Using Local Search Algorithms for Cryptanalysis of Playfair Cipher

Reinhard Denis Najogie | 13509097[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]*reinhard.denis@s.itb.ac.id*

***Abstract*—Cryptanalysis methods for playfair cipher rely on guessing and manual labors. These usually include counting word frequencies (n-gram), matching possible pairs of ciphertext and plaintext, etc. In short, this is a trial and error method and doesn't guarantee the decipherment will be successful. However, if we can give some limitation on the problem context, like short ciphertext, there are actually several algorithms that have been used to decipher playfair cipher. These algorithms mainly used in artificial intelligence problems, specifically local search and optimization problems. Some of the algorithms used in local search and optimization problems are hill climbing, simulated annealing and genetic algorithms. In this paper, we will try to use these algorithms to decipher ciphertext encrypted using playfair cipher.**

***Index Terms*—playfair cipher, local search, hill climbing, simulated annealing, genetic algorithms.**

I. Introduction

Classical cryptography methods often consist of letter substitutions. Playfair cipher is one of them. To make the matter of cryptanalysis harder, playfair cipher uses bigram (two-letters) substitution. This technique make it harder to crack the encryption since the single-letter frequency of the plaintext is well hidden now. The bigram frequency still can be helpful for cryptanalysis, though.

Since the invention of playfair cipher, methods for its cryptanalysis began to spread. Because cryptography era came quite long before computer era, methods that have been developed are methods that involved trial and error by hand. Cryptanalyst are forced to count the bigram frequency, make some guesses about the bigram substitution, the key that used to encrypt, etc. A handy guide on how to decrypt the playfair cipher this way has been published on [1].

Instead of doing trial and error by hand, in this paper we will explore smarter ways of doing the trial and error, i.e. by using local search algorithms. This paper is motivated by the work in [2], where simulated annealing algorithm was used to decrypt a short playfair ciphers (80-120 letters) without using a probable word. In addition of this technique, we will also implement genetic algorithm to decrypt playfair ciphers and compare their performance, i.e. the time required to correctly decrypt playfair ciphers.

II. Background Theory

A. The Playfair Cipher

The playfair cipher was named after Lord Playfair who promoted its use, albeit the original inventor was Wheatstone.

Playfair cipher uses a 5 x 5 square as a place to write the key and later to do the actual encryption process. The way key is used to encrypt in playfair cipher is unique. Say we want the word "UNIQUE" as the key. The 5 x 5 square will be like this:

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

Figure 1 – 5 x 5 square for key UNIQUE

From Figure 1 above, we can understand the method for filling the 5 x 5 square. That is, first to write the key without repetition of letters already present in the square. Next, we just continue filling alphabets from A-Z again not already present in the table and is not J (we assumed I=J to fit the alphabets in the 5 x 5 square).

Next, the plaintext will be encrypted bigramly (two by two) using the following algorithm described in [2]:

1. When both letters appear in the same row, replace them with letters directly right of them (wrap around for corners). For the example square above, "NQ" will be enciphered as "IE".
2. When both letters appear in the same column, replace them with letter directly below them (wrap around for corners). For the example square above, "IK" will be enciphered as "CR".
3. When both letters are in different row and column, replace them with letters that will form a rectangle edges.

For more practical explanation, refer to [2].

B. Local Search Algorithms

Local search algorithms are different from ordinary[1] search algorithms in the way they find the solution in the state space. In ordinary search, we are interested not only on the final solution or destination, but also on the path required to travel to the solution. In local search, we are interested only on the final solution. The way this is done is by evaluating and modifying current state(s) (local search) rather than systemically exploring paths from an initial state (ordinary search). This is explained in [3].

Some algorithms that frequently used to solve local search problems are hill climbing, simulated annealing, and genetic algorithms.

Hill climbing is easier to explain using the following figure:

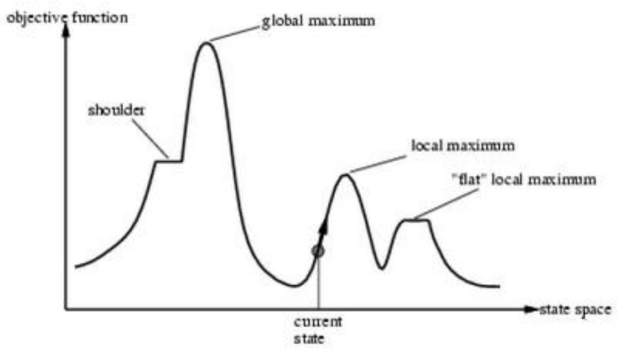

Figure 2 – Hill climbing illustration. Figure taken from [3]

In Figure 2, what a hill climbing algorithm will do is to go increasing the value of objective function until it hit the local maximum. In other words, hill climbing algorithm will stop just after it find a downhill. Of course this will not guarantee the best solution, but often times it is good enough.

It is known that the hill climbing often failed to get the best solution. The reason that it failed that it often stuck at the local maximum, where the real key is on the global maximum.

The pseudocode for hill climbing algorithm can be seen below:

```
function hill-climbing(problem) returns a state that is a local maximum
  current ←make-node(problem.INITIAL-STATE)
  loop do
    neighbor ← a highest-valued successor of current
    if neighbor.VALUE <= current.VALUE then return current.STATE
    current ← neighbor
```

Figure 3 – Hill climbing pseudocode. Source: [3]

To overcome the limitation of hill climbing algorithm, simulated annealing algorithm is used. Simulated annealing is similar to hill climbing algorithm. A hill climbing algorithm never makes downhill moves, whereas simulated annealing can do this move occasionally, with the intention of reaching the global maximum. A more thorough explanation of the logic behind simulated annealing can be found on [3].

The simulated annealing algorithm pseudocode can be seen below:

```
function simulated-annealing(problem,schedule) returns a solution state
  inputs: problem, a problem
  schedule, a mapping from time to "temperature"

  current ← make-node(problem.INITIAL-STATE)
  for t = 1 to INF do
    T ← schedule(t)
    if T = 0 then return current
    next ← a randomly selected successor of current
    ΔE ← next.VALUE - current.VALUE
    if ΔE > 0 then current ← next
    else current ← next only with probability e^E/T
```

Figure 4 – Simulated annealing pseudocode. Source: [3]

Another popular local search algorithm is genetic algorithm. Unlike simulated annealing, genetic algorithm uses "parents" to generate "child" states. In the beginning the algorithm will generate a population consist of individuals. Then these individuals will be selected randomly to generate the child state. At small probability, there will be mutation of the child state. This process is iterated until some individual is fit enough, or enough time elapsed. The pseudocode of this algorithm can be seen below:

```
function genetic-algorithm(population, FITNESS-FN) returns an individual
  inputs: population, a set of individuals
        fitness-fn, a function that measures the fitness of an individual

  repeat
    new_population ← empty set
    for i = 1 to size(population) do
      x ← random-selection(population, fitness-fn)
      y ← random-selection(population, fitness-fn)
      child ← reproduce(x,y)
      if(small random probability) then child ← mutate(child)
      add child to new_population
    population ← new_population
  until some individual is fit enough, or enough time has elapsed
  return the best individual in population, according to
```

[1] Ordinary here means that the search problem exhibits the following characteristics: observable, deterministic, and known environments. For more information check [3].

```
fitness-fn

function reproduce(x,y) returns an individual
  inputs: x,y,parent individuals

  n ← length(x)
  c ← random number from 1 to n
  return append(substring(x,1,c),substring(y,c+1,n))
```

Figure 5 – Genetic algorithm pseudocode. Source: [3]

C. Quadgrams Statistics

One of the technique often used in breaking classical cipher is using letters frequency analysis. For playfair cipher, we can make use of quadgram statistics instead of instead of unigram, bigram, or trigram.

From the work on [4], we know that quadgram frequencies work slightly better than trigrams. Hence for the implementation, we will use quadgram statistics for scoring the states on the hill climbing algorithm, simulated annealing algorithm and genetic algorithm.

III. Implementation

A. Implementation Environment and Test Cases

The programs in this paper ran using standard command line tools in Ubuntu 12.04 operating system. The programming language of choice is C, for the sake of speed.

Note the use of term like “fitness”, “similarity”, and “score” refers to the same concept. That is, how far is the guessed key by the algorithm to the real key that correctly deciphers the ciphertexts.

To simplify the experiment, we will use all-caps characters and without space, punctuation, nor any other type of characters whatsoever. The following test cases are the ciphertexts that we are going to use for the experiment:

Testcase #1[2]:
XZOGQRWVQWNROKCOAELBXZWGEQYLGDRZX
YZRQAEKLRHDUMNUXYXSXYEMXEHDGNXZYN
TZONYELBEUGYSCOREUSWTZRLRYBYCOLZYLE
MWNSXFBUSDBORBZCYLQEDMHQRWVQWAEDP
GDPOYHORXZINNYWPXZGROKCOLCCOCYTZUE
UIICERLEVHMVQWLNWPRYXHGNMLEKLRHDUY
SUCYRAWPUYECRYRYXHGNBLUYSCCOUYOHR
YUMNUXYXSXYEMXEHDGN

This test case consisted of 254 characters.

Testcase #1 solution:
THEPLAYFAIRCIPHERWASTHEFIRSTPRACTICAL
DIGRAPHSUBSTITUTIONCIPHERTHESCHEMEWA
SINVENTEDINBYCHARLESWHEATSTONEBUTWA
SNAMEDAFTERLORDPLAYFAIRWHOPROMOTED
THEUSEOFTHECIPHERTHETECHNIQUEXNCRYPT
SPAIRSOFLETXERSDIGRAPHSINSTEADOFSINGLE
LETXERSASINTHESIMPLESUBSTITUTIONCIPHER

Testcase #2:
HISZYTWQXBCADLMRTYYXFCZGBAILFHYIOAB
GILVCWYRXDAHQRAPHMYKEUDFPHISEIDOZUF
HISZYTWQXBCADLMRTYYXFCZGBAILFHYIOAB
GILVCWYRXDAHQRAPHMYKEUDFPHISEIDOZUF
HISZYTWQXBCADLMRTYYXFCZGBAILFHYIOAL
B

This test case consisted of 176 characters.

Testcase #2 solution:
THEQUICKBROWNFOXJUMPSOVERTHELAZYDO
GTHEQUICKBROWNFOXJUMPSOVERTHELAZYD
OGTHEQUICKBROWNFOXJUMPSOVERTHELAZY
DOGTHEQUICKBROWNFOXJUMPSOVERTHELAZ
YDOGTHEQUICKBROWNFOXJUMPSOVERTHELA
ZYDOG

Testcase #3:
SNHDURRYCSBSMFDCLKMDGESGCGDTPFQRMC
BVGVKLGBZXTCSMMUSPBSABITCFPRBGDVVPD
MPIHSBCPDUPGFFIRKGEARLCSBPRFCDOYQDCE
PDUCSRCWSBKDORLSVHMQYODGKKLBPQLCDH
BWBCPZIDSIKKZGKGIHPPRPRBGDVVPDMLKSY
MPSPWKEVKDKGSFKGUPVQGFVBCFGDMUGUV
HHNAGUGEUDTSMMCHQSIUPDABCMQCSGDBW
BHOGKGUPQHMHEPNFDOYQDCDPCSPYUCDOYQ
DCIPKLSMMCHQRIPFQDSFNSZBDCFGPDQCKZAR
BDBVFGRBYBSFUPIWHNQYPRSFPOFADGKGARK
YGDHDNKPR

This test case consisted of 354 characters.

Testcase #3 solution:
FIRSTLYTHESENDERANDRECEIVERMUSTAGREX
EONAKEYWORDINTHISEXAMPLETHEKEYWORDI
SWHEATSTONESNAMECHARLESTHELETXTERSO
FTHEALPHABETAREWRITXTENINASQUAREASXS
HOWNBEGINXNINGWITHTHEKEYWORDANDWIT
HIXICOMBINEDINTOXONEXELEMENTNOWCLICK
ONFORMDIGRAPHSTOBREAKTHEMESXSAGEINT
OPAIRSOFLETXTERSTHETWOLETXTERSINADIGR
APHMUSTBEDIFXFERENTSOANXHASBEXENADX
DEDTOSPLITXTHEDOUBLEMINHAMXMERSMITH

To get more accurate results, we will run the implementation of each algorithms several times, and then calculate the average to determine the time required to find the correct key for deciphering. This is because there is randomness involved in the algorithm (in swapping and generating the key).Also, due to limited time, we limit the time to 10 minutes. Above that, we consider the algorithm fail to solve the cipher.

[2] This testcase was taken from [5]. We use will use it to compare the performance with the genetic algorithm.

B. Hill Climbing

For the hill climbing algorithm, we first generate a random key as the starting state. Next we will alter this key by randomly swapping the characters in the key and measuring the score of each state. If we get a higher score, then the new state become the starting state and we start over again.

The results from hill climbing are:

Table 1 – Hill climbing results for test case #1

Run number	Time required (seconds)	Solved?
1	∞	No
2	∞	No
3	∞	No
4	∞	No
5	∞	No

Table 2 – Hill climbing results for test case #2

Run number	Time required (seconds)	Solved?
1	∞	No
2	∞	No
3	∞	No
4	∞	No
5	∞	No

Table 3 – Hill climbing results for test case #3

Run number	Time required (seconds)	Solved?
1	∞	No
2	∞	No
3	∞	No
4	∞	No
5	∞	No

C. Simulated Annealing

There are some details that need to be explained before implementing the simulated annealing algorithm. From the general algorithm described in section 2, we need to define what exactly are the input, output, and the states of the problem.

The problem we have is clear, that is, to break a playfair cipher without hints about the content or the key used. The way we are going to do this is first to start guessing with the key "ABCDEFGHIKLMNOPQRSTUVWSYZ". Actually we can just start with random string. This is just an example. From this key, next we will calculate the score, that is, the similarity of the decipherment result with a typical English passage. The way we do this is by using the quadgram statistics described earlier.

The process is repeated forever. We need to look at the output of the program to check whether a correct decipherment result has been found or not. At each loop, whenever the simulated annealing program found a better solution (bigger score that ever recorded) then it will be printed to the screen.

For brevity, in this implementation we don't code the simulated annealing from scratch. We will use the implementation in [5] and compare it with our own implementation of genetic algorithm in the next section. To better compare it, the first test case is the test case used by the work in [5].

The results from the simulated annealing are:

Table 4 – Simulated annealing results for test case #1

Run number	Time required (seconds)	Solved?
1	366.3	Yes
2	38.72	Yes
3	201.1	Yes
4	105.5	Yes
5	57.3	Yes

Table 5 – Simulated annealing results for test case #2

Run number	Time required (seconds)	Solved?
1	∞	No
2	∞	No
3	∞	No
4	∞	No
5	∞	No

Table 6 – Simulated annealing results for test case #3

Run number	Time required (seconds)	Solved?
1	131.43	Yes
2	30.36	Yes
3	31.89	Yes
4	55.32	Yes
5	169.37	Yes

D. Genetic Algorithm

The main difference between simulated annealing and genetic algorithm is in genetic algorithm we need a pair of "parents" to generate children. So, from two states we will generate a new state.

The parents will be the candidate key for the playfair cipher. Unlike in simulated annealing, in genetic algorithm we need to generate a population of parents first. After that, we will iterate to the size of population of parents and pick randomly two states. From these states we "marry" them and we get a new state. The fitness of the offspring is calculated using similarity of the resulting plaintext deciphered using this key to the quadgram statistics of English language, similar to the method employed at simulated annealing.

The results from the genetic algorithm are:

Table 7 – Genetic algorithm results for test case #1

Run number	Time required (seconds)	Solved?
1	200.22	Yes
2	103.41	Yes
3	150.51	Yes
4	78.8	Yes
5	67.73	Yes

Table 8 – Genetic algorithm results for test case #2

Run number	Time required (seconds)	Solved?
1	∞	No
2	∞	No
3	∞	No
4	∞	No
5	∞	No

Table 9 – Genetic algorithm results for test case #3

Run number	Time required (seconds)	Solved?
1	210.79	Yes
2	105.90	Yes
3	20.17	Yes
4	60.54	Yes
5	40.32	Yes

IV. Analysis

A. Analysis for Hill Climbing

We found that the results from hill climbing algorithm are unsatisfactory. This is due to the nature of hill climbing that will only find local maximum solution. Using this algorithm, when it found the local maximum it will just exit and think it is the best solution.

This results match with the theory that hill climbing hardly finish with the expected solution. Similar results where hill climbing couldn't find solution for playfair ciphers were reported on [2].

B. Analysis for Simulated Annealing

We found the results from simulated annealing were good. Most of the time it can find the solution within the constrained time.

However it failed to give the correct solution for characters that don't exhibit English letters frequencies. For example for repeated sentence in test case number 2, it failed to find the expected result even though the sentence consisted of legit English words.

C. Analysis for Genetic Algorithm

We found that the results from simulated annealing were good. It can be said that the results are equally good with simulated annealing, because the performance difference is not really big, and it suffers similar problem as simulated annealing (bad on sentences that don't exhibits English letter frequencies). Although the problem nature doesn't really match for the use of genetic algorithm (genetic algorithm heavily used in optimization problem), it still can produce a good results, even slightly better than the simulated annealing. This results proved that genetic algorithm is a good general-purpose algorithm for any problem that can be presented as a searching problem.

The only drawback of genetic algorithm is in that no one really knows why genetic algorithm can produce good results. It is not clear whether the good results come from their performance or from the origin of evolution theory. This problem is stated on [3]. For scientific purposes the mathematical explanation of the genetic algorithm is still lacking.

D. Summary

The performance comparison of the three algorithms can be seen below:

Table 10 – Summary of local search algorithm performances

Test Case	Algorithm	Average seconds needed
1	Hill Climbing	∞
	Simulated Annealing	153.784
	Genetic Algorithm	120.134
2	Hill Climbing	∞
	Simulated Annealing	∞
	Genetic Algorithm	∞
3	Hill Climbing	∞
	Simulated Annealing	83.674
	Genetic Algorithm	87.544

Thus, it can be inferred that hill climbing is out of the choice for cryptanalysis of playfair cipher, while simulated annealing and genetic algorithm performed almost equally good.

V. Conclusion

We found that of all the local search algorithms we used to break playfair ciphers, hill climbing is the worse. Apparently hill climbing is not a good choice for searching the solution of a cipher due to the nature that the key used for encryption usually lie on the global maximum point of the search states.

We also observed that longer ciphertext doesn't mean longer time to solve. This is shown by results from test case 1 and test case 3. Longer text actually will have more similarity to the frequency of English letters.

Simulated annealing is a good choice of algorithm for breaking the playfair cipher. The theory works well in practice although sometimes it can take a long time to find the solution due to randomness involved in the swapping of the characters in the key.

Genetic algorithm is also a good choice for breaking the playfair cipher. The performance in the experiment in this paper is similar to those of simulated annealing, but genetic algorithm usage is very broad and thus the result is quite appealing.

It also important to note that the solution key found using simulated annealing or genetic algoritms are not necessarily unique (i.e. there are some keys that can lead to correct deciphering of the ciphertext).

VI. Future Works

Future works could be done on optimizing the performance of simulated annealing and genetic algorithm.

The work in this paper exhibits limitation of the number of characters in the ciphertexts. The algorithms implemented are only capable of solving small input consisting of 100-300 characters. Therefore, future work on solving larger size of ciphertexts still can be done.

Another concern from our experiment is the running time of the algorithm. Future work may use the power of parallel computation through multiple computers or using the power of GPU to improve the running of the algorithms for solving the playfair ciphers.

The accuracy of the results on this paper is also not really high due to limited time to do experiment. For more accurate results, future works should be done using more test cases and more various type of test cases to compare the performance of these algorithms.

Lastly, the local search algorithms can be tinkered to solve other type of cryptographic scheme. For other classical cryptographic methods this seems not really hard to explore, but for modern cryptographic methods it might be a challenging task.

VII. Acknowledgment

The author thanks Dr. Rinaldi Munir as the lecturer of Cryptograhpy class in Informatics Engineering Institut Teknologi Bandung for giving the assignment so that this work is possible.

References

[1] DEPARTMENT OF THE ARMY. "Basic Cryptanalysis", FM 34-40-2, FIELD MANUAL,1990, ch. 7.
[2] M. J. Cowan, "Breaking Short Playfair Ciphers with the Simulated Annealing Algorithm", Cryptologia. Vol. 32, Iss. 1, 2008.
[3] S. J. Russel, et al, "Artificial intelligence: a modern approach", Upper Saddle River, NJ: Prentice hall, 2010, ch. 4.
[4] James Lyons, "Quadgrams Statistics as a Fitness Measure", http://practicalcryptography.com/cryptanalysis/text-characterisation/quadgrams/ - Accessed March 22nd 2013
[5] James Lyons, "Cryptanalysis of the Playfair Cipher", http://practicalcryptography.com/cryptanalysis/text-characterisation/quadgrams/ - Accessed March 22nd 2013

Statement

I hereby declare that this paper is the product of my own work, not an excerpt, translation, nor plagiarism.

Bandung, March 23 2013

[illegible]

ttd

Reinhard Denis Najogie | 13509097