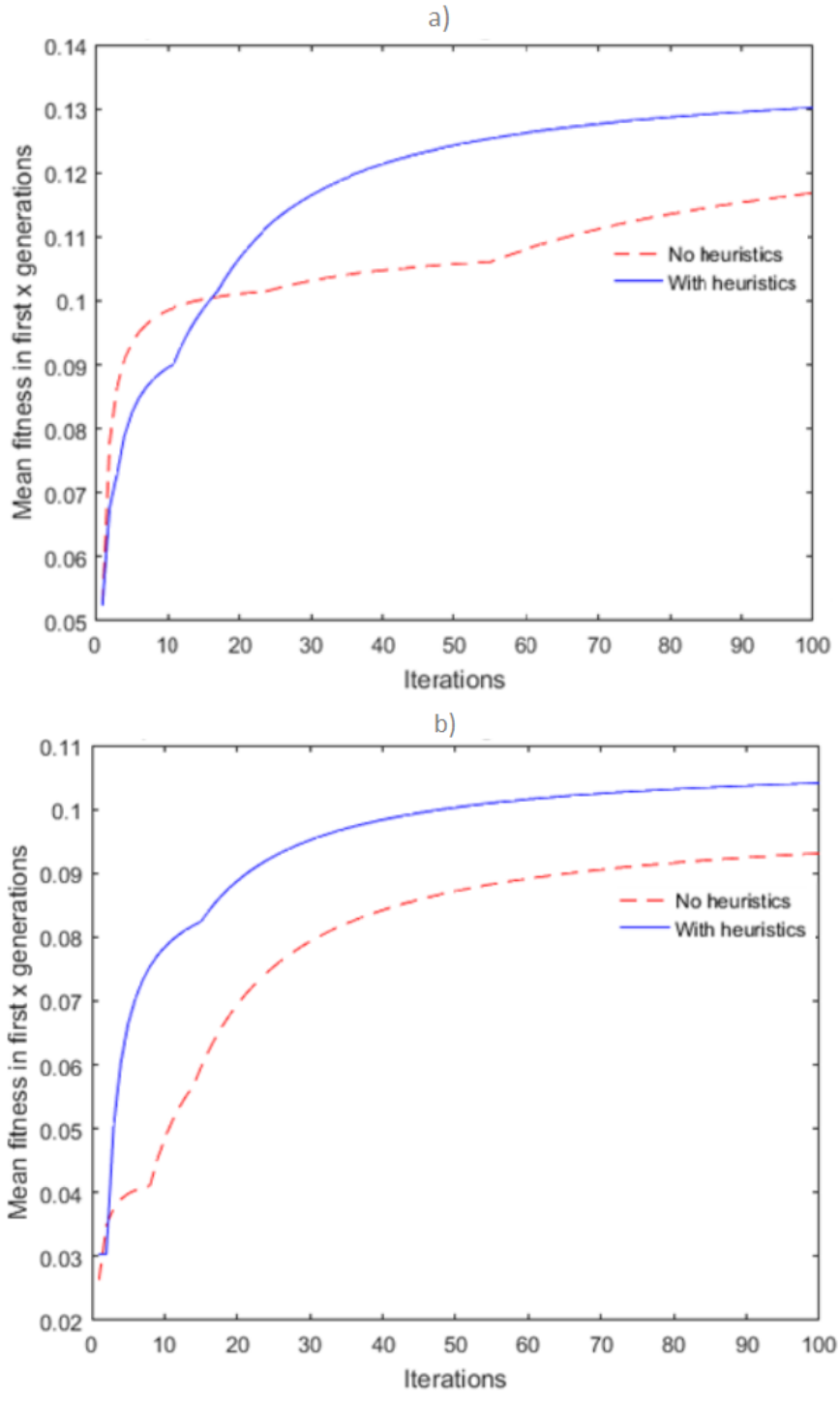

Figure 3: Comparison between Hill climbing with and without H: a) T=600; b) T=350

4 CHOOSING FITNESS FUNCTION

Nearly all fitness functions used to evaluate how well a particular key performs rely on some form of Frequency Analysis. The fundamental question when relying on Frequency Analysis is how much does the text obtained by decryption resemble English? A formal process of Frequency Analysis allows us to quantify how much a particular text looks like the language in which the plaintext was written.

We attempt to evaluate how well fitness functions perform, according to some measure. We define the distance between two keys as the number of positions at which they differ. In order to define efficiency of fitness function we introduced a time-dependent measure ”goodness”:

$$\boldsymbol{Goodness(t)} = \frac{\mathbf{1}}{\boldsymbol{t * d(t)}} \tag{7}$$

where $\boldsymbol{d(t)}$ is the distance between the best key until moment $\boldsymbol{t}$ and the original key used for encryption. We consider fitness function good if it finds the key close to the original one in a small number of iterations.

In order to determine which fitness function is good for a particular case, we tested fitness functions defined below. We ran simulations on 50 long-text and 50 short-text examples and calculated cumulative goodness measure at each time point, ie. iteration.

$$Fitness[cyan] = 1 - log_2\{\sum_{i=1}^{26}(|SF[i] - DF[i]| + 0.3\sum_{j=26}^{26}|SDF[i,j] - DDF[i,j]|)\} \tag{8}$$

$$Fitness[black] = 1 - log_2\{\sum_{i=1}^{26}(|SF[i] - DF[i]| + 0.5\sum_{j=26}^{26}|SDF[i,j] - DDF[i,j]|)\} \tag{9}$$

$$Fitness[green] = 1 - log_2\{\sum_{i=1}^{26}(|SF[i] - DF[i]| + \sum_{j=26}^{26}|SDF[i,j] - DDF[i,j]|)\} \tag{10}$$

$$Fitness[blue] = 1 - log_{10}\{\sum_{i=1}^{26}(|SF[i] - DF[i]| + \sum_{j=26}^{26}|SDF[i,j] - DDF[i,j]|)\} \tag{11}$$

The constant in front of the sum for bigrams mitigates the influence of that sum. The sum is bigger for short texts due to irregularity in bigrams frequency, while it gets smaller as the length of text increases.

4.1 Results for short-text examples

In Figure 4, we observe the results for short-text examples (up to 20 characters each). We can see the average value of goodness measure in each iteration. From the figure, it is clear that among the functions with log_2 term, the ones that have higher constant in front of sum for bigrams perform better and have higher average goodness. It seems that amplifying the influence of this sum forces Hill climbing algorithm to quickly find better solutions. Function with log_{10} term performs similar to the best one among those with log_2 term. These two functions have higher average value of goodness than the function used in Section 2.

We can also see that the overall goodness for short-text examples mostly decreases with the number of iterations, since the algorithm is unable to make any significant improvements after first few iterations, due to high irregularities in frequencies in short texts.

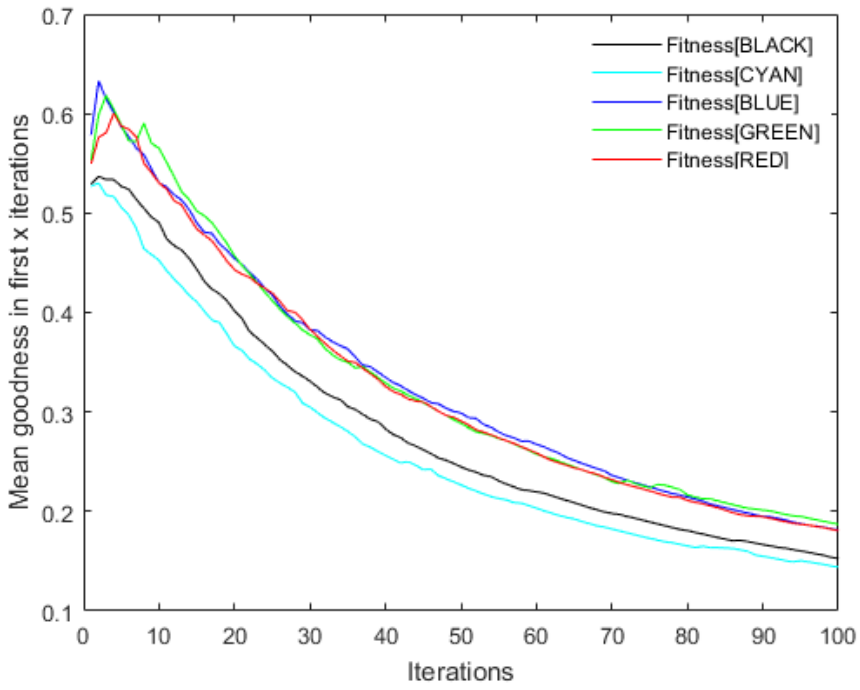

Figure 4: Average goodness of fitness functions in short-text examples

4.2 Results for long-text examples

In Figure 5, we observe the results for long-text examples (up to 500 characters each). We can see that the overall goodness is mostly increasing with the number of iterations, since the algorithm is able to find better solutions during these 100 iterations, due to more regularities in terms of frequencies of letters and bigrams.

In contrast with the short-texts case, functions with lower constant in front of sum for bigrams among fitness functions with $\boldsymbol{log_2}$ term perform better (have higher average goodness). It seems that weakening the influence of bigrams sum gives better goodness during decryption of long texts. These $\boldsymbol{log_2}$ functions also perform better than the function from Section 2, while the $\boldsymbol{log_{10}}$ function has lower goodness compared to this function.

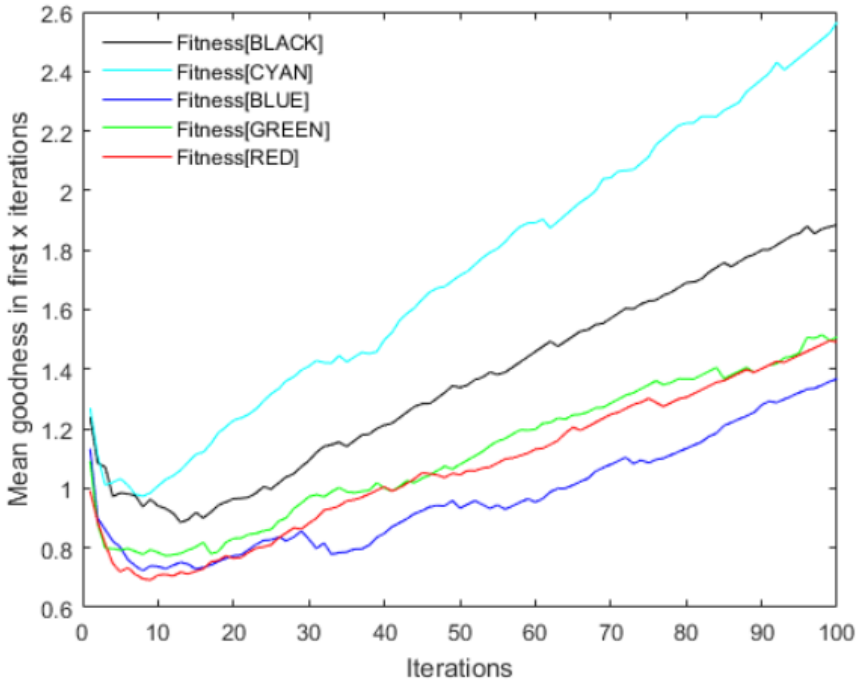

Figure 5: Average goodness of fitness functions in long-text examples

5 CONCLUSIONS

We explored the ability of Hill climbing technique to search through a space of all possible keys for the simple substitution cryptosystem. It performed well and was able to decrypt any English language text we tried (even shorter ones, which have larger deviation in terms of letter and bigram frequencies from standard English language frequencies). We used our heuristic function, which showed great increase in overall fitness, and thus enables Hill climbing to be even more successful in finding solution with less number of iterations and smaller threshold. We also showed how different fitness functions performed when applied to long texts and short texts, using time-dependent measure goodness, as well as showed which ones should be used in particular situation.

All this makes it possible to automatically decrypt substitution ciphertext in a reasonable amount of time with great precision.

REFERENCES

[1] R. Spillman, M. Janssen, B. Nelson, M. Kepner, "Use of a genetic algorithm in the cryptanalysis of simple substitution ciphers", *Cryptologia*, **1**(17) pp.31-44 (1993).

[2] Wikipedia, "Substitution cipher"

[3] B. Ycart, "Letter counting: a stem cell for Cryptology, Quantitative Linguistics, and Statistics", *Historiographia Linguistica,* **40** (3), 303-329 (2013).

[4] J. Grime, *An Introduction to Cryptography*, Enigma Project (2015).

[5] J. R. Stuart, P. Norvig, *Artificial Intelligence: A Modern Approach*, Prentice Hall (1994).

[6] S. Hasinoff, "Solving Substitution Ciphers", A Technical Report, Department of Computer Science, University of Toronto (2003).

[7] J. M. Carroll, L. Robbins, "The Automated Cryptanalysis of Polyalphabetic Ciphers", *Cryptologia,* **11**, 193-205 (1987).
[8] J. M. Carroll, L. Robbins, "Computer cryptanalysis of product ciphers", *Cryptologia,* **13**, 303-326 (1989).
[9] S. Peleg, A. Rosenfeld, "Breaking Substitution Ciphers using a Relaxation Algorithm", *Communications of the ACM,* **22,** 598-605 (1979).
[10] T. Jakobsen, "A fast method for cryptanalysis of substitution ciphers", *Cryptologia,* **19**, 265-274 (1995).
[11] A. J. Clark, "Optimisation Heuristics for Cryptology", PhD thesis (1998).

Received January 16, 2019